Project 4

A5: Prejudice Remover Regularizer

In [1]:
```
# A5
# I built three models, the differnces only come from X (features).
# The first model uses 5 features mentioned in A7's paper;
# The second model drops 2 more features based on A7's result;
# The third model chooses features based on correlation.
# Each model is compared to the logistic regression model.
# Due to randomness, we might need to reset the value of eta based
on the three graphs of accuracy, calibration, and parity.
```

In [2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import torch as t
import torch.nn as nn
from torch.nn import functional as F
import matplotlib.pyplot as plt
import random
random.seed(10)
```

In [3]:
```
# model I
# We only focused on age_cat, priors_count, gender_cat, charge_cat,
length_stay, and sensitive feature.
```

In [4]:
```
df=pd.read_csv('../output/cleaned_compas.csv')
df = df.drop(columns=["Unnamed: 0"])
```

In [5]:
```
df
```

Out[5]:

	age_cat	priors_count	two_year_recid	race_cat	gender_cat	charge_cat	length_stay
0	0.5	0.0	1	0	0	1	0.5
1	0.0	1.0	1	0	0	1	0.0
2	0.5	1.0	1	1	0	1	0.0
3	0.5	0.0	0	1	1	0	0.0
4	0.0	0.5	1	1	0	1	0.0
...	...	...	...	...	...	...	...
5910	0.5	0.0	1	0	0	0	0.0
5911	0.0	0.0	0	0	0	1	0.0
5912	0.0	0.0	0	0	0	1	0.0
5913	0.0	0.0	0	0	0	1	0.0
5914	0.5	0.5	0	0	1	0	0.0

5915 rows × 7 columns

In [6]:
```
# Data splitting: Training:Testing:Validation=5:1:1
df_a = df[(df['race_cat'] == 0)]
df_c = df[(df['race_cat'] == 1)]
df_a = df_a.drop(columns=["race_cat"])
df_c = df_c.drop(columns=["race_cat"])

X_a = df_a.drop(columns = ['two_year_recid']).copy()
X_c = df_c.drop(columns = ['two_year_recid']).copy()
y_a = df_a['two_year_recid']
y_c = df_c['two_year_recid']

X_a_train, X_a_tv, y_a_train, y_a_tv = train_test_split(X_a, y_a, t
rain_size=5/7)
X_a_test, X_a_valid, y_a_test, y_a_valid = train_test_split(X_a_tv,
y_a_tv, test_size=1/2)
X_c_train, X_c_tv, y_c_train, y_c_tv = train_test_split(X_c, y_c, t
rain_size=5/7)
X_c_test, X_c_valid, y_c_test, y_c_valid = train_test_split(X_c_tv,
y_c_tv, test_size=1/2)

X_train = pd.concat([X_a_train, X_c_train])
y_train = pd.concat([y_a_train, y_c_train])
X_test = pd.concat([X_a_test, X_c_test])
y_test = pd.concat([y_a_test, y_c_test])
X_valid = pd.concat([X_a_valid, X_c_valid])
y_valid = pd.concat([y_a_valid, y_c_valid])
```

In [7]:
```
X_a_train = t.tensor(np.array(X_a_train)).to(t.float32)
y_a_train = t.from_numpy(np.array(y_a_train).astype('float32')).res
hape(X_a_train.shape[0], 1)
X_c_train = t.tensor(np.array(X_c_train)).to(t.float32)
y_c_train = t.from_numpy(np.array(y_c_train).astype('float32')).res
hape(X_c_train.shape[0], 1)

X_a_test = t.tensor(np.array(X_a_test)).to(t.float32)
y_a_test = t.from_numpy(np.array(y_a_test).astype('float32')).resha
pe(X_a_test.shape[0], 1)
X_c_test = t.tensor(np.array(X_c_test)).to(t.float32)
y_c_test = t.from_numpy(np.array(y_c_test).astype('float32')).resha
pe(X_c_test.shape[0], 1)

X_a_valid = t.tensor(np.array(X_a_valid)).to(t.float32)
y_a_valid = t.from_numpy(np.array(y_a_valid).astype('float32')).res
hape(X_a_valid.shape[0], 1)
X_c_valid = t.tensor(np.array(X_c_valid)).to(t.float32)
y_c_valid = t.from_numpy(np.array(y_c_valid).astype('float32')).res
hape(X_c_valid.shape[0], 1)
```

In [8]:
```
# We used 0.5 as threshold, and used Accuracy, Calibration, and Par
ity as evaluation metrics.

def Evaluation(Model_a, Model_c, X_a, y_a, X_c, y_c):
    y_a_pred = (Model_a(X_a) >= 0.5)
    y_c_pred = (Model_c(X_c) >= 0.5)
    acc_a  = t.sum(y_a_pred.flatten() == y_a.flatten()) / X_a.shape
[0]
    acc_c  = t.sum(y_c_pred.flatten() == y_c.flatten()) / X_c.shape
[0]
    resid_a = t.sum(y_a_pred == True) / X_a.shape[0]
    resid_c = t.sum(y_c_pred == True) / X_c.shape[0]
    accuracy = (acc_c + acc_a) / 2
    calibration = t.abs(acc_a - acc_c)
    parity = t.abs(resid_a - resid_c)
    return round(accuracy.item(), 4), round(calibration.item(), 4),
round(parity.item(), 4)
```

In [9]:
```
class LogisticRegression(nn.Module):
    def __init__(self,df):
        super(LogisticRegression, self).__init__()
        self.w = nn.Linear(df.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 [10]:
```
class PRLoss():
    def __init__(self, eta=1.0):
        super(PRLoss, self).__init__()
        self.eta = eta
    def forward(self, output_a, output_c):
        # Approximating the true distribution of data by the sample
distribution
        # eqn(9) in paper: hat{Pr}[y|s] = sum{(xi,si), s.t. si=s} M
odel(y|xi,s;theta) / |D[xs]|
        #D[xs]
        N_a = t.tensor(output_a.shape[0])
        N_c = t.tensor(output_c.shape[0])
        D_xs = t.stack((N_a, N_c), axis=0)
        # Pr[y|s]
        y_pred_a = t.sum(output_a)
        y_pred_c = t.sum(output_c)
        P_y_s = t.stack((y_pred_a, y_pred_c), axis=0) / D_xs
        # eqn(10) in paper: hat{Pr}[y] = sum{(xi,si)} Model(y|xi,s
i;theta) / |D[xs]|
        P = t.cat((output_a, output_c), 0)
        P_y = t.sum(P) / (X_a_train.shape[0]+X_c_train.shape[0])
        # P(yi|si)
        P_1_1 = t.log(P_y_s[1]) - t.log(P_y)
        P_0_1 = t.log(1-P_y_s[1]) - t.log(1-P_y)
        P_1_0 = t.log(P_y_s[0]) - t.log(P_y)
        P_0_0 = t.log(1-P_y_s[0]) - t.log(1-P_y)
        # eqn(11) in paper: prejudice remover regularizer R_PR(D, t
heta)
        # R_PR = sum{xi,si}sum{y} Model(y|xi,s;theta) * ln(hat{P
r}[y|si]/hat{Pr}[y])
        R_PR_1_1 = output_c * P_1_1
        R_PR_0_1 =(1- output_c) * P_0_1
        R_PR_1_0 = output_a * P_1_0
        R_PR_0_0 = (1- output_a) * P_0_0
        R_PR = t.sum(R_PR_1_1) + t.sum(R_PR_0_1) + t.sum(R_PR_1_0)
+ t.sum(R_PR_0_0)
        R_PR = self.eta * R_PR
        return R_PR
```

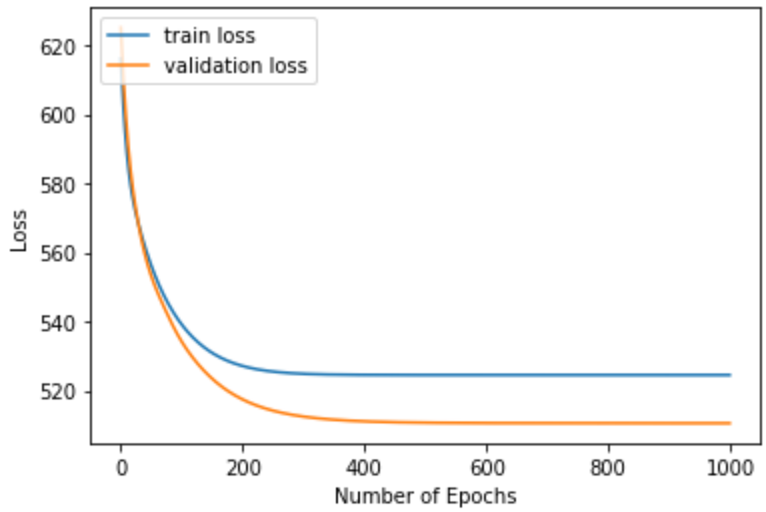
In [11]:

```
class PRLR():
    def __init__(self, eta = 0.0, epochs = 300, lr = 0.01):
        super(PRLR, self).__init__()
        self.eta = eta
        self.epochs = epochs
        self.lr = lr
    def fit(self, X_a_train, y_a_train, X_c_train, y_c_train, X_a_valid, y_a_valid,
            X_c_valid, y_c_valid, X_a_test, y_a_test, X_c_test, y_c_test):
        model_a = LogisticRegression(X_a_train)
        model_c = LogisticRegression(X_c_train)
        criterion = nn.BCELoss(reduction='sum')
        PI = PRLoss(eta=self.eta) # Prejudice index
        epochs = self.epochs
        optimizer = t.optim.Adam(list(model_c.parameters())+list(model_a.parameters()), self.lr, weight_decay=1e-5)
        train_losses = []
        valid_losses = []
        for epoch in range(epochs):
            model_c.train()
            model_a.train()
            optimizer.zero_grad()
            output_a = model_a(X_a_train)
            output_c = model_c(X_c_train)
            logloss = criterion(output_a, y_a_train) + criterion(output_c, y_c_train)
            PRloss = PI.forward(output_a, output_c)
            loss = (PRloss + logloss)/5
            loss.backward()
            optimizer.step()
            train_losses.append(loss.detach().numpy())
            output_a = model_a(X_a_valid)
            output_c = model_c(X_c_valid)
            logloss = criterion(output_a, y_a_valid) + criterion(output_c, y_c_valid)
            PRloss = PI.forward(output_a, output_c)
            loss = PRloss + logloss
            valid_losses.append(loss.detach().numpy())
        model_a.eval()
        model_c.eval()
        eva = Evaluation(model_a, model_c, X_a_train, y_a_train, X_c_train, y_c_train)
        eva_valid = Evaluation(model_a, model_c, X_a_valid, y_a_valid, X_c_valid, y_c_valid)
        eva_test = Evaluation(model_c, model_a, X_a_test, y_a_test, X_c_test, y_c_test)
        plt.plot(list(range(epochs)), train_losses, label="train loss")
        plt.plot(list(range(epochs)), valid_losses, label="validation loss")
        plt.legend(loc="upper left")
        plt.xlabel('Number of Epochs')
        plt.ylabel('Loss')
        plt.show()
```

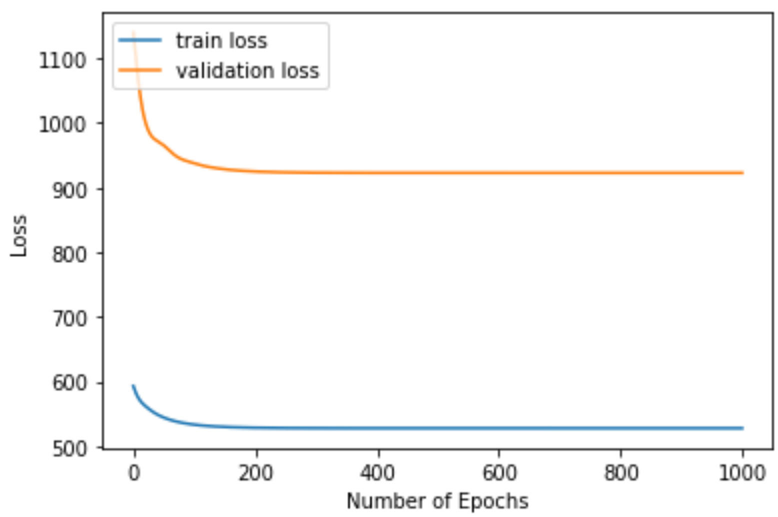
```
        return eva, eva_valid, eva_test
```

In [12]:
```
eta_value_I = [0, 1, 2, 3, 4, 5, 6, 8, 10, 15]
evalu_I = list()
evalu_valid_I = list()
evalu_test_I = list()
for i in range(0, len(eta_value_I)):
    print("eta Value: %d" % eta_value_I[i])
    PR_I = PRLR(eta = eta_value_I[i], epochs = 1000, lr = 0.01)
    eva_I, eva_valid_I, eva_test_I = PR_I.fit(X_a_train, y_a_train, 
X_c_train, y_c_train, X_a_valid, y_a_valid,
                                               X_c_valid, y_c_valid, 
X_a_test, y_a_test, X_c_test, y_c_test)
    evalu_I.append(eva_I)
    evalu_valid_I.append(eva_valid_I)
    evalu_test_I.append(eva_test_I)
```

