In [ ]:

```
#APPLIED DATA SCIENCE
#PROJECT 4 GROUP 6
```

In [1]:

```
import numpy as np
import pandas as pd
#!pip install torch
import torch as t
import torch.nn as nn
from torch.nn import functional as F
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

In [2]:

```
df=pd.read_csv('../data/compas-scores-two-years.csv')
```

In [3]:

```
df1=df[["sex","age_cat",'decile_score','priors_count','days_b_screening_arres
df1=df1[(df1['days_b_screening_arrest'] <= 30)|(df1['days_b_screening_arrest'
df1=df1[df1['is_recid'] != -1]
df1=df1[df1['c_charge_degree'] != "O"]
df1=df1[df1['score_text'] != 'N/A']

df1['length_of_stay']=df1['c_jail_out'].apply(pd.to_datetime) - df1['c_jail_i
df1['length_of_stay']=df1['length_of_stay'].dt.days
df1['length_of_stay'] = df1.length_of_stay.apply(lambda x:'greater than 100 d

df1=df1[(df1['race'] == 'Caucasian') | (df1['race'] == 'African-American')]
df1.loc[df1.race=='Caucasian','race']=1
df1.loc[df1.race=='African-American','race']=0

categorical_variabls = ["c_charge_degree","race","sex","age_cat","score_text"
for var in categorical_variabls:
    df1[var] = df1[var].astype('category').cat.codes

df1=df1[["sex","age_cat","score_text",'c_charge_degree',"race", 'two_year_rec
```

In [4]:

```
df1.groupby(['race', 'two_year_recid'])['sex'].agg([ 'count'])
```

Out[4]:

race	two_year_recid	count
0	0	1668
	1	1869
1	0	1421
	1	957

In [5]:

```
df_a=df1[(df1['race'] == 0)]
del df_a['race']
df_c=df1[(df1['race'] == 1)]
del df_c['race']
df_a.head()
```

Out[5]:

	sex	age_cat	score_text	c_charge_degree	two_year_recid
1	1	0	1	0	1
2	1	2	1	0	1
11	1	2	2	1	1
13	1	0	1	0	0
15	1	0	1	0	1

In [6]:

```
#split dataset so that training:validation:testing=5:1:1

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

df_a_X_train, df_a_X_rem, df_a_y_train, df_a_y_rem = train_test_split(X_a,y_a
df_a_X_valid, df_a_X_test, df_a_y_valid, df_a_y_test = train_test_split(df_a_

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

df_c_X_train, df_c_X_rem, df_c_y_train, df_c_y_rem = train_test_split(X_c,y_c
df_c_X_valid, df_c_X_test, df_c_y_valid, df_c_y_test = train_test_split(df_c_

X_train=pd.concat([df_a_X_train,df_c_X_train])
y_train=pd.concat([df_a_y_train,df_c_y_train])
X_valid=pd.concat([df_a_X_valid,df_c_X_valid])
y_valid=pd.concat([df_a_y_valid,df_c_y_valid])
X_test=pd.concat([df_a_X_test,df_c_X_test])
y_test=pd.concat([df_a_y_test,df_c_y_test])
```

A4: Fairness Beyond Disparate Treatment & Disparate Impact: Learning Classification without Disparate Mistreatment (DM and DM-sen)

In [ ]:

```
#implement A4
%run /test/demo_constraints.py
```

A5: Fairness-aware Classifier with Prejudice Remover Regularizer (PR)

In [7]:

```
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(d
df_a_X_train=t.tensor(np.array(df_a_X_train)).to(t.float32)
df_a_y_train=t.from_numpy(np.array(df_a_y_train).astype('float32')).reshape(d

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(d
df_a_X_valid=t.tensor(np.array(df_a_X_valid)).to(t.float32)
df_a_y_valid=t.from_numpy(np.array(df_a_y_valid).astype('float32')).reshape(d

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')).reshape(df_
df_a_X_test=t.tensor(np.array(df_a_X_test)).to(t.float32)
df_a_y_test=t.from_numpy(np.array(df_a_y_test).astype('float32')).reshape(df_

def accuracy( Model_c,Model_a, df_c_X_train,df_c_y_train,df_a_X_train,df_a_y_
    yc_pred = (Model_c(df_c_X_train) >= 0.5)
    ya_pred = (Model_a(df_a_X_train) >= 0.5)
    accu_c  = t.sum(yc_pred.flatten() == df_c_y_train.flatten()) / df_c_X_tra
    accu_a  = t.sum(ya_pred.flatten() == df_a_y_train.flatten()) / df_a_X_tra
    accuracy = (accu_c + accu_a) / 2
    calibration=abs(accu_c-accu_a)
    return round(accuracy.item(),4),round(calibration.item(),4)
    print("Accuracy : %.3f" % (accuracy * 100)+'%')
    print("Calibration : %.3f" % (calibration * 100)+'%')

def CVS(Model_c,Model_a,df_c_X_train,df_a_X_train):
    yc_pred = (Model_c(df_c_X_train) >= 0.5)
    ya_pred = (Model_a(df_a_X_train) >= 0.5)
    corr_c = t.sum(yc_pred == True)
    corr_a = t.sum(ya_pred == True)
    P_y1_s1 = corr_c / df_c_X_train.shape[0]
    P_y1_s0 = corr_a / df_a_X_train.shape[0]
    CV_score = t.abs(P_y1_s0 - P_y1_s1)
    return round(CV_score.item(),4)
```

Baseline Model: Logistic Regression

In [8]:

```
#Logistic refression with PR
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
```

With Prejudice Remover Regularizer

In [9]:

```
class PRLoss():#using linear
    def __init__(self, eta=1.0):
        super(PRLoss, self).__init__()
        self.eta = eta
    def forward(self,output_c,output_a):
        # For the mutual information,
        # eqn(9): Pr[y|s] = sum{(xi,si),si=s} sigma(xi,s) / D[xs]
        #D[xs]
        N_cau = t.tensor(output_c.shape[0])
        N_aa   = t.tensor(output_a.shape[0])
        Dxisi = t.stack((N_aa,N_cau),axis=0) #male sample, #female sample
        # Pr[y|s]
        y_pred_cau = t.sum(output_c)
        y_pred_aa   = t.sum(output_a)
        P_ys = t.stack((y_pred_aa,y_pred_cau),axis=0) / Dxisi
        # eqn(10): Pr[y]~sum{(xi,si)} sigma(xi,si) / |D[xs]|
        P = t.cat((output_c,output_a),0)
        P_y = t.sum(P) / (df_c_X_train.shape[0]+df_a_X_train.shape[0])
        # P(siyi)
        P_s1y1 = t.log(P_ys[1]) - t.log(P_y)
        P_s1y0 = t.log(1-P_ys[1]) - t.log(1-P_y)
        P_s0y1 = t.log(P_ys[0]) - t.log(P_y)
        P_s0y0 = t.log(1-P_ys[0]) - t.log(1-P_y)
        # eqn(11) RPR
        # PI=sum{xi,si}sum{y}M*ln(Pr[y|si]/Pr[y])=sum{xi,si}sum{y}M*ln(Pr[Y,S
        PI_s1y1 = output_c * P_s1y1
        PI_s1y0 =(1- output_c) * P_s1y0
        PI_s0y1 = output_a * P_s0y1
        PI_s0y0 = (1- output_a )* P_s0y0
        PI = t.sum(PI_s1y1) + t.sum(PI_s1y0) + t.sum(PI_s0y1) + t.sum(PI_s0y0
        PI = self.eta * PI
        return PI
```

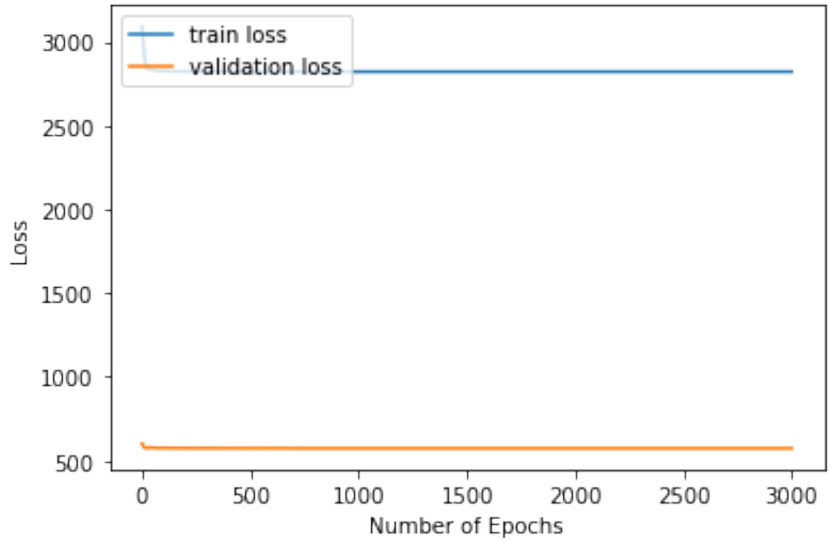
In [10]:

