In [1]:
```
%reload_ext autoreload
%autoreload 2

import pandas as pd
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestC
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.metrics import confusion_matrix, accuracy_score

from DataGenerator import *
from Competition import *
from config import *
```

In [2]:
```
BN_E0 = initialize_BN()
dataset = BN_E0.simulate(n_samples=50000, seed=SEED)
```

Generating for node: JOB: 100% 8/8 [00:00<00:00, 19.93it/s]

In [3]:
```
dataset
```

Out[3]:

| | SES | COLLEGE | SCHOOL | SAT | SEX | INTERN | CGPA | JOB |
|---|---|---|---|---|---|---|---|---|
| **0** | 2 | 1 | 1 | 2 | 1 | 1 | 2 | 1 |
| **1** | 2 | 0 | 2 | 1 | 1 | 1 | 2 | 0 |
| **2** | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 0 |
| **3** | 1 | 0 | 1 | 2 | 1 | 0 | 0 | 0 |
| **4** | 2 | 0 | 1 | 2 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **49995** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **49996** | 2 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| **49997** | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| **49998** | 2 | 1 | 1 | 1 | 1 | 1 | 2 | 1 |
| **49999** | 2 | 1 | 0 | 2 | 1 | 1 | 2 | 1 |

50000 rows × 8 columns

In [4]:
```
C1_label = 'COLLEGE'
C1_features = ['SES', 'SEX', 'SAT', 'SCHOOL']
C1 = Competition(dataset, C1_features, C1_label, ['SEX', 'SES'], [0,0]
```

In [5]:
```
C2_label = 'INTERN'
C2_features = ['SES', 'SEX', 'SAT', 'SCHOOL','COLLEGE','CGPA']
C2 = Competition(dataset, C2_features, C2_label, ['SEX', 'SES'], [0,0]
```

In [6]:
```
C3_label = 'JOB'
C3_features = ['SES', 'SEX', 'SAT', 'SCHOOL','COLLEGE','CGPA','INTERN'
C3 = Competition(dataset, C2_features, C2_label, ['SEX', 'SES'], [0,0]
```

In [7]:
```
X_train, y_train, X_test, y_test, X_val, y_val = C1.create_train_test_
X_train.shape, X_test.shape, X_val.shape
```

Out[7]:
```
((30000, 4), (10000, 4), (10000, 4))
```

In [8]:
```

```

```
C2.create_train_test_val_split(SEED=44)
C3.create_train_test_val_split(SEED=44)
```

Out[8]:

```
(       SES  SEX  SAT  SCHOOL  COLLEGE  CGPA
 7180     1    0    2       0        1     0
 23110    1    0    1       0        1     1
 39329    1    1    0       2        1     2
 319      2    1    1       1        0     2
 7916     2    0    1       0        1     1
 ...    ...  ...  ...     ...      ...   ...
 16848    2    0    2       0        1     1
 45072    1    1    0       1        0     2
 19667    2    1    2       1        1     0
 6728     2    0    2       1        1     1
 7086     0    1    0       2        1     2

 [30000 rows x 6 columns],
 7180     1
 23110    1
 39329    1
 319      1
 7916     1
         ..
 16848    1
 45072    0
 19667    1
 6728     1
 7086     1
 Name: INTERN, Length: 30000, dtype: int64,
        SES  SEX  SAT  SCHOOL  COLLEGE  CGPA
 49457    1    1    2       2        1     2
 20706    1    0    2       0        1     0
 46059    2    1    1       2        0     1
 38076    2    1    2       2        1     2
 8394     2    1    2       2        1     2
 ...    ...  ...  ...     ...      ...   ...
 34398    2    0    0       0        0     0
 41323    1    0    1       0        1     1
 17138    1    1    2       2        1     2
 14034    1    0    1       0        1     1
 31387    2    0    2       1        1     1

 [10000 rows x 6 columns],
 49457    0
 20706    1
 46059    0
 38076    1
 8394     1
         ..
 34398    1
 41323    1
 17138    1
```

```
14034    1
31387    1
Name: INTERN, Length: 10000, dtype: int64,
       SES  SEX  SAT  SCHOOL  COLLEGE  CGPA
48004    2    0    0       1        0     0
6168     2    0    1       2        0     1
31693    1    1    2       2        1     2
17547    1    0    2       0        1     0
41374    2    0    2       1        1     1
...    ...  ...  ...     ...      ...   ...
40325    0    1    2       2        1     0
1622     1    0    1       0        0     0
29904    1    0    0       0        0     0
12158    2    1    2       0        1     2
47080    1    1    2       2        1     1

[10000 rows x 6 columns],
48004    0
6168     0
31693    1
17547    0
41374    0
        ..
40325    0
1622     0
29904    0
12158    1
47080    1
Name: INTERN, Length: 10000, dtype: int64)
```

In [9]:
```
for i in C1.test_groups.keys():
    print(i, C1.test_groups[i].shape[0]/C1.X_test.shape[0])
```

```
SEX_SES_priv 0.1191
SEX_SES_dis 0.313
SEX_priv 0.6057
SEX_dis 0.3943
SES_priv 0.2004
SES_dis 0.7996
```

In [10]:
```
best_models = {}
#for model_name in model_specs.keys():
for model_name in ['lr']:
    print(model_name)
    search = GridSearchCV(model_specs[model_name]["base_model"], model
    model = search.fit(X_train, y_train)
    best_models[model_name] = model.best_estimator_

best_models
```

```
lr
Fitting 5 folds for each of 25 candidates, totalling 125 fits
```