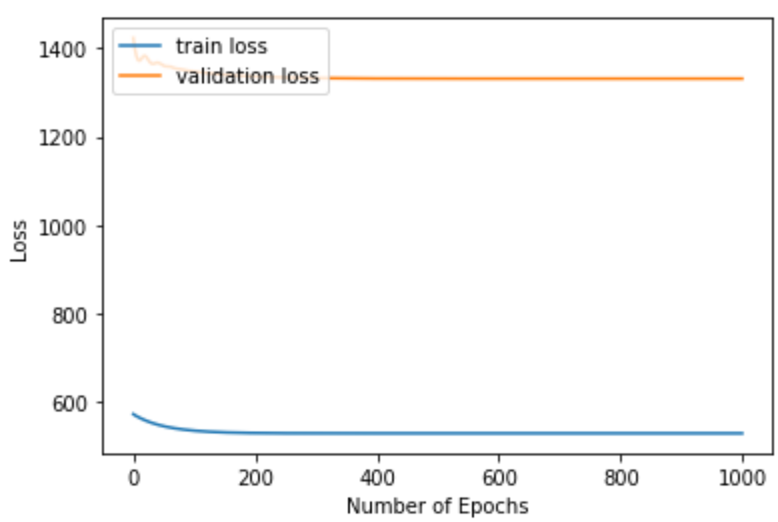
```
eta Value: 0
```



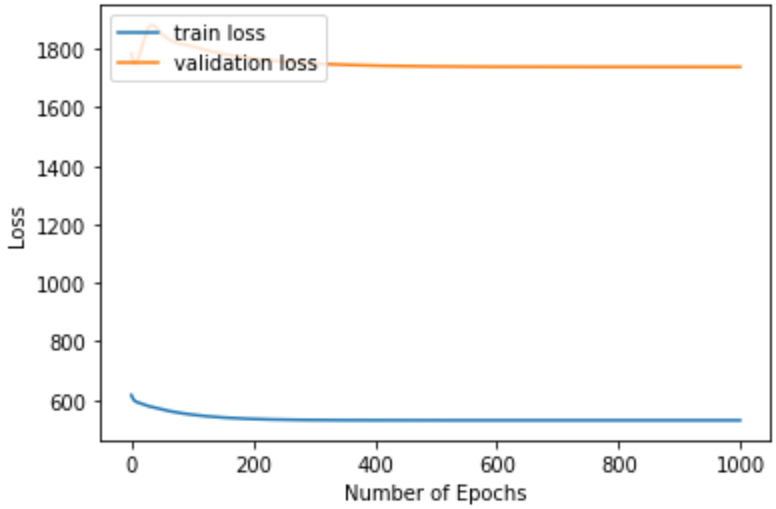

```
eta Value: 1
```



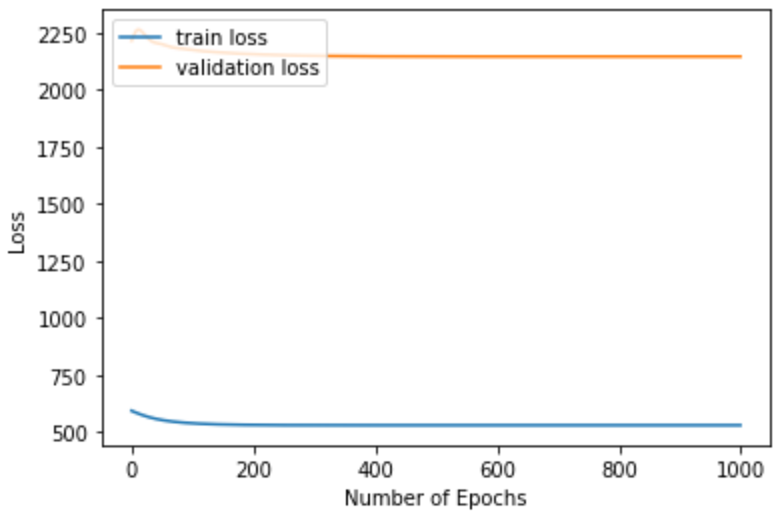

```
eta Value: 2
```



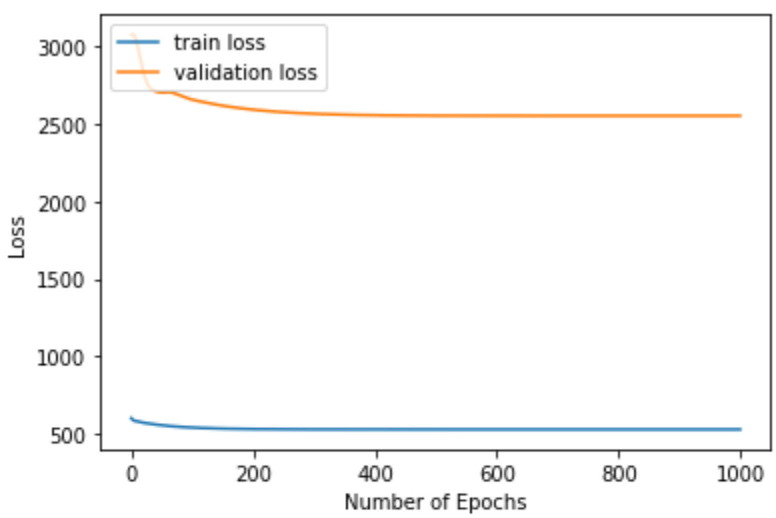

```
eta Value: 3
```



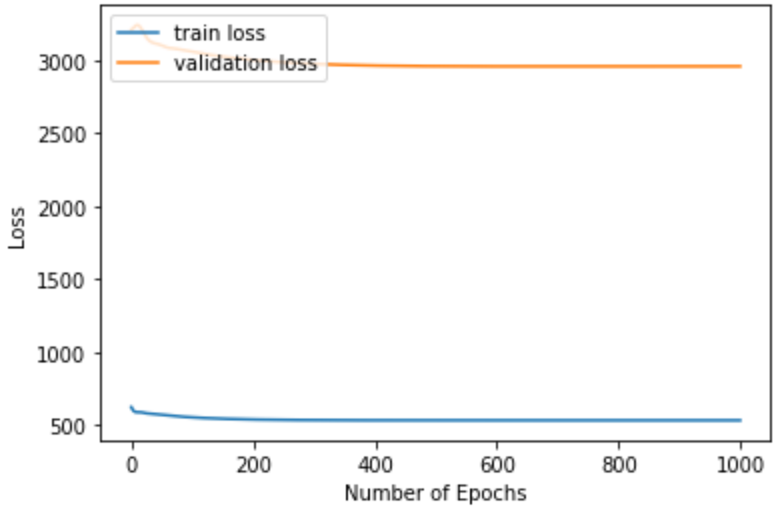
eta Value: 4



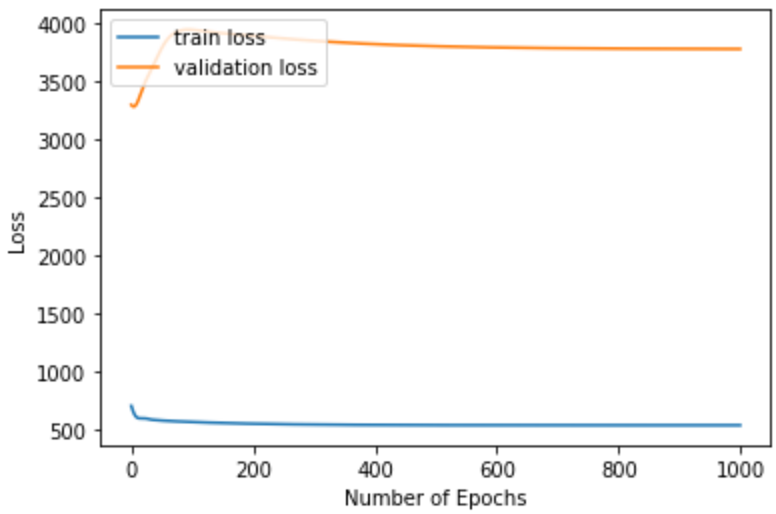
eta Value: 5



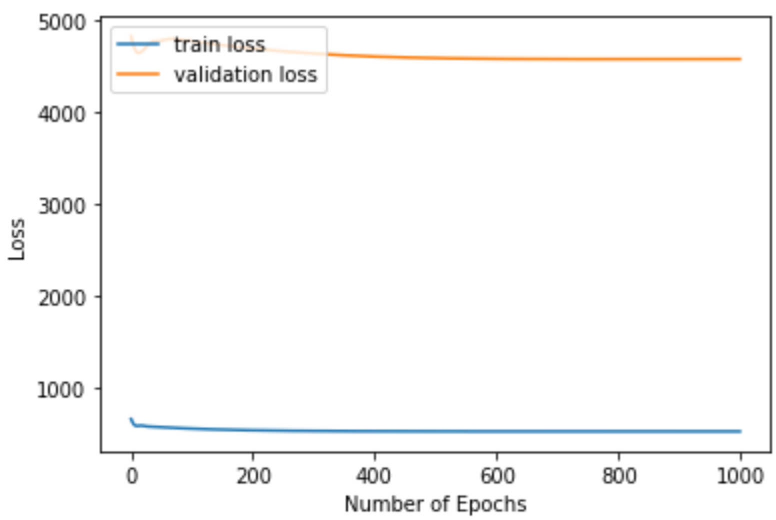
eta Value: 6



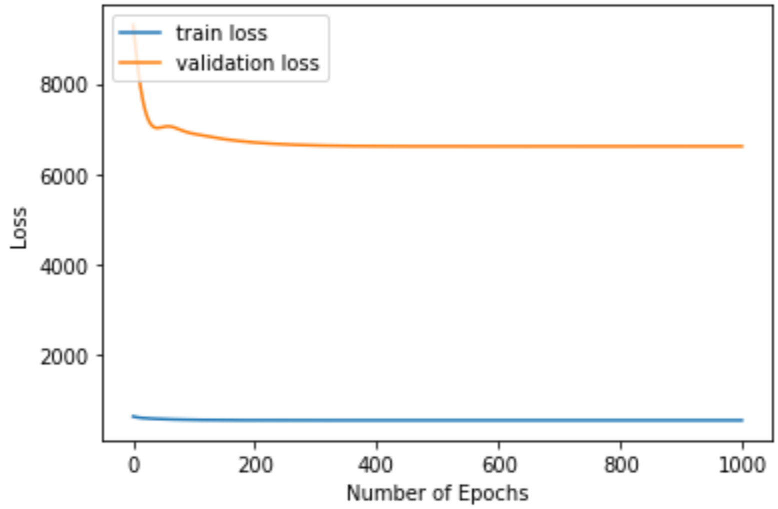
eta Value: 8



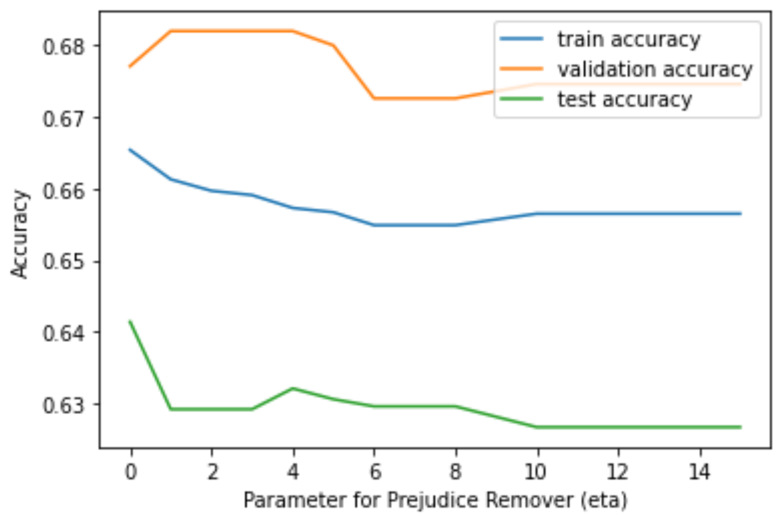
eta Value: 10



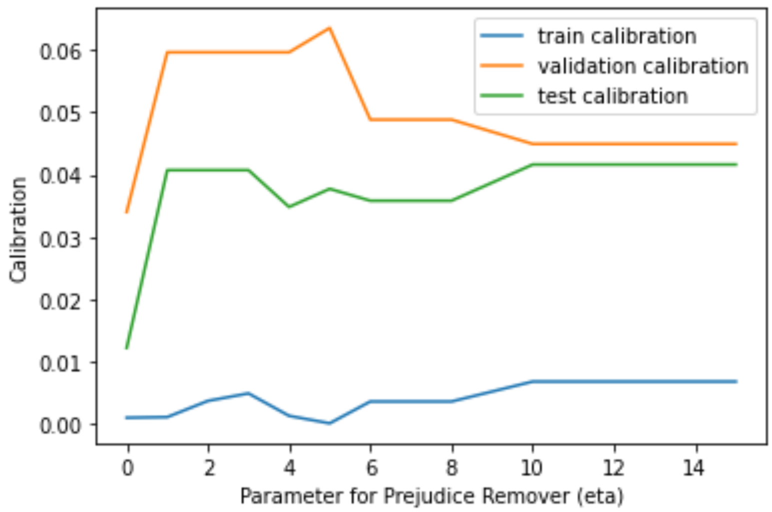
eta Value: 15



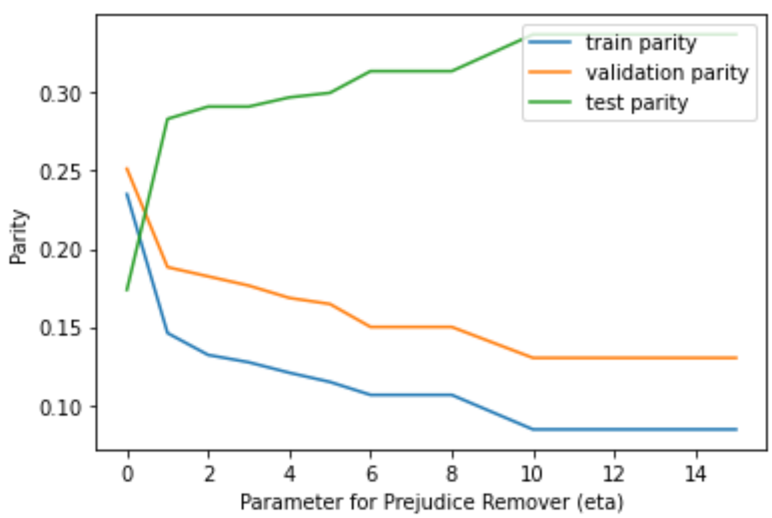
In [13]:
```
eta_acc_train_I = [x[0] for x in evalu_I]
eta_acc_valid_I = [x[0] for x in evalu_valid_I]
eta_acc_test_I = [x[0] for x in evalu_test_I]
plt.plot(eta_value_I, eta_acc_train_I, label="train accuracy")
plt.plot(eta_value_I, eta_acc_valid_I, label="validation accuracy")
plt.plot(eta_value_I, eta_acc_test_I, label="test accuracy")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Accuracy')
plt.legend(loc="upper right")
plt.show()
```



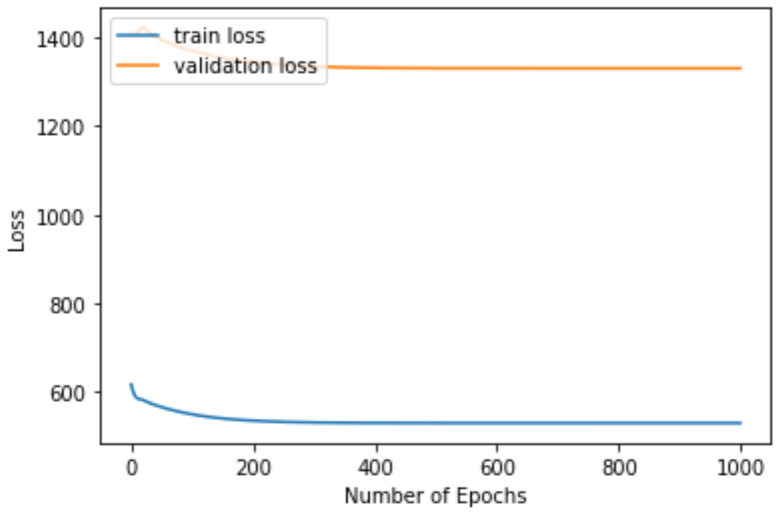
In [14]:
```
eta_cal_train_I = [x[1] for x in evalu_I]
eta_cal_valid_I = [x[1] for x in evalu_valid_I]
eta_cal_test_I = [x[1] for x in evalu_test_I]
plt.plot(eta_value_I, eta_cal_train_I, label="train calibration")
plt.plot(eta_value_I, eta_cal_valid_I, label="validation calibratio
n")
plt.plot(eta_value_I, eta_cal_test_I, label="test calibration")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Calibration')
plt.legend(loc="upper right")
plt.show()
```



