Problem Set 6: Correcting Bias in Classification

This problem set is addapted from the ML Failures lab: Correcting Bias by Nick Merrill, Inderpal Kaur, Samuel Greenberg, which is licensed under CC BY-NC-SA 4.0 (https://creativecommons.org/licenses/by-nc-sa/4.0)

Feedback

Students can provide feedback here (https://docs.google.com/forms/d/1jI8oXRkqD1I1ARuZR1y9W_qkOystPr-YEyywNDez46M/edit?ts=5efa772a&dods).

Background

The datasets we use to train machine learning models can often encode human biases. From a social and ethical standpoint, we want to remove or minimize this bias so that our models are not perpetuating harmful stereotypes or injustices. From a business and legal perspective, we want to produce effective models that adhere to industry standards of fairness.

There are several ways that we can tackle this problem, including pre-processing the data to remove bias before training, in-processing the model to change the way it learns from the data, and post-processing the results to correct for bias. In this assignment, we will be introducing two methods for correcting for bias: a post-processing method that uses alternative classification thresholds for different groups, and an in-processing method to train a logistic regression classifier that maximizes fairness while maintaining a certain level of accuracy. The in-processing method for correcting bias is based on Fairness Constraints: Mechanisms for Fair Classification (https://arxiv.org/pdf/1507.05259.pdf) by Zafar et al. (2017).

Agenda

- Introduction
- Part 1: Defining fairness
- Part 2: Observing a classifier's bias in our dataset
- Part 3: Post-processing with alternative thresholds
- Part 4: In-processing with fairness constraints
- Bonus question: Alternative fairness definitions

In [3]:

```
%%capture
import numpy as np
import pandas as pd
import generate_synthetic_data as generate
import classify_synthetic_data as classify
import matplotlib.pyplot as plt
%matplotlib inline
```

Introduction: Gender bias in hiring

Our dataset represents applicants for a job. We want an algorithm to help making hiring decisions, as this process is rife with human bias. However, studying past data would simply reinforce these biases (as we will see below).

Consider gender in hiring decisions. There is a well-documented gender pay gap (https://en.wikipedia.org/wiki/Gender_pay_gap): in general, people who identify as women are paid less for the same work than people who identify as men.

Now, imagine that we want to build a classifier that will make hiring decisions. If we train a classifier to make hiring decisions based on prior work experience and income, we would expect the classifier to *learn* a bias against women. Put another way, we can't make a classifier that is "gender-blind" by simply "throwing out" gender, excluding it from our dataset. Information about gender is already correlated with income.

We do not want our algorithm to learn this bias for gender. **Instead, we would like our algorithm to *correct* for this bias.**

To discuss this issue more deeply, let's introduce some key terms:

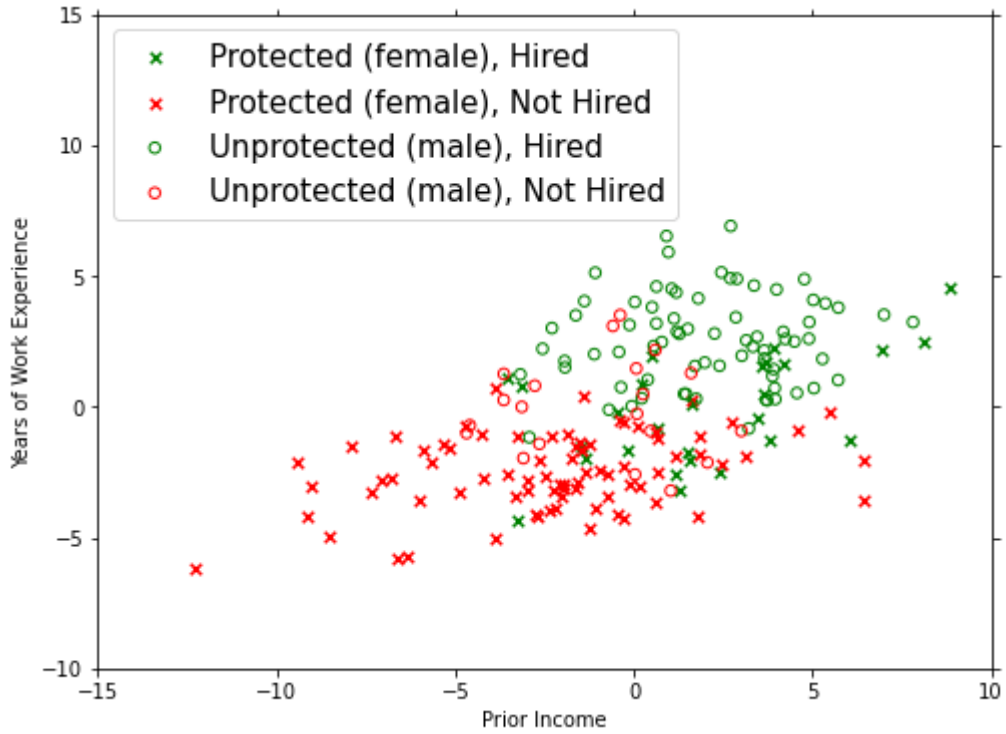
- **Sensitive** features: A feature is *sensitive* when it may contribute to bias. For example, gender is a *sensitive* feature.
- **Protected** classes: A class is *protected* when we expect there is bias against it. For example, being female is a *protected* class in this example.