Out[10]:
```
{'lr': LogisticRegression(C=1, max_iter=200, random_state=111, solver
='newton-cg')}
```

```
for model_name in model_specs.keys():
    print(model_name, best_models[model_name].score(X_val, y_val))
```

In [28]:
```
best_model = best_models['lr']
res_df = C1.fit_base_model(best_model)
__ = C1.compute_predictive_metrics()
```

```
/Users/falaaharifkhan/Documents/EO_Experiments/Competition.py:43: Run
timeWarning: invalid value encountered in long_scalars
  metrics['PPV'] = TP/(TP+FP)
```

```
res_df.to_csv("C1_results.csv", index=False)
```

In [12]:
```
res2_df = C2.fit_base_model(best_model)
```

In [13]:
```
res3_df = C3.fit_base_model(best_model)
```

In [14]:
```
C1.X_train
```

Out[14]:

| | SES | SEX | SAT | SCHOOL |
|---|---|---|---|---|
| **7180** | 1 | 0 | 2 | 0 |
| **23110** | 1 | 0 | 1 | 0 |
| **39329** | 1 | 1 | 0 | 2 |
| **319** | 2 | 1 | 1 | 1 |
| **7916** | 2 | 0 | 1 | 0 |
| **...** | ... | ... | ... | ... |
| **16848** | 2 | 0 | 2 | 0 |
| **45072** | 1 | 1 | 0 | 1 |
| **19667** | 2 | 1 | 2 | 1 |
| **6728** | 2 | 0 | 2 | 1 |
| **7086** | 0 | 1 | 0 | 2 |

30000 rows × 4 columns

In [15]:
```
def sigmoid(x):
    return np.array([_sigmoid_function(value) for value in x])

def _sigmoid_function(x):
    if x >= 0:
        z = np.exp(-x)
        return 1 / (1 + z)
    else:
        z = np.exp(x)
        return z / (1 + z)
```

In [16]:
```
def mse(y_true, y_pred, sample_weights=None):
    return np.average((y_true - y_pred) ** 2, axis=0, weights=sample_w
```

In [17]:
```
def cross_entropy(y_true, y_pred):
    # binary cross entropy
    y_zero_loss = y_true * np.log(y_pred + 1e-9)
    y_one_loss = (1-y_true) * np.log(1 - y_pred + 1e-9)
    return -np.mean(y_zero_loss + y_one_loss)
```

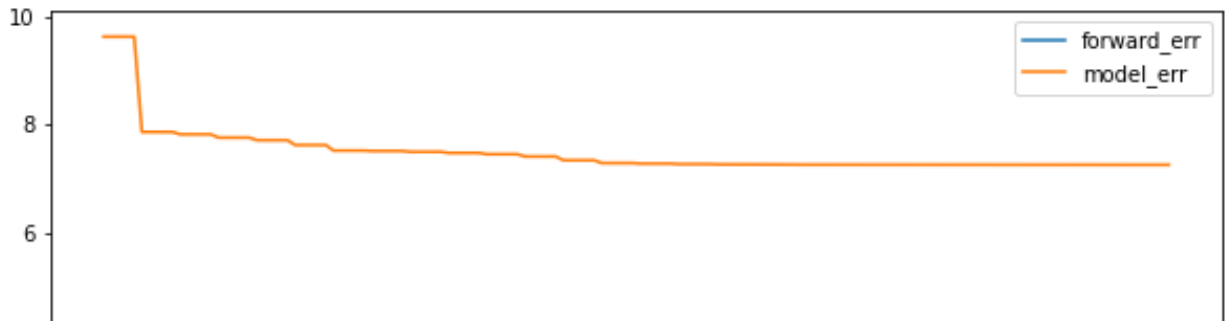
In [18]:
```
def C_forward_linear(X, Y):
    # SES, SEX, SAT, SCHOOL
    # 0 is priv
    beta = [-0.5, -0.2, -0.6, -0.1]
    beta_y = -0.8
    return sigmoid(np.matmul(X,beta)+beta_y*Y)
```

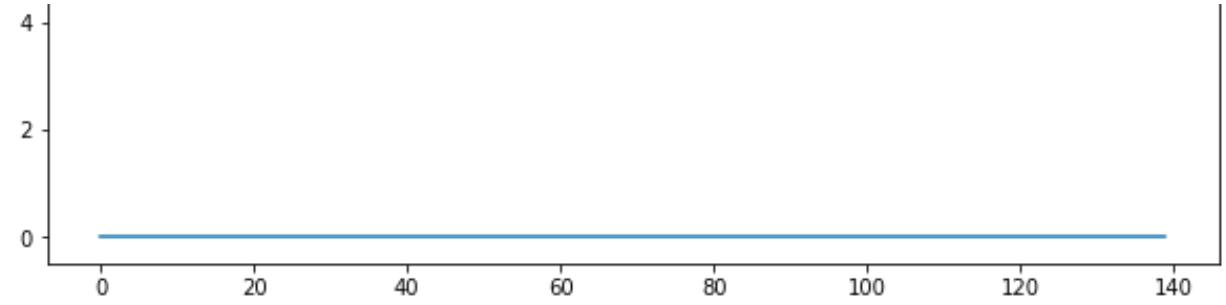
In [19]:
```
ass CustomLinearModel:
  """
  Linear model: Y = XB, fit by minimizing the provided loss_function
  with custom regularization
  """
  def __init__(self, loss_function=cross_entropy, C_forward=C_forward_
               X=None, Y=None, sample_weights=None, beta_init=None,
               regularization=0.0015):
      self.regularization = regularization
      self.beta = None
      self.loss_function = loss_function
      self.C_forward_fuction = C_forward
      self.sample_weights = sample_weights
      self.beta_init = beta_init

      self.model_error__ = []
      self.forward_error__ = []
      self.X = X
      self.Y = Y

  def predict(self, X):
      prediction = sigmoid(np.matmul(X, self.beta))
      return(prediction)

  def model_error(self):
      error = self.loss_function(
          self.predict(self.X), self.Y
      )
      #print("model error: ", error)
      self.model_error__.append(error)
      return(error)

  def C2_error(self):
      error = np.linalg.norm((self.C_forward_fuction(self.X, self.pred
      #print("forward error: ", error)
      self.forward_error__.append(error)
      return(error)

  def custom_foward_loss(self, beta):
      self.beta = beta
      return (self.model_error() + (self.regularization*self.C2_error(

  def l2_regularized_loss(self, beta):
      self.beta = beta
      return(self.model_error() + \
```

```
            sum(self.regularization*np.array(self.beta)**2))

    def plot_train_dynamics(self, scale_factor):
        plt.figure(figsize=(10,5))
        plt.plot(range(len(self.forward_error__)), [i * scale_factor for
        plt.plot(range(len(self.forward_error__)), self.model_error__, la
        plt.legend()
        plt.show()

    def fit(self, maxiter=250):
        # Initialize beta estimates (you may need to normalize
        # your data and choose smarter initialization values
        # depending on the shape of your loss function)
        if type(self.beta_init)==type(None):
            # set beta_init = 1 for every feature
            self.beta_init = np.array([0.1]*self.X.shape[1])
        else:
            # Use provided initial values
            pass

        if self.beta!=None and all(self.beta_init == self.beta):
            print("Model already fit once; continuing fit with more iter

        res = minimize(self.custom_foward_loss, self.beta_init,
                       method='BFGS', options={'maxiter': maxiter})
        self.beta = res.x
        self.beta_init = self.beta
```

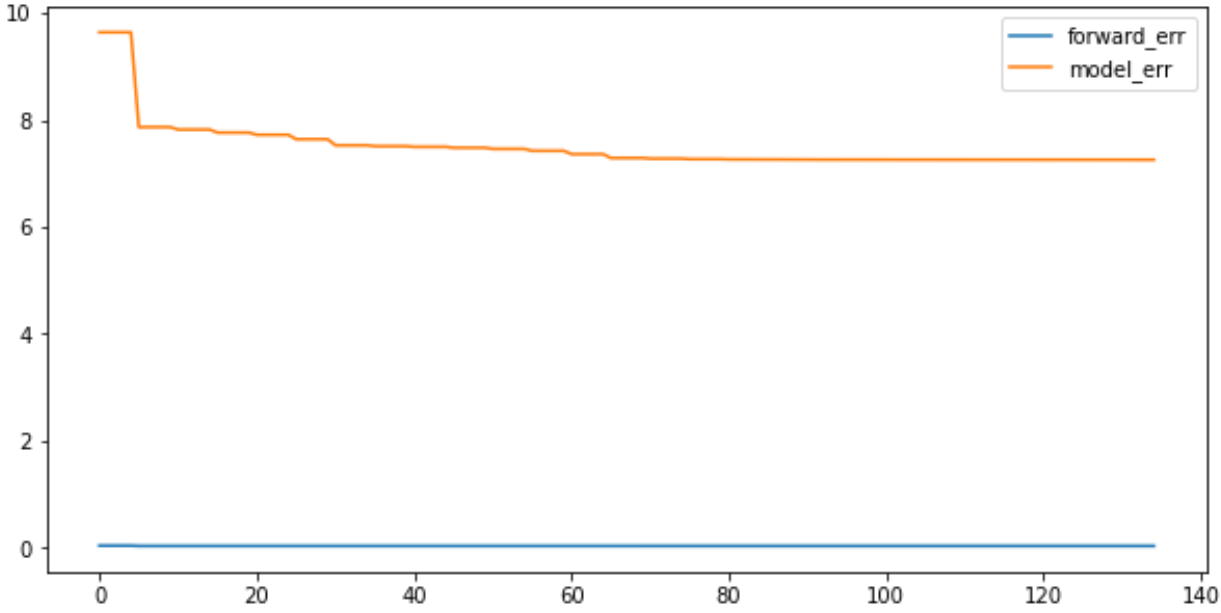
In [20]:
```
#lambda_range = [0, 0.0001, 0.001, 0.005, 0.01, 0.02, 0.05, 0.075, 0.1
lambda_range = [0, 0.0001, 0.001, 0.005, 0.01, 0.02, 0.05, 0.075, 0.1,
acc = []
for lambda_ in lambda_range:
    print(lambda_)
    c1_fair = CustomLinearModel(X=C1.X_train.values, Y=C1.y_train.valu
    c1_fair.fit(maxiter=250)
    c1_fair_preds = np.round(c1_fair.predict(C1.X_test.values))
    acc.append(accuracy_score(C1.y_test.values, c1_fair_preds))
    c1_fair.plot_train_dynamics(0.1*lambda_)

print(c1_fair.beta)
```

```
0
```

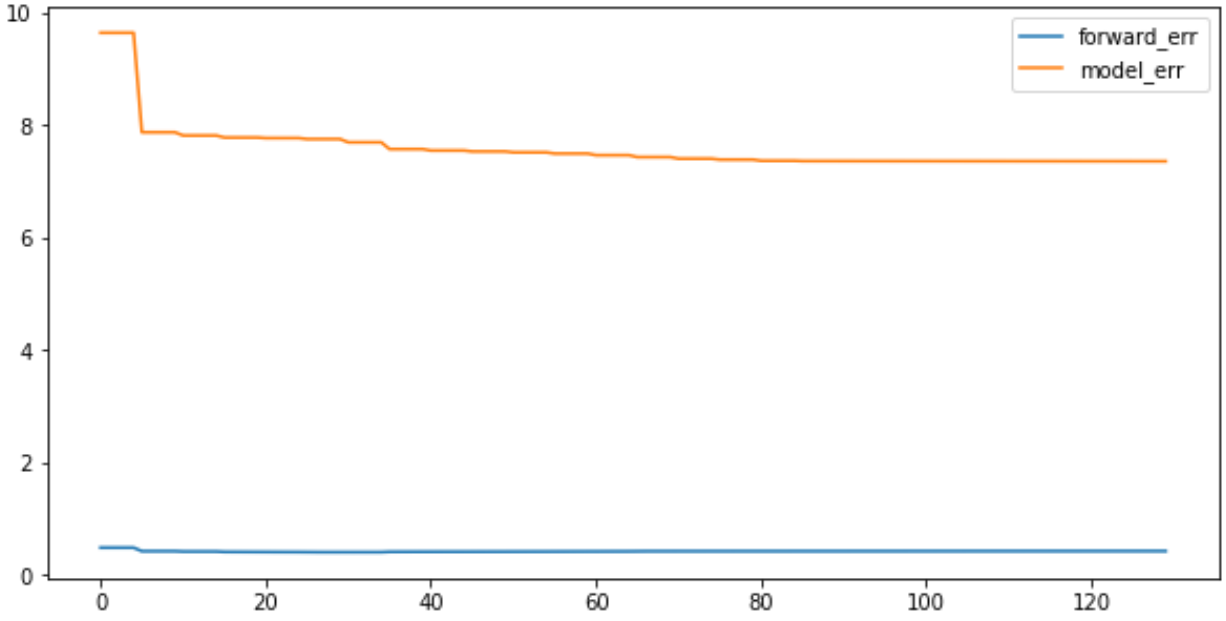




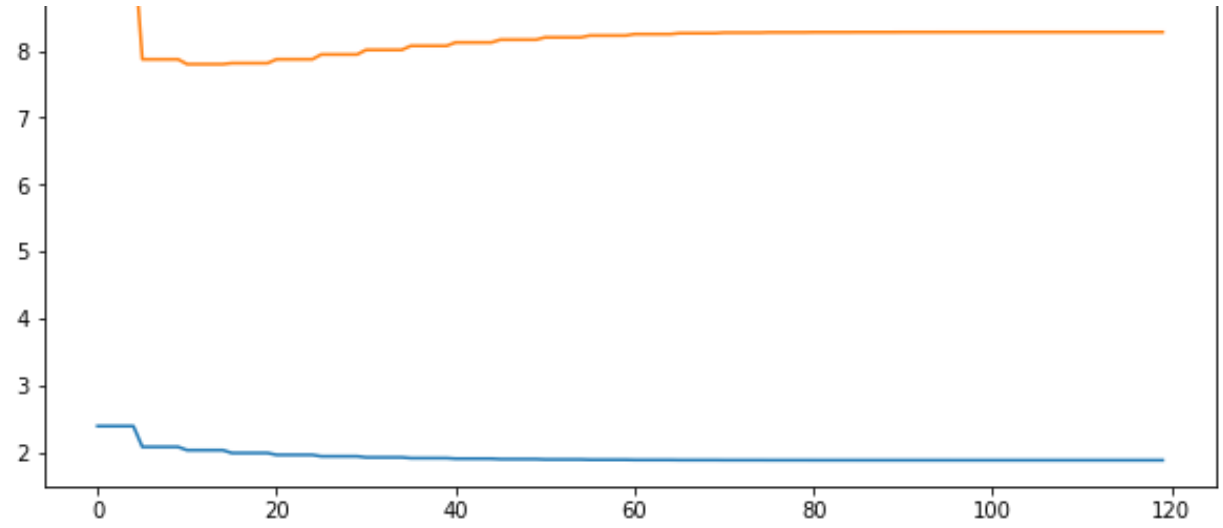

0.0001



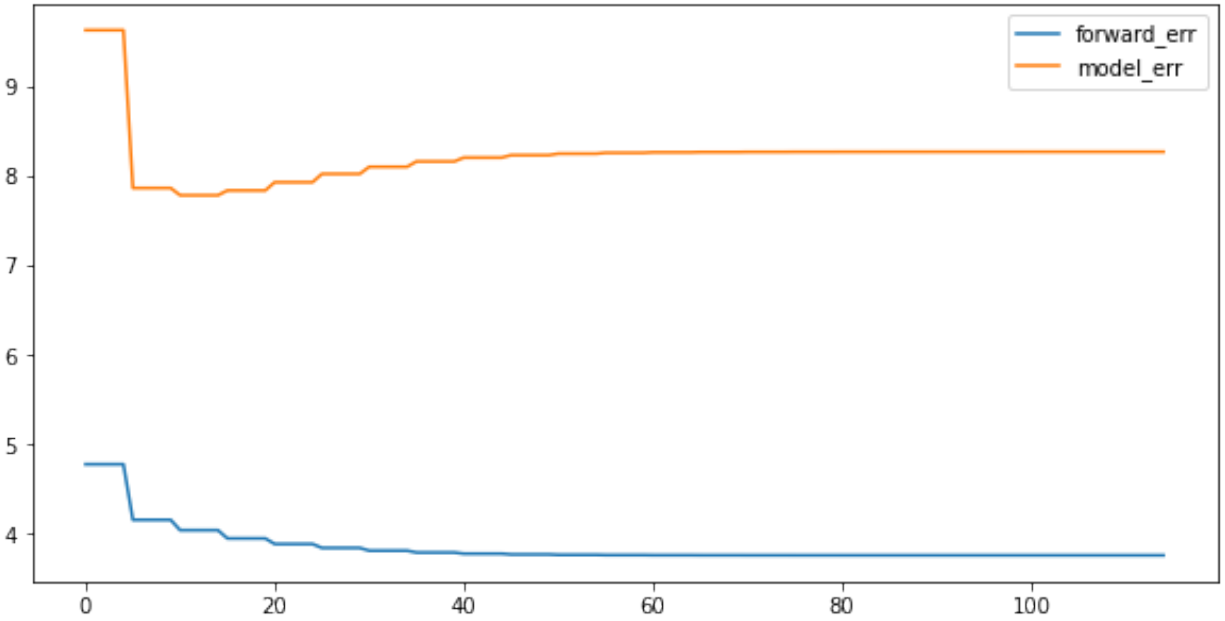

0.001



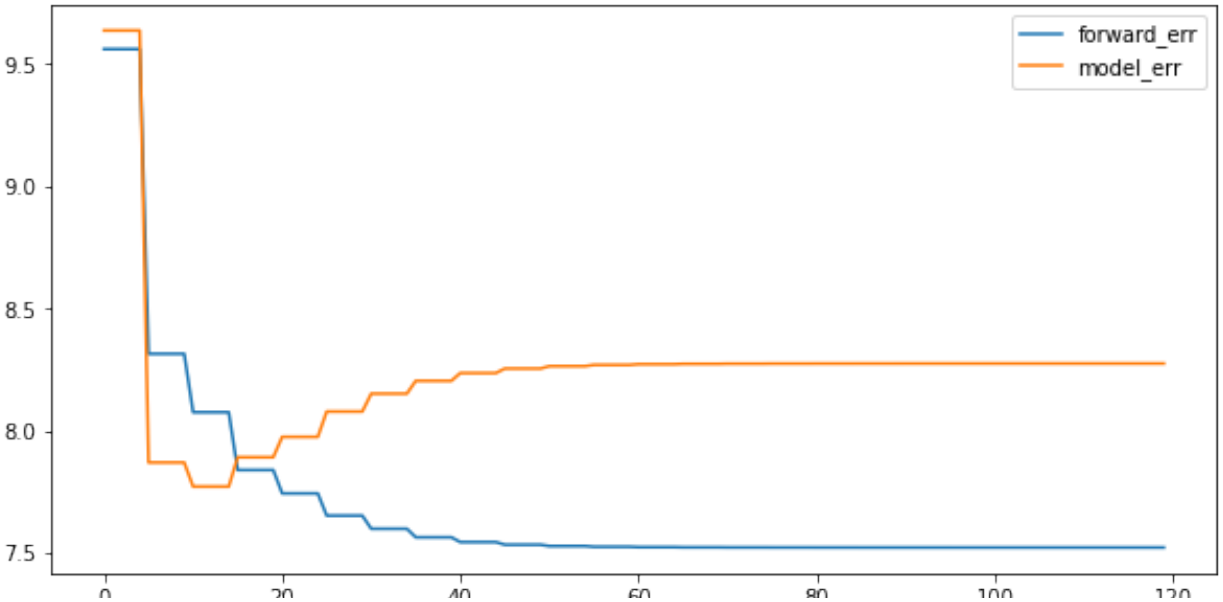

0.005


8
7
6
5
4
3
2
0
20
40
60
80
100
120

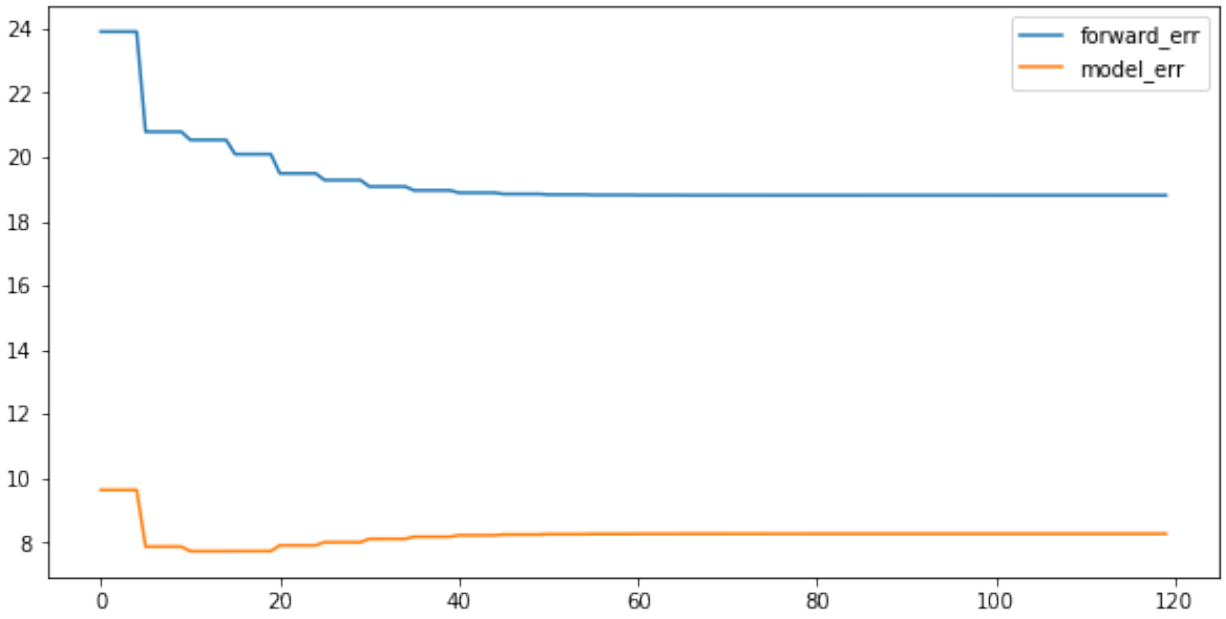
0.01


forward_err
model_err
9
8
7
6
5
4
0
20
40
60
80
100

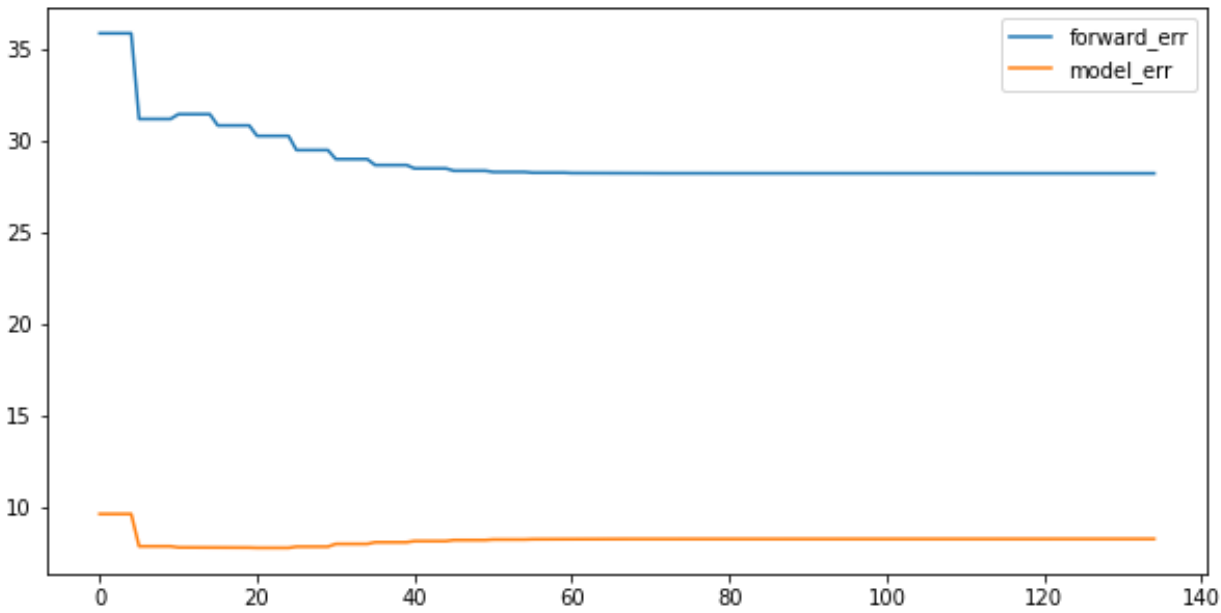
0.02


forward_err
model_err
9.5
9.0
8.5
8.0
7.5
0
20
40
60
80
100
120

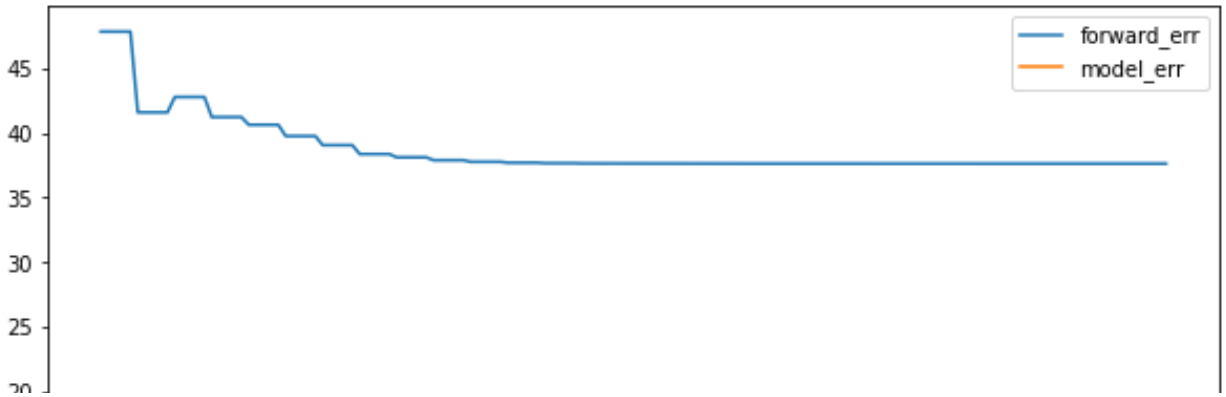
0.05

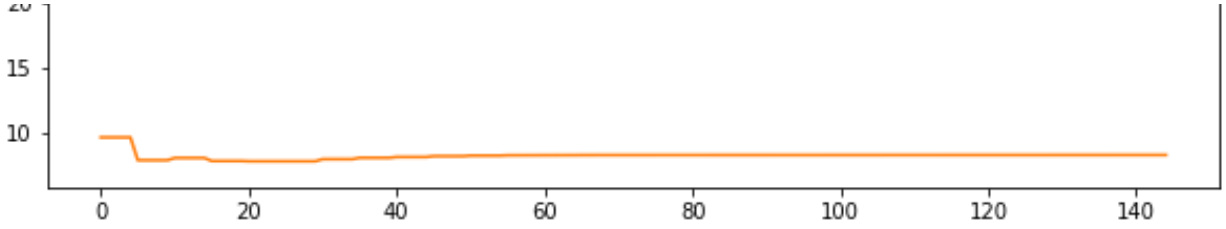

forward_err
model_err
24
22
20
18
16
14
12
10
8
0
20
40
60
80
100
120

0.075

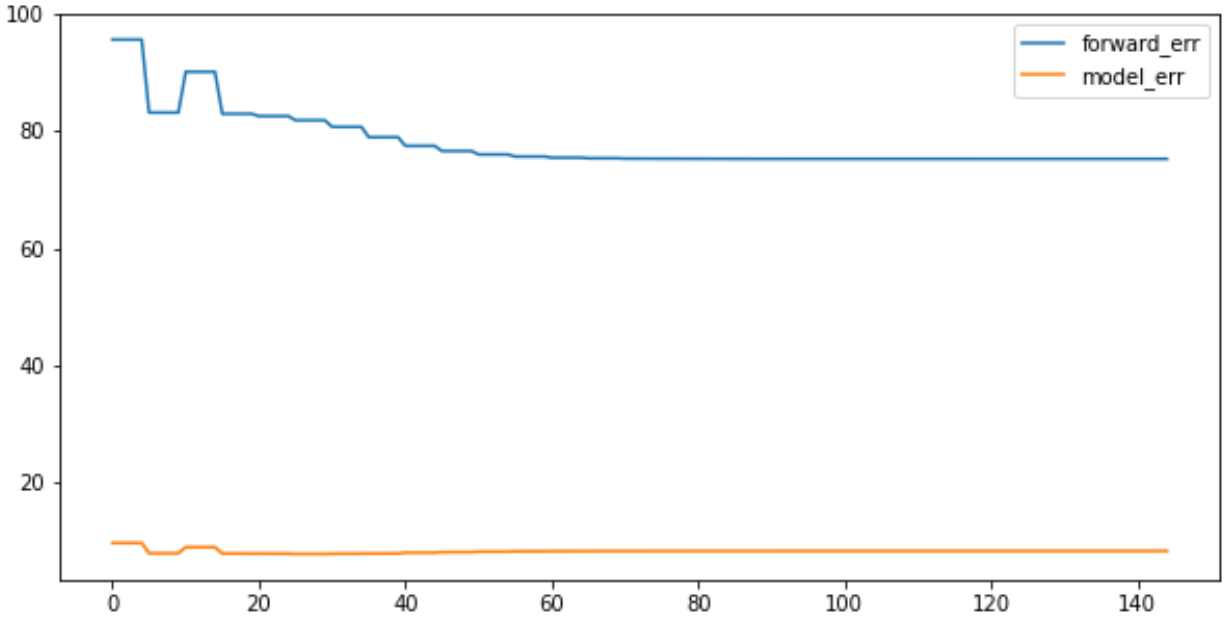