In [15]:
```
eta_par_train_I = [x[2] for x in evalu_I]
eta_par_valid_I = [x[2] for x in evalu_valid_I]
eta_par_test_I = [x[2] for x in evalu_test_I]
plt.plot(eta_value_I, eta_par_train_I, label="train parity")
plt.plot(eta_value_I, eta_par_valid_I, label="validation parity")
plt.plot(eta_value_I, eta_par_test_I, label="test parity")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Parity')
plt.legend(loc="upper right")
plt.show()
```

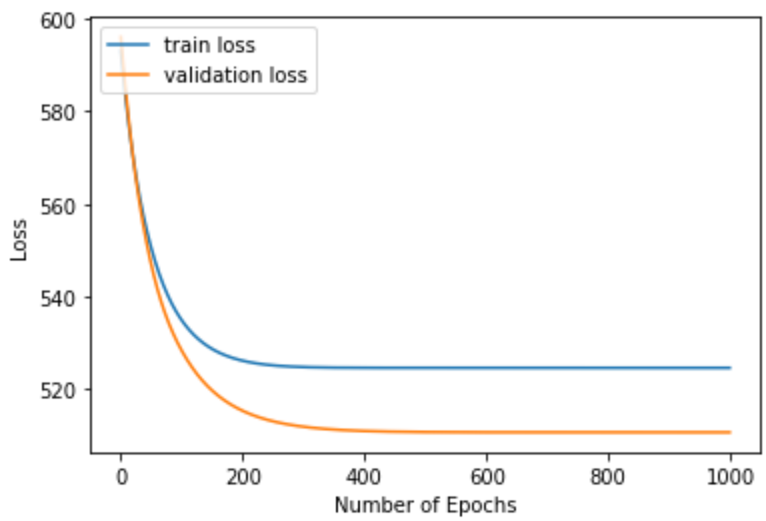


In [16]:
```
# Final model I
# To achieve high accuracy, low calibration and low parity, we deci
ded to choose eta = 2.
# The outputs are (accuracy, calibration, parity) of training , val
idation, and testing sets.
PR_final_I = PRLR(eta = 2, epochs = 1000, lr = 0.01)
PR_final_I.fit(X_a_train, y_a_train, X_c_train, y_c_train, X_a_vali
d, y_a_valid,
               X_c_valid, y_c_valid, X_a_test, y_a_test, X_c_test,
y_c_test)
```



Out[16]:
```
((0.6597, 0.0037, 0.1321), (0.682, 0.0596, 0.1824), (0.6292, 0.0407,
0.2911))
```

In [17]:
```
# Compared to logistic regression without prejudice remover regular
izer
PR_0 = PRLR(eta = 0, epochs = 1000, lr = 0.01)
PR_0.fit(X_a_train, y_a_train, X_c_train, y_c_train, X_a_valid, y_a
_valid,
         X_c_valid, y_c_valid, X_a_test, y_a_test, X_c_test, y_c_te
st)
```



Out[17]:
```
((0.6654, 0.001, 0.235), (0.6771, 0.034, 0.2513), (0.6414, 0.0122,
0.1738))
```

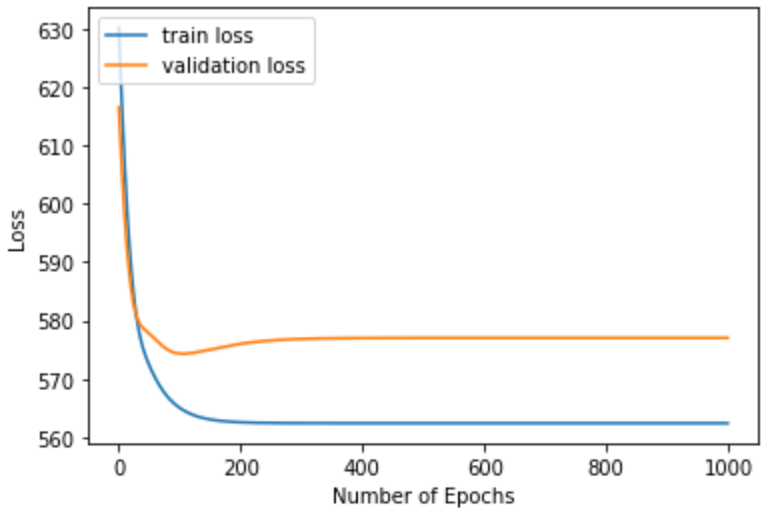
In [18]:
```
# model II
# Based on A7, we dropped "Age" and "Prior Count", and repeat the p
rocess above.
```

In [19]:
```
new_df_a = df_a.drop(columns=["age_cat", "priors_count"])
new_df_c = df_c.drop(columns=["age_cat", "priors_count"])

new_X_a = new_df_a.drop(columns = ['two_year_recid']).copy()
new_X_c = new_df_c.drop(columns = ['two_year_recid']).copy()

new_X_a_train, new_X_a_tv, y_a_train, y_a_tv = train_test_split(new
_X_a, y_a, train_size=5/7)
new_X_a_test, new_X_a_valid, y_a_test, y_a_valid = train_test_split
(new_X_a_tv, y_a_tv, test_size=1/2)
new_X_c_train, new_X_c_tv, y_c_train, y_c_tv = train_test_split(new
_X_c, y_c, train_size=5/7)
new_X_c_test, new_X_c_valid, y_c_test, y_c_valid = train_test_split
(new_X_c_tv, y_c_tv, test_size=1/2)

new_X_train = pd.concat([new_X_a_train, new_X_c_train])
y_train = pd.concat([y_a_train, y_c_train])
new_X_test = pd.concat([new_X_a_test, new_X_c_test])
y_test = pd.concat([y_a_test, y_c_test])
new_X_valid = pd.concat([new_X_a_valid, new_X_c_valid])
y_valid = pd.concat([y_a_valid, y_c_valid])
```

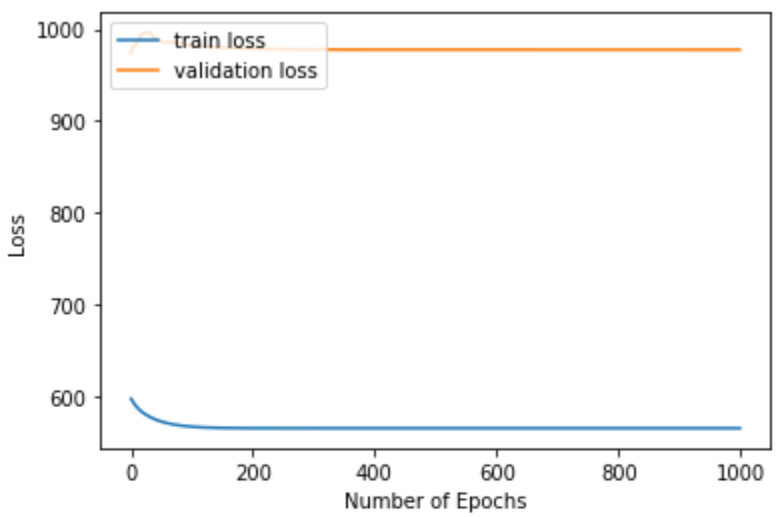
In [20]:
```
new_X_a_train = t.tensor(np.array(new_X_a_train)).to(t.float32)
y_a_train = t.from_numpy(np.array(y_a_train).astype('float32')).res
hape(new_X_a_train.shape[0], 1)
new_X_c_train = t.tensor(np.array(new_X_c_train)).to(t.float32)
y_c_train = t.from_numpy(np.array(y_c_train).astype('float32')).res
hape(new_X_c_train.shape[0], 1)

new_X_a_test = t.tensor(np.array(new_X_a_test)).to(t.float32)
y_a_test = t.from_numpy(np.array(y_a_test).astype('float32')).resha
pe(new_X_a_test.shape[0], 1)
new_X_c_test = t.tensor(np.array(new_X_c_test)).to(t.float32)
y_c_test = t.from_numpy(np.array(y_c_test).astype('float32')).resha
pe(new_X_c_test.shape[0], 1)

new_X_a_valid = t.tensor(np.array(new_X_a_valid)).to(t.float32)
y_a_valid = t.from_numpy(np.array(y_a_valid).astype('float32')).res
hape(new_X_a_valid.shape[0], 1)
new_X_c_valid = t.tensor(np.array(new_X_c_valid)).to(t.float32)
y_c_valid = t.from_numpy(np.array(y_c_valid).astype('float32')).res
hape(new_X_c_valid.shape[0], 1)
```

In [21]:
```
eta_value_II = [0, 1, 2, 3, 4, 5, 6, 8, 10, 15]
new_evalu = list()
new_evalu_valid = list()
new_evalu_test = list()
for i in range(0, len(eta_value_II)):
    print("eta Value: %d" % eta_value_II[i])
    PR_II = PRLR(eta = eta_value_II[i], epochs = 1000, lr = 0.01)
    new_eva, new_eva_valid, new_eva_test = PR_II.fit(new_X_a_train,
y_a_train, new_X_c_train, y_c_train,
                                                     new_X_a_valid,
y_a_valid, new_X_c_valid, y_c_valid,
                                                     new_X_a_test,
y_a_test, new_X_c_test, y_c_test)
    new_evalu.append(new_eva)
    new_evalu_valid.append(new_eva_valid)
    new_evalu_test.append(new_eva_test)
```