Let's generate a synthetic dataset that contains the prior income, work experience, and gender of 1,000 applicants who were hired and 1,000 applicants who were not. (For a more detailed explanation of how we generate this toy dataset, see the Appendix below.)

We represent each individual on the plot below according to their prior income and work experience (the non-sensitive features). The shape of the marker indicates the applicant's gender (the *sensitive* feature). Females, denoted by `x` s, are the *protected* class in this example. The color indicates whether or not they were hired.

In [4]:

```
X, y, x_sensitive = generate.generate_synthetic_data(plot_data=True)
```



The variables here are `x` (non-sensitive feature: years of prior work experience), `y` (label: hired/not-hired), and `x_sensitive` (male/female).

Here, we have effectively created a biased dataset where the sensitive feature *gender* is strongly related to the applicant's hiring status. A female applicant (`x`) seems less likely than a male applicant (`o`) to be hired. (Again, this is just example data---while gender gap in pay is real, this particular data is synthetic. See the appendix for details.)

Part 1: Defining Fairness

In order to remove bias from our model, we first need to define bias.

In this assignment, we will be using **disparate impact** (also known as the **p%-rule**) to measure the unfairness of a data set with respect to a particular sensitive feature. We can measure *how biased* our dataset is by calculating the p%-rule.

1.1 Calculate the p% rule

If we take $\mathbf{x}$ to represent a (non-sensitive) feature vector, y a 1/-1 class label, and z a 0/1 sensitive attribute, the p%-rule states that the ratio between the fraction of subjects assigned the positive decision outcome ($y = 1$) given that they have a sensitive attribute value and the fraction of subjects assigned the positive outcome given that they do not have that value should be no less than p%.

In the scenario we described in this lab, the non-sensitive feature vector $\mathbf{x}$ represents prior income and years of work experience, the class label y represents the hiring status (1 for hired, -1 for not hired), and the sensitive attribute z represents gender (0 for female, 1 for male). The p%-rule would tell us the ratio between the fraction of female applicants who were hired and the fraction of male applicants who were hired.

$$p = 100 \left(\frac{Pr(y = 1|z = 1)}{Pr(y = 1|z = 0)} \right)$$

In [5]:

```
sensitive_features_arr  = np.array(x_sensitive['s1'])
```

In [6]:

```
# TODO: Write a function compute_p_rule that takes as inputs a sensitive_features_ar
# hiring outcomes), and returns the p percent rule. Also have your function print ou
# in the protected and unprotected class, and the percent of observations in each cl

def calculate_p_rule(sensitive_features_arr, y):
    # TODO: Replace 0 in the next line with a calculation of the # of unprotected ob
    number_unprotected = np.count_nonzero(sensitive_features_arr == 1)

    # TODO: Replace 0 in the next line with a calculation of the # of protected obse
    number_protected = np.count_nonzero(sensitive_features_arr == 0)

    # TODO: Replace 0 in the next line with a calculation of the % of unprotected ob
    hire_gender = np.column_stack((y, sensitive_features_arr))
    number_hired_unpro = len(hire_gender[np.where((hire_gender[:,0] == 1) & (hire_ge
    #print(number_hired_unpro)
    percent_pos_unprotected = (number_hired_unpro / number_unprotected) * 100

    # TODO: Replace 0 in the next line  with a calculation of the % of protected obs
    number_hired_pro = len(hire_gender[np.where((hire_gender[:,0] == 1) & (hire_genc
    #print(number_hired_pro)
    percent_pos_protected =  (number_hired_pro / number_protected) * 100

    # TODO: Replace 0 with a calucalation of the p percent rule, using the previousl
    p_percent_rule = (percent_pos_protected / percent_pos_unprotected) * 100

    print('Number of protected observations: %i' % number_protected)
    print('Number of unprotected observations: %i' % number_unprotected)
    print('Protected in positive class: %i' % round(percent_pos_protected) + '%')
    print('Unprotected in positive class: %i' % round(percent_pos_unprotected) + '%'
    print('P-rule: %i' % round(p_percent_rule) + '%')

    return p_percent_rule
```

Now test your function using the sensitive_features_arr and hiring outcomes (*y*) from our dataset. Verify that your function is correct: it should output p-percent rule of 29.45%. If you get a different number, go back and correct your code!

In [7]:

```
calculate_p_rule(sensitive_features_arr, y)
```

```
Number of protected observations: 1075
Number of unprotected observations: 925
Protected in positive class: 24%
Unprotected in positive class: 81%
P-rule: 29%
```

Out[7]:

```
29.45216169814266
```

QUESTION A: Interpret the output of this method to describe the disparate impact between male and female applicants in the training data.

Answer: p-rule = 29% means that if the hiring rate of male is 100%, only 29% of female is hired, indicating a very unequal disparate impact between male and female applicants.

QUESTION B: Is there evidence of bias in the training data?

Answer:

- The evidence of bias in the data can be observed from the scatter plot above, in which we can clearly imagine an transparent line that seperates the green plots and the red plots. Green plots represent the privileged group (male)that has more years of working experience, while red plots represent the unprivileged group that has less years of working experience in the data. Also although not as obvious as years of working experience, female have lower prior income compared with male (ex. only female have prior income lower than -5 in this dataset).
- According to the Four-fifths rule, if the selection rate for a certain group is less than 80% of rate for group with highest selection rate, there is adverse impact. p% = 29% violates the Four-fifths rule.

Part 2: Machine learning without correcting for bias

Now let's turn back to our dataset. Let's see what happens when we *don't* correct for bias and train a machine learning model on our data.

First, we pre-process the data. We add an intercept column and split the data into train and test sets.

In [8]:

```
intercept = np.ones(X.shape[0]).reshape(X.shape[0], 1)
X = np.concatenate((intercept, X), axis = 1)
pd.DataFrame(X, columns=["intercept", "income", "experience"]).head() # to help us v
```

Out[8]:

	intercept	income	experience
0	1.0	-1.738503	-1.977414
1	1.0	-4.908148	-3.267064
2	1.0	3.006536	-0.925507
3	1.0	0.779730	2.473575
4	1.0	1.193537	-2.595904

In [9]:

```
# split the data into training and test using a 70/30 split
train_fold_size = 0.7
X_train, y_train, x_sensitive_train, X_test, y_test,  x_sensitive_test =\
    classify.split_into_train_test(X, y, x_sensitive, train_fold_size)
```

To set our baseline, we'll train a standard logistic regression classifier on our data and see how it performs. (See appendix for more on how this classifier training works).

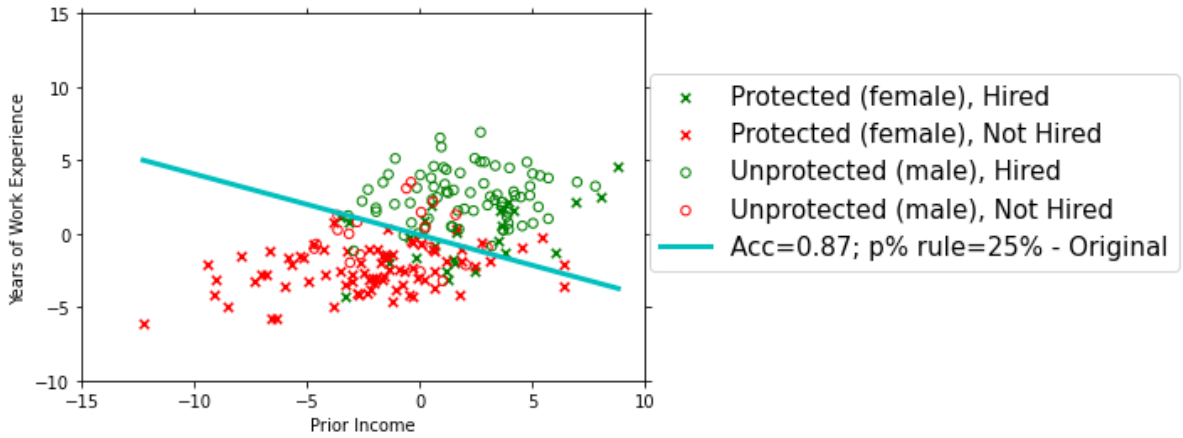
In [10]:

```
theta, p_rule, score, distances_from_decision_boundary = classify.train_test_classif
    X_train, y_train,  x_sensitive_train,
    X_test, y_test, x_sensitive_test,
    ['s1'],  # our list of sensitive features.
    apply_fairness_constraints=0 # We are NOT applying any fairness constraints this
)
```

```
Accuracy: 0.87
Protected/non-protected in positive class: 86% / 22%
P-rule achieved: 25%
Covariance between sensitive feature and decision from distance bounda
ry : 1.181
```

In [11]:

```
classify.plot_boundaries(X, y, np.array(x_sensitive['s1']), theta, p_rule[0], score)
```



The blue line represents the decision boundary for our standard logistic regression classifier. The model predicts points above this line as "hired" and points below as "not hired."

QUESTION A: Is our classifier accurate at predicting class labels (as determined by its accuracy rate)?

Answer:

- It is accurate at predicting class labels with a 0.87 accuracy. However, the bias in the data is neglected.

QUESTION B: Is our classifier fair (as determined by its p% rule)? Explain. Hint: female applicants would be labeled "hired" by this classifier at what rate relative to male applicants?

Answer: With a 25% p% rule, the classifier does not seems to be fair. The 25% p-rule means that female applicants would be labeled "hired" by this classifier at 25% relative to male applicants.

QUESTION C: How did the classifier learn bias for gender, even though gender was not included?

Answer:

- The bias of gender is composed by lower prior incomes and less years of working experience. Although gender is not included in traning this classifier, the classifier can still learn the gender bias by learning the

lower prior incomes and less years of working experience, which are the traits of the female group in this data.

Part 3: Post-processing with alternative thresholds

One option for correcting for this bias is postprocessing model outputs with a method called *thresholding*. This method makes no changes to the classifier itself, but instead processes classifier outputs differently for observations in the protected class and observations in the unprotected class. In the context of this problem set, we can think of thresholds as alternative restrictions on distance from the decision boundary: observations without the sensitive characteristic (males) must be above the decision boundary in order to be hired, but observations with the sensitive characteristic (females) can be up to a distance of *k* below the decision boundary.

3.1 Applying one alternative threshold

First, use the distances_from_decision_boundary obtained earlier from the train_test_classifier function to test out a threshold of 0 for males and a threshold of -1 for females. Identify which observations in the test set will be hired according to these thresholds, and use the compute_p_rule function you wrote earlier to calculate the p% rule.

In [12]:

```
sensitive_feature_arr = np.array(x_sensitive_test['s1'])
```

In [13]:

```
# TODO: Post-process the outputed distances_from_decision_boundary to apply a thresh
# protected observations (sensitive_feature = 1) and a threshold of -1 or over for u
# (sensitive_feature = 0). Calculate the p% rule using these post-processed outputs.
```

In [14]:

```
arr = np.column_stack((distances_from_decision_boundary, sensitive_feature_arr))
new_hired = list(map(lambda x,y : 1 if ((x >= -1 and y == 0) or (x >= 0 and y == 1))
```

In [15]:

```
calculate_p_rule(sensitive_feature_arr, new_hired)
```

```
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 30%
Unprotected in positive class: 86%
P-rule: 35%
```

Out[15]:

```
35.234308805318385
```

QUESTION A: Is this decision method more or less fair than the naive decision method with equal thresholding implemented earlier? Why?

Answer: Although the decision is still somehow unfair, it is more fair than the naive decision. Compared with the

naive decision (p% rule = 25%), the p% rule of this decision has increased to 35%, meaning that relative to male applicants, the rate of female applicants being labeled "hired" by the classifier has increased from 25% to 35%.

3.2 Varying the threshold

Now let's try out a set of different thresholds. Assume that we will leave the threshold for the unprotected class (males) at 0. Try out a set of thresholds for the protected class (females) between -8 and 0. Try out at least 20 thresholds in this range, and plot the results in a scatterplot, with the threshold on the x-axis and the p% rule on the y-axis. Calculate the threshold at which the p% rule is closest to 100%, and add a vertical line to the scatterplot showing this threshold. Remember to make sure your plot is readable and well-labeled

In [16]:

```
# TODO: Post-process the outputed distances_from_decision_boundary to apply a thresh
# observations and a threshold of k for unprotected observations (sensitive_feature
# p% rule. Experiment with at least 20 different values of k between -8 and 0.
```

In [17]:

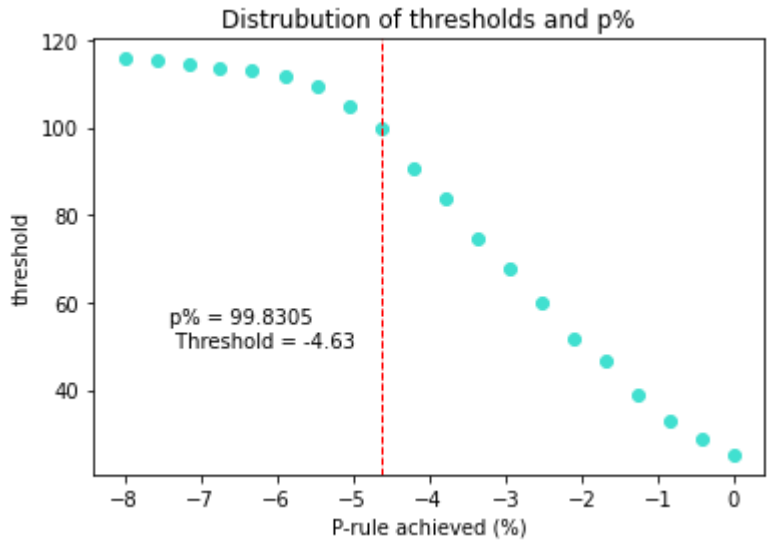
```
thresholds = np.round(np.linspace(0, -8, 20), 2)
p = {}

for i in thresholds:
    new_hired = list(map(lambda x,y : 1 if ((x >= i and y == 0) or (x >= 0 and y ==
    p.update({i: calculate_p_rule(sensitive_feature_arr, new_hired)})
```

```
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 22%
Unprotected in positive class: 86%
P-rule: 25%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 25%
Unprotected in positive class: 86%
P-rule: 29%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 28%
Unprotected in positive class: 86%
P-rule: 33%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 34%
Unprotected in positive class: 86%
P-rule: 39%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 40%
Unprotected in positive class: 86%
P-rule: 47%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 45%
Unprotected in positive class: 86%
P-rule: 52%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 52%
Unprotected in positive class: 86%
P-rule: 60%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 59%
Unprotected in positive class: 86%
P-rule: 68%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 64%
Unprotected in positive class: 86%
P-rule: 75%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 73%
Unprotected in positive class: 86%
P-rule: 84%
Number of protected observations: 335
```

```
Number of unprotected observations: 265
Protected in positive class: 79%
Unprotected in positive class: 86%
P-rule: 91%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 86%
Unprotected in positive class: 86%
P-rule: 100%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 91%
Unprotected in positive class: 86%
P-rule: 105%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 95%
Unprotected in positive class: 86%
P-rule: 110%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 96%
Unprotected in positive class: 86%
P-rule: 112%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 98%
Unprotected in positive class: 86%
P-rule: 113%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 98%
Unprotected in positive class: 86%
P-rule: 114%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 99%
Unprotected in positive class: 86%
P-rule: 114%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 100%
Unprotected in positive class: 86%
P-rule: 115%
Number of protected observations: 335
Number of unprotected observations: 265
Protected in positive class: 100%
Unprotected in positive class: 86%
P-rule: 116%
```

In [42]:

```
#scatter plot
for i in p.keys():
    plt.scatter(i, p[i], color="turquoise")

plt.axvline(-4.63, color='r', linestyle='dashed', linewidth=1) #add line
plt.text(-4.63*1.6, 50, 'p% = {:.4f} \n Threshold = -4.63'.format(p[-4.63])) #add av


plt.xlabel('P-rule achieved (%)')
plt.ylabel('threshold')
plt.title('Distrubution of thresholds and p%')
plt.show()
```



Part 4: In-processing with fairness constraints

An alternative method for correcting for bias is applying fairness constraints in training the machine learning algorithm.

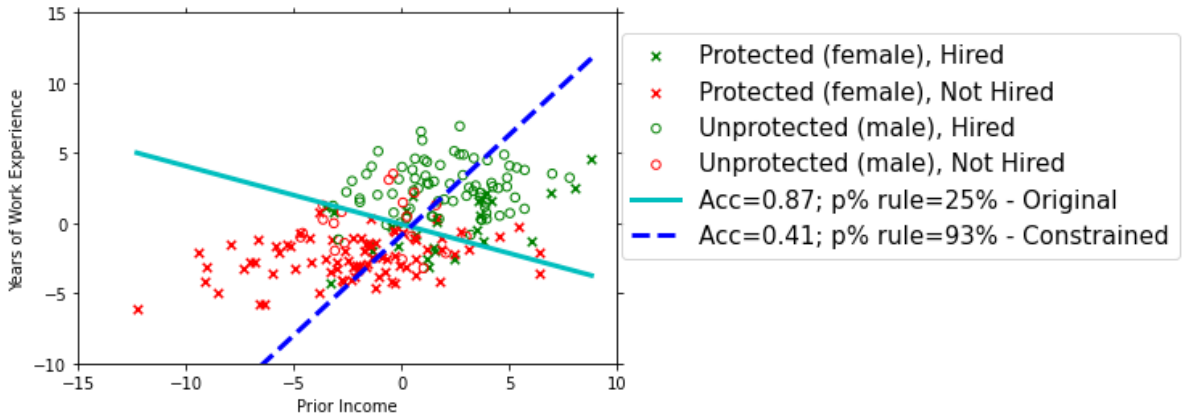
You can see the appendix for more information on how this process works (see Appendix: Applying fairness constraints. But it's best illustrated with an example.

Below, you're going to pick a **hyperparameter gamma** (γ). I've set a value of 1.5. See what happens.

In [19]:

```
theta1, p_rule1, score1, distances_from_decision_boundary = classify.train_test_clas
    X_train, y_train, x_sensitive_train,
    X_test, y_test, x_sensitive_test,
    ['s1'],  # our sensitive feature
    apply_accuracy_constraint=1, # applying our fairness constraint.
    gamma=1.5
    )

classify.plot_boundaries(X, y,
                         sensitive_features_arr,
                         theta,
                         p_rule[0],
                         score,
                         theta1,
                         p_rule1[0],
                         score1)
```

```
Accuracy: 0.41
Protected/non-protected in positive class: 60% / 56%
P-rule achieved: 93%
Covariance between sensitive feature and decision from distance bounda
ry : 0.002
```



The solid line represents the decision boundary from our naive classifier---the one that scored poorly on the p% rule. That's the same line we saw above.

The *dashed* line represents our new decision boundary: the one from our **constrained** model, to which we applied our gamma hyperparameter.

After applying our bias correction method, the decision boundary *rotates* to produce a more fair distribution of class labels with respect to the sensitive feature.

At a gamma of 1.5, this method achieves a 93% p-rule, *but* a less accurate classifier. That's our tradeoff: we're effectively saying the original class labels are biased, and therefore wrong. So we're consciously making this tradeoff.

4.1 Fairness-accuracy trade-off

Using this method, we can specify how much we are willing to let the accuracy change by choosing an appropriate γ parameter for the model. The larger γ is, the more loss we are willing to incur in our corrected model compared to the baseline model. Again, refer to Appendix: Applying fairness constraints to get a better understanding of how this works.

Let's visually explore the *fairness-accuracy trade-off* by varying the value of gamma. Test out at least 20 values of gamma between 0 and 5. For each value, train a constrained model and record the accuracy and p-rule. Then, produce three scatterplots: The first should show gamma on the x-axis and accuracy on the y-axis. The second should show gamma on the x-axis and the p-rule on the y-axis. The third should show the p-rule on the x-axis and accuracy on the y-axis. Remember to make sure the plots are easy to read and well-labeled.

In [20]:

```
# TODO: experiment with gamma values between 0 and 3. Create plots of gamma vs. p-ru
# and p-rule vs. accuracy.
```

In [21]:

```
Gamma = np.round(np.linspace(0, 5, 20), 2)
p_acc = {}

for i in Gamma:
    theta1, p_rule1, score1, distances_from_decision_boundary = classify.train_test_
        X_train, y_train, x_sensitive_train,
        X_test, y_test, x_sensitive_test,
        ['s1'],  # our sensitive feature
        apply_accuracy_constraint=1, # applying our fairness constraint.
        gamma=i
        )

    p_acc.update({i:[p_rule1[0], score1]})
```

```
Accuracy: 0.87
Protected/non-protected in positive class: 86% / 22%
P-rule achieved: 25%
Covariance between sensitive feature and decision from distance bounda
ry : 1.181

Accuracy: 0.88
Protected/non-protected in positive class: 86% / 21%
P-rule achieved: 25%
Covariance between sensitive feature and decision from distance bounda
ry : 0.485

Accuracy: 0.87
Protected/non-protected in positive class: 86% / 22%
P-rule achieved: 26%
Covariance between sensitive feature and decision from distance bounda
ry : 0.288

Accuracy: 0.83
Protected/non-protected in positive class: 83% / 30%
P-rule achieved: 36%
Covariance between sensitive feature and decision from distance bounda
ry : 0.165

Accuracy: 0.74
Protected/non-protected in positive class: 67% / 38%
P-rule achieved: 56%
Covariance between sensitive feature and decision from distance bounda
ry : 0.070

Accuracy: 0.59
Protected/non-protected in positive class: 87% / 79%
P-rule achieved: 91%
Covariance between sensitive feature and decision from distance bounda
ry : 0.000

Accuracy: 0.41
Protected/non-protected in positive class: 58% / 54%
P-rule achieved: 93%
Covariance between sensitive feature and decision from distance bounda
ry : 0.002

Accuracy: 0.41
Protected/non-protected in positive class: 57% / 54%
```

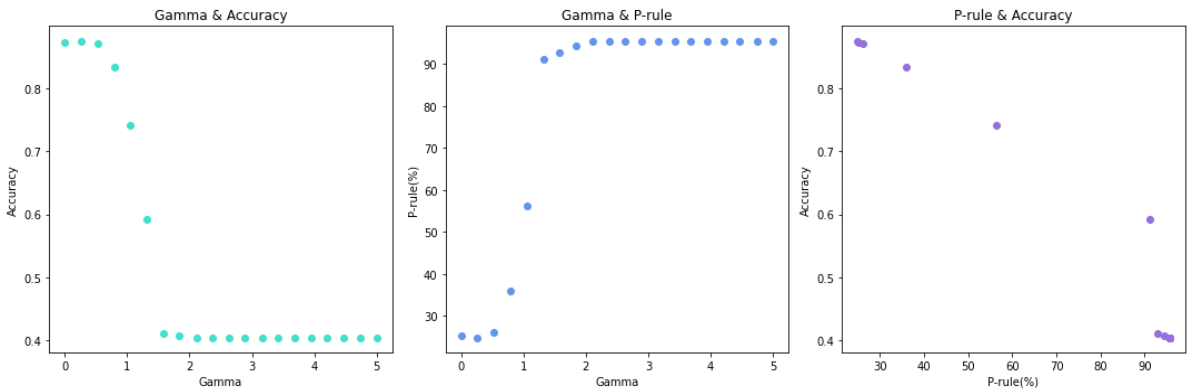
```
P-rule achieved: 94%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
```

```
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003

Accuracy: 0.41
Protected/non-protected in positive class: 56% / 54%
P-rule achieved: 96%
Covariance between sensitive feature and decision from distance bounda
ry : 0.003
```

In [104]:

```
print(p_acc)

fig, axs = plt.subplots(1, 3, figsize=(15,5))
# Gamma & acc
for i in p_acc.keys():
    axs[0].scatter(i, p_acc[i][1], color="turquoise")

axs[0].set_xlabel('Gamma')
axs[0].set_ylabel('Accuracy')
axs[0].set_title('Gamma & Accuracy')

#gamma & p-rule
for i in p_acc.keys():
    axs[1].scatter(i, p_acc[i][0], color="cornflowerblue")

axs[1].set_xlabel('Gamma')
axs[1].set_ylabel('P-rule(%)')
axs[1].set_title('Gamma & P-rule')

#gamma & p-rule
for i in p_acc.keys():
    axs[2].scatter(p_acc[i][0], p_acc[i][1], color="mediumpurple")

axs[2].set_xlabel('P-rule(%)')
axs[2].set_ylabel('Accuracy')
axs[2].set_title('P-rule & Accuracy')

fig.tight_layout()
plt.show()
```

```
{0.0: [25.216711203806298, 0.8733333333333333], 0.26: [24.871276803754
157, 0.875], 0.53: [26.021209740769834, 0.8716666666666667], 0.79: [3
5.95658073270013, 0.8333333333333334], 1.05: [56.31166202883886, 0.741
6666666666667], 1.32: [91.14211550940948, 0.5916666666666667], 1.58:
[92.88396726047183, 0.4116666666666667], 1.84: [94.29672827913413, 0.4
083333333333333], 2.11: [95.56245617549835, 0.405], 2.37: [95.56245617
549835, 0.405], 2.63: [95.56245617549835, 0.405], 2.89: [95.5624561754
9835, 0.405], 3.16: [95.56245617549835, 0.405], 3.42: [95.562456175498
35, 0.405], 3.68: [95.56245617549835, 0.405], 3.95: [95.5624561754983
5, 0.405], 4.21: [95.56245617549835, 0.405], 4.47: [95.56245617549835,
0.405], 4.74: [95.56245617549835, 0.405], 5.0: [95.56245617549835, 0.4
05]}
```



The following questions have no right answer. Instead, they are meant to make you think about the nuances of navigating correcting for bias in practice.

QUESTION A: What is the "right" value for γ? Why? Can you think of an empirical way to justify your choice for the "right" gamma value?

Answer:

- The choice of gamma should vary according to different senerios.
- In a hiring process, I think we should choose a gamma that sacrafice a bit accuray to get a higher p% rule, becuase gender inequality is already a well-known social problem in the hiring process.
- In this case, gamma = 1.05 gives us p = 56% & accuracy = 74%.

QUESTION B: What kinds of discussions or decision making processes could help us agree on γ?

Answer:

- Understanding how important the features (ex. prior working experience) are needed in each scenerio.
- For example, in this hiring case, we can try to discuss with the stakeholders about how important this role requires many years of working experience or a high prior income. If it turns out that some of these features are crucial, we can pick a gamma that provides higher accuracy. On the other hand, if either of the features are really important, we can choose a gamma that provides a higher p%.

QUESTION C: In the case of hiring, whom should these discussions involve? How about other cases? Remember that different stakeholders have different levels of understanding about machine learning, and different levels of understanding about social issues such as gender and racial bias.

Answer:

- Including stakeholders as diverse as possible, because most of the time people in the community can best understand their vulnerability. For example, in this case, including female stakeholdes who also belongs to the protected classes into discussion.
- Includeing the engineers who train the model or understand machine learning fairness to interpret the result of classifiers.

QUESTION D: Is there some point at which we should stop trying to correct for bias in our data? If so, how do we determine that point?

Answer:

1. When there seems to be no bias in our data should we stop correcting for bias. We should always check whether bias in data exist ex. use some statistical testings to check our data.

Bonus Question: Alternative Ways of Defining Fairness

In this assignment, we have used disparate impact -- as quantified by the p% rule -- to measure bias in our dataset and predictions. However, disparate impact is not the only way to quantify fairness. For this question, we'll look at another option for defining fairness: *disparate mistreatment*, introduced by Zafar et al. (2017b) (https://dl.acm.org/doi/10.1145/3038912.3052660). Disparate mistreatment focuses on the misclassification

rates between groups -- that is, it compares the *true positive rate* for the protected class to the *true positive rate* for the unprotected class. In the context of our dataset, disparate mistreatment compares the rate at which qualified females are hired in comaprison to the rate at which qualified men are hired.

Let $\hat{y}$ be the prediction of the classifier for an observation. Then disparate mistreatment is defined as:

$$m = 100 \left(\frac{Pr(\hat{y} = 1 | y = 1, z = 1)}{Pr(\hat{y} = 1 | y = 1, z = 0)} \right)$$

Write a function calculate_disparate_mistreatment that takes in a sensitive_features_arr and *y* (the true hiring labels), along with *distances_from_decision_boundary* as outputed by the *train_test_classifier* method, and returns *m*, the measure of disparate mistreatment. Also have your function print out the true positive rate for each of the classes.

In [100]:

```
# Code up disparate mistreatment
def calculate_disparate_mistreatment(sensitive_features_arr, y, distances_from_decis

    arr = np.column_stack((y, sensitive_features_arr, distances_from_decision_bounda

    y_hat = list(map(lambda x,y : 1 if ((x >= 0 and y == 0) or (x >= 0 and y == 1))
    arr = np.column_stack((arr, y_hat))

    # TODO: Replace 0 with the true positive rate in the unprotected class
    unpro_y = len(arr[np.where((arr[:,0] == 1) & (arr[:,1] == 1))])
    unpro_yhat = len(arr[np.where(((arr[:,3] == 1) & (arr[:,1] == 1)) & (arr[:,0] ==
    tpr_unprotected =  (unpro_yhat / unpro_y) * 100


    # TODO: Replace 0 with the the true positive rate in the protected class
    pro_y = len(arr[np.where((arr[:,0] == 1) & (arr[:,1] == 0))])
    pro_yhat = len(arr[np.where(((arr[:,3] == 1) & (arr[:,1] == 0)) & (arr[:,0] == 1
    tpr_protected =  (pro_yhat / pro_y) * 100

    # TODO: Replace 0 with your calculation of disparate mistreatment
    disparate_mistreatment = (tpr_protected / tpr_unprotected) * 100

    print('True positive rate for unprotected class: %i' % tpr_unprotected + '%')
    print('True positive rate for protected class: %i' % tpr_protected + '%')
    print('Disparate mistreatment: %i' % disparate_mistreatment + '%')

    return disparate_mistreatment
```

In [94]:

```
# Unconstrained classifier
theta, p_rule, score, distances_from_decision_boundary = classify.train_test_classif
    X_train, y_train,  x_sensitive_train,
    X_test, y_test, x_sensitive_test,
    ['s1'],  # our list of sensitive features.
    apply_fairness_constraints=0 # We are NOT applying any fairness constraints this
)
```

```
Accuracy: 0.87
Protected/non-protected in positive class: 86% / 22%
P-rule achieved: 25%
Covariance between sensitive feature and decision from distance bounda
ry : 1.181
```

In [101]:

```
# Test disparate mistreatment on the predictions of the unconstrained classifier
sensitive_features_arr = np.array(x_sensitive_test['s1'])
calculate_disparate_mistreatment(sensitive_features_arr, y_test, distances_from_deci
```

```
True positive rate for unprotected class: 97%
True positive rate for protected class: 63%
Disparate mistreatment: 65%
```

Out[101]:

```
65.58369167064818
```

In [102]:

```
# Constrained classifier
theta1, p_rule1, score1, distances_from_decision_boundary = classify.train_test_clas
    X_train, y_train, x_sensitive_train,
    X_test, y_test, x_sensitive_test,
    ['s1'],  # our sensitive feature
    apply_accuracy_constraint=1, # applying our fairness constraint.
    gamma=1.5
)
```

```
Accuracy: 0.41
Protected/non-protected in positive class: 60% / 56%
P-rule achieved: 93%
Covariance between sensitive feature and decision from distance bounda
ry : 0.002
```

In [103]:

```
# Test disparate mistreatment on the predictions of the constrained classifier
sensitive_features_arr = np.array(x_sensitive_test['s1'])
calculate_disparate_mistreatment(sensitive_features_arr, y_test, distances_from_deci
```

```
True positive rate for unprotected class: 58%
True positive rate for protected class: 26%
Disparate mistreatment: 44%
```

Out[103]:

```
44.940659340659344
```

QUESTION A: Did using a classifier that was constrained to reduce disparate impact improve disparate mistreatment?

Answer:

- No. The disparate impact of the naive classifer is 65%, with accuracy of male = 97% and female = 65%, while the disparate impact of the constrained classifer is 44%, with accuracy of male = 58% and female = 26%. ALthough the constrained classifer improve the p-rule %, it reduce the positive rate as well as the disparate mistreatment.

QUESTION B: In general, what is the relationship between disparate impact and disparate mistreatment? Why? *Hint*: A useful resource for this question is Barocas et al. (2019) (https://fairmlbook.org), Chapter 2. Barocas et al. use different terms for the fairness metrics we study here: disparate impact is referred to as *independence* and disparate mistreatment is referred to as *separation*.

Answer: In general, independence and sufficiency are mutually exclusive, the underlying assuptions are:

1. the sensitive features and the target variable are not independent
2. the target variable y is binary
3. y hat is not independent of the target variable y

QUESTION C: Tell a story where reducing disparate impact may unintentially increase disparate mistreatment. This story can be in the context of this problem set, but it doesn't have to be.

Answer:

QUESTION D: How might we decide which fairness metric is employed to test for bias in a classifier? Alternatively, how might we balance multiple fairness criteria? What kinds of decision mamking processes or tools could be employed, and who should be involved in these discussions?

Answer: Replace with your answer

Conclusion

Even if we exclude sensitive attributes from our training procedure, models trained on biased data with strong correlations between sensitive and non-sensitive attributes can still replicate or even emphasize that bias in

their predictions.

We can correct for that bias, but doing so will *always* incur tradeoffs in accuracy. In a way, our bias correction is a claim that the data are "misclassified" in the first place. In this lab's example, we think there are too few women labeled as "hired." So we're finding a compromise between correcting that bias and fitting the data we already have.

Based on a paper by Zafar et. al, we demonstrated a bias correction method we can control by setting the hyperparameter γ. But remember: different industries often have different standards for how fair is "fair enough" or how accurate is "accurate enough" for a model to be considered acceptable for use.

Appendix

Generating the toy dataset

To generate the non-sensitive features, we assign each class a bivariate Gaussian distribution and draw the 2 non-sensitive features for each data point from the appropriate distribution. To generate a biased sensitive feature correlated with the non-sensitive features, we rotate the non-sensitive features; the closer the rotation angle is to 0, the more correlated our sensitive feature will be to the non-sensitive features. We use these rotated coordinates to generate a Bernoulli distribution from which to draw the binary sensitive feature. We set the parameters of this distribution such that an individual in the dataset is more likely to be in the unprotected sensitive group (sensitive feature = 1) if the rotated non-sensitive features are also more likely to belong to the positive class (class label = 1).

Applying fairness constraints

To do so, we first define the *decision boundary covariance* as a measure of the covariance between the sensitive feature $\mathbf{z}$ and the classifier's decision $d_\theta(\mathbf{x})$. For a logistic regression classifier, the classifier's decision is based on the dot product of the parameters θ that the model learns and the feature values $\mathbf{x}$.

$$Cov(\mathbf{z}, d_\theta(\mathbf{x})) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{z}_i - \bar{\mathbf{z}})\theta^T\mathbf{x}_i$$

Now, the whole idea here is that we *don't* want the sensitive feature to influence our classifier's decision. Put another way, we want to demonstrate that the sensitive feature and the classifier's decision boundary don't covary---or, at least that the decision boundary covariance is close to 0. Now, we could adjust the classifier to minimize the decision boundary covariance, but this might lead to a significant loss in accuracy which could make our model unusable in practice.

Instead, we want to make our decisions as fairly as possible, with some limit on how low the accuracy can be to produce a functional classifier. So we can set up a mathematical way to frame the trade-off between accuracy and fairness: we can minimize the decision boundary covariance (maximize fairness) given some upper bound on the model's loss (an accuracy constraint).

If $Loss(\theta^*)$ represents the loss of our baseline classifier without any bias correction, we can set the parameter γ to specify how much additional loss we are willing to add to the baseline loss. So our bias correction method can be written:

$$\text{find } \theta \text{ that minimizes } \quad \left|Cov(\mathbf{z}, d_\theta(\mathbf{x}))\right|$$

$$\text{subject to } \quad Loss(\theta) \leq (1+\gamma)Loss(\theta^*)$$

We refer to this constraint as the hyperparameter gamma (γ).

In [ ]: