A7

April 12, 2023

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
import torch as t
import torch.nn as nn
import pandas as pd
import warnings
import math
import itertools
import copy
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from datetime import datetime, timedelta
warnings.filterwarnings("ignore")
```

0.1 1. Data preprocessing

[2]:
```
data = pd.read_csv("compas-scores-two-years.csv")
df = data[['age', 'c_charge_degree', 'race', 'age_cat', 'score_text', 'sex',␣
 ↪'priors_count',
           'decile_score', 'is_recid', 'two_year_recid', 'c_jail_in',␣
 ↪'c_jail_out','is_violent_recid']]


df = df.loc[df['race'].isin(('African-American', 'Caucasian'))]
df.loc[df["race"] == "African-American", "race"] = 0
df.loc[df["race"] == "Caucasian", "race"] = 1

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

df = df.loc[df['is_recid'] != -1]
df = df.loc[df['c_charge_degree'] != 'O']
df = df.loc[df['score_text'] != 'N/A']

df['length_of_stay'] = (df['c_jail_out'].apply(pd.to_datetime) -␣
 ↪df['c_jail_in'].apply(pd.to_datetime)).dt.days
df = df.dropna(subset = ['length_of_stay'])
```

```python
df['length_of_stay'] = df['length_of_stay'].apply(lambda length_of_stay: 0 if␣
↪length_of_stay <= 7 else (2 if length_of_stay > 90 else 1))
df = df.drop(columns=['c_jail_in', 'c_jail_out'])

df['priors_count'] = df['priors_count'].apply(lambda priors_count: 0 if␣
↪priors_count == 0 else (2 if priors_count > 3 else 1))

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

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

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

df = df.drop_duplicates()

df.tail()
```

[2]:

```
      age  c_charge_degree  race  age_cat  score_text  sex  priors_count  \
7205   23                1     1        0           2    1             2
7206   21                1     1        0           1    1             0
7207   30                1     0        1           0    1             0
7208   20                0     0        0           2    1             0
7212   33                1     0        1           0    0             1

      decile_score  is_recid  two_year_recid  is_violent_recid  length_of_stay
7205            10         1               1                 1               1
7206             6         1               1                 0               0
7207             2         1               1                 0               0
7208             9         0               0                 0               0
7212             2         0               0                 0               0
```

0.2 2. Train Test Validation set split

Chosing features based on the paper: 'c_charge_degree', 'age_cat', 'sex', 'priors_count', 'length_of_stay'. Splitting the data into train, test ,and validation in the ratio 5:1:1.

[11]:

```python
X = df.drop(columns=["two_year_recid"])
Y = df["two_year_recid"]

#split dataset so that training:validation:testing=5:1:1
df_X_train, df_X_rem, df_Y_train, df_Y_rem = train_test_split(X,Y, train_size=5/
↪7.0)
```

```
df_X_valid, df_X_test, df_Y_valid, df_Y_test = train_test_split(df_X_rem,␣
 ↪df_Y_rem, test_size = 0.5)

A7_df = df

label = "two_year_recid"
sensitive = "race"
features = ['c_charge_degree', 'age_cat', 'sex',␣
 ↪'is_violent_recid','priors_count', 'length_of_stay']
features_race = ['race', 'c_charge_degree', 'age_cat', 'sex',␣
 ↪'is_violent_recid','priors_count', 'length_of_stay']

train_A7 = A7_df[:int(len(A7_df) * (5/7))]
test_A7 = A7_df[int(len(A7_df) * (5/7)):int(len(A7_df) * (6/7))]
vali_A7 = A7_df[int(len(A7_df) * (6/7)):]

x_train1 = train_A7[features]
y_train1 = train_A7[label].to_numpy()
race_train1 = train_A7[sensitive]

x_test1 = test_A7[features]
y_test1 = test_A7[label].to_numpy()
race_test1 = test_A7[sensitive]

x_validation1 = vali_A7[features]
y_validation1 = vali_A7[label].to_numpy()
race_validation1 = vali_A7[sensitive]

x_train_race = train_A7[features_race]
x_test_race = test_A7[features_race]
x_validation_race = vali_A7[features_race]

x_validation1.head()
```

[11]:

```
      c_charge_degree  age_cat  sex  is_violent_recid  priors_count  \
6012                1        1    0                 0             1
6015                0        1    0                 0             2
6017                0        2    1                 0             2
6018                1        1    1                 0             2
6021                0        2    1                 0             2

      length_of_stay
6012               0
6015               0
6017               0
6018               0
6021               0
```

0.3 3. Baseline Model

Baseline model with the featrue race

[12]:
```
clf_race = LogisticRegression().fit(x_train_race, y_train1)
accuracy_race = clf_race.score(x_validation_race, y_validation1)
accuracy_race
```

[12]: 0.6232980332829047

Baseline model without the featrue race

[26]:
```
clf_withtout_race = LogisticRegression().fit(x_train1, y_train1)
accuracy_withtout_race = clf_withtout_race.score(x_validation1, y_validation1)
accuracy_withtout_race
```

[26]: 0.6248108925869894

0.4 4. A7 Information Theoretic Measures for Fairness-aware Feature Selection

[14]:
```
def uni_values_array(arr):

    # arr: n * m matrix
    # each elements in uni_values_array gives unique values from the matrix

    uni_values = []
    for col in range(arr.shape[1]):
        uni_values.append(np.unique(arr[:, col]).tolist())
    return uni_values

def power_func(seq):

    #This function create a generator that contains all subsets of seq

    if len(seq) <= 1:
        yield seq
        yield []
    else:
        for i in power_func(seq[1:]):
            yield [seq[0]] + i
            yield i

def unique_information(array_1, array_2):
    assert array_1.shape[0] == array_2.shape[0]

    n_rows = array_1.shape[0]
    n_col_array_1 = array_1.shape[1]

    concated_array = np.concatenate((array_1, array_2), axis=1)
```

```
    unique_array = uni_values_array(concated_array)
    cartesian_product = list(itertools.product(*unique_array))

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

    IQ = 0
    for i in cartesian_product:
        r1_r2 = len(np.where((concated_array == i).all(axis=1))[0]) / n_rows
        r1 = len(np.where((array_1 == i[:n_col_array_1]).all(axis=1))[0]) /␣
↪n_rows
        r2 = len(np.where((array_2 == i[n_col_array_1:]).all(axis=1))[0]) /␣
↪n_rows

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


def unique_infor_condi(array_1, array_2, conditional):
    assert (array_1.shape[0] == array_2.shape[0]) and (array_1.shape[0] ==␣
↪conditional.shape[0])

    n_rows = array_1.shape[0]
    n_col_array_1 = array_1.shape[1]
    n_col_array_2 = array_2.shape[1]

    concated_array_2_conditional = np.concatenate((array_2, conditional),␣
↪axis=1)
    concated_array_all           = np.concatenate((array_1,␣
↪concated_array_2_conditional), axis=1)
    unique_array                 = uni_values_array(concated_array_all)
    cartesian_product = list(itertools.product(*unique_array))

    IQ = 0
    for i in cartesian_product:
        r1_r2 = len(np.where((concated_array_all == i).all(axis=1))[0]) / n_rows
        r1    = len(np.where((array_1 == i[:n_col_array_1]).all(axis=1))[0]) /␣
↪n_rows
        r2    = len(np.where((concated_array_all[:, n_col_array_1:␣
↪-n_col_array_2] == i[n_col_array_1: -n_col_array_2]).all(axis=1))[0]) /␣
↪n_rows
```

```python
        try:
            r1_given_r2 = len(np.where((concated_array_all[:, :n_col_array_1] == i[ :n_col_array_1]).all(axis=1) & (concated_array_all[:, -n_col_array_2:] == i[-n_col_array_2:]).all(axis=1))[0]) / len(np.where((concated_array_all[:, -n_col_array_2:] == i[-n_col_array_2:]).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:
            IQ_iter = 0
        else:
            IQ_iter = r1_r2 * np.log(r1_r2 / r2) / r1_given_r2
        IQ += np.abs(IQ_iter)
    return IQ

def accuracy_coef(y, x_s, x_s_c, A):
    conditional = np.concatenate((x_s_c, A), axis=1)
    return unique_infor_condi(y, x_s, conditional)

def discrimination_coef(y, x_s, A):
    x_s_a = np.concatenate((x_s, A), axis=1)
    return unique_information(y, x_s_a) * unique_information(x_s, A) * unique_infor_condi(x_s, A, y)

def marginal_accuracy_coef(y_train, x_train, A, set_tracker):

    n_features = x_train.shape[1]
    feature_idx = list(range(n_features))
    feature_idx.pop(set_tracker)
    power_func_features = [x for x in power_func(feature_idx) if len(x) > 0]

    shapley_value =0
    for sc_idx in power_func_features:
            coef = math.factorial(len(sc_idx)) * math.factorial(n_features - len(sc_idx) - 1) / math.factorial(n_features)

            # Compute v(T  {i})
            idx_xs_ui = copy.copy(sc_idx)
            idx_xs_ui.append(set_tracker)
            idx_xsc_ui = list(set(list(range(n_features))).difference(set(idx_xs_ui))) # compliment of x_s
            vTU = accuracy_coef(y_train.reshape(-1, 1), x_train[:, idx_xs_ui], x_train[:, idx_xsc_ui], A.reshape(-1, 1))

             # Compute v(T)
            idx_xsc = list(range(n_features))
            idx_xsc.pop(set_tracker)
```

```
            idx_xsc = list(set(idx_xsc).difference(set(sc_idx)))
            vT = accuracy_coef(y_train.reshape(-1, 1), x_train[:, sc_idx],␣
↪x_train[:, idx_xsc], A.reshape(-1, 1))

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

def marginal_discrimination_coef(y_train, x_train, A, set_tracker):

    n_features = x_train.shape[1]
    feature_idx = list(range(n_features))
    feature_idx.pop(set_tracker)
    power_func_features = [x for x in power_func(feature_idx) if len(x) > 0]

    shapley_value =0
    for sc_idx in power_func_features:
            coef = math.factorial(len(sc_idx)) * math.factorial(n_features -␣
↪len(sc_idx) - 1) / math.factorial(n_features)

            # Compute v(T   {i})
            idx_xs_ui = copy.copy(sc_idx)
            idx_xs_ui.append(set_tracker)
            vTU = discrimination_coef(y_train.reshape(-1, 1), x_train[:,␣
↪idx_xs_ui], A.reshape(-1, 1))

            # Compute v(T)
            vT = discrimination_coef(y_train.reshape(-1, 1), x_train[:,␣
↪sc_idx], A.reshape(-1, 1))

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

    return shapley_value
```

0.5 5. Shapley Value

Takes 80 seconds for one feature. Expecting 8 minutes for calculating six feature's shapley.

[16]:
```
shapley_acc = []
shapley_disc = []
for i in range(6):
    acc_i = marginal_accuracy_coef(y_train1, x_train1.to_numpy(), race_train1.
↪to_numpy(), i)
    disc_i = marginal_discrimination_coef(y_train1, x_train1.to_numpy(),␣
↪race_train1.to_numpy(), i)
    shapley_acc.append(acc_i)
```

```
    shapley_disc.append(disc_i)

# DataFrame to compare shapely values
#should is_recid be in here given it's the response?

names = ['c_charge_degree', 'age_cat', 'sex',␣
↪'is_violent_recid','priors_count', 'length_of_stay']

shapley = pd.DataFrame(list(zip(names, shapley_acc, shapley_disc)),
                          columns=["Feature", "Accuracy",'Discrimination'])
shapley
```

[16]:

	Feature	Accuracy	Discrimination
0	c_charge_degree	0.981916	84457.538802
1	age_cat	1.195308	107576.605927
2	sex	0.903141	78210.136834
3	is_violent_recid	0.744201	72271.230359
4	priors_count	1.185834	108126.985005
5	length_of_stay	1.038081	101778.528671

Conclusion:

As per the algorithm outlined in 'Information Theoretic Measures for Fairness-aware Feature selection (FFS)', we computed both the marginal accuracy coefficient and the marginal discrimination coefficient. The obtained outcome demonstrates that Age and Priors Counts exhibit the most significant influence on accuracy while also serving as strong indicators for discrimination, corroborating the conclusion drawn in paper A7. As such, simply discarding either of these features could have a significant impact on both model accuracy and calibration.

To make a more informed decision regarding feature selection, we evaluated three fairness utility scores introduced in paper A7, each of which trades off between accuracy and discrimination, using different hyperparameters.

[42]:
```
def fairness_utility_score(Accuracy, Discr, alpha_value):
    fu_scores = []
    for i in range(6):
        fu_score = Accuracy[i] - alpha_value * Discr[i]
        fu_scores.append(fu_score)
    return fu_scores
```

[43]:
```
alpha1 = pd.DataFrame(list(zip(names, shapley_acc, shapley_disc,␣
↪fairness_utility_score(shapley['Accuracy'], shapley['Discrimination'], 0.
↪000001))),
                          columns=["Feature", "Accuracy",'Discrimination',␣
↪'F_score'])
alpha1
```

[43]:
```
             Feature  Accuracy  Discrimination   F_score
0    c_charge_degree  0.981916    84457.538802  0.897458
1            age_cat  1.195308   107576.605927  1.087731
2                sex  0.903141    78210.136834  0.824930
3  is_violent_recid  0.744201    72271.230359  0.671930
4       priors_count  1.185834   108126.985005  1.077707
5     length_of_stay  1.038081   101778.528671  0.936302
```

[44]:
```
alpha2 = pd.DataFrame(list(zip(names, shapley_acc, shapley_disc,␣
↪fairness_utility_score(shapley['Accuracy'], shapley['Discrimination'], 0.
↪00001))),
                      columns=["Feature", "Accuracy",'Discrimination',␣
↪'F_score'])
alpha2
```

[44]:
```
             Feature  Accuracy  Discrimination   F_score
0    c_charge_degree  0.981916    84457.538802  0.137340
1            age_cat  1.195308   107576.605927  0.119542
2                sex  0.903141    78210.136834  0.121039
3  is_violent_recid  0.744201    72271.230359  0.021489
4       priors_count  1.185834   108126.985005  0.104564
5     length_of_stay  1.038081   101778.528671  0.020295
```

[46]:
```
alpha3 = pd.DataFrame(list(zip(names, shapley_acc, shapley_disc,␣
↪fairness_utility_score(shapley['Accuracy'], shapley['Discrimination'], 0.
↪0001))),
                      columns=["Feature", "Accuracy",'Discrimination',␣
↪'F_score'])
alpha3
```

[46]:
```
             Feature  Accuracy  Discrimination   F_score
0    c_charge_degree  0.981916    84457.538802 -7.463838
1            age_cat  1.195308   107576.605927 -9.562353
2                sex  0.903141    78210.136834 -6.917873
3  is_violent_recid  0.744201    72271.230359 -6.482922
4       priors_count  1.185834   108126.985005 -9.626865
5     length_of_stay  1.038081   101778.528671 -9.139772
```

For the alpha1 = 0.000001, the F-score of 'is_violent_recid' is the lowest.

For the alpha2 = 0.00001, the F-score of 'length_of_stay' is the lowest.

For the alpha3 = 0.0001, the F-score of 'priors_count' and 'age_cat' are the lowest. However, due to high marginal accuracy, we cannot remove these two feature. Therefore, we perform logistic regression by removing the next features with lowest F-score, which is length_of_stay.

0.6 6. Logistic Regression After Shapley Features Selection

6.1.1 Based on Alpha 1, we remove is_violent_recid which has the lowest F-score.

[47]:
```
x_train_subset_vc = x_train1.drop(["is_violent_recid"],axis = 1)
x_test_subset_vc = x_test1.drop(["is_violent_recid"],axis = 1)
x_validation_vc = x_validation1.drop(["is_violent_recid"],axis = 1)

FFS_LogReg_vc = LogisticRegression(random_state = 0).fit(x_train_subset_vc,␣
 ↪y_train1)

accuracy_ffs_vc = FFS_LogReg_vc.score(x_validation_vc,y_validation1)
accuracy_ffs_vc
```

[47]: 0.5854765506807866

6.1.2 Based on Alpha 2 & 3, we remove length_of_stay which has the lowest F-score.

[50]:
```
x_train_subset_ls = x_train1.drop(["length_of_stay"],axis = 1)
x_test_subset_ls = x_test1.drop(["length_of_stay"],axis = 1)
x_validation_ls = x_validation1.drop(["length_of_stay"],axis = 1)

FFS_LogReg_ls = LogisticRegression(random_state = 0).fit(x_train_subset_ls,␣
 ↪y_train1)

accuracy_ffs_ls = FFS_LogReg_ls.score(x_validation_ls,y_validation1)
accuracy_ffs_ls
```

[50]: 0.6384266263237519

We will verify our results through calibration. In the following step, we will begin by training our baseline models on the complete dataset, including all six features. Then, we will remove one feature at a time from the set consisting of Gender, is_violent_recid, Length of Stay, c_charge_degree, Age_cat, and Prior Counts, respectively. Finally, we will compare the accuracy and calibration of these models.

0.7 7. Logistic Regression After calibration Features Selection

[56]:
```
def MyCalibration(sensitive_attr, y_pred, y_true):
    cau_index = np.where(sensitive_attr == 1)[0]
    african_index = np.where(sensitive_attr == 0)[0]

    y_pred_cau = y_pred[cau_index]
    y_true_cau = y_true[cau_index]
    Acc_cau = sum(y_pred_cau == y_true_cau)/len(y_pred_cau)

    y_pred_african = y_pred[african_index]
    y_true_african = y_true[african_index]
    Acc_african = sum(y_pred_african == y_true_african)/len(y_pred_african)

    calibration = abs(Acc_cau - Acc_african)
    return(calibration)
```

[60]:

```
Accuracy_lr = []
Calibration_lr = []

for i in ['base'] + features:
    if i == 'base':
        logReg = LogisticRegression(random_state = 0).fit(x_train1, y_train1)
        Accuracy_lr.append(logReg.score(x_test1, y_test1))
        Calibration_lr.append(MyCalibration(race_test1, logReg.
 ↪predict(x_test1), y_test1))
    else:
        x_train_subset = x_train1.drop([i],axis= 1)
        x_test_subset = x_test1.drop([i],axis = 1)
        logReg = LogisticRegression(random_state = 0).fit(x_train_subset,␣
 ↪y_train1)
        Accuracy_lr.append(logReg.score(x_test_subset, y_test1))
        Calibration_lr.append(MyCalibration(race_test1, logReg.
 ↪predict(x_test_subset), y_test1))

col_names = ['base'] + names
Conclusion_lr = pd.DataFrame(list(zip(col_names, Accuracy_lr, Calibration_lr)),
                          columns=["Eliminating Feature", "Accuracy",␣
 ↪"Calibration"])
Conclusion_lr
```

[60]:

	Eliminating Feature	Accuracy	Calibration
0	base	0.665152	0.015793
1	c_charge_degree	0.662121	0.016577
2	age_cat	0.660606	0.026210
3	sex	0.654545	0.009297
4	is_violent_recid	0.596970	0.018033
5	priors_count	0.624242	0.010977
6	length_of_stay	0.657576	0.014673

Final Conclusion: As the calibration for sex is the lowest, we condiser sex to be the second featrue to be removed. Therefore we conclude the model without length_of_stay and sex is our final model.

[62]:

```
x_train_subset_final = x_train1.drop(["length_of_stay","sex"],axis = 1)
x_test_subset_final = x_test1.drop(["length_of_stay","sex"],axis = 1)
x_validation_final = x_validation1.drop(["length_of_stay","sex"],axis = 1)



FFS_LogReg_final = LogisticRegression(random_state = 0).
 ↪fit(x_train_subset_final, y_train1)

accuracy_ffs_final = FFS_LogReg_final.score(x_validation_final,y_validation1)
accuracy_ffs_final
```

```
[62]: 0.642965204236006
```

0.8 9. Citations

https://towardsdatascience.com/optimization-with-scipy-and-application-ideas-to-machine-learning-81d39c7938b8

https://github.com/mbilalzafar/fair-classification/tree/master/disparate_impact

https://github.com/TZstatsADS/fall2022-project4-group-10

https://www.propublica.org/datastore/dataset/compas-recidivism-risk-score-data-and-analysis

https://github.com/TZstatsADS/Fall2021-Project4-group4

https://github.com/SreeranjaniD/Fairness-in-Classification-using-SVM

https://arxiv.org/abs/2106.00772