```
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,df_c_X_train,df_c_y_train,df_a_X_train,df_a_y_train,df_c_X_v
        model_c = LogisticRegression(df_c_X_train)
        model_a = LogisticRegression(df_a_X_train)
        criterion = nn.BCELoss(reduction='sum')
        PI = PRLoss(eta=self.eta)
        epochs = self.epochs
        optimizer = t.optim.Adam(list(model_c.parameters())+ list(model_a.par
        train_losses = []
        val_losses = []
        for epoch in range(epochs):
            model_c.train()
            model_a.train()
            optimizer.zero_grad()
            output_c = model_c(df_c_X_train)
            output_a = model_a(df_a_X_train)
            logloss = criterion(output_c, df_c_y_train)+ criterion(output_a, 
            PIloss = PI.forward(output_c,output_a)
            loss = PIloss +logloss
            loss.backward()
            optimizer.step()
            train_losses.append(loss)
            output_c = model_c(df_c_X_val)
            output_a = model_a(df_a_X_val)
            logloss = criterion(output_c, df_c_y_val)+ criterion(output_a, df_
            PIloss = PI.forward(output_c,output_a)
            loss = PIloss +logloss
            val_losses.append(loss)
        model_c.eval()
        model_a.eval()
        accu = accuracy(model_c,model_a,df_c_X_train,df_c_y_train,df_a_X_trai
        accu_val = accuracy(model_c,model_a,df_c_X_val,df_c_y_val,df_a_X_val,
        accu_test = accuracy(model_c,model_a,df_c_X_test,df_c_y_test,df_a_X_t
        plt.plot(list(range(epochs)),train_losses, label="train loss")
        plt.plot(list(range(epochs)),val_losses, label="validation loss")
        plt.legend(loc="upper left")
        plt.xlabel('Number of Epochs')
        plt.ylabel('Loss')
        plt.show()
        cvs = CVS(model_c,model_a,df_c_X_train, df_a_X_train)
        return accu, accu_val, accu_test,cvs
```

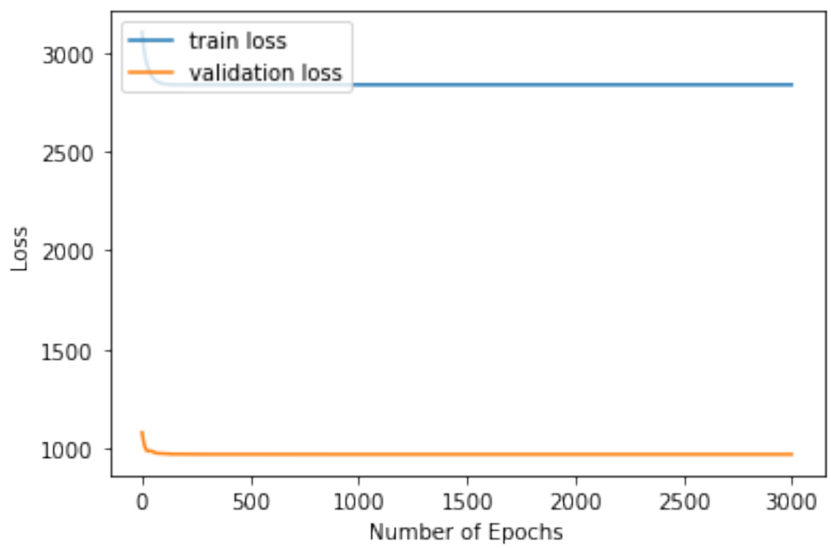
In [11]:

