Machine Learning Fairness Algorithms Evaluation

(STAT 4243 Project 4 - Group 7)

As machine learning is growing in popularity and being used more frequently in decision-making, there is a growing concern that some algorithms are unfairly and disproportionately predicting a certain outcome towards a particular group of people. More importantly, is that this unfair treatment may hurt or benefit this group of people due to their sensitive attributes (for example, race and sex).

As such, our goal is to implement algorithms from the assigned papers built to maximize accuracy and fairness. And then evaluate the performance of the algorithms. The evaluation metrics are based on the trade-off between accuracy and fairness in prediction, which the accuracy and calibration score will illustrate.

Lastly, the data set used to train and test the model in this project is the COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) data set. The data set contains has a variety of features ranging from criminal history to demographics with recidivism as the prediction lable. Our goal is build a model that accurately predicts recidivism without any discrimination based onf race or gender.

Importing Dependencies and Reading in Data

In [1]:
```python
import os, sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from dython.nominal import associations
from mlxtend.plotting import plot_decision_regions
from itertools import chain
import math
from random import seed, shuffle
from scipy.optimize import minimize
from multiprocessing import Pool, Process, Queue
from collections import defaultdict
from copy import deepcopy
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score as accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import LogisticRegression

sys.path.insert(1, '../lib/')
```

```
import util as ut

sns.set_style('darkgrid')
%matplotlib inline

import warnings
import time
warnings.filterwarnings("ignore")
```

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

Accuracy Constraints (A3 Method)

In this section, to achieve fairness we will use an inprocessing method, where we modify the whole optimization problem. We aim to implement a convex margin-based classifier that avoids Disparate Treatment and Disparate Impact and still achieves "business necessity."

The decision-making process is unfair and has disparate treatment if its decisions are based on the subject's sensitive attribute information. It has a disparate impact if it disproportionately affects people with specific sensitive attribute values (e.g., females, blacks). It is crucial to keep in mind that more than simply ignoring sensitive attributes will be required, as models may still produce a disparate impact on outcomes - especially if a group with certain values for these attributes has been unfairly treated in the past.

The algorithm implemented are essentially leveraging the covariance between the sensitive attributes and the distance between the subjects' attribute vectors and the decision boundary of the classifier. This covariance is a proxy to the p% rule. Moreover, because it is a convex function (unlike the p% rule), we can optimize it to our needs.

This algorithm guarantees fulfilling disparate impact by maximizing fairness subject to accuracy constraints, which in simple terms, ensures no disparate impact by focusing on fairness while keeping the accuracy.

Preprocessing Data for A3

In [3]:
```
data = df[(df['race']=='African-American')|(df['race']=='Caucasian')]
data["length_of_stay"] = (pd.to_datetime(data.c_jail_out) - pd.to_datetime(data.c_jail
data = data[(data.days_b_screening_arrest <= 30) & (data.days_b_screening_arrest >= -3
data = data[data.is_recid != -1]
data = data[data.c_charge_degree != "O"]
data = data[data.score_text != "N/A"]
data.drop(['first','last','c_case_number','c_charge_desc','violent_recid','vr_charge_c
          'c_arrest_date','r_jail_out','r_jail_in','r_days_from_arrest','r_charge_desc
          'r_offense_date','r_case_number','r_charge_degree'], axis=1,inplace=True)
data['c_days_from_compas'] = data['c_days_from_compas'] .fillna(data['c_days_from_comp
data = data.dropna()
X = data.drop(['id','two_year_recid','dob','name','v_type_of_assessment','type_of_asse
X.set_index(data.id,inplace=True)
y = data[['two_year_recid']]
y.set_index(data.id,inplace=True)
```

```
X = X.drop(['c_offense_date','c_jail_in','c_jail_out','out_custody','in_custody','scre
#convert categorical variables into numeric
X['race']= X['race'].replace(['African-American','Caucasian'],[0,1.0])
X['sex'] = X['sex'].replace(['Male','Female'],[0,1.0])
X['age_cat'] = X['age_cat'].replace(['25 - 45','Less than 25', 'Greater than 45'],[0,1
X['c_charge_degree'] = X['c_charge_degree'].replace(['M','F'],[0,1.0])
X['score_text'] = X['score_text'].replace(['Low', 'High', 'Medium'],[0,1.0,2.0])
X['v_score_text'] = X['v_score_text'].replace(['Low', 'High', 'Medium'],[0,1.0,2.0])
X = X.fillna(0)


y=np.array(y)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=.1,random_state=44)
x_sensitive_tr = X_train.loc[:, 'race']
x_sensitive_te = X_test.loc[:, 'race']
X_train = X_train.loc[:, X_train.columns != 'race']
X_test = X_test.loc[:, X_test.columns != 'race']

X_train=X_train.reset_index()
X_test=X_test.reset_index()
x_sensitive_tr=x_sensitive_tr.reset_index()
x_sensitive_te=x_sensitive_te.reset_index()

X_train=X_train.drop(['id'], axis=1)
X_test=X_test.drop(['id'], axis=1)
x_sensitive_tr=x_sensitive_tr.drop(['id'], axis=1)
x_sensitive_te=x_sensitive_te.drop(['id'], axis=1)
```