forward_err
model_err
35
30
25
20
15
10
0
20
40
60
80
100
120
140

0.1

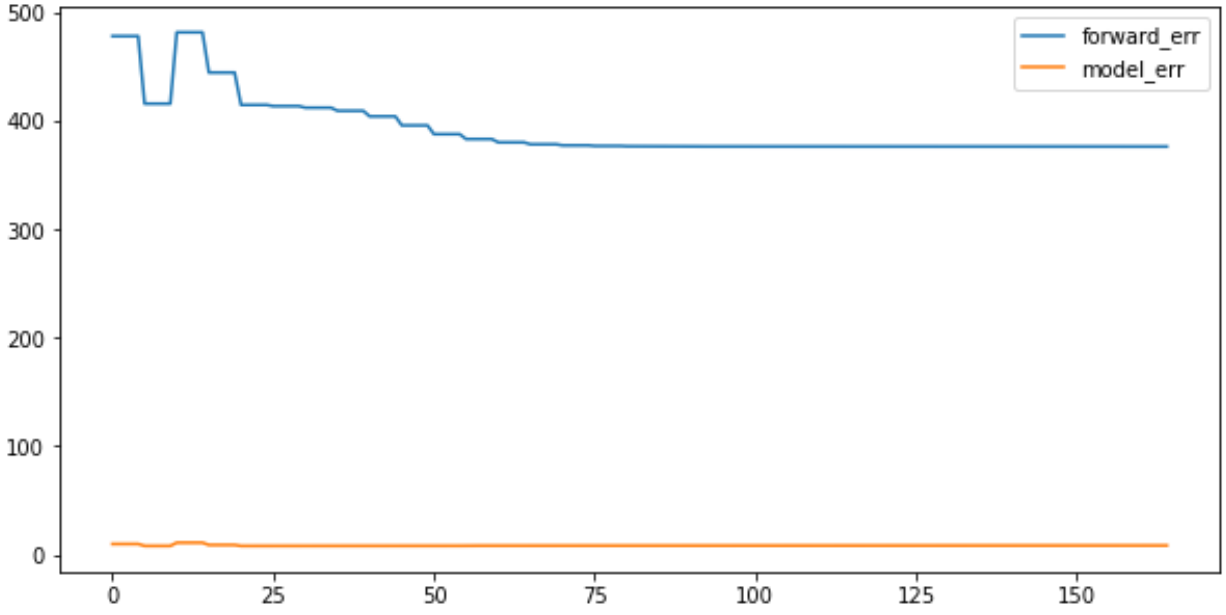

forward_err
model_err
45
40
35
30
25




0.2

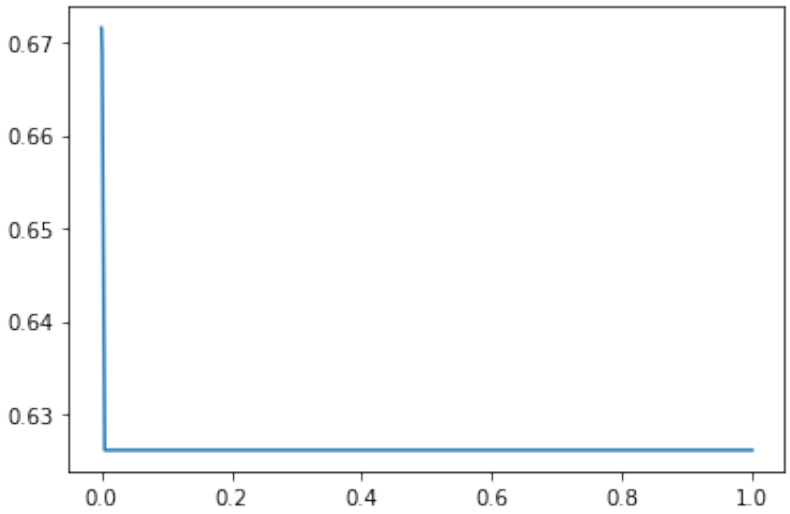



1




[22.86087245 16.32019886 16.07965857 18.56244053]

In [22]:
```
plt.plot(lambda_range, acc)
plt.show()
```




In [25]:
```
c1_fair = CustomLinearModel(X=C1.X_train.values, Y=C1.y_train.values,
c1_fair.fit(maxiter=250)
c1_fair_preds = np.round(c1_fair.predict(C1.X_test.values))
```

In [30]:
```
c1_fair_preds
```

Out[30]:
```
array([1., 1., 1., ..., 1., 1., 1.])
```

In [35]:
```
c1_fair_metrics = {}

c1_fair_metrics['overall'] = compute_metrics(C1.y_test, c1_fair_preds)

for group_name in C1.test_groups.keys():
    X_test_group = C1.test_groups[group_name]
    c1_fair_metrics[group_name] = compute_metrics(C1.y_test[C1.test_gr
```

In [50]:
```
pd.DataFrame(C1.metrics)
```

Out[50]:

| | overall | SEX_SES_priv | SEX_SES_dis | SEX_priv | SEX_dis | SES_priv | SES_di |
|---|---|---|---|---|---|---|---|
| **TPR** | 0.751161 | 0.000000 | 0.963347 | 0.616722 | 0.893451 | 0.216769 | 0.80022 |
| **TNR** | 0.738590 | 1.000000 | 0.290323 | 0.808608 | 0.546512 | 0.955776 | 0.61535 |
| **PPV** | 0.799707 | NaN | 0.846046 | 0.758536 | 0.832728 | 0.612717 | 0.80582 |
| **FNR** | 0.248839 | 1.000000 | 0.036653 | 0.383278 | 0.106549 | 0.783231 | 0.19977 |
| **FPR** | 0.261410 | 0.000000 | 0.709677 | 0.191392 | 0.453488 | 0.044224 | 0.38464 |
| **Accuracy** | 0.745900 | 0.853904 | 0.830032 | 0.713885 | 0.795080 | 0.775449 | 0.73849 |
| **F1** | 0.774674 | 0.000000 | 0.900894 | 0.680317 | 0.862022 | 0.320242 | 0.80301 |
| **Statistical_Parity** | 0.939295 | 0.000000 | 1.138645 | 0.813043 | 1.072920 | 0.353783 | 0.99305 |

In [51]:
```
pd.DataFrame(c1_fair_metrics)
```

Out[51]:

| | overall | SEX_SES_priv | SEX_SES_dis | SEX_priv | SEX_dis | SES_priv | SES_di |
|---|---|---|---|---|---|---|---|
| **TPR** | 0.989682 | 0.655172 | 1.000000 | 0.979933 | 1.000000 | 0.877301 | 1.00000 |
| **TNR** | 0.120908 | 0.497542 | 0.000000 | 0.164982 | 0.000000 | 0.333993 | 0.00000 |
| **PPV** | 0.610028 | 0.182400 | 0.801917 | 0.533600 | 0.716460 | 0.298331 | 0.66608 |
| **FNR** | 0.010318 | 0.344828 | 0.000000 | 0.020067 | 0.000000 | 0.122699 | 0.00000 |
| **FPR** | 0.879092 | 0.502458 | 1.000000 | 0.835018 | 1.000000 | 0.666007 | 1.00000 |
| **Accuracy** | 0.626100 | 0.520571 | 0.801917 | 0.567278 | 0.716460 | 0.466567 | 0.66608 |
| **F1** | 0.754804 | 0.285357 | 0.890071 | 0.690956 | 0.834811 | 0.445252 | 0.79958 |
| **Statistical_Parity** | 1.622356 | 3.591954 | 1.247012 | 1.836455 | 1.395752 | 2.940695 | 1.50131 |