Learning Fair Representations

Notations:

$\mathbb{X}$ denotes all individual in dataset with D features

$\mathbb{S}$ denotes whether individual are protected or not. In our case,race is the sensitive feature that need to be protected. $\mathbb{S} = 1$ indicate that individual are African-American (protected), $\mathbb{S} = 0$ indicates individual are Caucasian (unprotected).

$\mathbb{X}^-$ denotes subset of dataset that individual belongs to unprotected group($\mathbb{S} = 0$), whereas $\mathbb{X}^+$ denotes subset of individual belongs to ptotected group($\mathbb{S} = 1$).

V_k denotes the vector with dimension D associated with "prototypes". Prototypes comes from the variable $\mathbb{Z} \sim multinorm(k, n, p_1, p_2, \ldots\ldots p_k)$,where k values represent the intermediate prototypes.

$\mathbb{Y}$ denotes the binary variable that classify each individuals. In our case, $\mathbb{Y} = 1$ denotes individual are classfied as "Recided in two years", and $\mathbb{Y} = 0$ denotes individual are classfied as "Not Recided in two years".

Goal of algorithms:

Objective function:

Minimize $L = A_z \cdot L_z + A_x \cdot L_x + A_y \cdot L_y$

LFR Model:

Find the intermidiate representation by encoding the original dataset such that it preserves as much as possible information of individual's features and it is blinded to distinguish between protected group and unprotected group.

That is, when the algorithms maps the original dataset $\mathbb{X}$ to prototypes, the infomation that whether or not the individual belongs to protected group are removed. To achieve this, formula of statistical parity are used:

$$P(Z = k|x^+ \in \mathbb{X}^+) = P(Z = k|x^- \in \mathbb{X}^-)\ for\ all\ k$$

Where the probability are formulated by Softmax:

$$P(Z = k|X = x) = \frac{exp(-dist(x, v_k)}{\sum_{j=1}^{K} exp[-dist(x, v_j)]}$$

So after mapping $\mathbb{X}$ to Z, we hope the expectation value of probability mapping $\mathbb{X}^-$ to v_k equals to expectation value of probability mapping $\mathbb{X}^+$ to v_k.

That is,

$$M_k^+ = M_k^-$$

where

$$M_k^+ = \mathbb{E}_{x \in X^+} P(Z = k|X) = \frac{1}{|X^+|} \sum_{n \in X^+} M_{n,k}$$

And

$$M_{n,k} = P(Z = k|X_n)\ \ for\ all\ \ n, k$$

The algorithm is aimed to minimize objective function by the following three terms:

$$L_z = \sum_{k=1}^{K} |M_k^+ - M_k^-|$$
$$L_x = \sum_{n=1}^{N} (x_n - \hat{x}_n)^2\ \ where\ \ \hat{x}_n = \sum_{k=1}^{K} M_{n,k} V_k$$
$$L_y = \sum_{n=1}^{N} -y_n log\hat{y}_n - (1 - y_n) log(1 - \hat{y}_n)\ \ where\ \ \hat{y}_n = \sum_{k=1}^{K} M_{n,k} w_k$$

Define functions needed in algotithm

In [1]:
```
import numpy as np
import pandas as pd
```

In [7]:
```
def distances(X, V, alpha):
  N,D = X.shape
  K,D = V.shape
  dists = np.zeros((N, D))
  for i in range(N):
    for d in range(D):
      for k in range(K):
        dists[i, k] += (X[i, d] - V[k, d]) * (X[i, d] - V[k, d]) * alpha[d]
    return dists
```

In [8]:
```
def M_nks(dists,K):
  N,D = dists.shape
  upper = np.zeros((N,K))
  lower = np.zeros(N)
  M_nks = np.zeros((N,K))
  for i in range(N):
    for j in range(K):
      upper[i,j] = np.exp(-dists[i,j])
      lower[i] += upper[i,j]
    for j in range(K):
      if lower[i]:
        M_nks[i,j] = upper[i,j]/lower[i]
      else:
        M_nks[i, j] = upper[i, j] / 1e-6
  return M_nks
```

In [9]:
```
def M_ks (X,M_nks):
  N,K = M_nks.shape
  M_ks = np.zeros(K)
  for i in range(K):
    for j in range(N):
      M_ks[i] += M_nks[j,i]
    M_ks[i] = M_ks[i]/N
  return M_ks
```