```
eta_value = [0.0,1.0,2.0,3.0,4.0,5.0,10.0,15.0,20.0,25.0,30.0]
accur = list()
accur_val = list()
accur_test = list()
cvss = list()
for e in range(0,len(eta_value)):
    print("Theta Value: %d" % eta_value[e])
    PR = PRLR(eta = eta_value[e], epochs = 3000, lr = 0.01)
    accur_eta,accur_val_eta,accur_test_eta,cvs = PR.fit(df_c_X_train,df_c_y_t
    accur.append(accur_eta)
    accur_val.append(accur_val_eta)
    accur_test.append(accur_test_eta)
    cvss.append(cvs)
```

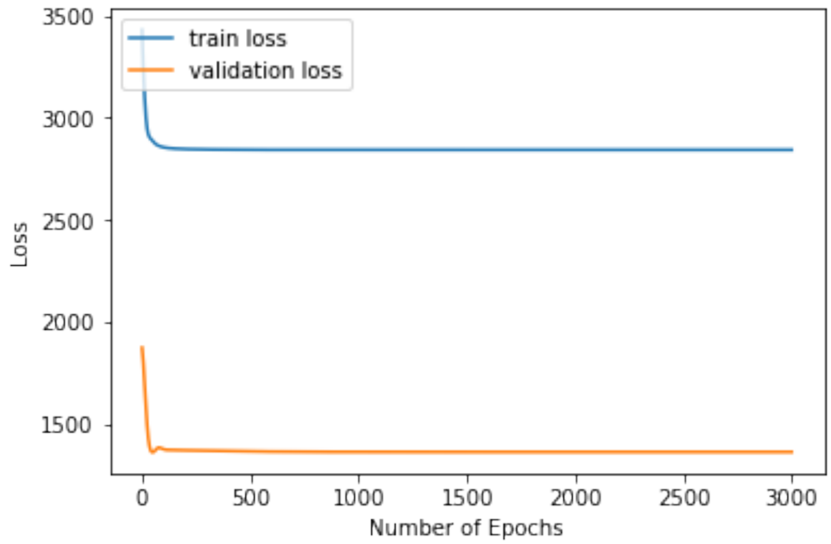
Theta Value: 0



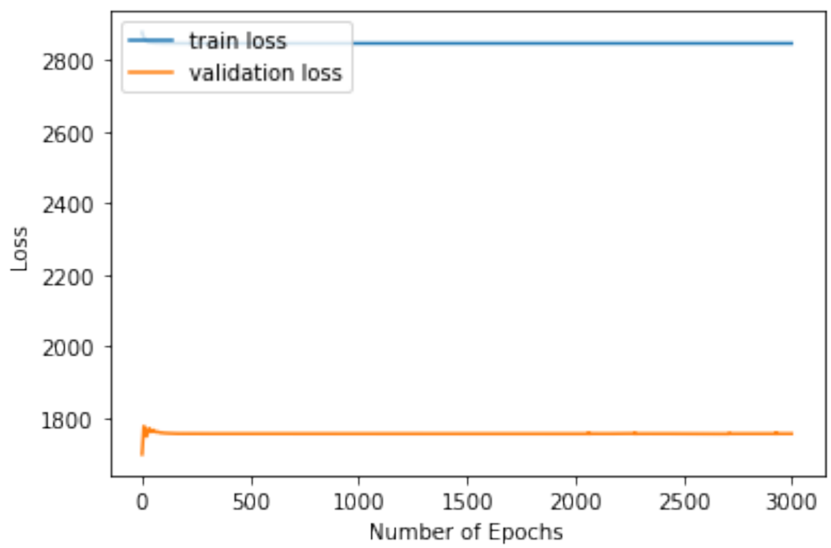
Theta Value: 1



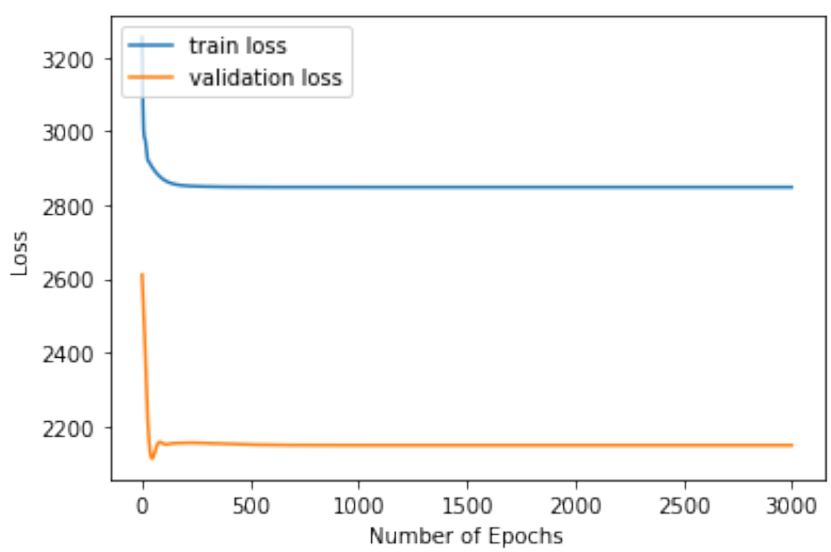
Theta Value: 2



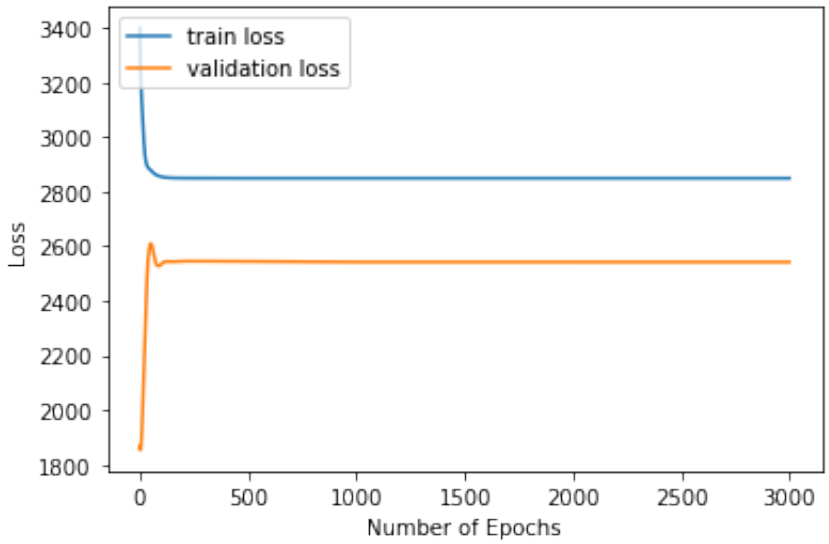
Theta Value: 3



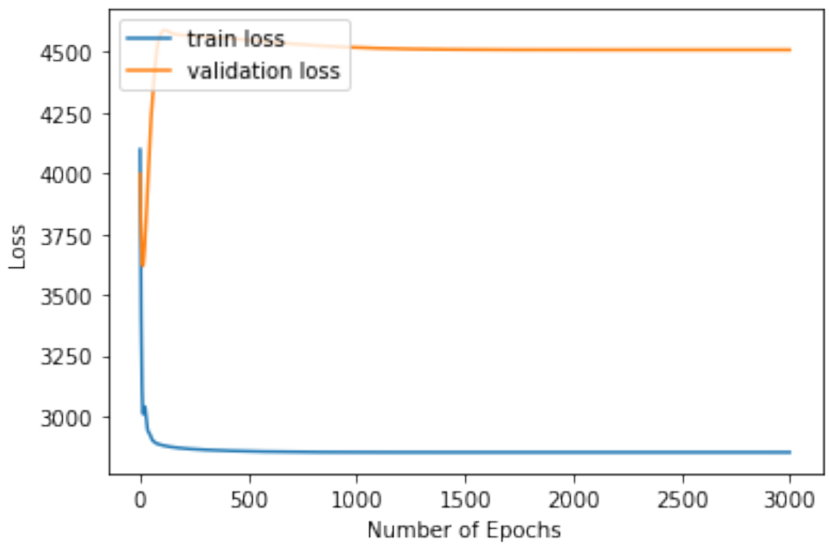