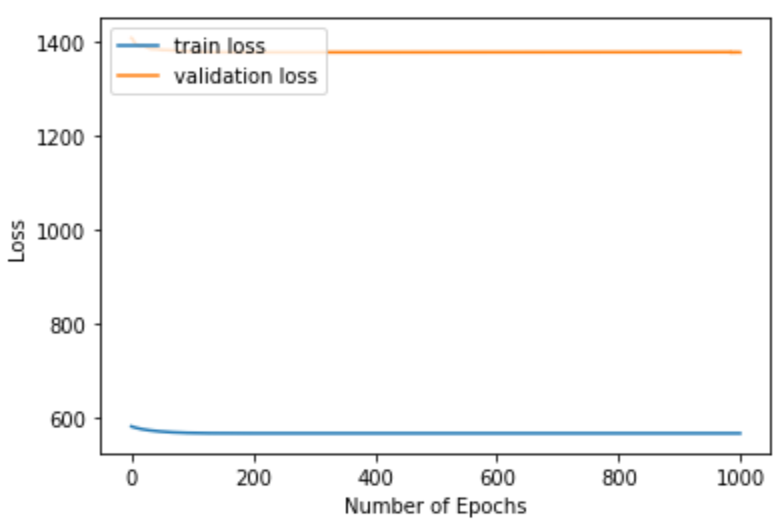
```
eta Value: 0
```



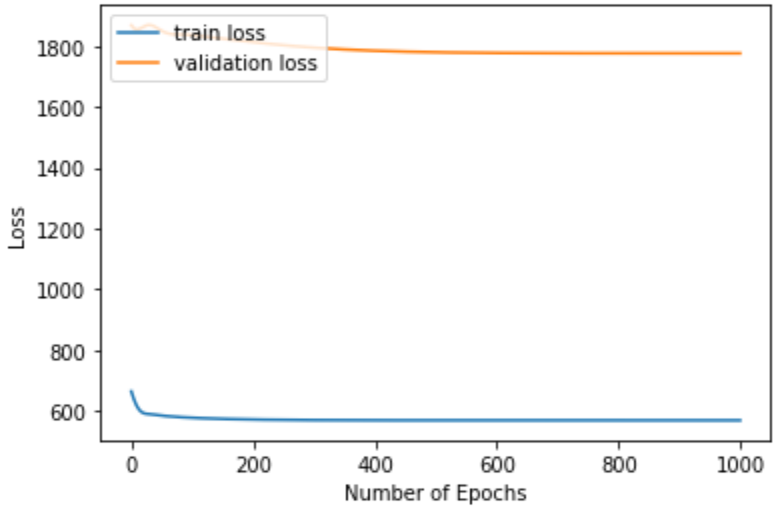

```
eta Value: 1
```



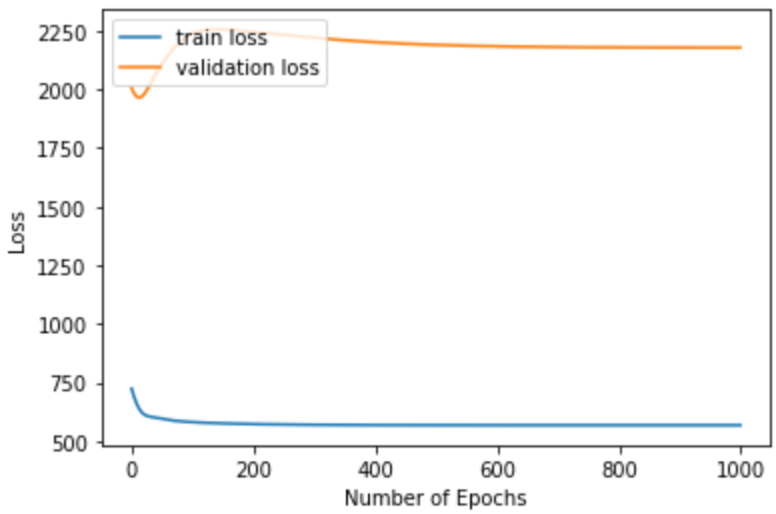

```
eta Value: 2
```



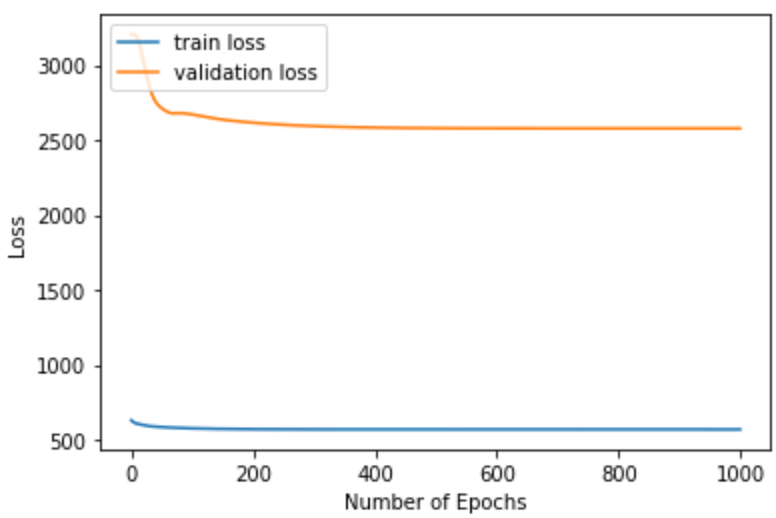

```
eta Value: 3
```



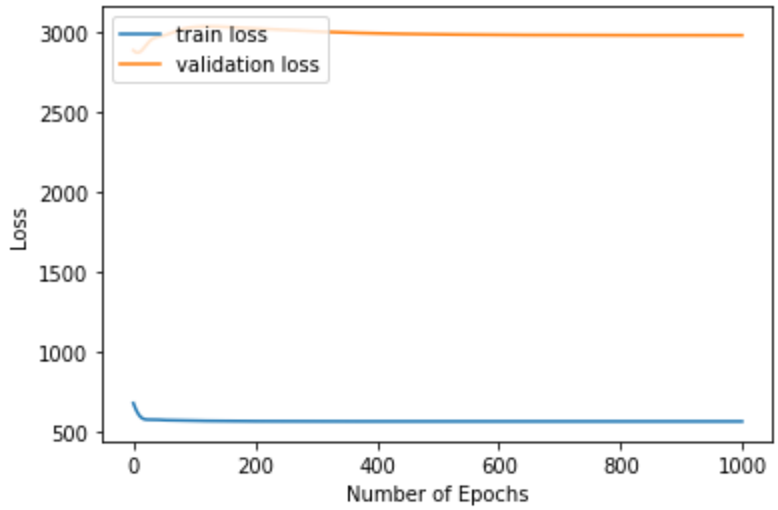
eta Value: 4



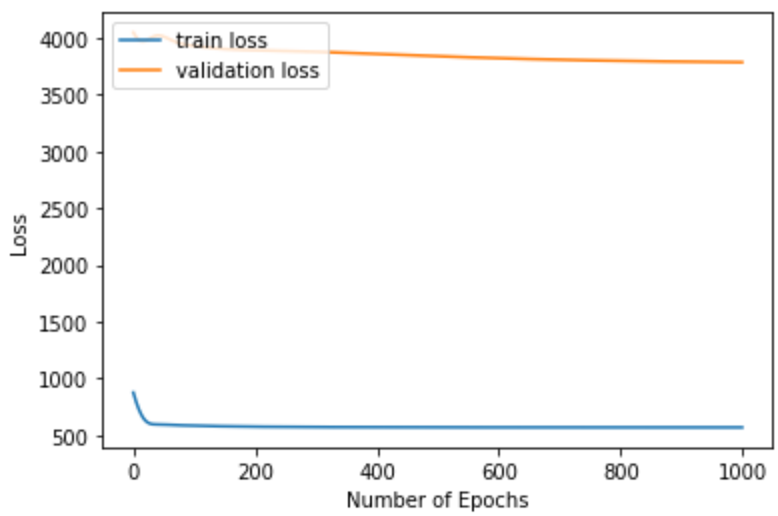
eta Value: 5



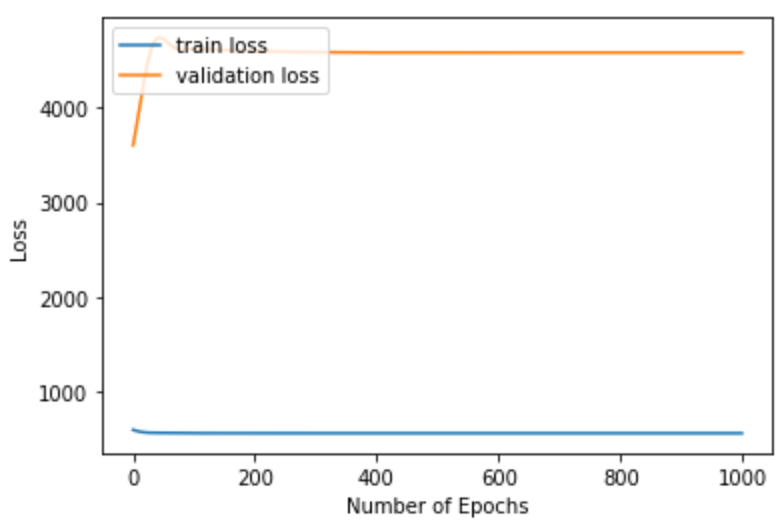
eta Value: 6



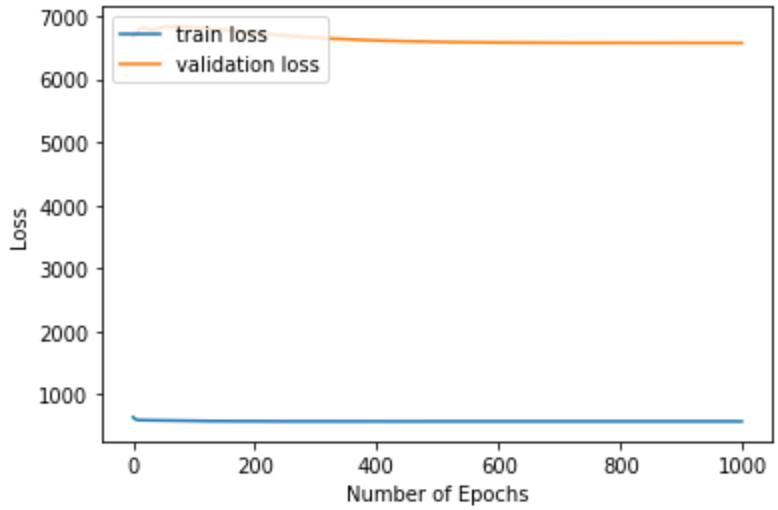
eta Value: 8



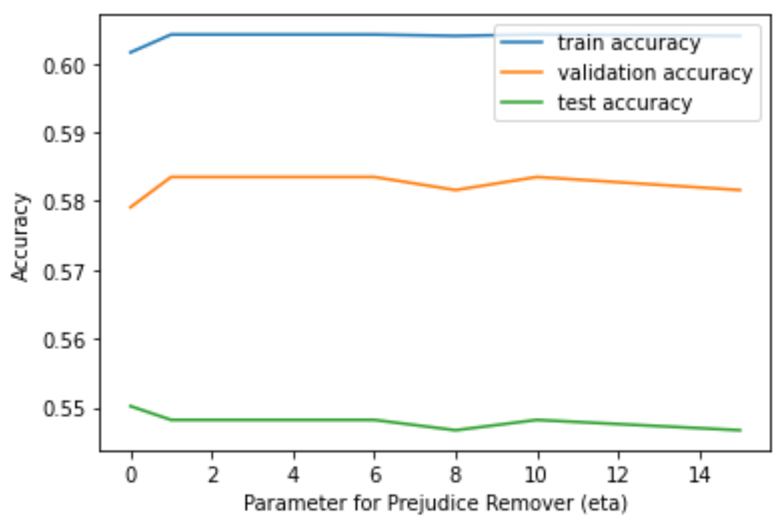
eta Value: 10



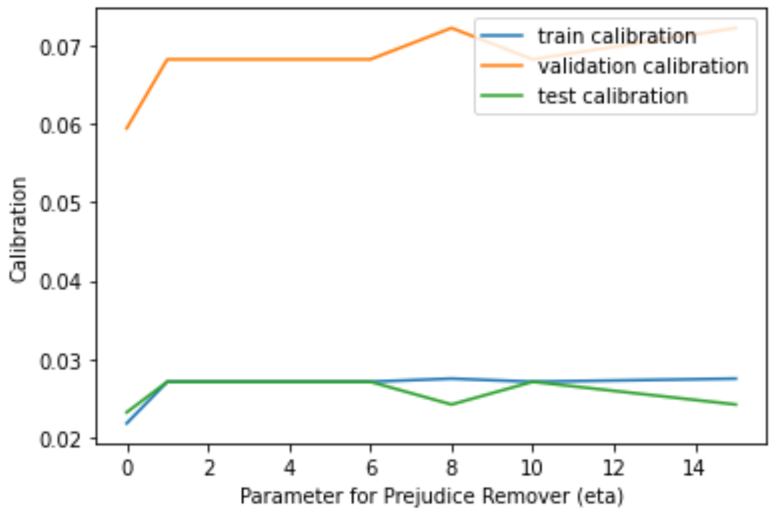
eta Value: 15



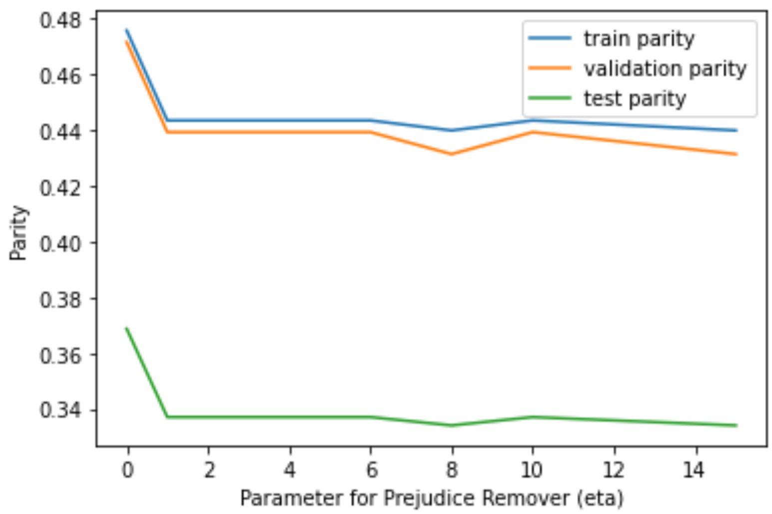
In [22]:
```
new_eta_acc_train = [x[0] for x in new_evalu]
new_eta_acc_valid = [x[0] for x in new_evalu_valid]
new_eta_acc_test = [x[0] for x in new_evalu_test]
plt.plot(eta_value_II, new_eta_acc_train, label="train accuracy")
plt.plot(eta_value_II, new_eta_acc_valid, label="validation accurac
y")
plt.plot(eta_value_II, new_eta_acc_test, label="test accuracy")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Accuracy')
plt.legend(loc="upper right")
plt.show()
```



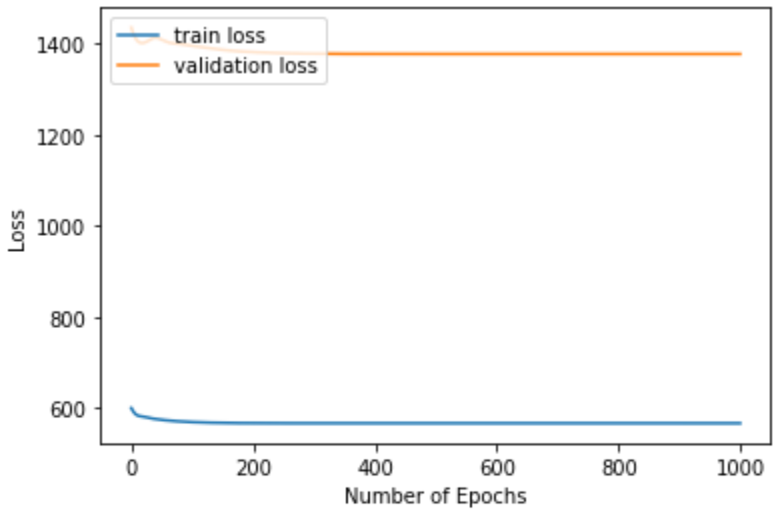
In [23]:
```
new_eta_cal_train = [x[1] for x in new_evalu]
new_eta_cal_valid = [x[1] for x in new_evalu_valid]
new_eta_cal_test = [x[1] for x in new_evalu_test]
plt.plot(eta_value_II, new_eta_cal_train, label="train calibratio
n")
plt.plot(eta_value_II, new_eta_cal_valid, label="validation calibra
tion")
plt.plot(eta_value_II, new_eta_cal_test, label="test calibration")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Calibration')
plt.legend(loc="upper right")
plt.show()
```



In [24]:
```
new_eta_par_train = [x[2] for x in new_evalu]
new_eta_par_valid = [x[2] for x in new_evalu_valid]
new_eta_par_test = [x[2] for x in new_evalu_test]
plt.plot(eta_value_II, new_eta_par_train, label="train parity")
plt.plot(eta_value_II, new_eta_par_valid, label="validation parit
y")
plt.plot(eta_value_II, new_eta_par_test, label="test parity")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Parity')
plt.legend(loc="upper right")
plt.show()
```

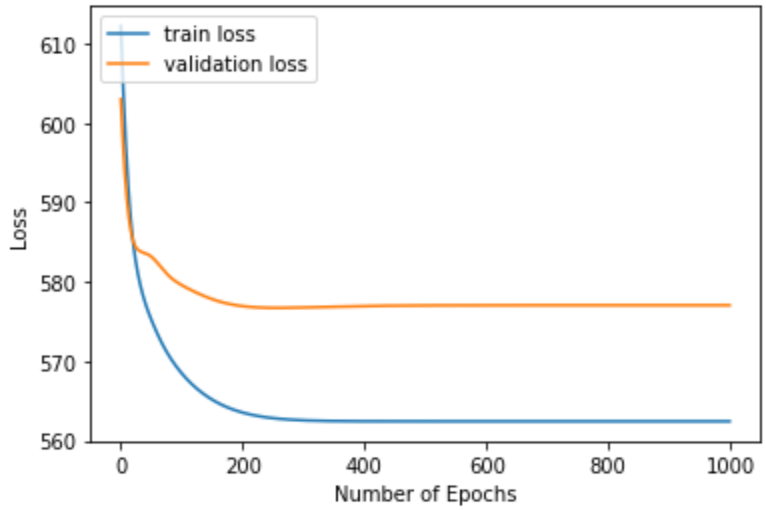


In [25]:
```
# Final model II
# To achieve high accuracy, low calibration and low parity, we deci
ded to choose eta = 2.
# The outputs are (accuracy, calibration, parity) of training , val
idation, and testing sets.
new_PR_final = PRLR(eta = 2, epochs = 1000, lr = 0.01)
new_PR_final.fit(new_X_a_train, y_a_train, new_X_c_train, y_c_trai
n, new_X_a_valid, y_a_valid,
                 new_X_c_valid, y_c_valid, new_X_a_test, y_a_test,
new_X_c_test, y_c_test)
```