In [10]:
```
def X_hat(X,V,M_nks):
  N,D = X.shape
  K,D = V.shape
  X_hat = np.zeros((N,D))
  for i in range(N):
    for d in range(D):
      for j in range(K):
        X_hat[i,d] += M_nks[i,j]*V[j,d]
  return X_hat
```

In [11]:
```
def l_x(X,X_hat):
  l_x = 0
  N,D = X.shape
  for i in range(N):
    for d in range(D):
      l_x += (X[i,d]-X_hat[i,d])**2
  return l_x
```

In [12]:
```
def Y_hat(M_nks,W):
  N,K = M_nks.shape
  Y_hat = np.zeros(N)
  for i in range(N):
    for j in range(K):
      Y_hat[i] += M_nks[i,j]*W[j]
    Y_hat[i] = 1e-6 if Y_hat[i] <= 0 else Y_hat[i]
    Y_hat[i] = 0.999 if Y_hat[i] >= 1 else Y_hat[i]
  return Y_hat
```

In [13]:
```
def l_y(Y,Y_hat):
  l_y = 0
  N = len(Y)
  for i in range(N):
    l_y += -Y[i]*np.log(Y_hat[i]) - (1-Y[i])*(np.log(1-Y_hat[i]))
  return l_y
```

In [14]:
```
def LFR(params,X_sen,X_unsen,Y_sen,Y_unsen, k=10, A_x = 1e-4, A_y = 0.1, A_z = 1000, result=0):
  LFR.iters+=1
  N1,P = X_sen.shape
  N0,_ = X_unsen.shape

  alpha0 = params[:P]
  alpha1 = params[P: 2 * P]
  W = params[2 * P: (2 * P) + k]
  V = np.matrix(params[(2 * P) + k:]).reshape((k, P))
  dists_sen = distances(X_sen, V, alpha1)
  dists_unsen = distances(X_unsen, V, alpha0)
  M_nks_sen = M_nks(dists_sen,k)
  M_nks_unsen = M_nks(dists_unsen,k)
  M_ks_sen = M_ks(X_sen,M_nks_sen)
  M_ks_unsen = M_ks(X_unsen,M_nks_unsen)

  X_hat_sen = X_hat(X_sen,V,M_nks_sen)
  X_hat_unsen = X_hat(X_unsen,V,M_nks_unsen)

  L_x_1 =  l_x(X_sen,X_hat_sen)
  L_x_0 = l_x(X_unsen,X_hat_unsen)

  Y_hat_sen = Y_hat(M_nks_sen,W)
  Y_hat_unsen = Y_hat(M_nks_unsen,W)

  L_y_1 = l_y(Y_sen,Y_hat_sen)
  L_y_0 = l_y(Y_unsen,Y_hat_unsen)

  L_X = L_x_1+L_x_0
  L_Y = L_y_1+L_y_0

  L_z = 0

  for i in range(k):
    L_z += abs(M_ks_sen[i]-M_ks_unsen[i])

  L_obj = A_z*L_z + A_x*L_X + A_y*L_Y

  if LFR.iters % 250 == 0:
    print(LFR.iters, L_obj)

  if result:
    return Y_hat_sen, Y_hat_unsen, M_nks_sen , M_nks_unsen
  else:
    return L_obj
```

Data

In [15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import random
import scipy
from scipy import optimize

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

In [53]:
```
def remove_missing_records(df: pd.DataFrame, threshold: float = 0.5) -> pd.DataFrame:
    """Remove records with missing values above a threshold."""
    # Get the number of missing values per column
    missing = df.isna().sum()
    # Get the columns with missing values above the threshold
    cols = missing[missing > threshold * df.shape[0]].index
    # Remove the columns
    df = df.drop(cols, axis=1)
    # Remove the rows with missing values
    df = df.dropna()
    return df

def inpute_missing_data(df):
    """Inpute missing data with the mean."""
    # Get the number of missing values per column
    missing = df.isna().sum()
    # Get the columns with missing values
    cols = missing[missing > 0].index
    # Inpute the missing values with the mean
    for col in cols:
        df[col] = df[col].fillna(df[col].mean())
    return df

def preprocess_data(df: pd.DataFrame) -> pd.DataFrame:
    """Preprocess the data."""
    # Remove the records with missing values
    df = remove_missing_records(df)
    # Inpute the missing values
    df = inpute_missing_data(df)
    return df
```

In [38]:
```
data = pd.read_csv('https://raw.githubusercontent.com/propublica/compas-analysis/master/compas-scores-two-years.csv')
```

In [39]:
```
data.head()
```

Out[39]:

	id	name	first	last	compas_screening_date	sex	dob	age	age_cat	race	...	v_decile_score	v_score_text	v_screening_date	in_custody
0	1	miguel hernandez	miguel	hernandez	2013-08-14	Male	1947-04-18	69	Greater than 45	Other	...	1	Low	2013-08-14	2014-07-07
1	3	kevon dixon	kevon	dixon	2013-01-27	Male	1982-01-22	34	25 - 45	African-American	...	1	Low	2013-01-27	2013-01-26
2	4	ed philo	ed	philo	2013-04-14	Male	1991-05-14	24	Less than 25	African-American	...	3	Low	2013-04-14	2013-06-16
3	5	marcu brown	marcu	brown	2013-01-13	Male	1993-01-21	23	Less than 25	African-American	...	6	Medium	2013-01-13	NaN
4	6	bouthy pierrelouis	bouthy	pierrelouis	2013-03-26	Male	1973-01-22	43	25 - 45	Other	...	1	Low	2013-03-26	NaN

5 rows × 53 columns

In [40]:
```
data = data[['age', 'c_charge_degree', 'race', 'age_cat', 'score_text', 'sex', 'priors_count',
             'days_b_screening_arrest', 'decile_score', 'is_recid', 'two_year_recid', 'c_jail_in', 'c_jail_out']]
data = data.loc[(data['days_b_screening_arrest'] <= 30) & (data['days_b_screening_arrest'] >= -30)]
data = data.loc[data['is_recid'] != -1]
data = data.loc[data['score_text'] != 'N/A']
data["length_of_stay"] = (pd.to_datetime(data['c_jail_out'])-pd.to_datetime(data['c_jail_in'])).dt.days
data = data.drop(columns=['c_jail_in', 'c_jail_out'])
data = data.loc[(data["race"] == "African-American") | (data["race"] == "Caucasian")]
data = data.replace({'race': 'Caucasian'}, 1)
data = data.replace({'race': 'African-American'}, 0)

data = data.replace({'sex': 'Male'}, 1)
data = data.replace({'sex': 'Female'}, 0)

data = data.replace({'age_cat': 'Less than 25'}, 0)
data = data.replace({'age_cat': '25 - 45'}, 1)
data = data.replace({'age_cat': 'Greater than 45'}, 2)

data = data.replace({'c_charge_degree': 'F'}, 0)
data = data.replace({'c_charge_degree': 'M'}, 1)

data = data.replace({'score_text': 'Low'}, 0)
data = data.replace({'score_text': 'Medium'}, 1)
data = data.replace({'score_text': 'High'}, 2)
```

In [19]:
```
data.head()
```

Out[19]:

	age	c_charge_degree	race	age_cat	score_text	sex	priors_count	days_b_screening_arrest	decile_score	is_recid	two_year_recid	length_of_stay
1	34	0	0	1	0	1	0	-1.0	3	1	1	10
2	24	0	0	0	0	1	4	-1.0	4	1	1	1
6	41	0	1	1	1	1	14	-1.0	6	1	1	6
8	39	1	1	1	0	0	0	-1.0	1	0	0	2
10	27	0	1	1	0	1	0	-1.0	4	0	0	1

In [42]:
```
data.shape
```

Out[42]:
```
(5278, 12)
```

In [43]:
```
X = data.drop(columns=["two_year_recid"])
y = data["two_year_recid"]
```

In [44]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
X_train_a = X_train[(X_train['race'] == 0)]
X_train_c = X_train[(X_train['race'] == 1)]
y_train_a = y_train[(X_train['race'] == 0)]
y_train_c = y_train[(X_train['race'] == 1)]
X_test_a = X_test[(X_test['race'] == 0)]
X_test_c = X_test[(X_test['race'] == 1)]
y_test_a = y_test[(X_test['race'] == 0)]
y_test_c = y_test[(X_test['race'] == 1)]
#X_train_a = X_train_a.drop(columns=["race"])
#X_train_c = X_train_c.drop(columns=["race"])
#X_test_a = X_test_a.drop(columns=["race"])
#X_test_c = X_test_c.drop(columns=["race"])
X_train_a = t.tensor(np.array(X_train_a)).to(t.float32)
y_train_a = t.from_numpy(np.array(y_train_a).astype('float32')).reshape(X_train_a.shape[0], 1)
X_train_c = t.tensor(np.array(X_train_c)).to(t.float32)
y_train_c = t.from_numpy(np.array(y_train_c).astype('float32')).reshape(X_train_c.shape[0], 1)

X_test_a = t.tensor(np.array(X_test_a)).to(t.float32)
y_test_a = t.from_numpy(np.array(y_test_a).astype('float32')).reshape(X_test_a.shape[0], 1)
X_test_c = t.tensor(np.array(X_test_c)).to(t.float32)
y_test_c = t.from_numpy(np.array(y_test_c).astype('float32')).reshape(X_test_c.shape[0], 1)
```

In [45]:
```
X_train_a.shape
```

Out[45]:
```
torch.Size([2542, 11])
```

In [46]:
```
LFR.iters = 0
# Initialize V_ks
#Number of prototypes:K

k=10
D = 11
rez = np.random.uniform(size=D * 2 + k + D * k)

bnd = []
for i, k2 in enumerate(rez):
    if i < D * 2 or i >= D * 2 + k:
        bnd.append((None, None))
    else:
        bnd.append((0, 1))

rez = optimize.fmin_l_bfgs_b(LFR, x0=rez, epsilon=1e-3,
                             args=(X_train_a, X_train_c,
                                   y_train_a, y_train_c, k, 1e-4,
                                   0.1, 1000, 0),
                             bounds = bnd, approx_grad=True, maxfun=2000,
                             maxiter=2000)

w = rez[0][D*2:D*2+k]
v = rez[0][D*2+k:].reshape(k,D)
```

```
250 tensor([1949.0575])
500 tensor([1938.3824])
750 tensor([1898.1364])
1000 tensor([1812.1644])
1250 tensor([1607.1807])
1500 tensor([1630.9956])
1750 tensor([1470.9220])
2000 tensor([1388.2178])
```

In [47]:
```
y_test_a = y_test_a.flatten()
y_test_a = list(y_test_a)
y_test_c = y_test_c.flatten()
y_test_c = list(y_test_c)
y_test= y_test_a + y_test_c
y_test = np.array(y_test)
```

In [48]:
```
LFR.iters=0
yhat_a, yhat_c, Mnk_a,Mnk_c = LFR(rez[0],
                                  X_train_a,
                                  X_train_c,
                                  y_train_a,
                                  y_train_c,10, 1e-4, 0.1, 1000, result=1)

yhat_a_test, yhat_c_test, Mnk_a_test,Mnk_c_test = LFR(rez[0],
                                  X_test_a,
                                  X_test_c,
                                  y_test_a,
                                  y_test_c,10, 1e-4, 0.1, 1000, result=1)
```

In [49]:
```
Y_pred_a = [1 if y >= 0.5 else 0 for y in yhat_a_test]
Y_pred_c = [1 if y >= 0.5 else 0 for  y in yhat_c_test]
```

In [52]:
```
def metrics_call( y_pred_a, y_a, y_pred_c, y_c):
    #y_pred_a = (1 if i >= 0.5 else 0 for i in y_pred_a)
    #y_pred_c = (1 if i >= 0.5 else 0 for  i in Y_pred_c)
    accuracy_a  = np.sum(y_pred_a == y_a.flatten()) / y_a.shape[0]
    accuracy_c  = np.sum(y_pred_c == y_c.flatten()) / y_c.shape[0]
    accuracy = (accuracy_a + accuracy_c) / 2
    discrimination = abs(sum(y_pred_a)/len(y_pred_a)-sum(y_pred_c)/len(y_pred_c))
    calibration = abs(accuracy_a - accuracy_c)
    return round(accuracy.item(),4), round(calibration.item(),4), round(discrimination.item(),4)
```

In [56]:
```
results = metrics_call(np.array(Y_pred_a), np.array(y_test_a), np.array(Y_pred_c), np.array(y_test_c))
results
```

Out[56]:
```
(0.4778, 0.1092, 0.0008)
```

In [58]:
```
from google.colab import drive
drive.mount('/content/drive')

with open('/content/drive/My Drive/results_LFR.txt', 'w') as f:
    np.savetxt(f, results)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```