Theta Value: 4



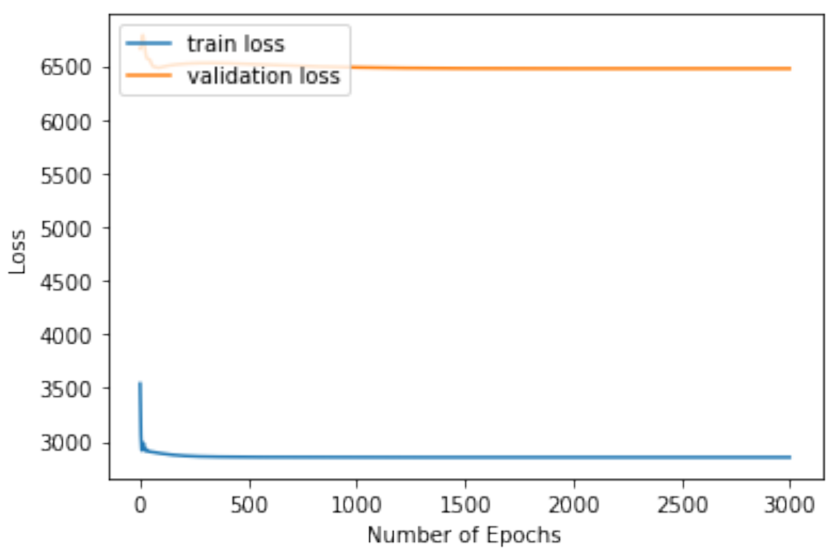
Theta Value: 5



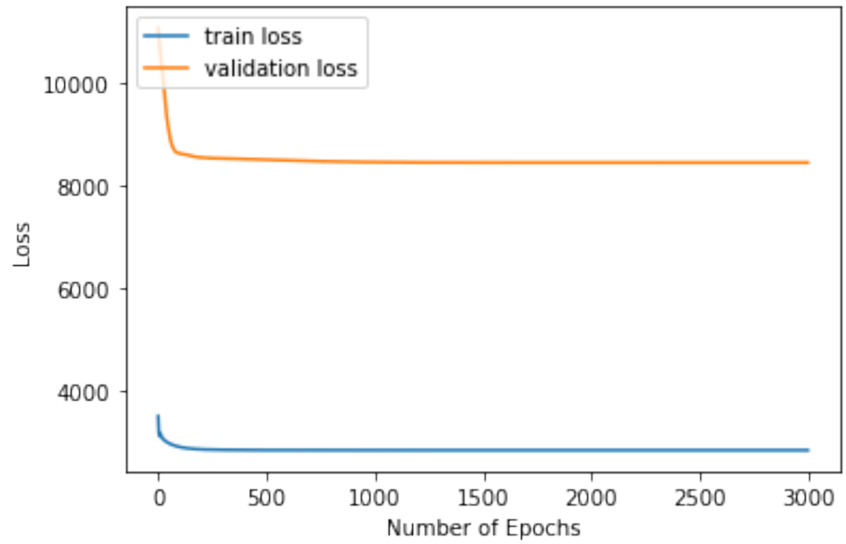
Theta Value: 10



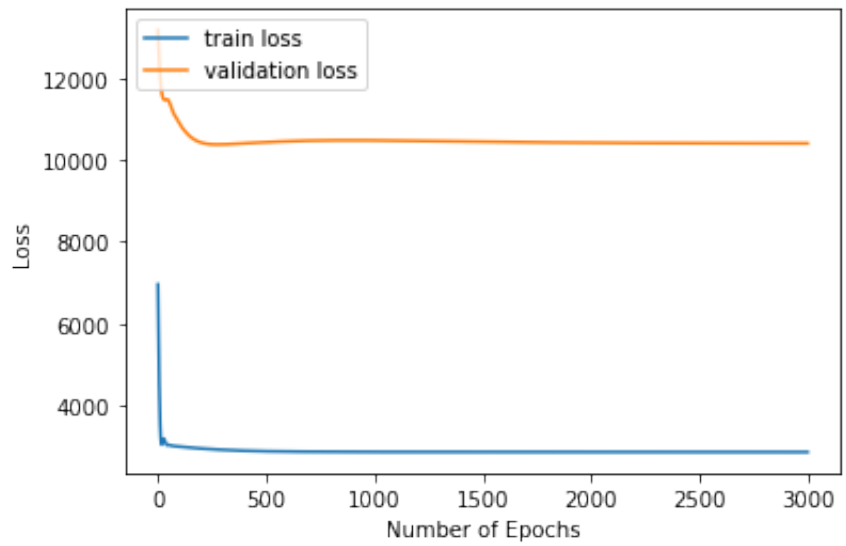
Theta Value: 15



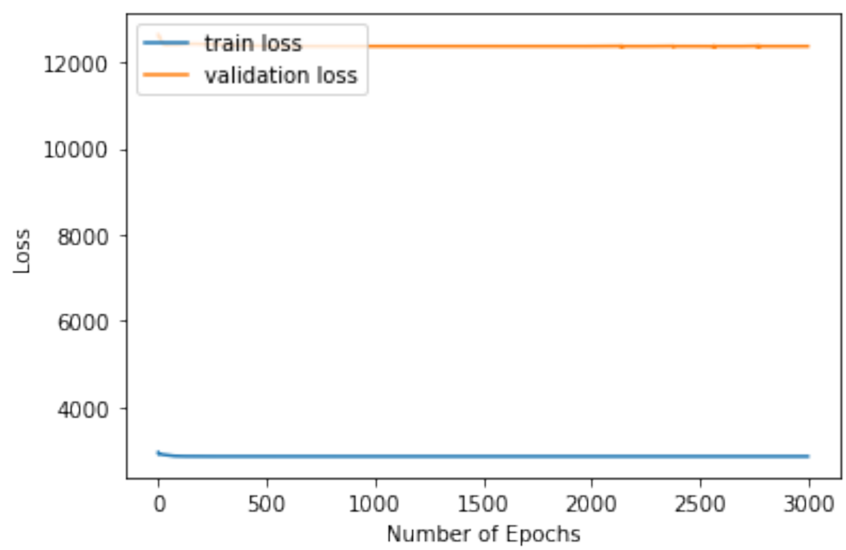
Theta Value: 20



Theta Value: 25

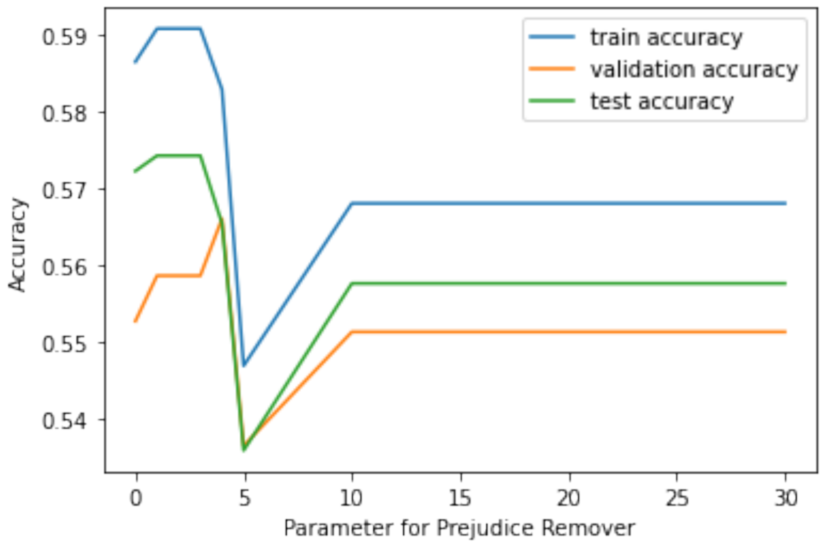


Theta Value: 30



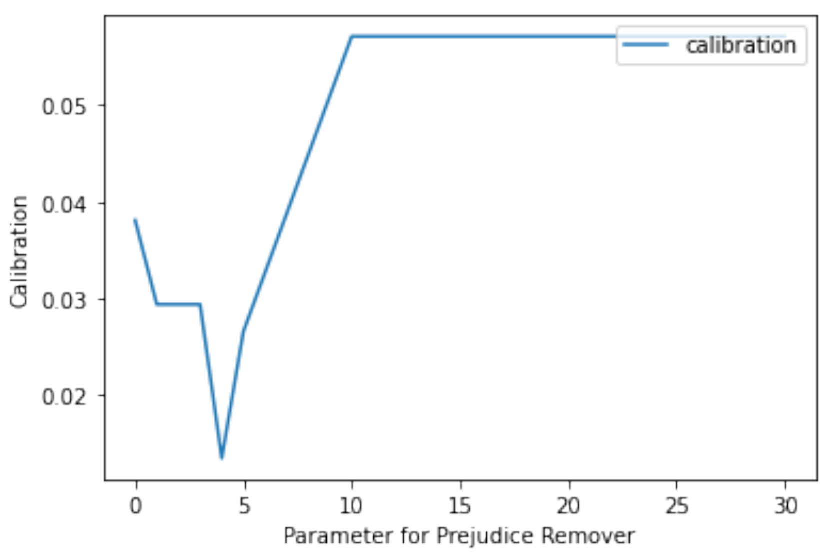
In [12]:

