Open in Colab

Exploring the ProPublica COMPAS Analysis

In this notebook you'll get a chance to examine the data used in the ProPublica story yourself.

Disclaimer: Please don't over interpret what you find in the data. We know from our discussions that methodology is key to being able to properly interpret findings. Our goal here will be to reproduce results from the readings we did for last class.

First, we'll download and parse the data into a data frame.

In [0]:

```
import pandas as pd

!wget https://raw.githubusercontent.com/propublica/compas-analysis/master/compas-scores-two-years.csv
df = pd.read_csv('compas-scores-two-years.csv')
df
```

In the ProPublica dataset they only used data where the "days_b_screening_arrest" feature was in the range [-30, 30].

In [0]:

```
# filter these based on propublica analysis (not sure why this doesn't match https://fairmlbook.org/classification.html)
df = df[(df['days_b_screening_arrest'] <= 30) | (df['days_b_screening_arrest'] >= -30)]
```

Reproducing Calculations of False Positive Rate and Positive Predictive Value

From a statistical point of view (but not necessarily a social justice point of view) the debate between Propublica and NorthPointe boiled down to what is the rate way to measure bias in an algorithm. As you saw in the first part of the assignmnet, ProPublica used the evidence that the false positive rate differed between blacks and whites as evidence of bias. Northpointe argued used the fact that the positive predictive values across the two groups were the same as evidence *against* bias.

In Northpointe's report, they have a table which lists various statistics for blacks versus whites using the COMPAS risk scores as the predictor. Here is the relevant information from their report.

Race	$\hat{y} = 1$	True positive rate	False Positive Rate	Positive Predictive Value	Negative Predictive Value
white	*decile* ≥ 1	1.00	1.00	0.39	1.00
	decile ≥ 2	0.85	0.64	0.46	0.79
	decile ≥ 3	0.74	0.47	0.5	0.76
	decile ≥ 4	0.64	0.35	0.54	0.74
	decile ≥ 5	0.52	0.23	0.59	0.71
	decile ≥ 6	0.41	0.15	0.64	0.69
	decile ≥ 7	0.29	0.09	0.68	0.66
	decile ≥ 8	0.20	0.05	0.71	0.65
	decile ≥ 9	0.12	0.03	0.70	0.63
	decile ≥ 10	0.05	0.01	0.70	0.61
black	*decile* ≥ 1	1.00	1.00	0.51	1.00
	decile ≥ 2	0.95	0.83	0.55	0.77
	decile ≥ 3	0.89	0.68	0.58	0.73
	decile ≥ 4	0.81	0.56	0.60	0.69
	decile ≥ 5	0.72	0.45	0.63	0.65

Race	$\hat{y} = 1$	True positive rate	False Positive Rate	Positive Predictive Value	Negative Predictive Value
	$decile \geq 6$	0.63	0.34	0.66	0.62
	$decile \geq 7$	0.51	0.25	0.69	0.59
	$decile \geq 8$	0.39	0.16	0.72	0.57
	$decile \geq 9$	0.26	0.09	0.74	0.54
	$decile \geq 10$	0.12	0.03	0.79	0.51

Notebook Exercise 1

Write code to reproduce the table above. Here are some hints to help you.

- If you're fuzzy on what each of these statistic means (false positive rate, true positive rate, etc.), consider checking out Binary Diagnostic Tests.
- You'll want to use the column `df['two_year_recid']` as your indicator of true positive versus true negative (positive means that the person recidivated).
- To select narrow the data frame to just contain people of a particular race you can use the following snippet (`race` would be a string that is either "Caucasian" or "African-American").

```
df_for_race = df[df['race'] == race]
```

- To generate a particular row of the table, you'll want to loop over all possible thresholds where the model would predict recidivate ($\hat{y} = 1$).
- You can count the number of elements in a Pandas series that satisfy some criterion using the following technique. For instance, if we wanted to calculate the number of elements in "some_column" that are greater than 0 and less than 30, we could use the following code.

```
((df['some column'] > 0).sum() & (df['some column'] < 30).sum()
```

- It's up to you how you want to generate the table. You can simply print out the values within a loop as you compute them, or you could populate a data frame with your calculations and then plot them (this is what we did in the solution).

In [0]:

```
# ***Solution***

# we're going to create a data frame to hold all of the results.  Think of this
# as a represntation of the
results = pd.DataFrame(columns=['race', 'decile >=', 'true_positive_rate', 'false_positive_rate', '
positive_predictive_value', 'negative_predictive_value'])

for race in ['Caucasian', 'African-American']:
    df_for_race = df[df['race'] == race]
    y = df_for_race['two_year_recid']
    for thresh in range(1, 11):
        yhat = df_for_race['decile_score'] >= thresh
        true_positive_rate = ((y == 1) & (yhat == 1)).sum() / (y == 1).sum()
        false_positive_rate = ((y == 0) & (yhat == 1)).sum() / (y == 0).sum()
        if (yhat == 1).sum() == 0:
            positive_predictive_value = float('nan')
        else:
            positive_predictive_value = ((y == 1) & (yhat == 1)).sum() / (yhat == 1).sum()

        if (yhat == 0).sum() == 1:
            negative_predictive_value = float('nan')
        else:
            negative_predictive_value = ((y == 0) & (yhat == 0)).sum() / (yhat == 0).sum()

        results = results.append({'race': race,
                                  'decile >=': str(thresh),
                                  'true_positive_rate': true_positive_rate,
                                  'false_positive_rate': false_positive_rate,
                                  'positive_predictive_value': positive_predictive_value,
                                  'negative_predictive_value': negative_predictive_value}, ignore_i
dex=True)
results
```

In [0]:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
```

```
legend_strings = []
for race, df_by_race in results.groupby('race'):
    plt.plot(df_by_race['false_positive_rate'], df_by_race['true_positive_rate'])
    for _, row in df_by_race.iterrows():
        ax.annotate(str(row[1]), (row[3], row[2]))
    legend_strings.append(race)

plt.legend(legend_strings)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```

Notice how for a given threshold (annotated as text), the curves have vastly different false positive rates.

In [0]:

```
fig, ax = plt.subplots()
legend_strings = []
for race, df_by_race in results.groupby('race'):
    plt.plot(df_by_race['negative_predictive_value'], df_by_race['positive_predictive_value'])
    for _, row in df_by_race.iterrows():
        ax.annotate(str(row[1]), (row[5], row[4]))
    legend_strings.append(race)

plt.legend(legend_strings)
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.xlabel('negative predictive value')
plt.ylabel('positive predictive value')
plt.show()
```

Notice how for a threshold of 4 or 5, the positive predictive values are almost identical.

Sanity Check Using Sklearn

We can compare our calculations to the ROC curve (false positive rate versus true postive rate).

In [0]:

```
from sklearn import metrics

for race in ['African-American', 'Caucasian']:
    df_filtered = df[df['race'] == race]
    y = df_filtered['two_year_recid']
    scores = df_filtered['decile_score']
    fpr, tpr, thresholds = metrics.roc_curve(y, scores)
    plt.plot(fpr, tpr)
plt.legend(['Caucasian', 'African-American'])
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```