Baseline Model

```
In [4]: model_base = LogisticRegression(fit_intercept=False).fit(X_train,y_train.flatten())
        base_y_train = model_base.predict(X_train)
        base_y_test = model_base.predict(X_test)

        theta_star = model_base.coef_.flatten()

        #metrics
        print('train accuracy: ' + str(model_base.score(X_train,y_train.flatten())))
        print('test accuracy: ' + str(model_base.score(X_test,y_test.flatten())))

        base_calib_train = ut.calibration(base_y_train, y_train.flatten(),x_sensitive_tr['race
        base_calib_test = ut.calibration(base_y_test, y_test.flatten(),x_sensitive_te['race'].

        print('train calibration: ' + str(base_calib_train))
        print('test calibration: ' + str(base_calib_test))
```

```
train accuracy: 0.9639225181598063
test accuracy: 0.9564270152505446
train calibration: 0.0013216642186006933
test calibration: -0.03842692232439837
```

Logistic Regression with Accuracy Constraints (A3)

As this algorithm's goal is to ensure compliance with the disparate impact by maximizing the fairness subject to accuracy constraints. The formal definitions is defined below:

The general case is using one gamma across individuals. This allows a relaxation on the accuracy

from the unconstrained case with regard to accuracy and allows the algorithm to minimize
unfairness.

A3_algo_gamma.png

```
In [5]:   for g in [0,0.001,0.01,0.05,0.1,0.15,0.2]:
              print('Using gamma = ',g)

              theta_g = ut.model_gamma(X_train,y_train.flatten(),x_sensitive_tr['race'].values,t

              #to use score
              model_gamma = LogisticRegression(fit_intercept=False)
              model_gamma.coef_ = theta_g.reshape((1,-1))
              model_gamma.intercept_ = 0
              model_gamma.classes_ = np.array([0,1.0])
              g_y_train = model_gamma.predict(X_train)
              g_y_test = model_gamma.predict(X_test)

              #metrics
              print('train accuracy: ' + str(model_gamma.score(X_train,y_train.flatten())))
              print('test accuracy: ' + str(model_gamma.score(X_test,y_test.flatten())))

              g_calib_train = ut.calibration(g_y_train, y_train.flatten(),x_sensitive_tr['race']
              g_calib_test = ut.calibration(g_y_test, y_test.flatten(),x_sensitive_te['race'].va

              print('train calibration difference: ' + str(g_calib_train))
              print('test calibration difference: ' + str(g_calib_test))
```

```
Using gamma =  0
train accuracy: 0.47893462469733655
test accuracy: 0.5032679738562091
train calibration difference: 0.13196630217683586
test calibration difference: 0.15824691841126987
Using gamma =  0.001
train accuracy: 0.47893462469733655
test accuracy: 0.5032679738562091
train calibration difference: 0.13196630217683586
test calibration difference: 0.15824691841126987
Using gamma =  0.01
train accuracy: 0.47893462469733655
test accuracy: 0.5032679738562091
train calibration difference: 0.13196630217683586
test calibration difference: 0.15824691841126987
Using gamma =  0.05
train accuracy: 0.47893462469733655
test accuracy: 0.5032679738562091
train calibration difference: 0.13196630217683586
test calibration difference: 0.15824691841126987
Using gamma =  0.1
train accuracy: 0.47893462469733655
test accuracy: 0.5032679738562091
train calibration difference: 0.13196630217683586
test calibration difference: 0.15824691841126987
Using gamma =  0.15
train accuracy: 0.47893462469733655
test accuracy: 0.5032679738562091
train calibration difference: 0.13196630217683586
test calibration difference: 0.15824691841126987
Using gamma =  0.2
train accuracy: 0.47893462469733655
test accuracy: 0.5032679738562091
train calibration difference: 0.13196630217683586
test calibration difference: 0.15824691841126987
```

From the results above, we see that the accuracy of the model decrecases from 0.95 (baseline model) to 0.55. And that the calibration has increased from 0.001 to 0.055.

Logistic Regression with Fine-Grained Accuracy Constraints - Algorithm 2 (A3)

In the previous case, we do not account for individuals of the sensitive class being predicted fairly. Thus, we produce an algorithm with a gamma specific to the individual that we can manipulate.

A3_algo_fine_gamma.png

```python
In [6]:  for g in [0.1]:

    print('Using gamma = ',g)


    theta_fg = ut.model_fg(X_train.values,y_train.flatten(),x_sensitive_tr['race'].val

    #to use score
```

```python
        model_fine_gamma = LogisticRegression(fit_intercept=False)
        model_fine_gamma.coef_ = theta_fg.reshape((1,-1))
        model_fine_gamma.intercept_ = 0
        model_fine_gamma.classes_ = np.array([0,1.0])
        fg_y_train = model_fine_gamma.predict(X_train)
        fg_y_test = model_fine_gamma.predict(X_test)

        #metrics
        print('train accuracy: ' + str(model_fine_gamma.score(X_train,y_train.flatten())))
        print('test accuracy: ' + str(model_fine_gamma.score(X_test,y_test.flatten())))

        fg_calib_train = ut.calibration(fg_y_train, y_train.flatten(),x_sensitive_tr['race
        fg_calib_test = ut.calibration(fg_y_test, y_test.flatten(),x_sensitive_te['race'].

        print('train calibration difference: ' + str(fg_calib_train))
        print('test calibration difference: ' + str(fg_calib_test))
```

```
Using gamma =  0.1
train accuracy: 0.47046004842615013
test accuracy: 0.49019607843137253
train calibration difference: 0.13672643056429867
test calibration difference: 0.16288397573860303
```

Similar to the results above, the test accuracy has also decreased to 0.49, with the calibration increasing to 0.05. This is not particularly ideal as the test accuracy has dropper significanlty.

Handling Conditional Discrimination (A6 Method)

In A6, instead of using an in-processing method to deal with discrimination like we did above, we are going to use pre-processing methods. The idea is to balance the data set before using it to train the model. To be more specific, we implemented the local massaging and local preferential sampling algorithm as pre-processing methods below.

Preprocessing Data for A6

A6 Method

```python
In [7]:  data = df[(df['race']=='African-American')|(df['race']=='Caucasian')]
         data.drop(['violent_recid','vr_charge_degree','vr_case_number','vr_offense_date','vr_c
                  'c_arrest_date','r_jail_out','r_jail_in','r_days_from_arrest','r_charge_desc
                  'r_offense_date','r_case_number','r_charge_degree'], axis=1,inplace=True)
         data['c_offense_date'] = pd.to_datetime(data['c_offense_date'])
         data['c_jail_in']= pd.to_datetime(data['c_jail_in'])
         data['c_jail_out']= pd.to_datetime(data['c_jail_out'])
         data['out_custody']= pd.to_datetime(data['out_custody'])
         data['in_custody']= pd.to_datetime(data['in_custody'])
         data['screening_date']= pd.to_datetime(data['screening_date'])
         data['compas_screening_date']= pd.to_datetime(data['compas_screening_date'])
         data['v_screening_date']= pd.to_datetime(data['v_screening_date'])
         data['c_days_from_compas'] = data['c_days_from_compas'] .fillna(data['c_days_from_comp
         X = data.drop(['id','two_year_recid','c_case_number','sex','dob','name'], axis=1)
         X.set_index(data.id,inplace=True)
         af = data[data.race=='African-American']
         ca = data[data.race =='Caucasian']
```

```python
rate_af = round(af[af['two_year_recid']==1].shape[0]/af.shape[0],2)
rate_ca = round(ca[ca['two_year_recid']==1].shape[0]/af.shape[0],2)
print(f'The rate of Recidivism for African-American is {rate_af}\nThe rate of Recidivi
print(f'The Corrected Recidivism rate should be {(rate_af+rate_ca)/2:,} ')
y = data[['two_year_recid']]
y.set_index(data.id,inplace=True)
```

```
The rate of Recidivism for African-American is 0.51
The rate of Recidivism for Caucasian is 0.26
The Corrected Recidivism rate should be 0.385
```
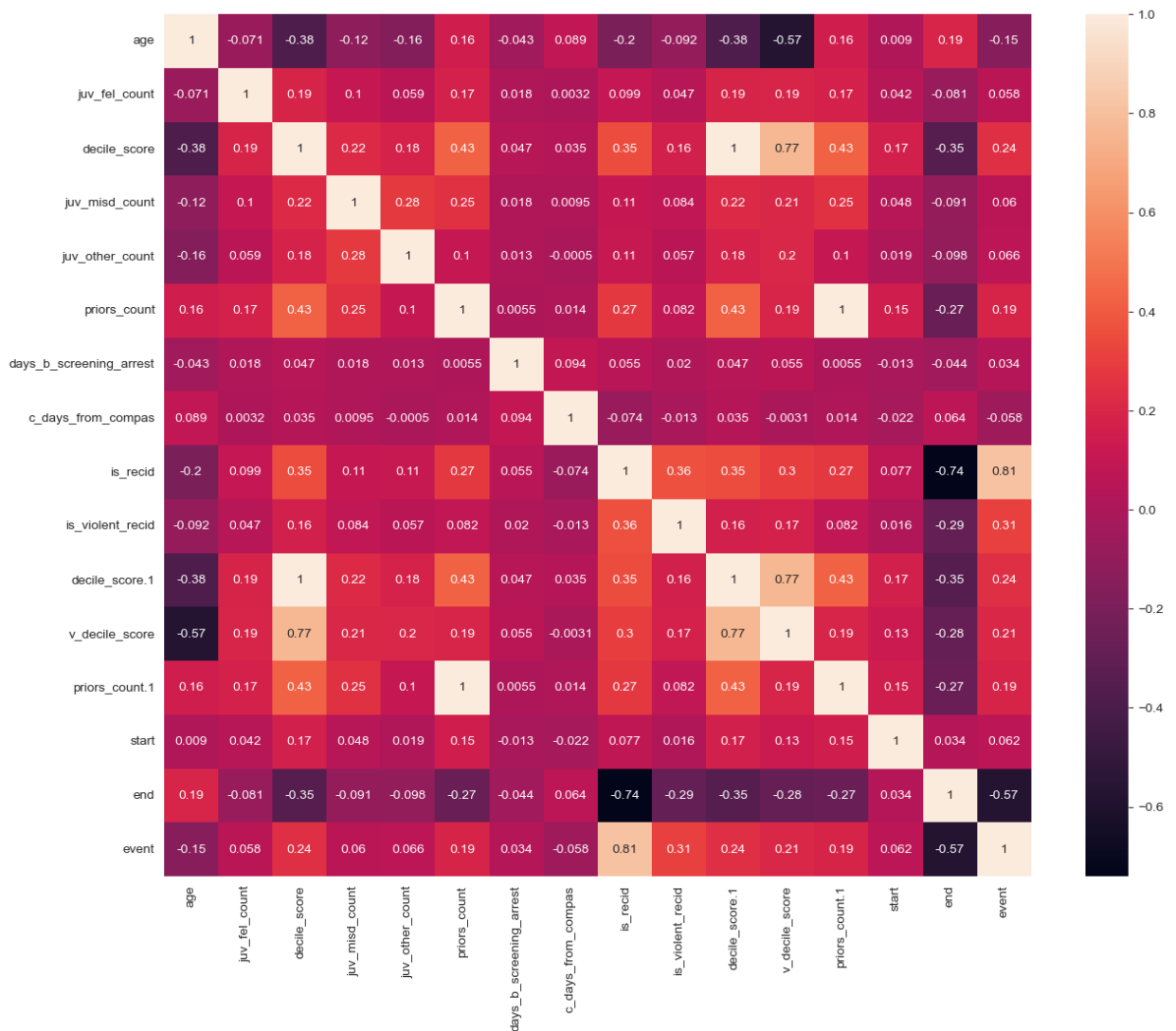
In [8]:
```python
corr = X.corr()
fig,ax = plt.subplots(figsize=(15,12))
sns.heatmap(corr, annot=True)
plt.show()
```


In [9]:
```python
enc = OneHotEncoder(handle_unknown='ignore')
X_new = enc.fit(X)
X_new = enc.transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_new,y,test_size=.3,random_state=
```

Baseline Model

In order to show the trade-off between accuracy and fairness, a baseline model is needed as a comparison to show the change. To that end, a simple random forest classfier is implemented below. The baseline random forest classifier is trained on the same set of data but not processed.

In [10]:
```python
forest = RandomForestClassifier(random_state=44).fit(X_train, y_train)
y_pred_test = forest.predict(X_test)
print(accuracy_score(y_test, y_pred_test))
print(confusion_matrix(y_test, y_pred_test))
cali_X = X.loc[y_test.index]
cali_X['label'] = y_test
cali_X['pred'] = y_pred_test
af_X = cali_X[cali_X.race=='African-American']
ca_X =  cali_X[cali_X.race=='Caucasian']
print(np.abs(accuracy_score(af_X.label, af_X.pred)-accuracy_score(ca_X.label, ca_X.pre
```

```
0.9766937669376694
[[932  43]
 [  0 870]]
0.02107992678462478
```

As we can see from the results above, the model achieves a prediction accuracy of 0.97. However, the model also illustrated that African-Americans have a higher recidivism rate compared to Caucasians. As shown by the calibration score of 0.013.

This is problematic as the data is unfairly treating the African-American when the rate should be pretty much similar. It highlights that a simple random forest classifier is likely prone to unfairness, as it seems to rely heavily on race (a sensitive attribute) as a key predictor. While at the same time, many other features can contribute to a person's recidivism rate.

To continue, a correlation table of the categorical features is displayed below:

In [11]:
```python
categorical_features = ['first','last','age_cat','race','c_charge_degree','c_charge_de
                        'score_text','v_type_of_assessment','v_score_text']
```

In [12]:
```python
selected_column= X[categorical_features]
categorical_df = selected_column.copy()
```

In [13]:
```python
categorical_correlation= associations(categorical_df, filename= 'categorical_correlati
```
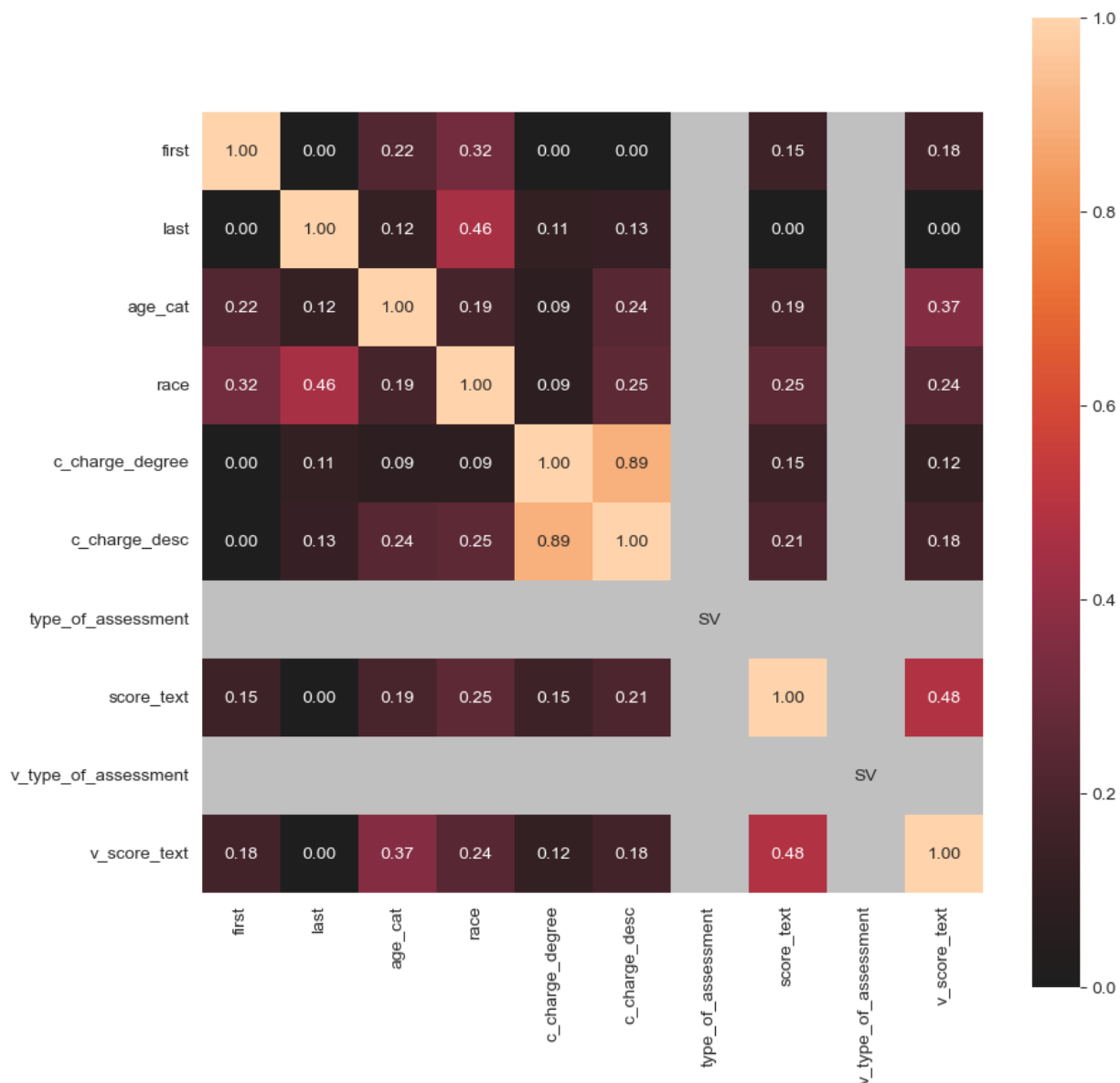
Setting the correlation threshold to be no less than 0.2, we get that e_i to be:

1) first 2) last 3) c_charge_desc 4) score_text 5) v_score_text

These are the features that would not require local massaging or local preferential sampling, while all other features will be need to be slightly pre-processed before being used to train the model.

```
In [14]:  e_list = ['first', 'last', 'c_charge_desc', 'score_text' , 'v_score_text']
          exclude_e_s = [
           'compas_screening_date',
           'age',
           'age_cat',
           'juv_fel_count',
           'decile_score',
           'juv_misd_count',
           'juv_other_count',
           'priors_count',
           'days_b_screening_arrest',
           'c_jail_in',
```

```
      'c_jail_out',
      'c_offense_date',
      'c_days_from_compas',
      'c_charge_degree',
      'is_recid',
      'is_violent_recid',
      'type_of_assessment',
      'decile_score.1',
      'screening_date',
      'v_type_of_assessment',
      'v_decile_score',
      'v_screening_date',
      'in_custody',
      'out_custody',
      'priors_count.1',
      'start',
      'end',
      'event']
e = X[e_list]
new_X = X[exclude_e_s]
s = X[['race']]
```

Local Massing - Algorithm 1 (A6)

In the localy massaging algorithm, we alter the training data to achieve our "debiasing" goal. Specifically, we relabel data points that are close to the decision boundary.

a1.jpg

```
In [15]:   New_X = ut.local_massaging(new_X,s,e,y)
```

```
In [16]:   af = New_X[New_X.race=='African-American']
           ca = New_X[New_X.race =='Caucasian']
           rate_af = round(af[af['new_label']==1].shape[0]/af.shape[0],2)
           rate_ca = round(ca[ca['new_label']==1].shape[0]/ca.shape[0],2)
```

```
In [17]:   rate_af,rate_ca
```

```
Out[17]:   (0.46, 0.44)
```

After implementing algorithm 1, to local massage the non-e_i features, we see that the discrimation is being balanced. With the proportion of African-American and Causcasian with the same label being pretty much the same.

```
In [18]:   New_y = New_X.new_label
           New_X_copy = New_X.copy()
           New_X_copy.drop(['new_label'],axis=1,inplace=True)
           X_new = enc.fit(New_X_copy)
           X_new = enc.transform(New_X_copy)
           X_train, X_test, y_train, y_test = train_test_split(X_new,New_y,test_size=.3,random_st
           forest = RandomForestClassifier(random_state=42).fit(X_train, y_train)
           y_pred_test = forest.predict(X_test)
```

```
In [19]:   print(accuracy_score(y_test, y_pred_test))
```

```
print(confusion_matrix(y_test, y_pred_test))
cali_X = New_X.loc[y_test.index]
cali_X['label'] = y_test
cali_X['pred'] = y_pred_test
af_X = cali_X[cali_X.race=='African-American']
ca_X =  cali_X[cali_X.race=='Caucasian']
print(np.abs(accuracy_score(af_X.label, af_X.pred)-accuracy_score(ca_X.label, ca_X.pre
```

```
0.9463414634146341
[[936  70]
 [ 29 810]]
0.043960286817429695
```

As we can see, after the local massaging of the data, the accuracy of the Random Forest
Classifier has decreased from 0.97 to 0.93. This decrease in accuracy is marginal and is
acceptable, as the classifier is still a very accurate one.

On the other hand, the calibration score has increased from 0.013 to 0.55. While the accuracy
difference between the two race groups has increased, the data is now much more balanced,
which shows that with local massaging, the model can predict with a high level of fairness at a
small cost of accuracy.

Local Preferential Sampling - Algorithm 2 (A6)

Moving on to the second algorithm, where local preferential sampling is used. The intuition
behind this is to remove the samples and resample them close to the decision boundary.

a2.jpg

```
In [20]: X_New = ut.local_preferential_sampling(new_X,s,e,y)
         af = X_New[X_New.race=='African-American']
         ca = X_New[X_New.race =='Caucasian']
         rate_af = round(af[af['new_label']==1].shape[0]/af.shape[0],2)
         rate_ca = round(ca[ca['new_label']==1].shape[0]/ca.shape[0],2)
         rate_af,rate_ca
```

```
Out[20]: (0.5, 0.4)
```

After using local preferential sampling, we see that the data is now balanced. With the
proportion of African-Americans and Caucasians with the label (1) being fairly similar. This is
desired, as it achieves our goal of eliminating discrimination and unfairness in the model.

```
In [21]: y_New = X_New.new_label
         X_New_copy = X_New.copy()
         X_New_copy.drop(['new_label'],axis=1,inplace=True)
         X_new = enc.fit(X_New_copy)
         X_new = enc.transform(X_New_copy)
         X_train, X_test, y_train, y_test = train_test_split(X_new,y_New,test_size=.3,random_st
         forest = RandomForestClassifier(random_state=42).fit(X_train, y_train)
         y_pred_test = forest.predict(X_test)
```

```
In [22]: print(accuracy_score(y_test, y_pred_test))
         print(confusion_matrix(y_test, y_pred_test))
```

```
cali_X = X_New.loc[y_test.index]
cali_X['label'] = y_test
cali_X['pred'] = y_pred_test
af_X = cali_X[cali_X.race=='African-American']
ca_X =  cali_X[cali_X.race=='Caucasian']
print(np.abs(accuracy_score(af_X.label, af_X.pred)-accuracy_score(ca_X.label, ca_X.pre
```

```
0.9588075880758807
[[929  73]
 [  3 840]]
0.06168413311270449
```

From the result above, we see that the accuracy score similarly has decreased from 0.97 to 0.95. But, again, this slight decrease in accuracy is negligible as the model is still highly accurate.

Meanwhile, the calibration score has also increased from the original 0.013 to 0.054. Again, this is a tiny increase in accuracy difference across the two race groups. This is a good sign as it shows that the model can include fairness in its prediction while still preserving its accuracy.

Conclusion

To conclude, after using two different methods to balance the model's fairness, namely: The in-processing method from A3 and the pre-processing method in A6, we see that the ML models can achieve a fair prediction with reasonably high accuracy (in A6). Although the accuracy in the models implemented has decreased from the baseline models, the goal of achieving fair predictions was fulfilled. In A3, we did this by adding constraints to the model to achieve either accuracy or fairness while optimizing the other. In A6, we pre-processed the data using local massaging and preferential sampling to achieve a balanced data set.

To this end, we believe that implementing the models above, does result in a drop in accuracy, but in return for compliance with model fairness, is worth a shot.