```
eta_accu_train = [i[0] for i in accur]
eta_accu_valid = [i[0] for i in accur_val]
eta_accuv_test = [i[0] for i in accur_test]
plt.plot(eta_value,eta_accu_train, label="train accuracy")
plt.plot(eta_value,eta_accu_valid, label="validation accuracy")
plt.plot(eta_value,eta_accuv_test, label="test accuracy")
plt.xlabel('Parameter for Prejudice Remover')
plt.ylabel('Accuracy')
plt.legend(loc="upper right")
plt.show()
```



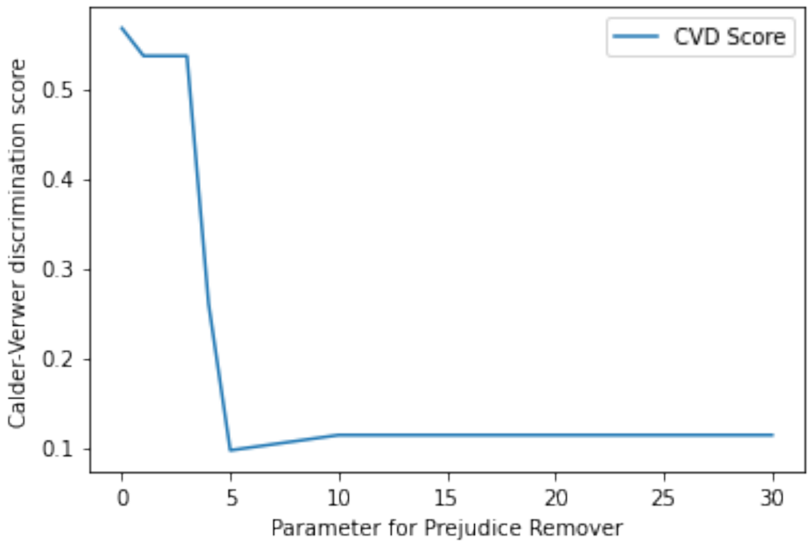
In [13]:

