Pre-Process

In [16]:
```
import pandas as pd
import numpy as np
df = pd.read_csv('../data/compas-scores-two-years.csv')
df.columns
```

Out[16]:
```
Index(['id', 'name', 'first', 'last', 'compas_screening_date', 'sex', 'dob',
       'age', 'age_cat', 'race', 'juv_fel_count', 'decile_score',
       'juv_misd_count', 'juv_other_count', 'priors_count',
       'days_b_screening_arrest', 'c_jail_in', 'c_jail_out', 'c_case_number',
       'c_offense_date', 'c_arrest_date', 'c_days_from_compas',
       'c_charge_degree', 'c_charge_desc', 'is_recid', 'r_case_number',
       'r_charge_degree', 'r_days_from_arrest', 'r_offense_date',
       'r_charge_desc', 'r_jail_in', 'r_jail_out', 'violent_recid',
       'is_violent_recid', 'vr_case_number', 'vr_charge_degree',
       'vr_offense_date', 'vr_charge_desc', 'type_of_assessment',
       'decile_score.1', 'score_text', 'screening_date',
       'v_type_of_assessment', 'v_decile_score', 'v_score_text',
       'v_screening_date', 'in_custody', 'out_custody', 'priors_count.1',
       'start', 'end', 'event', 'two_year_recid'],
      dtype='object')
```

In [17]:
```
compas_df = df
races = ['African-American','Caucasian']

compas_df=compas_df[compas_df.race.isin(races)]

compas_df.head()

#use only specific columns for future prediction
compas_df_filtered = compas_df.loc[:, ['race']].join(compas_df.select_dtypes(include=['int64']))
#drop id and duplicate columns
compas_df_filtered = compas_df_filtered.drop(['id','decile_score.1', 'priors_count.1'], axis=1)
compas_df_filtered.head()
```

Out[17]:

	race	age	juv_fel_count	decile_score	juv_misd_count	juv_other_count	priors_count	is_recid	is_violent_recid	v_decile_score	start	
1	African-American	34	0	3	0	0	0	1	1	1	9	
2	African-American	24	0	4	0	1	4	1	0	3	0	
3	African-American	23	0	8	1	0	1	0	0	6	0	1
6	Caucasian	41	0	6	0	0	14	1	0	2	5	
8	Caucasian	39	0	1	0	0	0	0	0	1	2	

In [18]:

```
compas_df_filtered["race"].replace(['African-American', 'Caucasian'],[0, 1], inplace=True)
```

In [19]:

```
y = compas_df_filtered["two_year_recid"]
X = compas_df_filtered.drop(["two_year_recid"], axis=1)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X,y,test_size=0.2, random_state=50, stratify=y)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
X_train, X_valid, y_train, y_valid=train_test_split(X_train,y_train,test_size=0.2, random_state=67, stratify=y_train)
print(X_train.shape, y_train.shape, X_valid.shape, y_valid.shape)
```

```
(4920, 13) (4920,) (1230, 13) (1230,)
(3936, 13) (3936,) (984, 13) (984,)
```

Baseline

In [20]:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression().fit(X_train, y_train)
clf.score(X_test,y_test)
```

```
C:\Users\zhang\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818: ConvergenceWarning: lbfgs failed to c
onverge (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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Out[20]:

```
0.9829268292682927
```

In [21]:

```
index_b= (X_test["race"]==0)
X_test_b=X_test[index_b]
y_test_b=y_test[index_b]
X_test_w=X_test[-index_b]
y_test_w=y_test[-index_b]
```

In [22]:

```
index_b= (X_train["race"]==0)
X_train_b=X_train[index_b]
y_train_b=y_train[index_b]
X_train_w=X_train[-index_b]
y_train_w=y_train[-index_b]
```

In [24]:

```
#calibration
print(clf.score(X_test_b,y_test_b))
print(clf.score(X_test_w,y_test_w))
print("calibration score: ",abs(clf.score(X_test_b,y_test_b)-clf.score(X_test_w,y_test_w)))
```

```
0.9807692307692307
0.9860557768924303
calibration score:  0.005286546123199565
```