Out[25]:
```
((0.6042, 0.0271, 0.4435), (0.5835, 0.0682, 0.4393), (0.5482, 0.027
1, 0.337))
```

In [26]:
```
# Compared to logistic regression without prejudice remover regular
izer
PR_0 = PRLR(eta = 0, epochs = 1000, lr = 0.01)
PR_0.fit(new_X_a_train, y_a_train, new_X_c_train, y_c_train, new_X_
a_valid, y_a_valid,
         new_X_c_valid, y_c_valid, new_X_a_test, y_a_test, new_X_c_
test, y_c_test)
```



Out[26]:
```
((0.6016, 0.0218, 0.4758), (0.5791, 0.0594, 0.4716), (0.5502, 0.023
2, 0.3687))
```

In [27]:
```
# model III
# If we choose different features (which have higher correlation wi
th y):
```

In [28]:

```python
compas = pd.read_csv("../data/compas-scores-two-years.csv")

compas = compas[(compas["race"]=='Caucasian') | (compas["race"]=='African-American')]
compas["race_cat"] = compas["race"].apply(lambda x: 1 if x == "Caucasian" else 0)
compas = compas.drop(columns = "race")
compas["gender_cat"] = compas["sex"].apply(lambda x: 1 if x == "Female" else 0)
compas = compas.drop(columns = "sex")
compas["charge_cat"] = compas["c_charge_degree"].apply(lambda x: 1 if x == "F" else 0)
compas = compas.drop(columns = "c_charge_degree")
compas["length_stay"] = pd.to_datetime(compas["c_jail_out"]) - pd.to_datetime(compas['c_jail_in'])
compas["length_stay"] = compas["length_stay"].apply(lambda x: x.days)
compas = compas.drop(columns = ["c_jail_in","c_jail_out"])
compas['length_stay'] = compas["length_stay"].apply(lambda x: 0 if x <= 7 else 0.5 if 7< x <= 90 else 1)
compas["priors_count"] = compas["priors_count"].apply(lambda x: 0 if x==0 else 0.5 if 1<=x<=3 else 1)
compas['age_cat'] = compas['age_cat'].apply(lambda x:0 if x == "Less than 25" else 0.5 if x == "25 - 45" else 1)
```

In [29]:

```python
print(compas.corr(method="kendall")['two_year_recid'].sort_values(ascending=False))
```

```
two_year_recid             1.000000
is_recid                   0.938762
event                      0.779733
is_violent_recid           0.351163
decile_score               0.303077
decile_score.1             0.303077
v_decile_score             0.257347
priors_count               0.256350
priors_count.1             0.256134
juv_other_count            0.148087
juv_misd_count             0.129031
start                      0.128130
juv_fel_count              0.109672
charge_cat                 0.097831
length_stay                0.081030
days_b_screening_arrest    0.064763
r_days_from_arrest         0.025697
id                         0.009568
c_days_from_compas        -0.052252
gender_cat                -0.099973
race_cat                  -0.118481
age_cat                   -0.153750
age                       -0.155349
end                       -0.600067
violent_recid                   NaN
Name: two_year_recid, dtype: float64
```

In [30]:
```
data = compas[['race_cat', 'is_recid', 'is_violent_recid', 'decile_
score', 'age_cat', 'priors_count', 'two_year_recid']]
data = data.dropna()
```

In [31]:
```
data
```

Out[31]:

	race_cat	is_recid	is_violent_recid	decile_score	age_cat	priors_count	two_year_recid
1	0	1	1	3	0.5	0.0	1
2	0	1	0	4	0.0	1.0	1
3	0	0	0	8	0.0	0.5	0
6	1	1	0	6	0.5	1.0	1
8	1	0	0	1	0.5	0.0	0
...	...	...	...	...	...	...	...
7207	0	1	0	2	0.5	0.0	1
7208	0	0	0	9	0.0	0.0	0
7209	0	0	0	7	0.0	0.0	0
7210	0	0	0	3	0.0	0.0	0
7212	0	0	0	2	0.5	0.5	0

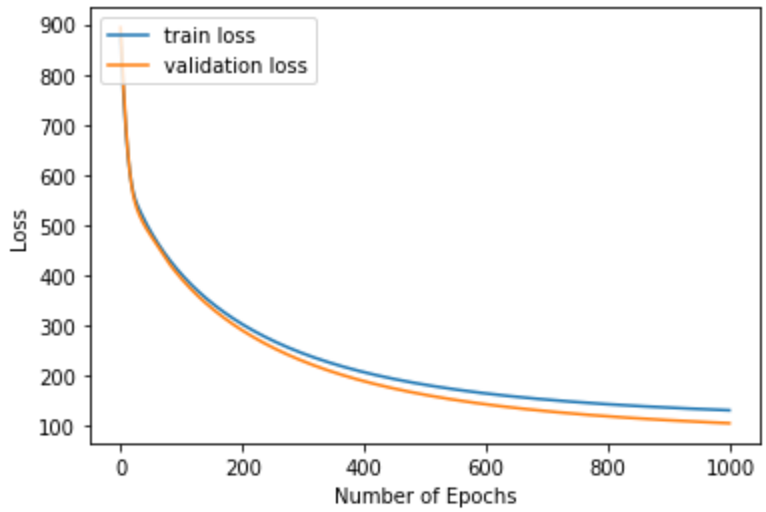
6150 rows × 7 columns

In [32]:
```
data_a = data[(data['race_cat'] == 0)]
data_c = data[(data['race_cat'] == 1)]
data_a = data_a.drop(columns=["race_cat"])
data_c = data_c.drop(columns=["race_cat"])

Xa = data_a.drop(columns = ['two_year_recid']).copy()
Xc = data_c.drop(columns = ['two_year_recid']).copy()
ya = data_a['two_year_recid']
yc = data_c['two_year_recid']

Xa_train, Xa_tv, ya_train, ya_tv = train_test_split(Xa, ya, train_s
ize=5/7)
Xa_test, Xa_valid, ya_test, ya_valid = train_test_split(Xa_tv, ya_t
v, test_size=1/2)
Xc_train, Xc_tv, yc_train, yc_tv = train_test_split(Xc, yc, train_s
ize=5/7)
Xc_test, Xc_valid, yc_test, yc_valid = train_test_split(Xc_tv, yc_t
v, test_size=1/2)

Xtrain = pd.concat([Xa_train, Xc_train])
ytrain = pd.concat([ya_train, yc_train])
Xtest = pd.concat([Xa_test, Xc_test])
ytest = pd.concat([ya_test, yc_test])
Xvalid = pd.concat([Xa_valid, Xc_valid])
yvalid = pd.concat([ya_valid, yc_valid])
```

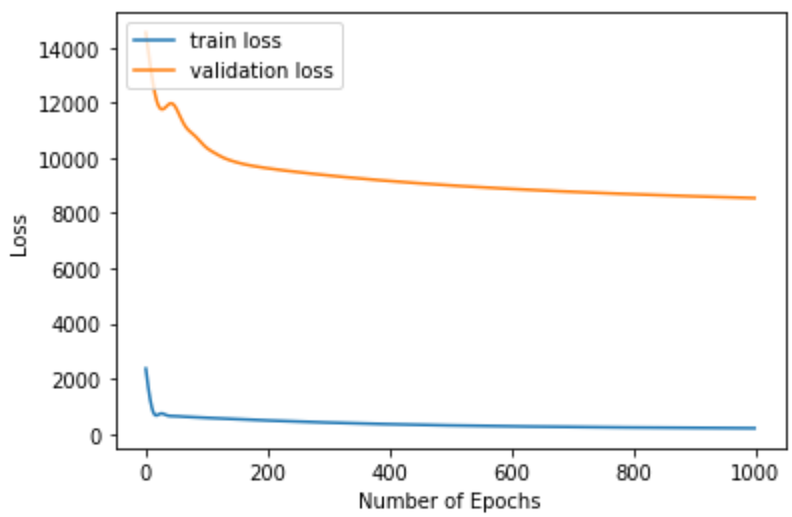
In [33]:
```
Xa_train = t.tensor(np.array(Xa_train)).to(t.float32)
ya_train = t.from_numpy(np.array(ya_train).astype('float32')).resha
pe(Xa_train.shape[0], 1)
Xc_train = t.tensor(np.array(Xc_train)).to(t.float32)
yc_train = t.from_numpy(np.array(yc_train).astype('float32')).resha
pe(Xc_train.shape[0], 1)

Xa_test = t.tensor(np.array(Xa_test)).to(t.float32)
ya_test = t.from_numpy(np.array(ya_test).astype('float32')).reshape
(Xa_test.shape[0], 1)
Xc_test = t.tensor(np.array(Xc_test)).to(t.float32)
yc_test = t.from_numpy(np.array(yc_test).astype('float32')).reshape
(Xc_test.shape[0], 1)

Xa_valid = t.tensor(np.array(Xa_valid)).to(t.float32)
ya_valid = t.from_numpy(np.array(ya_valid).astype('float32')).resha
pe(Xa_valid.shape[0], 1)
Xc_valid = t.tensor(np.array(Xc_valid)).to(t.float32)
yc_valid = t.from_numpy(np.array(yc_valid).astype('float32')).resha
pe(Xc_valid.shape[0], 1)
```

In [34]:
```
eta_value_III = [0, 20, 40, 60, 80, 90, 95, 100, 105, 110, 115, 12
0]
evalu_III = list()
evalu_valid_III = list()
evalu_test_III = list()
for i in range(0, len(eta_value_III)):
    print("eta Value: %d" % eta_value_III[i])
    PR_III = PRLR(eta = eta_value_III[i], epochs = 1000, lr = 0.01)
    eva_III, eva_valid_III, eva_test_III = PR_III.fit(Xa_train, ya_
train, Xc_train, yc_train, Xa_valid, ya_valid,
                                                      Xc_valid, yc_
valid, Xa_test, ya_test, Xc_test, yc_test)
    evalu_III.append(eva_III)
    evalu_valid_III.append(eva_valid_III)
    evalu_test_III.append(eva_test_III)
```

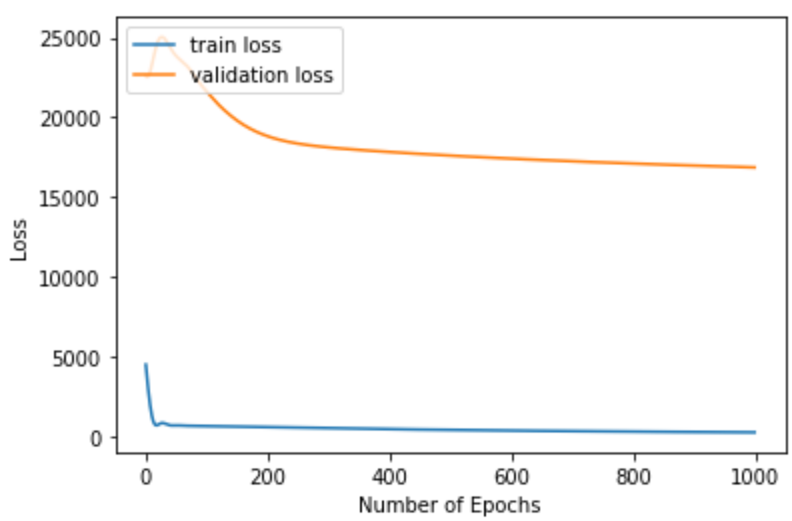
```
eta Value: 0
```



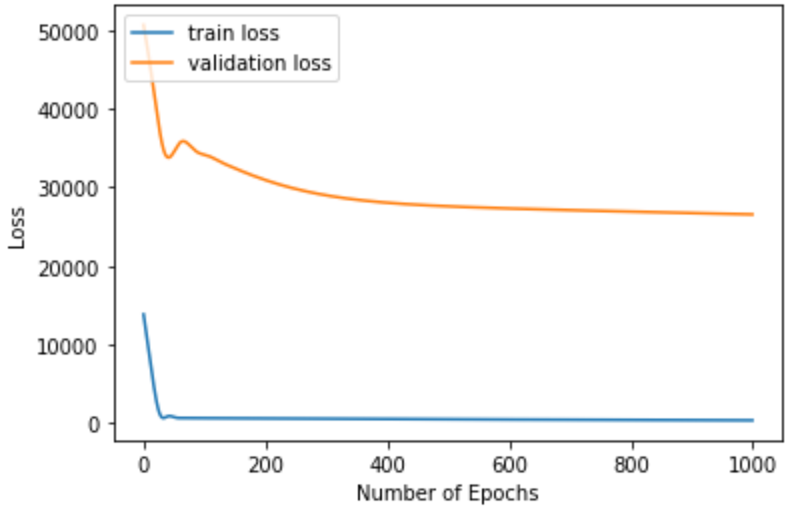

```
eta Value: 20
```



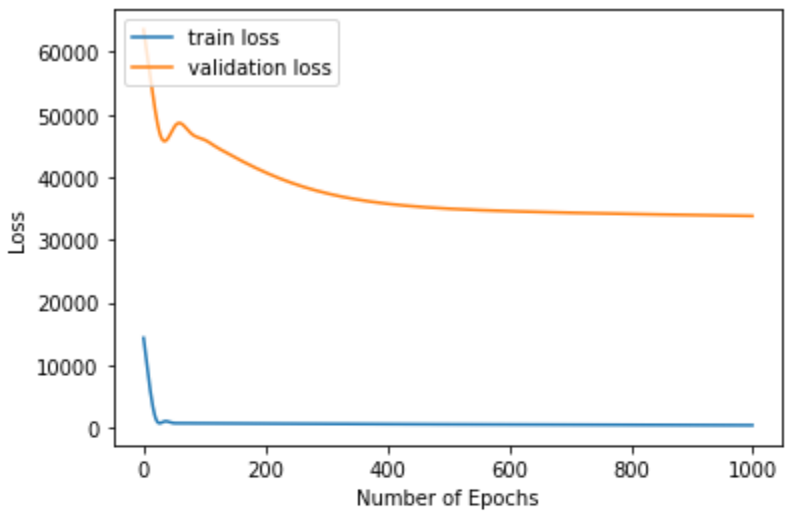

```
eta Value: 40
```



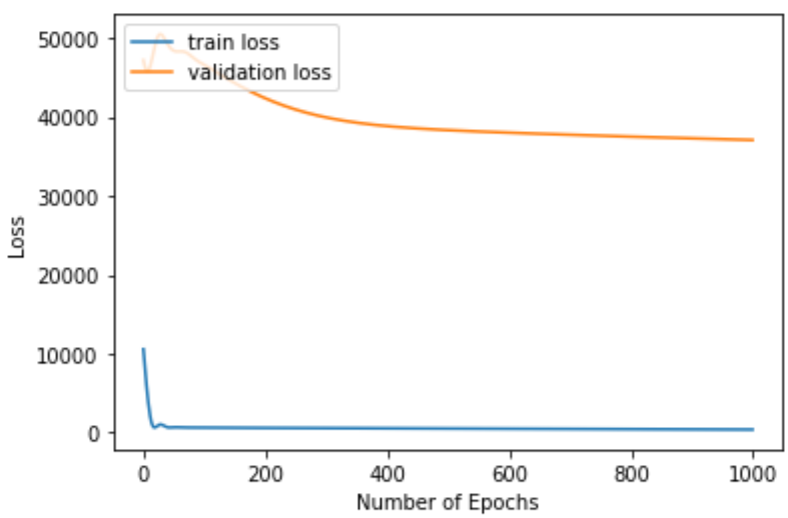

```
eta Value: 60
```



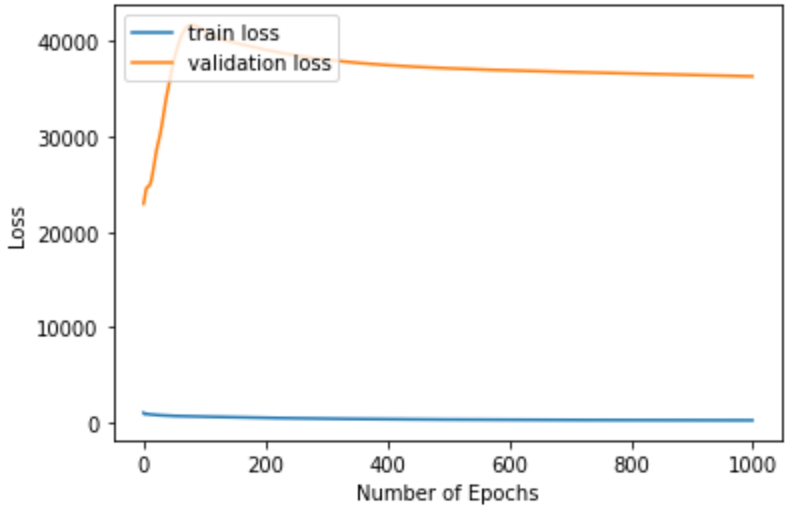
eta Value: 80



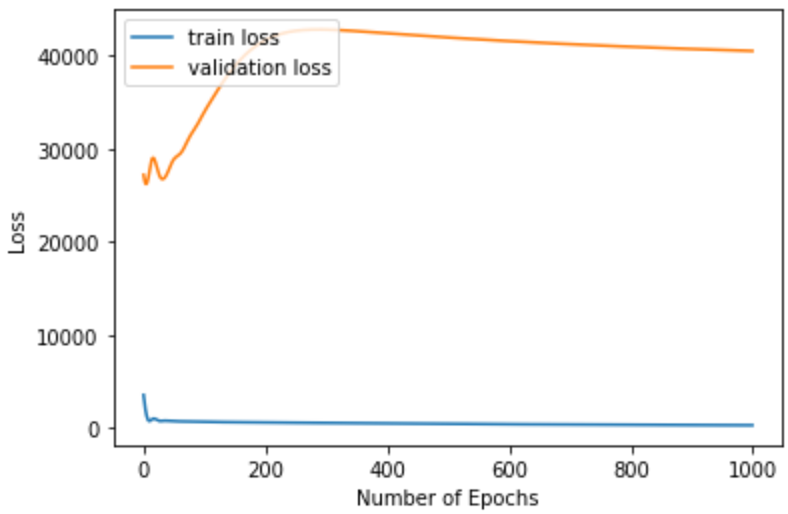
eta Value: 90



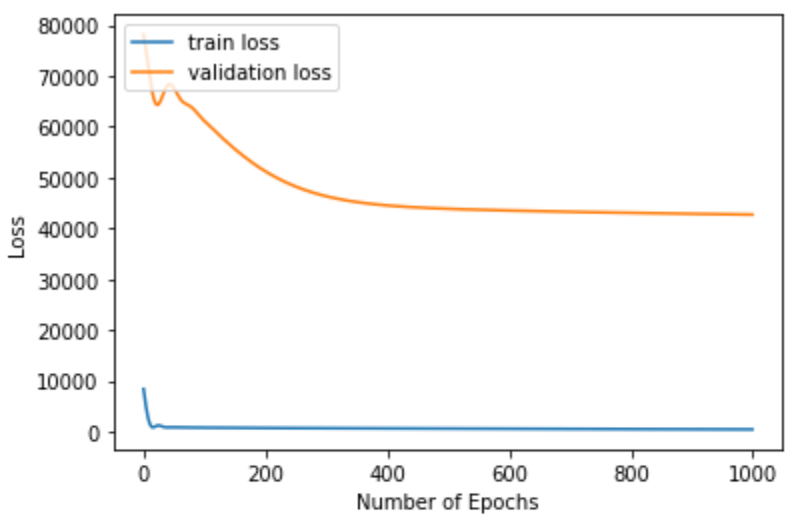
eta Value: 95



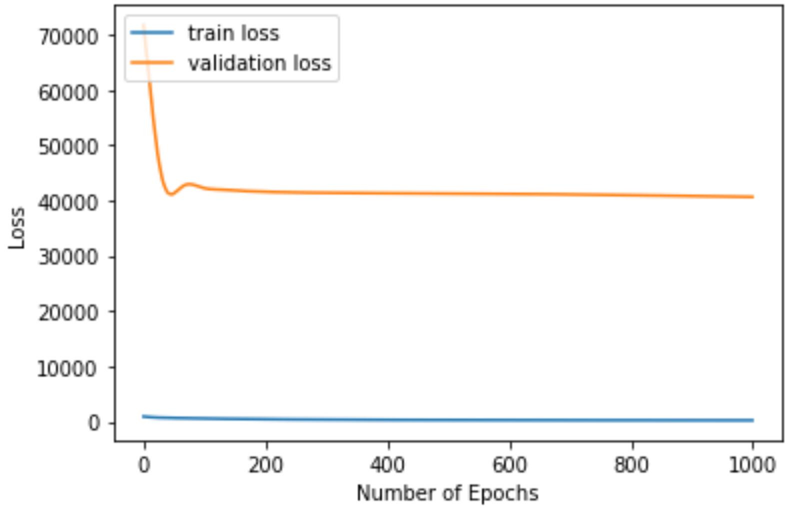

```
eta Value: 100
```



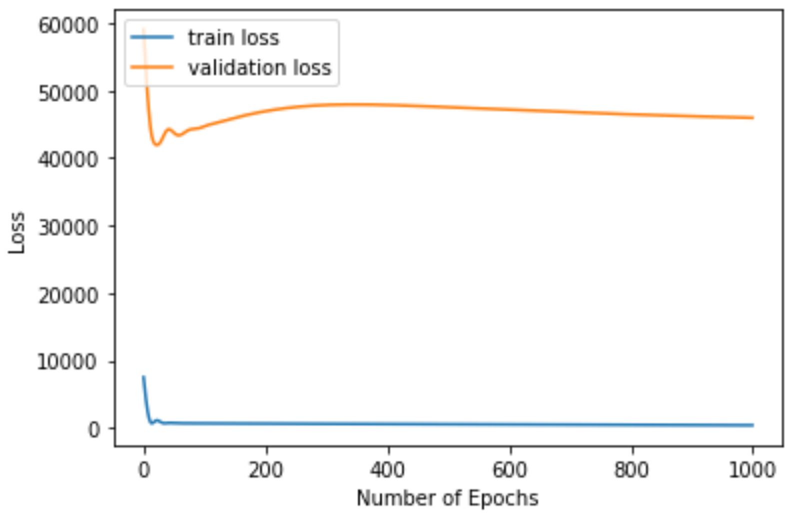

```
eta Value: 105
```



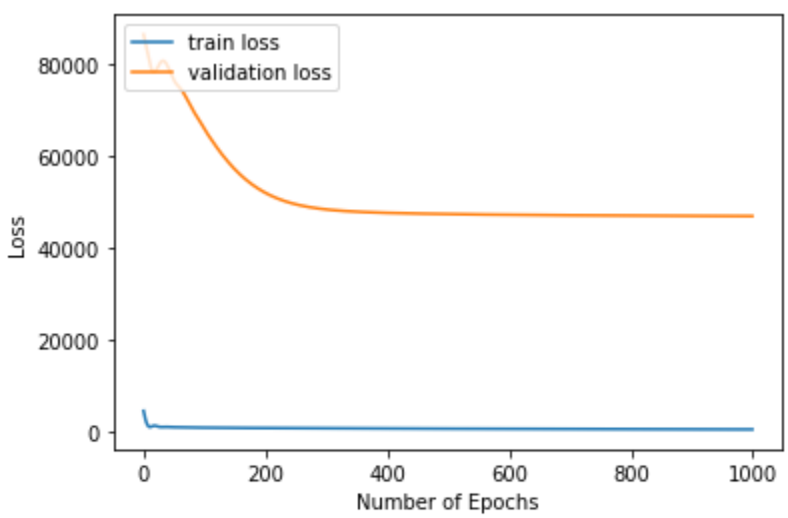

```
eta Value: 110
```



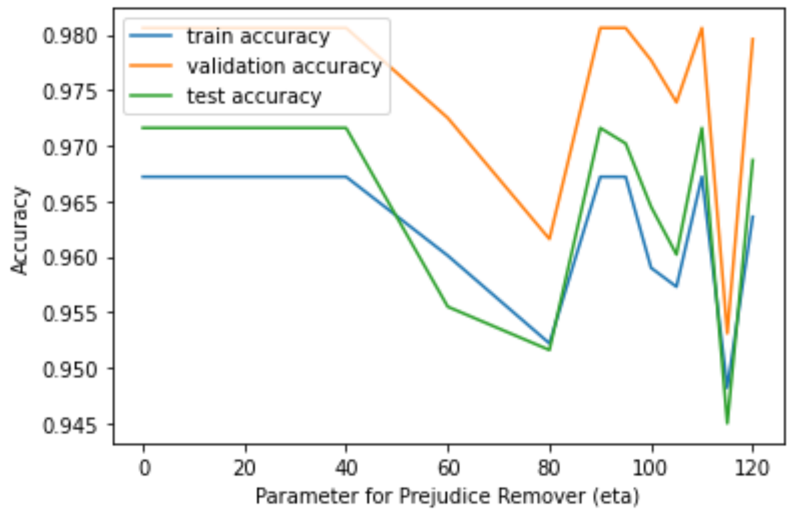
eta Value: 115



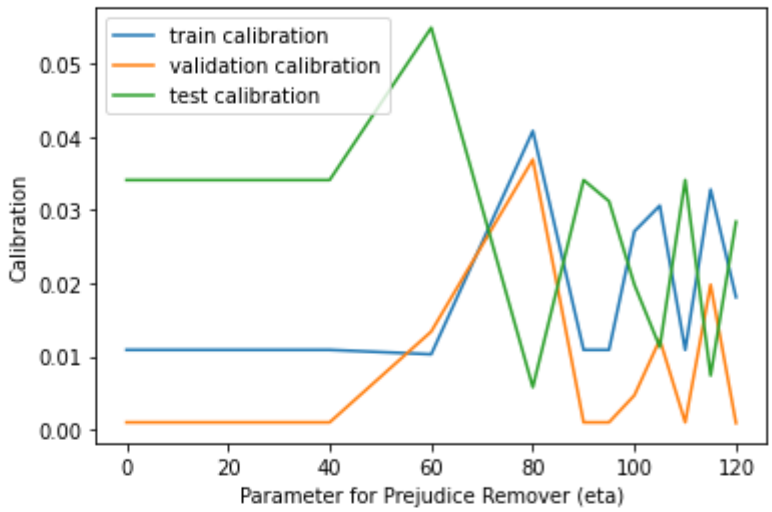
eta Value: 120



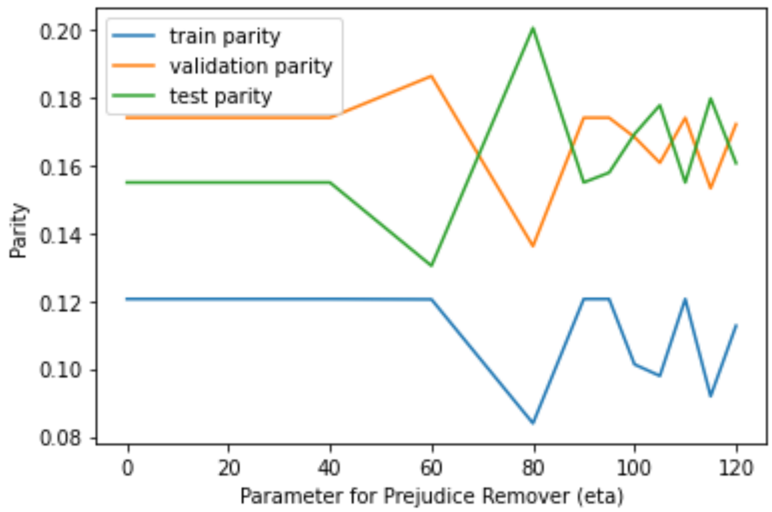
In [35]:
```
eta_acc_train_III = [x[0] for x in evalu_III]
eta_acc_valid_III = [x[0] for x in evalu_valid_III]
eta_acc_test_III = [x[0] for x in evalu_test_III]
plt.plot(eta_value_III, eta_acc_train_III, label="train accuracy")
plt.plot(eta_value_III, eta_acc_valid_III, label="validation accura
cy")
plt.plot(eta_value_III, eta_acc_test_III, label="test accuracy")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Accuracy')
plt.legend(loc="upper left")
plt.show()
```



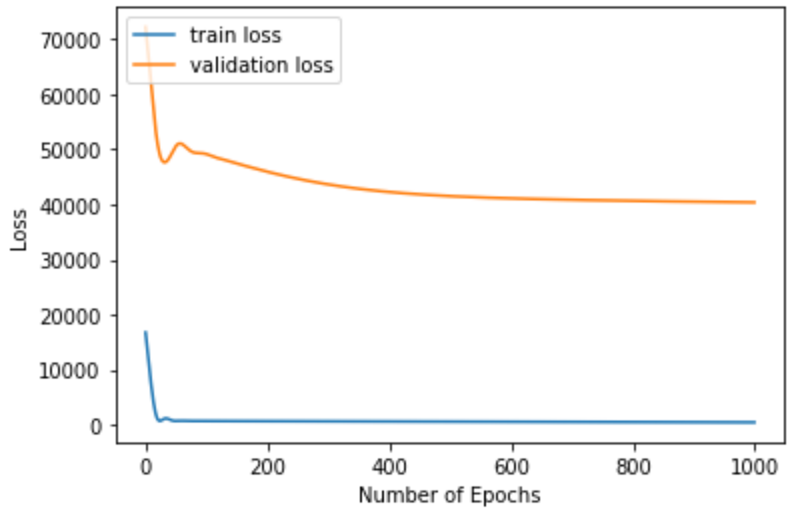
In [36]:
```
eta_cal_train_III = [x[1] for x in evalu_III]
eta_cal_valid_III = [x[1] for x in evalu_valid_III]
eta_cal_test_III = [x[1] for x in evalu_test_III]
plt.plot(eta_value_III, eta_cal_train_III, label="train calibratio
n")
plt.plot(eta_value_III, eta_cal_valid_III, label="validation calibr
ation")
plt.plot(eta_value_III, eta_cal_test_III, label="test calibration")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Calibration')
plt.legend(loc="upper left")
plt.show()
```



