Information Theoretic Measures for Fairness-aware Feature selection

In general, fairness and bias are considered relevant when decisions that impact people's lives, particularly with respect to a set of variables considered sensitive, such as gender, ethnicity, sexual orientation, disability, etc. In Machine Learning models, Outcomes might be skewed by a range of factors and thus might be considered unfair concerning specific groups or individuals.

Features relevant for accurate decisions may lead to explicit or implicit forms of discrimination against unprivileged groups, such as those of a certain race or gender. This happens due to existing biases in the training data, which are often replicated by the learning algorithm.

This model tries to tackle it by using information-theoretic measures which quantify the impact of different subsets of features on the accuracy and discrimination of the dependent variable(Outcome Variable). Then use the Shapley value function to quantify the marginal impact of each feature. This method focuses on the impact of features on discriminatory predictions and does not focus on a specific classifier design.

Loading the used packages

In [16]:
```
#Load the required packages
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.utils import shuffle
import numpy as np
from sklearn.metrics import log_loss
from scipy import optimize
import copy
import itertools
import math
from sklearn.svm import SVC
from sklearn.utils import shuffle
```

In [17]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

1. Data Processing:

Loading the data:

In [18]:
```
#data = pd.read_csv("compas-scores-two-years.csv")
data = pd.read_csv("../data/compas-scores-two-years.csv")

data.head()
```

Out[18]:

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

5 rows × 53 columns

Selecting the relevant features and pre processing:

In [19]:
```
def process_compas_dataset(df):
    df = df[["sex","age","age_cat","race","priors_count","c_charge_degree","c_jail_in", "c
    df["two_year_recid"] = df["two_year_recid"].apply(lambda x: -1 if x==0 else 1)

    #Only select Caucasian/African American, encode to 0/1
    df = df[df["race"].isin(["Caucasian","African-American"])]


    #categorical encoding race, gender, charge_degree
    df["race"] = df["race"].apply(lambda x: 1 if x == "Caucasian" else 0)
    df["gender_cat"] = df["sex"].apply(lambda x: 1 if x == "Female" else 0)
    df = df.drop(columns = "sex")
    df["charge_cat"] = df["c_charge_degree"].apply(lambda x: 1 if x == "F" else 0)
    df = df.drop(columns = "c_charge_degree")


    #Calculate length of stay
    df["length_stay"] = pd.to_datetime(df["c_jail_out"]) - pd.to_datetime(df['c_jail_in'])
    df["length_stay"] = df["length_stay"].apply(lambda x: x.days)
    df = df.drop(columns = ["c_jail_in","c_jail_out"])
    df['length_stay'] = df["length_stay"].apply(lambda x: 0 if x <= 7 else x)
    df['length_stay'] = df["length_stay"].apply(lambda x: 1 if 7< x <= 90 else x)
    df['length_stay'] = df["length_stay"].apply(lambda x: 2 if x > 90 else x)


    #Categorize priors count into 3 categories
    df["priors_count"] = df["priors_count"].apply(lambda x: 0 if x==0 else x)
    df["priors_count"] = df["priors_count"].apply(lambda x: 1 if (1<=x<=3) else x)
    df["priors_count"] = df["priors_count"].apply(lambda x: 2 if x>3 else x)

    df['age_cat'].replace(['Greater than 45', '25 - 45', 'Less than 25'],
                          [0, 1, 2], inplace=True)
    df = df.drop(columns = ["age"])

    print(len(df.index))
    df = df.dropna()
    print(len(df.index))

    y_label = df["two_year_recid"]
    protected_attribute = df["race"]
    df = df.drop(columns=["two_year_recid","race"])
    y_label, protected_attr, df = shuffle(y_label, protected_attribute, df, random_state =

    return y_label.to_numpy(), protected_attr.to_numpy(), df.to_numpy()
```

Constructing our training and test set:

In [20]:
```
#Still using compas dataset for evaluation
y_label, protected_attr, X =  process_compas_dataset(data)

train_index = int(len(X)*.80)
x_train, y_train, race_train = X[:train_index], y_label[:train_index], protected_attr[:tra
x_test, y_test, race_test = X[train_index:], y_label[train_index:],protected_attr[train_in
```

```
6150
5915

<ipython-input-19-195d6a5f3a17>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df["two_year_recid"] = df["two_year_recid"].apply(lambda x: -1 if x==0 else 1)
```

Implementation FFS:

In [21]:
```
"""This cell contains utility functions called in the proceeding cells."""

def get_uniq_vals_in_arr(arr):
    """Returns unique values in array.

    :param arr (np.array) n * m matrix
    :return (list) uniq_vals[i] contains unique values of ith column in arr
    """
    uniq_vals = []
    for id_col in range(arr.shape[1]):
        uniq_vals.append(np.unique(arr[:, id_col]).tolist())
    return uniq_vals


def powerset(seq):
    """
    Returns all the subsets of this set. This is a generator.
    """
    if len(seq) <= 1:
        yield seq
        yield []
    else:
        for item in powerset(seq[1:]):
            yield [seq[0]]+item
            yield item
```

In [22]:
```
"""This cell contains code for all the routines needed to calculate the Shapley coefficien

def get_info_coef(left, right):
    # Both arrays NEED same number of rows
    assert left.shape[0] == right.shape[0]
    num_rows = left.shape[0]
    num_left_cols = left.shape[1]

    concat_mat = np.concatenate((left, right), axis=1)
    concat_uniq_vals = get_uniq_vals_in_arr(concat_mat)
    concat_combos = list(itertools.product(*concat_uniq_vals))
    p_sum = 0
    for vec in concat_combos:
        p_r1_r2 = len(np.where((concat_mat == vec).all(axis=1))[0]) / num_rows
        p_r1 = len(np.where((left == vec[:num_left_cols]).all(axis=1))[0]) / num_rows
        p_r2 = len(np.where((right == vec[num_left_cols:]).all(axis=1))[0]) / num_rows

        if p_r1_r2 == 0 or p_r1 == 0 or p_r2 == 0:
            p_iter = 0
        else:
            p_iter = p_r1_r2 * np.log(p_r1_r2 / p_r1) / p_r1
        p_sum += np.abs(p_iter)
    return p_sum


def get_conditional_info_coef(left, right, conditional):
    assert (left.shape[0] == right.shape[0]) and (left.shape[0] == conditional.shape[0])
    num_rows = left.shape[0]
    num_left_cols = left.shape[1]
    num_right_cols = right.shape[1]

    right_concat_mat = np.concatenate((right, conditional), axis=1)
    concat_mat = np.concatenate((left, right_concat_mat), axis=1)
    concat_uniq_vals = get_uniq_vals_in_arr(concat_mat)
    concat_combos = list(itertools.product(*concat_uniq_vals))
    p_sum = 0
    for vec in concat_combos:
        p_r1_r2 = len(np.where((concat_mat == vec).all(axis=1))[0]) / num_rows
        p_r1 = len(np.where((left == vec[:num_left_cols]).all(axis=1))[0]) / num_rows
        p_r2 = len(np.where((concat_mat[:, num_left_cols: -num_right_cols] == vec[num_left

        try:
            p_r1_given_r3 = len(np.where((concat_mat[:, :num_left_cols] == vec[:num_left_c
        except ZeroDivisionError:
            p_r1_given_r3 = 0

        if p_r1_r2 == 0 or p_r1 == 0 or p_r2 == 0 or p_r1_given_r3 == 0:
            p_iter = 0
        else:
            p_iter = p_r1_r2 * np.log(p_r1_r2 / p_r2) / p_r1_given_r3
        p_sum += np.abs(p_iter)
    return p_sum


def get_acc_coef(y, x_s, x_s_c, protected_attr):
    conditional = np.concatenate((x_s_c, protected_attr), axis=1)
```

```
    return get_conditional_info_coef(y, x_s, conditional)


def get_disc_coef(y, x_s, protected_attr):
    x_s_a = np.concatenate((x_s, protected_attr), axis=1)
    return get_info_coef(y, x_s_a) * get_info_coef(x_s, protected_attr) * get_conditional


def get_shapley_acc_i(y, x, protected_attr, i):
    """Returns Shapley coeffecient of ith feature in x."""

    num_features = x.shape[1]
    lst_idx = list(range(num_features))
    lst_idx.pop(i)
    power_set = [x for x in powerset(lst_idx) if len(x) > 0]

    shapley = 0
    for set_idx in power_set:
        coef = math.factorial(len(set_idx)) * math.factorial(num_features - len(set_idx) -

        # Calculate v(T U {i})
        idx_xs_incl = copy.copy(set_idx)
        idx_xs_incl.append(i)
        idx_xsc_incl = list(set(list(range(num_features))).difference(set(idx_xs_incl)))
        acc_incl = get_acc_coef(y.reshape(-1, 1), x[:, idx_xs_incl], x[:, idx_xsc_incl], p

        # Calculate v(T)
        idx_xsc_excl = list(range(num_features))
        idx_xsc_excl.pop(i)
        idx_xsc_excl = list(set(idx_xsc_excl).difference(set(set_idx)))
        acc_excl = get_acc_coef(y.reshape(-1, 1), x[:, set_idx], x[:, idx_xsc_excl], prote

        marginal = acc_incl - acc_excl
        shapley = shapley + coef * marginal
    return shapley


def get_shapley_disc_i(y, x, protected_attr, i):
    """Returns Shapley coeffecient of ith feature in x."""

    num_features = x.shape[1]
    lst_idx = list(range(num_features))
    lst_idx.pop(i)
    power_set = [x for x in powerset(lst_idx) if len(x) > 0]

    shapley = 0
    for set_idx in power_set:
        coef = math.factorial(len(set_idx)) * math.factorial(num_features - len(set_idx) -

        # Calculate v_D(T U {i})
        idx_xs_incl = copy.copy(set_idx)
        idx_xs_incl.append(i)
        disc_incl = get_disc_coef(y.reshape(-1, 1), x[:, idx_xs_incl], protected_attr.resh

        # Calculate v_D(T)
        disc_excl = get_disc_coef(y.reshape(-1, 1), x[:, set_idx], protected_attr.reshape(
```

```
        marginal = disc_incl - disc_excl
        shapley = shapley + coef * marginal
    return shapley
```

In [23]:
```
# Calculate Shapley disc, acc coefs for each feature over training data
shap_acc = []
shap_disc = []
for i in range(5):
    acc_i = get_shapley_acc_i(y_train, x_train, race_train, i)
    disc_i = get_shapley_disc_i(y_train, x_train, race_train, i)

    shap_acc.append(acc_i)
    shap_disc.append(disc_i)

# Build Shapley output
feature_names = ["Age Categorical","Prior Count", "Gender", "Charge Degree", "Length of St
shapley_df = pd.DataFrame(list(zip(feature_names, shap_acc, shap_disc)),
                          columns=["Feature", "Shapley Accuracy", "Shapley Discrimination"
shapley_df = shapley_df.sort_values(by=["Shapley Discrimination"], ascending=[False]).res
#shapley_df.to_csv("../output/compas-data-shapley-table.csv")
```

In [24]:
```
pd.set_option('display.float_format', lambda x: '%.2E' % x)
print(shapley_df)
```

	Feature	Shapley Accuracy	Shapley Discrimination
0	Prior Count	1.26E+00	5.44E+04
1	Age Categorical	1.23E+00	5.38E+04
2	Length of Stay	1.09E+00	5.32E+04
3	Charge Degree	1.07E+00	4.36E+04
4	Gender	9.91E-01	4.29E+04

In [25]:
```
test_acc = []
test_cal = []

# Build model for overall data inclusive of all features
svm = SVC(kernel="linear").fit(x_train, y_train)
idx_race_1, idx_race_0  = np.where(race_test == 1)[0], np.where(race_test == 0)[0]
test_acc.append(svm.score(x_test, y_test))
test_cal.append(svm.score(x_test[idx_race_1], y_test[idx_race_1]) - svm.score(x_test[idx_r

# Eliminate one feature at a time build model
for id_feature in range(x_train.shape[1]):
    idxs = list(range(x_train.shape[1]))
    idxs.pop(id_feature)
    x_train_mod = x_train[:, idxs]
    x_test_mod = x_test[:, idxs]

    svm = SVC(kernel="linear").fit(x_train_mod, y_train)
    acc = svm.score(x_test_mod, y_test)
    cal = svm.score(x_test_mod[idx_race_1], y_test[idx_race_1]) - svm.score(x_test_mod[idx

    test_acc.append(acc)
    test_cal.append(cal)


index_names = ["None",  "Age Categorical", "Prior Count", "Gender", "Charge Degree", "Leng
test_acc = [x * 100 for x in test_acc]
test_cal = [x * 100 for x in test_cal]
results = pd.DataFrame(list(zip(index_names, test_acc, test_cal)),
                       columns=["Eliminating Feature", "Accuracy (%)", "Calibration (%)
```

In [26]:
```
pd.set_option('display.float_format', lambda x: '%.2f' % x)
print(results)
```

	Eliminating Feature	Accuracy (%)	Calibration (%)
0	None	65.77	2.47
1	Age Categorical	61.54	2.43
2	Prior Count	59.76	5.48
3	Gender	63.06	-1.57
4	Charge Degree	66.02	1.69
5	Length of Stay	65.85	2.33

In [ ]:
```

```