A5

In [25]:

```
import torch as t
import torch.nn as nn
```

Process the data for the model

In [26]:
```
data_filtered_afram = compas_df_filtered[compas_df_filtered['race']==0]
data_filtered_cau = compas_df_filtered[compas_df_filtered['race']==1]

#X_afram = data_filtered_afram[["race","age","priors_count","juv_count",'score_text',"sex","c_charge_degree","decile_sc
#y_afram = data_filtered_afram[['two_year_recid']]

X_afram = data_filtered_afram.drop(["two_year_recid"], axis=1)
y_afram = data_filtered_afram[["two_year_recid"]]

X_cau = data_filtered_cau.drop(["two_year_recid"], axis=1)
y_cau = data_filtered_cau[["two_year_recid"]]

#X_cau = data_filtered_cau[["race","age","priors_count","juv_count",'score_text',"sex","c_charge_degree","decile_score"
#y_cau = data_filtered_cau[['two_year_recid']]
```

In [27]:
```
X_train_afram, X_test_afram, y_train_afram, y_test_afram = train_test_split(X_afram, y_afram, test_size=0.2)
X_train_cau, X_test_cau, y_train_cau, y_test_cau = train_test_split(X_cau, y_cau, test_size=0.2)
```

Use tensor flow the modify the dataset

In [28]:
```
def tensorX(X):
    return t.tensor(np.array(X)).to(t.float32)

def tensorY(X,Y):
    return t.from_numpy(np.array(Y).astype('float32')).reshape(X.shape[0], 1)
```

In [29]:
```
X_train_afram = tensorX(X_train_afram)
X_test_afram = tensorX(X_test_afram)
X_train_cau = tensorX(X_train_cau)
X_test_cau = tensorX(X_test_cau)

y_train_afram = tensorY(X_train_afram,y_train_afram)
y_test_afram = tensorY(X_test_afram,y_test_afram)
y_train_cau = tensorY(X_train_cau,y_train_cau)
y_test_cau = tensorY(X_test_cau,y_test_cau)
```

Algorithm: Prejudice Remover Regularizer Loss Function

In [30]:
```
#update LogisticRegression function
class LogisticRegression(nn.Module):
```

```
    def __init__(self,data):
        super(LogisticRegression, self).__init__()
        self.w = nn.Linear(data.shape[1], out_features=1, bias=True)
        self.sigmod = nn.Sigmoid()
    def forward(self, x):
        w = self.w(x)
        output = self.sigmod(w)
        return output
```

In [31]:

```
#Define the Prejudice Remover Regularizer Loss Function
class PRLoss():
    def __init__(self, eta=1.0):
        super(PRLoss, self).__init__()
        self.eta = eta

    def PI(self,output_afram,output_cau):
        N_afram = t.tensor(output_afram.shape[0])
        N_cau   = t.tensor(output_cau.shape[0])
        # calculate P[y|s]
        P_ys = t.stack((t.sum(output_afram),t.sum(output_cau)),axis=0) / t.stack((N_afram,N_cau),axis=0)
        # calculate P[y]
        P_y = t.sum(t.cat((output_afram,output_cau),0)) / (X_train_afram.shape[0]+X_train_cau.shape[0])
        # calculate PI
        PI_s1y1 = output_afram * (t.log(P_ys[1]) - t.log(P_y))
        PI_s1y0 =(1- output_afram) *(t.log(1-P_ys[1]) - t.log(1-P_y))
        PI_s0y1 = output_cau * (t.log(P_ys[0]) - t.log(P_y))
        PI_s0y0 = (1- output_cau)* (t.log(1-P_ys[0]) - t.log(1-P_y))
        PI = t.sum(PI_s1y1) + t.sum(PI_s1y0) + t.sum(PI_s0y1) + t.sum(PI_s0y0)
        PI = self.eta * PI
        return PI
```

Algorithm:Logistical regression with Prejudice Remover Regularizer

In [32]:

```
#This function has a hyperparameter eta which is the size of the regulating term in the loss function.
#The fit method will give us the accuracy and caliberation of the fitted model
class LogisticRegressionWithPRR():

    def __init__(self, eta=0.0, epochs=100, lr = 0.01):
        super(LogisticRegressionWithPRR, self).__init__()
        self.eta = eta
        self.epochs = epochs
        self.lr = lr
```

```python
    def fit(self,X_train_afram,y_train_afram,X_train_cau,y_train_cau,X_test_afram, y_test_afram, X_test_cau, y_test_cau
        #LogisticRegression model
        model_afram = LogisticRegression(X_train_afram)
        model_cau = LogisticRegression(X_train_cau)

        criterion = nn.BCELoss()
        PI = PRLoss(eta=self.eta)
        epochs = self.epochs

        #L2 regularization (non-zero weight_decay)
        optimizer = t.optim.Adam(list(model_afram.parameters())+ list(model_cau.parameters()), self.lr, weight_decay=1e

        for epoch in range(epochs):
            #train
            model_afram.train()
            model_cau.train()

            #zero out the gradients
            optimizer.zero_grad()

            #compute loss
            output_afram = model_afram(X_train_afram)
            output_cau = model_cau(X_train_cau)
            log_loss = criterion(output_afram, y_train_afram)+ criterion(output_cau, y_train_cau)
            PI_loss = PI.PI(output_afram,output_cau)
            loss = PI_loss +log_loss

            loss.backward()
            optimizer.step()
        #eval
        model_afram.eval()
        model_cau.eval()

        #calculate accuracy
        #Accuracy is the average of correctly predicted labels of two groups
        #Caliberation is the difference of accuracy bewtween the two groups
        y_pred_afram = (model_afram(X_test_afram) >= 0.5)
        y_pred_cau = (model_cau(X_test_cau) >= 0.5)

        #sum of correct prediction/total num
        accuracy_afram  = t.sum(y_pred_afram == y_test_afram) / y_test_afram.shape[0]
        accuracy_cau  = t.sum(y_pred_cau == y_test_cau) / y_test_cau.shape[0]

        accuracy = (accuracy_afram + accuracy_cau) / 2
        calibration = t.abs(accuracy_afram - accuracy_cau)
```

```python
        return accuracy.item(), calibration.item(), accuracy_afram.item() , accuracy_cau.item()
```

In [41]:
```python
PR = LogisticRegressionWithPRR(eta = 860, epochs = 100, lr = 1e-04)
PR.fit(X_train_afram,y_train_afram,X_train_cau,y_train_cau, X_test_afram, y_test_afram, X_test_cau, y_test_cau)
```

Out[41]:
```
(0.43259480595588684,
 0.15643197298049927,
 0.5108107924461365,
 0.3543788194656372)
```

Tuning Hyperparameter eta

In [42]:
```python
#Tuning the hyperparameter to find the best model
eta=np.linspace(0, 100, num=1000)
hist_acc=np.zeros(1000)
hist_cal=np.zeros(1000)
hist_afram = np.zeros(1000)
hist_cau = np.zeros(1000)
for i in range(1000):
    PR = LogisticRegressionWithPRR(eta = eta[i], epochs = 100, lr = 1e-04)
    results = PR.fit(X_train_afram,y_train_afram,X_train_cau,y_train_cau, X_test_afram, y_test_afram, X_test_cau, y_tes
    hist_acc[i]=results[0]
    hist_cal[i]=results[1]
    hist_afram[i]=results[2]
    hist_cau[i]=results[3]
```

In [43]:
```python
clf_accuracy = np.max(hist_acc)
print("Accuracy:")
np.max(hist_acc)
```

```
Accuracy:
```

Out[43]:
```
0.7574337720870972
```

In [44]:
```python
A5_cal_score = np.min(hist_cal)
print("Calibration Score:")
hist_cal[np.argmax(hist_acc)] #calibration decreases
```

```
Calibration Score:
```

Out[44]:
```
0.0851324200630188
```

In [45]:
```
#clf_accuracy = hist_acc[np.argmin(hist_cal)]
print("Accuracy:")
hist_acc[np.argmin(hist_cal)]
```

```
Accuracy:
```

Out[45]:
```
0.6058223247528076
```

In [46]:
```
#A5_cal_score = np.min(hist_cal)
print("Calibration Score:")
np.min(hist_cal) #minimum calibration
```

```
Calibration Score:
```

Out[46]:
```
0.0018687844276428223
```

In [47]:
```
#np.sort(hist_cal)[0:5]
hist_cal_new = hist_cal
np.sort(hist_cal_new)[0:5]
```

Out[47]:
```
array([0.00186878, 0.00220457, 0.00247699, 0.00727418, 0.00727418])
```

In [55]:
```
np.where(hist_cal <= 0.0025)
```

Out[55]:
```
(array([158, 745, 897], dtype=int64),)
```

In [56]:
```
hist_acc[158]
```

Out[56]:
```
0.6203831434249878
```

In [57]:
```
A5_acc = hist_acc[158]
A5_calibration = np.sort(hist_cal_new)[2]
```

A6

In [96]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
```

Data Loading and Preprocessing

In [97]:
```
compas_df= pd.read_csv('../data/compas-scores-two-years.csv')

#Use the data of African American and Caucasian

races = ['African-American','Caucasian']

compas_df=compas_df[compas_df.race.isin(races)]

compas_df.head()

#use only specific columns for future prediction
compas_df_filtered = compas_df.loc[:, ['race']].join(compas_df.select_dtypes(include=['int64']))
#drop id and duplicate columns
compas_df_filtered = compas_df_filtered.drop(['id','decile_score.1', 'priors_count.1'], axis=1)
#compas_df_filtered = df[["two_year_recid","race","age","priors_count","juv_count",'score_text',"sex","c_charge_degree"
#compas_df_filtered.head()
```

Implemntaion of algorithms

In [98]:
```
#for the algorithms below, we apply on the whole dataset
X=compas_df_filtered
s=X.race
e=X.decile_score
y=X.two_year_recid



def PARTITION(X, e):
    unique_values = e.unique()  # find unique values of e
    groups = {}  # initialize empty dictionary to store resulting DataFrames

    for val in unique_values:
        group_df = X[e == val]  # filter rows in X where e equals current unique value
        groups[val] = group_df  # store resulting DataFrame in dictionary with key = unique value

    return groups  # return dictionary of resulting DataFrames
```

In [99]:
```
resulting_groups = PARTITION(X, e)
first_group_df = resulting_groups[1]
#print(first_group_df.head)

unique_values = e.unique()
print(unique_values)
```

```
[ 3  4  8  6  1 10  5  9  2  7]
```

In [100…
```
def delta(X, gender):
    # calculate the baseline probabilities
    p_pos_base = X.two_year_recid.mean()

    # calculate probabilities conditioned on gender
    p_pos_gender = X[X.race == gender].two_year_recid.mean()


    # calculate the number of gender people in X
    G = len(X[X.race == gender])

    # calculate delta

    delta_val = G * abs(p_pos_gender - p_pos_base)


    return int(delta_val)
```

In [101…
```
for group in resulting_groups.values():
    print(delta(group,'Caucasian'))
```

```
11
10
3
1
5
4
3
1
1
2
```

Algorithm: Local massaging

In [102…

```python
#Algorithm 1: Local massaging
resulting_groups = PARTITION(X, e)
updated_groups_1 = []
for group in resulting_groups.values():
    males = group[group.race == 'Caucasian']
    females = group[group.race == 'African-American']

    # learn a ranker H for this group
    H = LogisticRegression(random_state=42,max_iter=500)
    H.fit(group.drop(['race','two_year_recid'], axis=1), group.two_year_recid)

    # rank and relabel males
    males_females = pd.concat([males,females])
    males_females['proba'] = H.predict_proba(males_females.drop(['two_year_recid','race'], axis=1))[:, 1]


    males_females = males_females.sort_values(by='proba')


    delta_males = int(delta(group, 'Caucasian'))

    males_to_relabel = males_females[(males_females['race'] == 'Caucasian') & (males_females['two_year_recid'] == 0) &
    group.loc[males_to_relabel.index, 'two_year_recid'] = 1 - males_to_relabel.two_year_recid

    # rank and relabel females
    delta_females = int(delta(group, 'African-American'))
    females_to_relabel = males_females[(males_females['race'] == 'African-American') & (males_females['two_year_recid']
    group.loc[females_to_relabel.index, 'two_year_recid'] = 1 - females_to_relabel.two_year_recid

    updated_groups_1.append(group)
```

```
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
C:\Users\zhang\Anaconda3\lib\site-packages\pandas\core\indexing.py:1773: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy
  self._setitem_single_column(ilocs[0], value, pi)
```

In [103…

```
updated_df_1 = pd.concat(updated_groups_1)
updated_df_1 = updated_df_1.sort_index()
print(sum(updated_df_1['two_year_recid']!=compas_df_filtered['two_year_recid']))
#67 labels have been changed
```

```
67
```

Algorithm: Local preferential sampling

In [104…

```
#Algorithm 2:  Local preferential sampling
resulting_groups = PARTITION(X, e)
updated_groups_2 = []
for group in resulting_groups.values():
    males = group[group.race == 'Caucasian']
    females = group[group.race == 'African-American']

    # learn a ranker H for this group
    H = LogisticRegression(random_state=42,max_iter=500)
```

```
    H.fit(group.drop(['race','two_year_recid'], axis=1), group.two_year_recid)

    # rank delete,and relabel males
    males_females = pd.concat([males, females])
    males_females['proba'] = H.predict_proba(males_females.drop(['two_year_recid', 'race'], axis=1))[:, 1]
    males_females = males_females.sort_values(by='proba')
    delta_males = int(0.5*delta(group, 'Caucasian'))


    males_to_delete = males_females[(males_females['race'] == 'Caucasian') & (males_females['two_year_recid'] == 1) & (
    males_to_duplicate = males_females[(males_females['race'] == 'Caucasian') & (males_females['two_year_recid'] == 0)
    group = group[~group.index.isin(males_to_delete.index)]
    group = pd.concat([group, males_to_duplicate], ignore_index=True)

    # rank delete,and relabel females
    delta_females = int(0.5*delta(group, 'African-American'))
    females_to_delete = males_females[(males_females['race'] == 'African-American') & (males_females['two_year_recid']
    females_to_duplicate = males_females[(males_females['race'] == 'African-American') & (males_females['two_year_recid
    group = group[~group.index.isin(females_to_delete.index)]
    group = pd.concat([group, females_to_duplicate], ignore_index=True)

    updated_groups_2.append(group)

updated_df_2 = pd.concat(updated_groups_2)

#Now we have 3 data frames.compas_df_filtered is the original one. updated_df_1 is the one of algorithm1 updated_df_1 i
```

In [105…

```
# split the data into train, validation and test set

# Split data into training and validation/test sets
train_df, val_test_df = train_test_split(compas_df_filtered, test_size=0.2, random_state=42)

# Split validation/test set into validation and test sets
val_df, test_df = train_test_split(val_test_df, test_size=0.5, random_state=42)



# Split data into training and validation/test sets
train_df_algo1, val_test_df_algo1 = train_test_split(updated_df_1, test_size=0.2, random_state=42)

# Split validation/test set into validation and test sets
val_df_algo1, test_df_algo1 = train_test_split(val_test_df_algo1, test_size=0.5, random_state=42)
```

```python
# Split data into training and validation/test sets
train_df_algo2, val_test_df_algo2 = train_test_split(updated_df_2, test_size=0.2, random_state=42)

# Split validation/test set into validation and test sets
val_df_algo2, test_df_algo2 = train_test_split(val_test_df_algo2, test_size=0.5, random_state=42)
```

In [106…

```python
X_train1 = train_df.drop(['is_recid','race'], axis=1)
y_train1 = train_df['is_recid']

X_train2 = train_df_algo1.drop(['is_recid','race'], axis=1)
y_train2 = train_df_algo1['is_recid']

X_train3 = train_df_algo2.drop(['is_recid','race','proba'], axis=1)
y_train3 = train_df_algo2['is_recid']



# Fit logistic regression model for X_train1
logreg1 = LogisticRegression()
logreg1.fit(X_train1, y_train1)


# Fit logistic regression model for X_train2
logreg2 = LogisticRegression()
logreg2.fit(X_train2, y_train2)

# Fit logistic regression model for X_train3
logreg3 = LogisticRegression()
logreg3.fit(X_train3, y_train3)
```

```
C:\Users\zhang\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818: ConvergenceWarning: lbfgs failed to c
onverge (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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
C:\Users\zhang\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818: ConvergenceWarning: lbfgs failed to c
onverge (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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
C:\Users\zhang\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:818: ConvergenceWarning: lbfgs failed to c
onverge (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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

Out[106]:

```
LogisticRegression()
```

In [107…

```
#Get results for validation set
# For X_val1
X_val1 = val_df.drop(['is_recid','race'], axis=1)
y_val1 = val_df['is_recid']
y_pred_val1 = logreg1.predict(X_val1)


# For X_val2
X_val2 = val_df_algo1.drop(['is_recid','race'], axis=1)
y_val2 = val_df_algo1['is_recid']
y_pred_val2 = logreg2.predict(X_val2)


# For X_val3
X_val3 = val_df_algo2.drop(['is_recid','race','proba'], axis=1)
```

```
y_val3 = val_df_algo2['is_recid']
y_pred_val3 = logreg3.predict(X_val3)
```

In [108…

```
#Get results for test set
# For X_test1
X_test1 = test_df.drop(['is_recid','race'], axis=1)
y_test1 = test_df['is_recid']
y_pred_test1 = logreg1.predict(X_test1)


# For X_test2
X_test2 = test_df_algo1.drop(['is_recid','race'], axis=1)
y_test2 = test_df_algo1['is_recid']
y_pred_test2 = logreg2.predict(X_test2)


# For X_test3
X_test3 = test_df_algo2.drop(['is_recid','race','proba'], axis=1)
y_test3 = test_df_algo2['is_recid']
y_pred_test3 = logreg3.predict(X_test3)
```

In [109…

```
# Make a copy of the original test dataframes
test_df1 = test_df.copy()
test_df2 = test_df_algo1.copy()
test_df3 = test_df_algo2.copy()


test_df1["y_pred"] = y_pred_test1
test_df2["y_pred"] = y_pred_test2
test_df3["y_pred"] = y_pred_test3
```

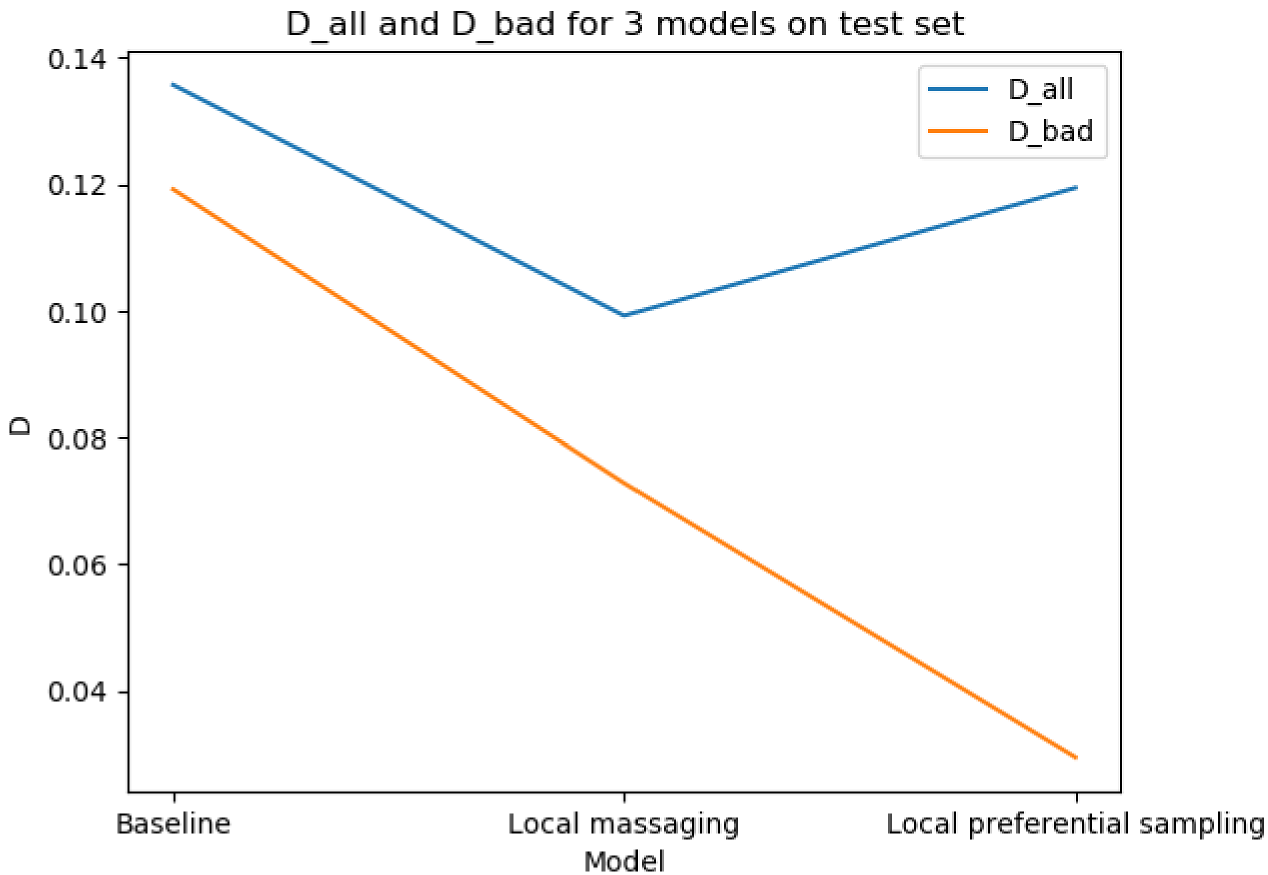
In [110…

```
# Define function to compute D_all and D_bad



def compute_D(test_df, y_pred):
    # Compute D_all
    D_all = test_df.loc[test_df['race'] == 'African-American', y_pred].mean() - \
            test_df.loc[test_df['race'] == 'Caucasian', y_pred].mean()

    # Compute D_bad
    decile_scores = list(range(1, 6))
    D_bad = 0
```

```
    for score in decile_scores:
        P_score = test_df.loc[(test_df['race'] == 'African-American') & (test_df['decile_score'] == score), y_pred].mea
        P_score -= test_df.loc[(test_df['race'] == 'Caucasian') & (test_df['decile_score'] == score), y_pred].mean()
        P_y = test_df.loc[test_df['decile_score'] == score, y_pred].mean()
        D_bad += ((P_score) * (P_y))




    return D_all, D_bad

# Compute D_all and D_bad for the three logistic regression models on the three test sets
D_all_1, D_bad_1 = compute_D(test_df1, 'two_year_recid')
D_all_2, D_bad_2 = compute_D(test_df2, 'two_year_recid')
D_all_3, D_bad_3 = compute_D(test_df3, 'two_year_recid')
```

In [111…

```
# Calculate D_all and D_bad for each model on the test set
D_all_test = [D_all_1,D_all_2,D_all_3]
D_bad_test = [D_bad_1, D_bad_2, D_bad_3]

plt.plot(['Baseline', 'Local massaging', 'Local preferential sampling'], D_all_test, label='D_all')
plt.plot(['Baseline', 'Local massaging', 'Local preferential sampling'], D_bad_test, label='D_bad')

# Add axis labels and title
plt.xlabel('Model')
plt.ylabel('D')
plt.title('D_all and D_bad for 3 models on test set')

# Add legend
plt.legend()

# Show the plot
plt.show()


print(D_all_test)
print(D_bad_test)
```



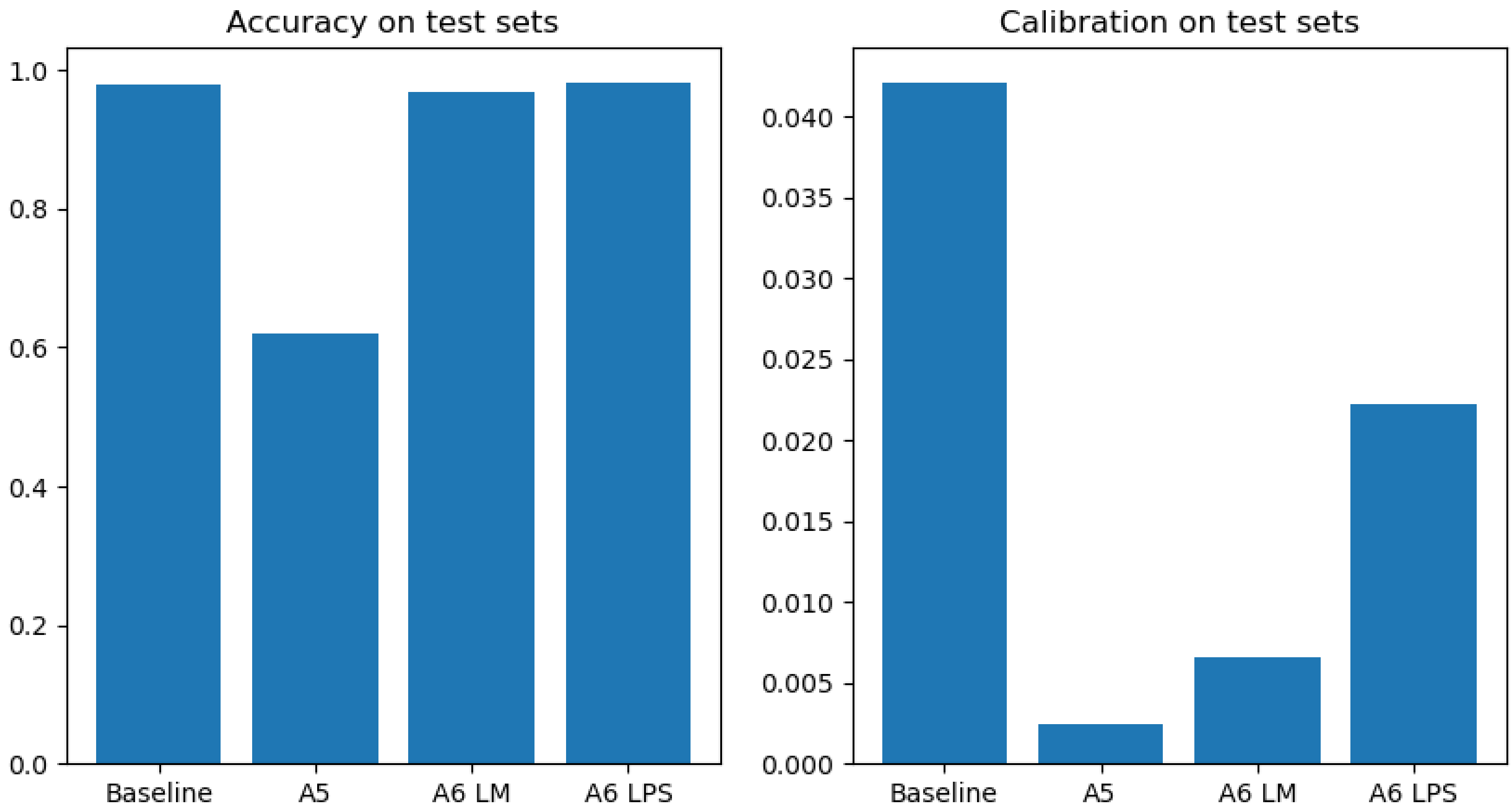

```
[0.13573754620665374, 0.09924529132195037, 0.11945715462632062]
[0.11922600741704245, 0.07281751731787592, 0.02948165449549718]
```

Comparision plots:

In [112…

```python
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Define function to compute calibration
def compute_calibration(df):
    african_american = df[df['race'] == 'African-American']
    caucasian = df[df['race'] == 'Caucasian']
    african_american_accuracy = accuracy_score(african_american['two_year_recid'], african_american['y_pred'])
    caucasian_accuracy = accuracy_score(caucasian['two_year_recid'], caucasian['y_pred'])
    return abs(african_american_accuracy - caucasian_accuracy)
```

```python
# Compute accuracy and calibration for test set 1
accuracy_test1 = accuracy_score(test_df1['two_year_recid'], test_df1['y_pred'])
calibration_test1 = compute_calibration(test_df1)
baseline_cal=calibration_test1*len((e.unique()))

# Compute accuracy and calibration for test set 2
accuracy_test2 = accuracy_score(test_df2['two_year_recid'], test_df2['y_pred'])
calibration_test2 = compute_calibration(test_df2)

# Compute accuracy and calibration for test set 3
accuracy_test3 = accuracy_score(test_df3['two_year_recid'], test_df3['y_pred'])
calibration_test3 = compute_calibration(test_df3)

A5_acc = hist_acc[158]
A5_calibration = np.sort(hist_cal_new)[2]

# Plot results
accuracies = [accuracy_test1,A5_acc, accuracy_test2, accuracy_test3]
calibrations = [baseline_cal, A5_calibration,calibration_test2, calibration_test3]



models = ['Baseline', 'A5', 'A6 LM','A6 LPS']
accs = accuracies
cals = calibrations

fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
axs[0].bar(models, accuracies)
axs[0].set_title('Accuracy on test sets')
axs[1].bar(models, calibrations)
axs[1].set_title('Calibration on test sets')
plt.show()
```



Accuracy on test sets
1.0
0.8
0.6
0.4
0.2
0.0
Baseline
A5
A6 LM
A6 LPS
Calibration on test sets
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Baseline
A5
A6 LM
A6 LPS