In [37]:
```
eta_par_train_III = [x[2] for x in evalu_III]
eta_par_valid_III = [x[2] for x in evalu_valid_III]
eta_par_test_III = [x[2] for x in evalu_test_III]
plt.plot(eta_value_III, eta_par_train_III, label="train parity")
plt.plot(eta_value_III, eta_par_valid_III, label="validation parit
y")
plt.plot(eta_value_III, eta_par_test_III, label="test parity")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Parity')
plt.legend(loc="upper left")
plt.show()
```

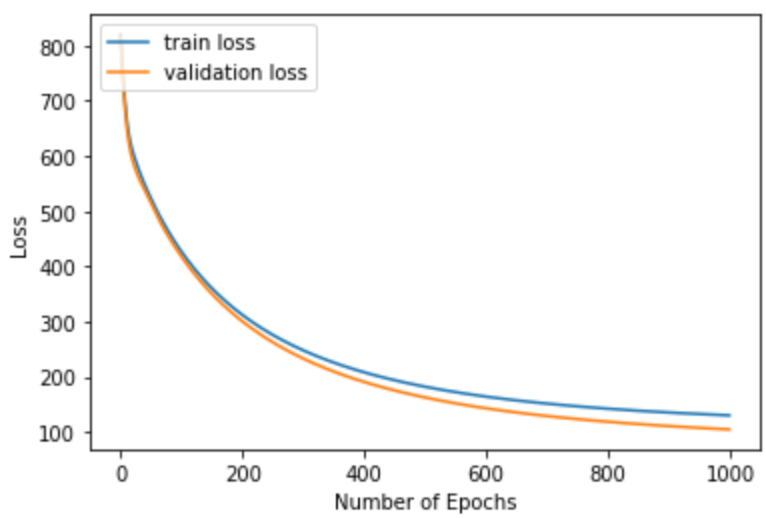


In [38]:
```
# Final model
# To achieve high accuracy, low calibration and low parity, we deci
ded to choose eta = 100.
# The outputs are (accuracy, calibration, parity) of training , val
idation, and testing sets.
PRfinal = PRLR(eta = 100, epochs = 1000, lr = 0.01)
PRfinal.fit(Xa_train, ya_train, Xc_train, yc_train, Xa_valid, ya_va
lid,
            Xc_valid, yc_valid, Xa_test, ya_test, Xc_test, yc_test)
```



Out[38]:
```
((0.966, 0.0086, 0.1229), (0.9763, 0.0096, 0.1826), (0.9725, 0.0322,
0.1531))
```

In [39]:
```
# Compared to logitstic regression without prejudice remover regula
rizer
PR_0 = PRLR(eta = 0, epochs = 1000, lr = 0.01)
PR_0.fit(Xa_train, ya_train, Xc_train, yc_train, Xa_valid, ya_vali
d,
         Xc_valid, yc_valid, Xa_test, ya_test, Xc_test, yc_test)
```



Out[39]:
```
((0.9672, 0.0109, 0.1206), (0.9806, 0.001, 0.1741), (0.9716, 0.0341,
0.155))
```

In [40]:
```
# model IV
# If we choose all the features:
```

In [41]:
```
raw_data=pd.read_csv(data_dir + 'compas-scores-two-years.csv')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent cal
l last)
Input In [41], in <cell line: 1>()
----> 1 raw_data=pd.read_csv(data_dir + 'compas-scores-two-years.csv
')

NameError: name 'data_dir' is not defined
```

In [ ]:
```
df = raw_data[['age', 'c_charge_degree', 'race', 'age_cat',
                    'score_text', 'sex', 'priors_count', 'days_b_sc
reening_arrest',
                    'decile_score', 'is_recid', 'c_jail_in',
                    'c_jail_out', 'two_year_recid']]\
                    .query('days_b_screening_arrest <= 30')\
                    .query('days_b_screening_arrest >= -30')\
                    .query('is_recid != -1')\
                    .query('c_charge_degree != "O"')\
                    .query('score_text != "N/A"')
```

In [ ]:
```
df['length_of_stay']=df['c_jail_out'].apply(pd.to_datetime) - df['c
_jail_in'].apply(pd.to_datetime)
df['length_of_stay']=df['length_of_stay'].dt.days
races = ['African-American', 'Caucasian']
df = df[df.race.isin(races)]
df.loc[df.race=='Caucasian','race']=1
df.loc[df.race=='African-American','race']=0

cat_var = ['c_charge_degree','race','sex','age_cat','score_text','i
s_recid','two_year_recid','length_of_stay']

for var in cat_var:
    df[var] = df[var].astype('category').cat.codes

df=df[['sex','age_cat','decile_score','priors_count','days_b_screen
ing_arrest','c_charge_degree','is_recid','score_text','length_of_st
ay',"race", 'two_year_recid']]
```

In [ ]:
```
df_aa=df[(df['race'] == 0)]
del df_aa['race']
df_c=df[(df['race'] == 1)]
del df_c['race']
X_aa = df_aa.drop(columns = ['two_year_recid']).copy()
y_aa = df_aa['two_year_recid']

X_c = df_c.drop(columns = ['two_year_recid']).copy()
y_c = df_c['two_year_recid']
```

In [ ]:

```
df_aa_X_train, df_aa_X_rest, df_aa_y_train, df_aa_y_rest = train_te
st_split(X_aa,y_aa, train_size=5/7.0)
df_aa_X_valid, df_aa_X_test, df_aa_y_valid, df_aa_y_test = train_te
st_split(df_aa_X_rest,df_aa_y_rest, test_size=0.5)

df_c_X_train, df_c_X_rest, df_c_y_train, df_c_y_rest = train_test_s
plit(X_c,y_c, train_size=5/7.0)
df_c_X_valid, df_c_X_test, df_c_y_valid, df_c_y_test = train_test_s
plit(df_c_X_rest,df_c_y_rest, test_size=0.5)

X_train=pd.concat([df_aa_X_train,df_c_X_train])
y_train=pd.concat([df_aa_y_train,df_c_y_train])
X_valid=pd.concat([df_aa_X_valid,df_c_X_valid])
y_valid=pd.concat([df_aa_y_valid,df_c_y_valid])
X_test=pd.concat([df_aa_X_test,df_c_X_test])
y_test=pd.concat([df_aa_y_test,df_c_y_test])

df_c_X_train=t.tensor(np.array(df_c_X_train)).to(t.float32)
df_c_y_train=t.from_numpy(np.array(df_c_y_train).astype('float32
')).reshape(df_c_X_train.shape[0],1)
df_aa_X_train=t.tensor(np.array(df_aa_X_train)).to(t.float32)
df_aa_y_train=t.from_numpy(np.array(df_aa_y_train).astype('float32
')).reshape(df_aa_X_train.shape[0],1)

df_c_X_valid=t.tensor(np.array(df_c_X_valid)).to(t.float32)
df_c_y_valid=t.from_numpy(np.array(df_c_y_valid).astype('float32
')).reshape(df_c_X_valid.shape[0],1)
df_aa_X_valid=t.tensor(np.array(df_aa_X_valid)).to(t.float32)
df_aa_y_valid=t.from_numpy(np.array(df_aa_y_valid).astype('float32
')).reshape(df_aa_X_valid.shape[0],1)

df_c_X_test=t.tensor(np.array(df_c_X_test)).to(t.float32)
df_c_y_test=t.from_numpy(np.array(df_c_y_test).astype('float32')).r
eshape(df_c_X_test.shape[0],1)
df_aa_X_test=t.tensor(np.array(df_aa_X_test)).to(t.float32)
df_aa_y_test=t.from_numpy(np.array(df_aa_y_test).astype('float32
')).reshape(df_aa_X_test.shape[0],1)
```

In [ ]:

```
eta_value_IV = [0, 1, 2, 3, 4, 5, 6, 8, 10, 15, 20, 30, 50, 100]
evalu_IV = list()
evalu_valid_IV = list()
evalu_test_IV = list()
for i in range(0, len(eta_value_IV)):
    print("eta Value: %d" % eta_value_IV[i])
    PR_IV = PRLR(eta = eta_value_IV[i], epochs = 1000, lr = 0.01)
    eva_IV, eva_valid_IV, eva_test_IV = PR_IV.fit(X_a_train, y_a_tr
ain, X_c_train, y_c_train, X_a_valid, y_a_valid,
                                              X_c_valid, y_c_valid,
X_a_test, y_a_test, X_c_test, y_c_test)
    evalu_IV.append(eva_I)
    evalu_valid_IV.append(eva_valid_IV)
    evalu_test_IV.append(eva_test_IV)
```

In [ ]:
```
eta_acc_train_IV = [x[0] for x in evalu_IV]
eta_acc_valid_IV = [x[0] for x in evalu_valid_IV]
eta_acc_test_IV = [x[0] for x in evalu_test_IV]
plt.plot(eta_value_IV, eta_acc_train_IV, label="train accuracy")
plt.plot(eta_value_IV, eta_acc_valid_IV, label="validation accurac
y")
plt.plot(eta_value_IV, eta_acc_test_IV, label="test accuracy")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Accuracy')
plt.legend(loc="upper right")
plt.show()
```

In [ ]:
```
eta_cal_train_IV = [x[1] for x in evalu_IV]
eta_cal_valid_IV = [x[1] for x in evalu_valid_IV]
eta_cal_test_IV = [x[1] for x in evalu_test_IV]
plt.plot(eta_value_IV, eta_cal_train_IV, label="train calibration")
plt.plot(eta_value_IV, eta_cal_valid_IV, label="validation calibrat
ion")
plt.plot(eta_value_IV, eta_cal_test_IV, label="test calibration")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Calibration')
plt.legend(loc="upper right")
plt.show()
```

In [ ]:
```
eta_cal_train_IV = [x[2] for x in evalu_IV]
eta_cal_valid_IV = [x[2] for x in evalu_valid_IV]
eta_cal_test_IV = [x[2] for x in evalu_test_IV]
plt.plot(eta_value_IV, eta_cal_train_IV, label="train parity")
plt.plot(eta_value_IV, eta_cal_valid_IV, label="validation parity")
plt.plot(eta_value_IV, eta_cal_test_IV, label="test parity")
plt.xlabel('Parameter for Prejudice Remover (eta)')
plt.ylabel('Parity')
plt.legend(loc="upper right")
plt.show()
```

In [ ]:
```
# Final model IV
# To achieve high accuracy, low calibration and low parity, we deci
ded to choose eta = 5.
# The outputs are (accuracy, calibration, parity) of training , val
idation, and testing sets.
PR_final_IV = PRLR(eta = 5, epochs = 1000, lr = 0.01)
PR_final_IV.fit(X_a_train, y_a_train, X_c_train, y_c_train, X_a_val
id, y_a_valid,
               X_c_valid, y_c_valid, X_a_test, y_a_test, X_c_test,
y_c_test)
```

In [ ]:
```
# Compared to logitstic regression without prejudice remover regula
rizer
PR_0 = PRLR(eta = 0, epochs = 1000, lr = 0.01)
PR_0.fit(X_a_train, y_a_train, X_c_train, y_c_train, X_a_valid, y_a
_valid,
               X_c_valid, y_c_valid, X_a_test, y_a_test, X_c_test,
y_c_test)
```

In [ ]:
```
# Summary
# as eta increases, accuracy will decrease since it is sacrisfied f
or fairness, calibration will also decrease, but parity will increa
se.
# From the figures, we cannot achieve low calibration and low parit
y at the same time for this problem.
# For this problem, the fairness looks good (calibration below 5% f
or all models), so the prejudice remover regularizer does not work
well.
```

A7: Fairness-aware Feature Selection

In [1]:
```
import pandas as pd
import numpy as np
import warnings
import itertools
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import copy
import math
warnings.filterwarnings('ignore')
```

In [2]:
```
compas_scores = pd.read_csv('../data/compas-scores-two-years.csv')
protected_attributes = ['sex','race']
```

Data Cleaning

Columns removed

- columns with more than 10% of data missing

Rows removed

- recidivist flag -- is_recid -- to be -1 (when no compas case would be found)
- charge date of a defendants Compas scored crime was not within 30 days from when the person was arrested
- ordinary traffic offenses -- those with a c_charge_degree of 'O'

In [3]:
```
def categorize_numerical_col(num, lim1, lim2):
    if num <= lim1:
        return 0
    elif lim1 < num <= lim2:
        return 1
    elif num > lim2:
        return 2
    else:
        raise('Invalid row')
def categorize_age(age_cat):
    if age_cat=='Less than 25':
        return 0
    elif age_cat=='25 - 45':
        return 1
    elif age_cat=='Greater than 45':
        return 2
    else:
        raise('Invalid row')
```

In [4]:
```
# Data Cleaning

# Remove NaNs
percent_missing = compas_scores.isnull().sum() * 100 / len(compas_s
cores)
missing_value_df = pd.DataFrame({'column_name': compas_scores.colum
ns,
                                 'percent_missing': percent_missin
g})
missing_value_df.sort_values('percent_missing', inplace=True, ascen
ding=False)
cols2keep_df = missing_value_df[~(missing_value_df.percent_missing>
10)]
cols2keep_df_list = cols2keep_df.column_name.tolist()
compas_scores_cols_trim = compas_scores[cols2keep_df_list]
compas_scores_cols_trim_dropna = compas_scores_cols_trim.dropna()

# Apply cleaning descibed in publication of data HERE: https://gith
ub.com/propublica/compas-analysis/blob/master/Compas%20Analysis.ipy
nb
compas_df = compas_scores_cols_trim_dropna[(compas_scores_cols_trim
_dropna['days_b_screening_arrest']<= 30) &
                               (compas_scores_cols_trim_dropna['day
s_b_screening_arrest']>= -30) &
                               (compas_scores_cols_trim_dropna['is_
recid']!= -1) &
                               (compas_scores_cols_trim_dropna['c_c
harge_degree']!= "O") &
                               (compas_scores_cols_trim_dropna['sco
re_text']!= 'N/A')
                              ]

# Select columns described in https://arxiv.org/abs/2106.00772 only
which are (Age, Charge Degree, Gender, Prior Counts, Length Of Sta
y, race)

compas_subset_df = compas_df[["sex","age","age_cat","race","priors_
count.1","c_charge_degree","c_jail_in", "c_jail_out","two_year_reci
d"]]


# Select only African American and Caucasian
compas_subset_df = compas_subset_df[(compas_subset_df["race"]=='Cau
casian') |(compas_subset_df["race"]=='African-American') ]

# Add length of stay and drop "c_jail_in", "c_jail_out"

compas_subset_df["length_stay"] = pd.to_datetime(compas_subset_df["
c_jail_out"]) - pd.to_datetime(compas_subset_df['c_jail_in'])
compas_subset_df["length_stay"] = compas_subset_df["length_stay"].a
pply(lambda x: x.days)
compas_subset_df = compas_subset_df.drop(columns = ["c_jail_in","c_
jail_out"])
compas_subset_df['length_stay'] = compas_subset_df["length_stay"].a
pply(categorize_numerical_col, lim1=7, lim2=90)
```

```
# Categorize prior counts according to https://arxiv.org/abs/2106.0
0772
compas_subset_df['priors_count.1'] = compas_subset_df["priors_coun
t.1"].apply(categorize_numerical_col, lim1=0, lim2=3)

# Categorize age according to https://arxiv.org/abs/2106.00772
compas_subset_df['age_cat'] = compas_subset_df["age_cat"].apply(cat
egorize_age)

# Encode categories
race_encoder = OrdinalEncoder(dtype='int')
compas_subset_df['race']  = race_encoder.fit_transform(compas_subse
t_df[['race']])

sex_encoder = OrdinalEncoder(dtype='int')
compas_subset_df['sex']  = sex_encoder.fit_transform(compas_subset_
df[['sex']])

charge_encoder = OrdinalEncoder(dtype='int')
compas_subset_df['c_charge_degree']  = charge_encoder.fit_transform
(compas_subset_df[['c_charge_degree']])

# Create protected attribute
protected_attribute = compas_subset_df["race"]

# Target Variable
target_variable = compas_subset_df['two_year_recid']

# Feature set
feature_df = compas_subset_df.drop(['two_year_recid','race','age'],
axis=1)

X_train, X_test, y_train, y_test, protected_train, protected_test =
train_test_split(
    feature_df.to_numpy(), target_variable.to_numpy(), protected_at
tribute.to_numpy(), test_size=0.2, random_state=42)
```

Implementation of Fairness Feature Selection Algorithm

```
In [5]: def unique_info_array(data):
    """Get unique information from input arrays"""
    unique_list = []

    for idx in range(data.shape[1]):
        unique_list.append(np.unique(data[:, idx]).tolist())
    return unique_list

def unique_information_conditional(y, x_s, x_s_c, protected_attr, f
orm=1):
    """Get unique information from input arrays with conditional pr
obabilities taken into account"""

    # Using
    # IQ(T; R1|R2) = ∑t,r1,r2 QT ,R1,R2 (t, r1, r2) log((QT |R1,R2
(t|r1,r2))/ (QT |R2 (t|r2)))

    if form == 1:

        row_count     = len(y)
        col_count_y   = y.shape[1]
        col_count_xs = x_s.shape[1]
        y_xs_protected_attr_xsc = np.concatenate((y, x_s, x_s_c, pr
otected_attr), axis=1)
        ui_array = unique_info_array(y_xs_protected_attr_xsc)
        ui_array_cat_product = list(itertools.product(*ui_array)) #
compute the cartesian product of all arrays
    else:
        row_count     = len(y)
        col_count_y   = x_s.shape[1]
        col_count_xs = protected_attr.shape[1]
        y_xs_protected_attr_xsc = np.concatenate((x_s, protected_at
tr, y), axis=1)
        ui_array = unique_info_array(y_xs_protected_attr_xsc)
        ui_array_cat_product = list(itertools.product(*ui_array)) #
compute the cartesian product of all arrays

    IQ = 0
    for array in ui_array_cat_product:
        r1_r2 = len(np.where((y_xs_protected_attr_xsc == array).all
(axis=1))[0]) / row_count
        r1 = len(np.where((y == array[:col_count_y]).all(axis=
1))[0]) / row_count
        r2 = len(np.where((y_xs_protected_attr_xsc[:, col_count_y:
-col_count_xs] == array[
            col_count_y: -col_count_xs]).all(axis=1))[0]) / row_cou
nt

        try:
            r1_given_r2 = len(np.where((y_xs_protected_attr_xsc[:,
:col_count_y] == array[:col_count_y]).all(axis=1)
                                       & (y_xs_protected_attr_xsc
[:, -col_count_xs:]  == array[
                                           -col_count_xs:]).all(axi
s=1))[0]) / len(np.where( \
```

```
                (y_xs_protected_attr_xsc[:, -col_count_xs:] == arra
y[-col_count_xs:]).all(axis=1))[0])
        except ZeroDivisionError:
            r1_given_r2 = 0

        if r1_r2 == 0 or r1 == 0 or r2 == 0 or r1_given_r2 == 0:
            temp = 0
        else:
            temp = r1_r2 * np.log(r1_r2 / r2) / r1_given_r2
        IQ += np.abs(temp)

    return IQ
```

In [6]:

```
def unique_information(array_1, array_2):
    """Get unique information from input arrays"""

    row_count          = len(array_1)
    col_count_array_1  = array_1.shape[1]

    features_combined = np.concatenate((array_1, array_2), axis=1)
    ui_array = unique_info_array(features_combined)
    ui_array_cat_product = list(itertools.product(*ui_array))

    # Using
    # IQ(T; R1|R2) = ∑t,r1,r2 QT ,R1,R2 (t, r1, r2) log((QT |R1,R2
(t|r1,r2))/ (QT |R2 (t|r2)))

    row_count          = len(array_1)
    col_count_array_1  = array_1.shape[1]

    IQ = 0
    for array in ui_array_cat_product:
        r1_r2 = len(np.where((features_combined == array).all(axis=
1))[0]) / row_count
        r1 = len(np.where((array_1 == array[:col_count_array_1]).al
l(axis=1))[0]) / row_count
        r2 = len(np.where((array_2 == array[col_count_array_1:]).al
l(axis=1))[0]) / row_count

        if r1_r2 == 0 or r1 == 0 or r2 == 0:
            temp = 0
        else:
            temp = r1_r2 * np.log(r1_r2 / r1) / r1
        IQ += np.abs(temp)
    return IQ
```

In [7]:

```
def get_feature_subsets(sc):
    """
    Generate all subsets of feature set
    """
    if len(sc) <= 1:
        yield sc
        yield []
    else:
        for item in get_feature_subsets(sc[1:]):
            yield [sc[0]]+item
            yield item

def acc_coef(y, x_s, x_s_c, protected_attr):
    return unique_information_conditional(y, x_s, x_s_c, protected_
attr)

def disc_coef(y, x_s, x_s_c, protected_attr):
    return unique_information(y, np.concatenate((x_s, protected_att
r), axis=1)) * unique_information(x_s, protected_attr) * unique_inf
ormation_conditional(y, x_s, x_s_c, protected_attr,form=2)

def marginal_acc_coef(y_train, X_train, protected_attr, set_tracke
r):
    """compute  marginal accuracy coefficient"""
    num_features = X_train.shape[1]
    feat_idx = list(range(num_features))
    feat_idx.pop(set_tracker)
    feature_subsets = [x for x in get_feature_subsets(feat_idx) if
len(x) > 0]
    shapley_value =0

    for sc_idx in feature_subsets:
            coef = math.factorial(len(sc_idx)) * math.factorial(num
_features - len(sc_idx) - 1) / math.factorial(num_features)

            # Compute v(T ∪ {i})
            idx_xs_ui = copy.deepcopy(sc_idx) # create copy of subs
et list
            idx_xs_ui.append(set_tracker) # append feature index
            idx_xsc_ui = list(set(list(range(num_features))).differ
ence(set(idx_xs_ui))) # compliment of x_s
            vTU = acc_coef(y_train.reshape(-1, 1), X_train[:, idx_x
s_ui], X_train[:, idx_xsc_ui], protected_attr.reshape(-1, 1))

             # Compute v(T)
            idx_xsc = list(range(num_features))
            idx_xsc.pop(set_tracker)
            idx_xsc = list(set(idx_xsc).difference(set(sc_idx)))
            vT = acc_coef(y_train.reshape(-1, 1), X_train[:, sc_id
x], X_train[:, idx_xsc], protected_attr.reshape(-1, 1))

            marginal = vTU - vT
            shapley_value = shapley_value + coef * marginal
    return shapley_value
```

```
def marginal_disc_coef(y_train, X_train, protected_attr, set_tracke
r):
    """compute marginal discrimination coefficient"""
    num_features = X_train.shape[1]
    feat_idx = list(range(num_features))
    feat_idx.pop(set_tracker)
    feature_subsets = [x for x in get_feature_subsets(feat_idx) if
len(x) > 0]
    shapley_value =0

    for sc_idx in feature_subsets:
            coef = math.factorial(len(sc_idx)) * math.factorial(num
_features - len(sc_idx) - 1) / math.factorial(num_features)

            # Compute v(T ∪ {i})
            idx_xs_ui = copy.deepcopy(sc_idx) # create copy of subs
et list
            idx_xs_ui.append(set_tracker) # append feature index
            vTU = disc_coef(y_train.reshape(-1, 1), X_train[:, idx_
xs_ui],X_train[:, idx_xs_ui], protected_attr.reshape(-1, 1))

             # Compute v(T)
            idx_xsc = list(range(num_features))
            idx_xsc.pop(set_tracker)
            idx_xsc = list(set(idx_xsc).difference(set(sc_idx)))
            vT = disc_coef(y_train.reshape(-1, 1), X_train[:, sc_id
x], X_train[:, sc_idx], protected_attr.reshape(-1, 1))

            marginal = vTU - vT
            shapley_value = shapley_value + coef * marginal
    return shapley_value
```

In [28]:
```
%%time
# shapley values for accuracy and discrimination
shapley_acc = []
shapley_disc = []
for i in range(5):
    acc_i = marginal_acc_coef(y_train, X_train, protected_train, i)
    disc_i = marginal_disc_coef(y_train, X_train, protected_train,
i)


    shapley_acc.append(acc_i)
    shapley_disc.append(disc_i)

# DataFrame to compare shapely values
feature_names = ["Gender", "Age", "Prior Count", "Charge Degree", "
Length of Stay"]
shapley_df = pd.DataFrame(list(zip(feature_names, shapley_acc, shap
ley_disc)),
                          columns=["Feature", "Accuracy",'Discrimin
ation'])
shapley_df
```

Wall time: 11.1 s

Out[28]:

	Feature	Accuracy	Discrimination
0	Gender	0.973917	729.645575
1	Age	1.181441	939.740547
2	Prior Count	1.229856	982.431358
3	Charge Degree	1.046473	765.737748
4	Length of Stay	1.028396	908.017124

We observe that Prior count and Age have the strongest discrimatory coefficients but also have the largest impact on the accuracy

In [58]:
```
shapley_df["F"] = shapley_df.Accuracy - 0.00125*shapley_df.Discrimi
nation
#shapley_df.Discrimination = shapley_df.Discrimination.apply(lambda
x:"%E"%x)
shapley_df = shapley_df.sort_values(by=["F"], ascending=[False]).re
set_index(0, True)
```

In [59]:
```
shapley_df.to_csv("../output/score values table new.csv")
shapley_df
```

Out[59]:

	Feature	Accuracy	Discrimination	F
0	Charge Degree	1.046473	765.737748	0.089301
1	Gender	0.973917	729.645575	0.061860
2	Age	1.181441	939.740547	0.006765
3	Prior Count	1.229856	982.431358	0.001817
4	Length of Stay	1.028396	908.017124	-0.106625

Prediction model using logistic Regression

We apply a logistic regression to the feature set and observe the impact on accuracy when we eliminate an individual feature and copare this with the discriminatory impact said feature has on the overall model

In [49]:

```
accuracy = []
calibration = []

# Build model testing impact of each feature on model accuracy

log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

black  = np.where(protected_test == 1)[0] # African American
white  = np.where(protected_test == 0)[0] # Caucasian
accuracy.append(log_reg.score(X_test, y_test))
calibration.append(log_reg.score(X_test[black], y_test[black]) - lo
g_reg.score(X_test[white], y_test[white]))

# Test impact of each feature on model
for i in range(X_train.shape[1]):
    features = list(range(X_train.shape[1]))
    features.pop(i)
    X_train_subset = X_train[:, features]
    X_test_subset = X_test[:, features]

    log_reg = LogisticRegression()
    log_reg.fit(X_train_subset, y_train)
    acc_subset = log_reg.score(X_test_subset, y_test)
    cal_subset = abs(log_reg.score(X_test_subset[black],
                               y_test[black]) - log_reg.score(X_tes
t_subset[white],
                                                              y_tes
t[white]))

    accuracy.append(acc_subset)
    calibration.append(cal_subset)


col_names = ["None", "Sex", "Age", "Prior Count", "Charge Degree",
"Length of stay"]
accuracy = [x * 100 for x in accuracy]
calibration = [x * 100 for x in calibration]
analysis = pd.DataFrame(list(zip(col_names, accuracy, calibratio
n)),
                          columns=["Eliminating Feature", "Accuracy
(%)", "Calibration (%)"])
analysis
```

Out[49]:

	Eliminating Feature	Accuracy (%)	Calibration (%)
0	None	66.919431	0.627451
1	Sex	65.876777	1.191410
2	Age	63.601896	0.515406
3	Prior Count	58.578199	6.715219
4	Charge Degree	67.014218	0.468721
5	Length of stay	66.161137	1.503268

In [50]:
```
analysis = analysis.sort_values(by=["Calibration (%)"], ascending=
[False]).reset_index(0, True)
analysis
```

Out[50]:

	Eliminating Feature	Accuracy (%)	Calibration (%)
0	Prior Count	58.578199	6.715219
1	Length of stay	66.161137	1.503268
2	Sex	65.876777	1.191410
3	None	66.919431	0.627451
4	Age	63.601896	0.515406
5	Charge Degree	67.014218	0.468721

In [51]:
```
analysis.to_csv("../output/A7 log analysis.csv")
```

It can be observed that eliminating Prior count results in the strongest drop in accuracy despite it's high discriminatory effect. We also see that age has a significant drop in accuracy despite its high discrimatory effect.

In [ ]:
```

```