```
eta_accu_train = [i[1] for i in accur]
plt.plot(eta_value,eta_accu_train, label="calibration")
plt.xlabel('Parameter for Prejudice Remover')
plt.ylabel('Calibration')
plt.legend(loc="upper right")
plt.show()
```



In [14]:

```
plt.plot(eta_value,cvss, label="CVD Score")
plt.xlabel('Parameter for Prejudice Remover')
plt.ylabel('Calder-Verwer discrimination score')
plt.legend(loc="upper right")
plt.show()
```



In [15]:

```
#train
accur
```

Out[15]:

```
[(0.5864, 0.0381),
 (0.5907, 0.0294),
 (0.5907, 0.0294),
 (0.5907, 0.0294),
 (0.5828, 0.0135),
 (0.5469, 0.0266),
 (0.568, 0.0571),
 (0.568, 0.0571),
 (0.568, 0.0571),
 (0.568, 0.0571),
 (0.568, 0.0571)]
```

In [16]:

```
#validation
accur_val
```

Out[16]:

```
[(0.5527, 0.0005),
 (0.5586, 0.0114),
 (0.5586, 0.0114),
 (0.5586, 0.0114),
 (0.566, 0.0033),
 (0.5365, 0.0082),
 (0.5513, 0.0143),
 (0.5513, 0.0143),
 (0.5513, 0.0143),
 (0.5513, 0.0143),
 (0.5513, 0.0143)]
```

In [17]:

```
#test
accur_test
```

Out[17]:

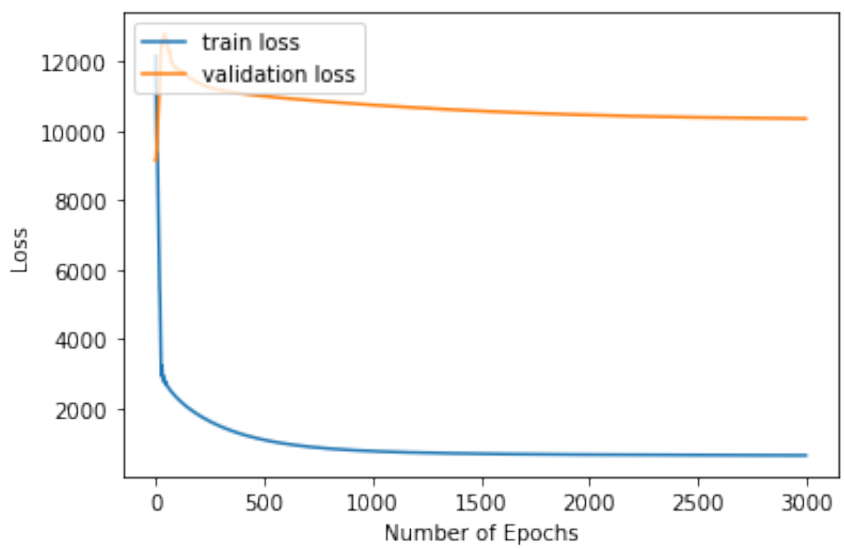
```
[(0.5722, 0.0614),
 (0.5742, 0.0575),
 (0.5742, 0.0575),
 (0.5742, 0.0575),
 (0.5654, 0.0398),
 (0.5359, 0.0164),
 (0.5576, 0.0034),
 (0.5576, 0.0034),
 (0.5576, 0.0034),
 (0.5576, 0.0034),
 (0.5576, 0.0034)]
```

Final Model

Baeline model: accuracy of testing dataset is 0.5722, calibration is 0.0614 Choose eta=4.0, accuracy of testing dataset is 0.5654, calibration is 0.0398. Calibration better than baseline model, PR reduce validation.

In [156…

```
#PR_eta1 = PRLR(eta = 5.0, epochs = 3000, lr = 0.01)
#PR.fit(df_c_X_train,df_c_y_train,df_a_X_train,df_a_y_train,df_c_X_valid,df_c_
```



Out[156…

```
((0.9704, 0.0097), (0.9645, 0.024), (0.9665, 0.0199), 0.1333)
```

In [ ]:

In [ ]: