Fairness in AI Systems

How to Detect and Mitigate Bias in Machine Learning Models

Autor: Ramiz Abusabbah

www.consileon.ai

Information Deficits and Fairness

Some say our choices shape our destiny. And we choose all the time: hardly a few seconds go by without us making some decision. You are reading this article because you have decided to do so and keep deciding to go on. Some decisions entail severe risks or grave consequences. In predictive policing, for example, decisions are made on how to deal with convicts considered prone to recidivism. Banks classify loan applicants as either good or bad risks. Employers need to decide which candidate to hire.

The higher the stakes, the more difficult the decision, especially when predictive information is sparse. And the harder information is to come by, the more we tend to rely on prejudice. That is why we tend to favour members of a group we identify with, or traits we seem to share. When such bias leads your staff into making discriminatory decisions, however, your organization may run into ethical or legal issues. Good business practice hence involves decision-making that is not only economically sound but also fair. Fairness, to that end, is the absence of any degree of prejudice or favouritism towards any individual or a group based on any trait or other inherent, acquired or merely supposed quality. So, when it comes to making high-stakes decisions, wouldn‘t it be better to leave it to a computer?

AI-Driven Decision-Making

Across industries, more and more organizations are looking to artificial intelligence (AI) to help them overcome human limitations. Unlike people, machines get neither bored nor tired. Moreover, they can take far more dimensions and factors into account than humans. Supervised machine learning (ML) algorithms are hence increasingly used for making high-stakes, multifactor decisions. Supervised ML models predict the outcome of a new instance by analysing training data that comprises many historical instances and their outcomes. In other words, they generalize from parallels and patterns they dig up in the training set to produce a well-founded guess on how the new instance is likely to turn out.

However, what seems to be the greatest virtue of such models can be a drawback too: they do their maths stoically, with notorious indifference to any bias that the training data may or may not contain. If your training data reflects social or ideological prejudice, your ML system will reproduce that bias, thus possibly yielding undesirable or even illegal results such as discrimination against protected demographic groups. Employers, for example, are legally required to ignore an applicant‘s gender in their hiring decision even if that criterion has proven statistically relevant.

In addition to reproducing ubiquitous prejudice, there is the risk of generating bias by over- or undersampling demographic groups. This is referred to as sampling bias. It creeps in whenever the composition of the training dataset differs from real-world distribution.

Now that we know how ML models become biased, we need to answer three more questions before we start developing a fair model: how do we define fairness, how do we measure it and when would we be justified in claiming that a model is fair?

Fairness Defined

To make fair, non-discriminatory decisions we need to define the notion of fairness. Philosophers and psychologists have tried to do that long before computer science. But they are still far from unanimous about it. After finding twenty-one definitions of fairness, we concluded our literature search. In all that variety, however, we found two general approaches to the idea: individual versus group fairness.

Group fairness versus individual fairness

individual fairness

- *Treat similar cases similarly*
- *Use distance metric as similarity measure*
- *Ignore group membership in decision-making*

group fairness

- *Split population in two groups (privileged, unprivileged) based on protected attributes such as age, gender or disability*
- *Apply a statistical measure that results in similar outcomes across groups*

Individual fairness means treating similar individuals similarly regardless of their relation with any group. Similarity, to that end, is measured by distance metrics such as Euclidean or Manhattan distance. Individuals who are close to one another as defined by those metrics must have similar traits. Group fairness, by contrast, refers to treating all demographic groups equally. To detect illegal or undesirable discrimination, the population is split into two subsets based on one or multiple sensitive attributes. One group possesses values for these attributes which afford its members preferential treatment, and the other lacks those attribute values. The first group is referred to as privileged while the second as unprivileged or ***protected*** depending on the context. The sensitive attributes are also called ***protected attributes*** because it is illegal or undesirable to discriminate based on their values. Common metrics and formulae for quantifying group fairness in machine learning models are listed below.

Statistical parity difference

$$\text{statistical parity difference} = \text{favourable outcome rate}_{\text{unprivileged}} - \text{favourable outcome rate}_{\text{privileged}}$$

Disparate impact

$$\text{disparate impact} = \frac{\text{favourable outcome rate}_{\text{unprivileged}}}{\text{favourable outcome rate}_{\text{privileged}}}$$

Equal opportunity difference

$$\text{equal opportunity difference} = \text{true positive rate}_{\text{unprivileged}} - \text{true positive rate}_{\text{privileged}}$$

Average odds difference

$$\begin{aligned}&\text{average odds difference}\\ &= \frac{(\text{true positive rate}+\text{false positive rate})_{\text{unprivileged}} - (\text{true positive rate}+\text{false positive rate})_{\text{privileged}}}{2}\end{aligned}$$

True negative rate parity

$$\text{true negative rate parity} = \text{true negative rate}_{\text{unprivileged}} - \text{true negative rate}_{\text{privileged}}$$

Another inequality metric is the Theil index, which quantifies the divergence of the current distribution of resources within and among demographic groups.

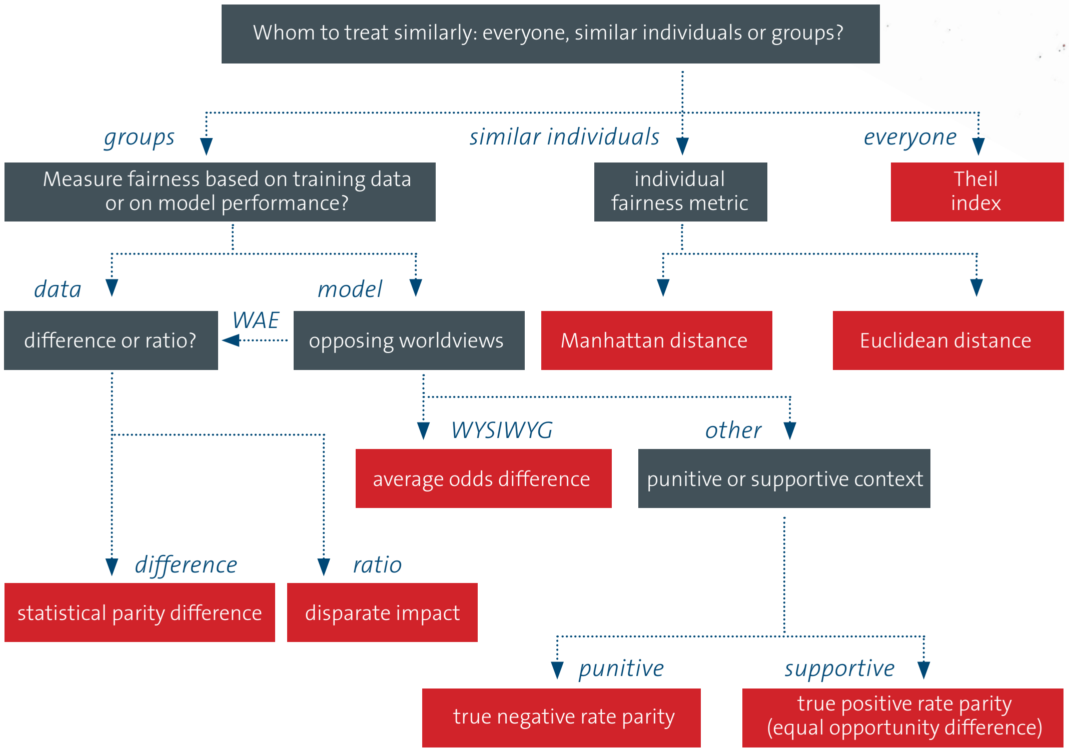
When to Use Fairness Metrics

The above metrics help us find bias either when we review our training data before building the ML model or when we evaluate the model's performance after completing the training. In the first case, bias is identified with respect to the real outcome values in the training dataset. In the second case, we assess the legal or ethical acceptability of the trained model's predictions. The following chart shows you when to apply which metric.

<table>
<tr><th></th><th>data</th><th colspan="2">model performance</th></tr>
<tr><td rowspan="5">group-fairness metrics</td><td colspan="3">statistical parity difference</td></tr>
<tr><td colspan="3">disparate impact</td></tr>
<tr><td></td><td>equal opportunity difference</td><td>true negative rate parity</td></tr>
<tr><td></td><td colspan="2">average odds difference</td></tr>
<tr><td colspan="3">Theil index</td></tr>
<tr><td rowspan="2">individual-fairness metrics</td><td colspan="3">Manhattan distance</td></tr>
<tr><td colspan="3">Euclidean distance</td></tr>
</table>

The choice of the metric depends on the context in which fairness is to be checked. Suppose we are evaluating the bias in the model's predictions. While the Theil index shows us whether resource distribution among demographic groups and their members is even, the statistical parity difference reveals how likely each group is to achieve a favourable outcome regardless of whether or to what extent group members qualify for that achievement. In other words, individuals pertaining to the same group are considered equally eligible to attain a positive outcome. This view of fairness is called ***we're all equa***l (WAE).

When we need to factor in that only qualified group members are likely to reach a favourable outcome, we can measure the true positive rate parity, also referred to as equal opportunity difference. Another concept is ***what you see is what you get,*** better known by its acronym WYSIWYG. It helps us balance the rate of positive outcomes of qualified to unqualified individuals across groups. In such cases, the average odds difference allows us to offset qualified individuals from one group with unqualified individuals from another. For guidance on choosing a fairness metric that fits your context, see the following decision tree.


Whom to treat similarly: everyone, similar individuals or groups?
groups
similar individuals
everyone
Measure fairness based on training data or on model performance?
individual fairness metric
Theil index
data
model
difference or ratio?
WAE
opposing worldviews
Manhattan distance
Euclidean distance
WYSIWYG
other
average odds difference
punitive or supportive context
difference
ratio
statistical parity difference
disparate impact
punitive
supportive
true negative rate parity
true positive rate parity (equal opportunity difference)

Bias Mitigation Algorithms

Bias mitigation algorithms can be applied in any phase of the ML workflow:

- **Preprocessing algorithms.** In our context, preprocessing means taking place before model training. Preprocessing algorithms transform the original data to remove implicit discrimination.

- **Inprocessing algorithms** are applied while training the model. They change or modify state-of-the-art learning algorithms in order to remove discrimination. This is done by imposing changes to the objective function or incorporating constraints.

- **Postprocessing algorithms** are applied after the training. They revise the labels assigned to the data by the trained model. These techniques are applied when the model can only be seen as a black box without any ability to modify the training data or the learning algorithm.

Literature abounds with mitigation algorithms. The following table subsumes popular specimens under the three classes introduced above.

preprocessing	inprocessing	postprocessing
reweighing	prejudice remover	reject-option calssification
optimized preprocessing	meta-fair classifier	equalized odds postprocessing
disparate impact remover		

In the demo presented later, the reweighing algorithm is applied to mitigate bias in the dataset. But before we come to that, let us explain the idea behind that algorithm and see how it deals with biased data. The following section includes technical content. This can be skipped as desired by the reader.

Reweighing

The reweighing algorithm enables a model to learn a classifier that excludes protected attributes from determining its predictions. It balances biased training data by assigning weights to samples without changing label or feature values. This is achieved by neutralizing the label's statistical dependence on the protected attribute. The balanced or transformed dataset is generated by sampling the original set and adjusting the weights of the samples. After that, the model learns the classifier from the balanced/transformed data.

Reweighing is based on, but differs slightly from, the idea of under- and oversampling. While undersampling balances class distribution by assigning weights of zero to problematic samples, thus effectively removing them from the dataset, reweighing keeps all samples, assigning weights above zero in a way that equalizes the distribution of labels across groups. Unlike oversampling, which assigns positive integer weights only, the reweighing algorithm uses real numbers, which qualifies it as an instance of cost-sensitive learning. The weights enter into the learning algorithm. A sample's weight quantifies the extent to which the algorithm will be "punished" when it fails to predict that sample correctly. Errors in the prediction of higher-weighted samples are thus more expensive than errors on samples with lower weights.

To understand how the algorithm calculates weights for each sample, let us assume the following parameters:

- a **binary protected attribute** such as **gender (G) = {f, m},** where **f** is the unprivileged group
- a **label** such as **class (C) = {+,-}**, where **+** is the favourable outcome
- **N:** total number of training samples
- **N_{conditon}:** number of training samples that meet the specified condition

If the training set is unbiased, the class label should be statistically independent of the protected attribute. The probability that an individual is female (f) and labelled as positive (+) should be:

$$\Pr_{exp}(G = f \wedge C = +) = \Pr(G = f) \times \Pr(C = +) = \frac{N_{G=f}}{N} \times \frac{N_{C=+}}{N}$$

However, the actual probability we observe is:

$$\Pr_{act}(G = f \wedge C = +) = \frac{N_{G=f \wedge C=+}}{N}$$

If the actual probability turns out lower than expected, women are less likely to obtain the positive outcome. To counter that disadvantage, we assign the following weight to female individuals with positive labels:

$$W(G = f \wedge C = +) = \frac{\mathrm{Pr}_{exp}(G = f \wedge C = +)}{\mathrm{Pr}_{act}(G = f \wedge C = +)}$$

By analogy, we divide expected by actual probability to assign weights to female individuals with negative labels as well as to male individuals with either positive or negative labels.

Demo: Bias Identification and Mitigation

In this section, we demonstrate how to mitigate bias in machine learning models. We walk you through the process step by step and provide the corresponding Python code.

Based on the Strategeion Résumé Skills dataset, we examine a hiring process for bias or discrimination against a protected demographic group. Strategeion Résumé Skills is a synthetic dataset of skills extracted from fictional CVs published on the web for AI specialists to evaluate the fairness of decisions made by ML models. It comprises 1968 records, each representing one fictional job applicant. Each record features 222 binary attributes that serve as input for training the model. They fall into two categories:

- 218 attributes refer to **skills**. They indicate whether or not a CV includes a skill in question. The skills were drawn from popular entries in LinkedIn such as ***automotive, branding*** or ***data analysis.***

- The dataset also contains four **protected demographic attributes** which we will be using to detect bias:

  veteran: 1 if the applicant is a veteran, else 0
  female: 1 if the applicant is a woman, else 0
  URM: 1 if the applicant belongs to an underrepresented minority, else 0
  disability: 1 if the applicant has a disability, else 0

Records, too, are marked with a binary label which serves as an output target in the supervised training of our ML model. The label is stored as an attribute named ***interview,*** with 1 indicating that the applicant has been invited, else 0.

In the demo below, we examine a hiring process for bias based on the ***female*** protected attribute. To that end, we split the applicants' data in two subsets, a privileged one comprising men (attribute value ***female = 0***), and a protected group of women (attribute value ***female = 1***). To amplify the bias reflected in the dataset, we manipulate the label values to the effect that male candidates receive more interview requests.

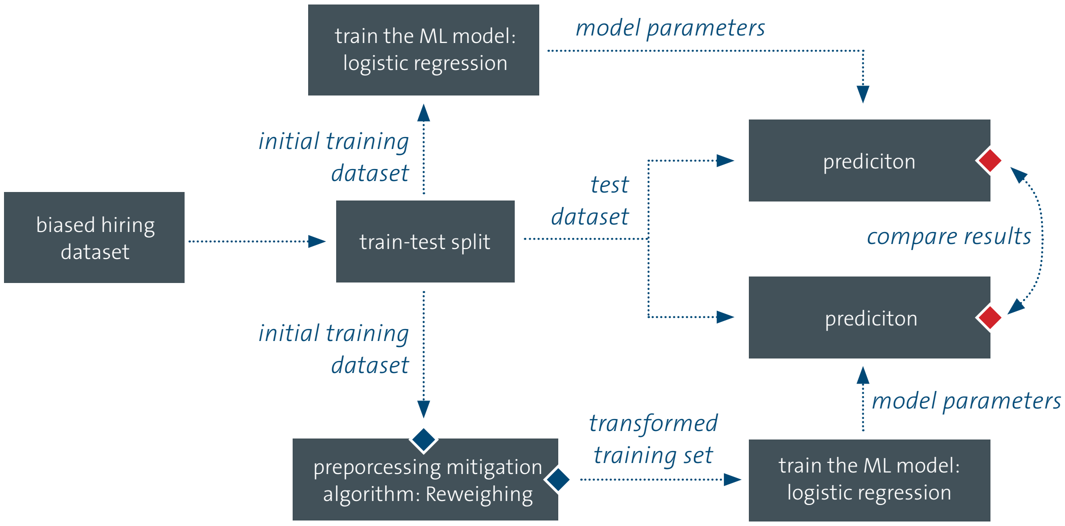
Before we scan the hiring data for discrimination, we need to clarify which outcome we would consider fair. So for the purpose of our demo, let's define that:

The model is ideally fair only if the same percentage of male and female job applicants receives an interview request.

Group fairness metrics such as statistical parity difference or disparate impact would meet that definition. Both quantify the discrepancy in favourable outcomes between protected and privileged demographic groups.

We start by splitting the original hiring data into a training and a test set. By running the reweighing algorithm on a copy of the training set, we generate a third, unbiased dataset. For comparison, we develop two versions of the ML model, both based on logistic regression. One version is trained on the original biased dataset to serve as benchmark for the other, which learns from the bias-free dataset generated by the reweighing algorithm. After training, we compare the performance of the two models regarding discrimination. This is done by computing and comparing the fairness metrics with respect to the predictions of both models. The chart below illustrates the entire bias detection and mitigation process.

Bias detection and mitigation process in our fairness demo



🔷 *Compute fairness metrics on data*
🟥 *Compute fairness metrics on model performance*

In the following steps, we explain how we built our demo. Before starting our analysis, we need to import the tools (step 1) and dependencies (required software components, step 2) we will be using.

Step 1: import and install AIF360

In this demo, we use the AI Fairness 360 toolkit, an open source library developed by researchers at IBM. AIF360 provides a comprehensive set of metrics and algorithms to test for and mitigate bias. For the demo to succeed, please import and install the toolkit within the environment you are working in.

```
pip install ‚aif360[all]' –user

pip install ‚aif360[LFR,OptimPreproc]' #Some algorithms require further
dependencies.
```

Step 2: import dependencies

As in any Python program, we need to import the dependencies we will be working with such as the ***NumPy*** library for handling arrays and mathematical operations as well as the ***pandas*** library for data analysis and manipulation. From scikit-learn, we import the logistic regression algorithm for training a classifier on our dataset. From AIF360, we import various metrics to check for bias, and classes relevant to our bias mitigation algorithm.

```
import NumPy as np

import pandas as pd


np.random.seed(0)


#Import logistic regression to develop machine learning model.

from sklearn.linear_model import LogisticRegression


#Import BinaryLabelDatasetMetric class from toolkit. It includes fairness

metrics needed to detect bias.

from aif360.metrics import BinaryLabelDatasetMetric


#From the toolkit, import reweighing mitigation algorithm.

from aif360.algorithms.preprocessing import reweighing


#Import Markdown for textual explanation of steps.

from IPython.display import Markdown, display
```

Step 3: import and read original data

After importing the libraries, we load our original hiring dataset. Like AIF360, the dataset must be available in the code's runtime environment. In our case, the dataset is a CSV (comma separated values) file. To read the file into a DataFrame and access its content, we insert its path in ***pandas'*** `read_csv` built in function.

```
df=pd.read_csv(„BiasedDatasetHiring.csv“, sep=“;“)
```

Step 4: exploratory data analysis

Before analysing any dataset for real-world insight, it pays to survey its structure and content, and gain some meta-information so we know what to look for later on. This preparatory step is called data exploration or exploratory data analysis. To that end, we can use the following functions.

- ***info():*** ***pandas*** function that summarizes DataFrames
- ***isnull():*** function that helps us detect missing values
- ***head():*** ***pandas*** function that returns the first five observations from the data

```
#Explore data.
display(Markdown('###Show dataset properties'))
print(df.info())
display(Markdown('###Are there any values missing?'))
print(df.isnull().sum())
display(Markdown('###Show first five records'))
df.head()
```

Output

```
Show dataset properties
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1986 entries, 0 to 1985
columns: 224 entries, column1 to disability
dtypes: int64(224)
memory usage: 3.4 MB
none
Are there any values missing?
column1          0
interview        0
Adobe Illustrator    0
Adobe Photoshop  0
agile methods    0
                 ..
writing          0
veteran          0
female           0
URM              0
disability       0
length: 224, dtype: int64
```

Step 5: preprocessing

Our data exploration has shown that the first column of the set merely serves to number the samples. Since it does not contain any factual information and is irrelevant to the model's training, we eliminate it with the del command. As all attributes are binary {0,1}, preprocessing comprises just this step.

```
del df["column1"]
```

Step 6: convert dataset into a standard format as required by AIF360 toolkit

To apply the methods and metrics from AIF360, we must convert our data into a compliant format. We initialize the dataset within a Python class called RecruitingDataset, which inherits all attributes and functions of superior classes from the toolkit such as StandardDataset, BinaryLabelDataset, StructuredDataset, Dataset. To initialize the set, we need to feed the ***__init__*** method with the desired specifications.

- ***label_name:*** name of the dataset's label column. In the original hiring dataset, the ***interview*** attribute indicates whether or not a job applicant has been invited. This is going to be our label attribute.

- ***favourable_classes:*** label values considered favourable. Applicants labelled with ***interview = 1*** have received an interview request.

- ***protected_attribute_names:*** a list of the protected attribute names in the dataset. In our case, they comprise four demographic categories: veteran, female, URM, disability.

- ***privileged_classes:*** a list of values which are considered privileged with respect to the protected attributes. Privileged classes refer to men (female = 0) who are neither veterans (veteran = 0) nor members of an underrepresented minority (URM = 0) nor disabled (disability = 0).

- ***instance_weights_name:*** name of the dataset's instance weight column. Since our hiring dataset does not include any instance weights, we set this argument *to* ***none***.

- ***categorical_features:*** a list of the names of categorical features in the dataset. AIF360 provides a method that uses one-hot encoding to transform categorical into numerical features. Since categorical features are inexistent in the hiring dataset, we leave that field blank.

- ***features_to_keep:*** names of columns to keep. All others are dropped except those in *protected_attribute_names, categorical_features, label_name or* ***instance_weights_name.*** As we will be using all skill attributes from the original dataset to check the outcomes of our model, we leave this argument blank.

- ***features_to_drop:*** names of columns to drop. This argument overrides *features_to_keep.*

- ***na_values:*** a list of strings to treat as missing values. In our demo, we leave this optional argument blank.

- ***custom_preprocessing:*** additional preprocessing specified by the user. Not needed for our demo.

- ***metadata:*** an argument for appending metadata.

To call the superclasses of RecruitingDataset, we use ***super().***

```
from aif360.datasets import StandardDataset
import os
class RecruitingDataset(StandardDataset):
def __init__(self, label_name='interview', favourable_classes=[1],\
    protected_attribute_names=['veteran','female','URM','disability'],\
    privileged_classes=[[0],[0],[0],[0]],\
    instance_weights_name=none,\
    categorical_features=[],\
    features_to_keep=[], features_to_drop=[],\
    na_values=[], custom_preprocessing=none,\
    metadata=none):

    super(RecruitingDataset, self).__init__(df=df, label_name=label_name,
      favourable_classes=favourable_classes,
      protected_attribute_names=protected_attribute_names,
      privileged_classes=privileged_classes,
      instance_weights_name=instance_weights_name,
      categorical_features=categorical_features,
      features_to_keep=features_to_keep,
      features_to_drop=features_to_drop, na_values=na_values,
      custom_preprocessing=custom_preprocessing, metadata=metadata)
```

Step 7: specify dataset settings and perform a train-test-split

In the demo, we split the dataset into two groups with respect to the protected attribute ***female*** and then check for bias by comparing the outcomes of both groups in the training dataset as well as later in the predictions. To do so, we first initiate an instance of class RecruitingDataset. All protected attributes except for ***female*** are dropped as they are not needed in the observed context.

After the initialization of the dataset, a training-test split is performed. We shuffle the original samples to assign them randomly to the training and test sets, then split them at a 70:30 ratio, using the larger share for training and the rest for testing the model's accuracy. Finally, we set two variables for the privileged (0) and unprivileged (1) values of the ***female*** attribute. These are key inputs for detecting and mitigating bias in step 8 and step 9.

```
#dataset settings

dataset_orig = RecruitingDataset (protected_attribute_names=['female'],
privileged_classes=[[0]],\

              features_to_drop=['veteran','URM','disability'])

#train–test split

dataset_orig_train, dataset_orig_test = dataset_orig.split([0.7], shuffle=true)

#privileged & unprivileged groups

privileged_groups = [{'female': 0}]

unprivileged_groups = [{'female': 1}]
```

If we were to check a model for bias against a combination of protected attributes such as `protected_attribute_names=['female','disability']`, we would split the dataset into a privileged and an unprivileged group based on the `privileged_classes` argument. Individuals privileged on both protected attributes would form the privileged group, while all others would belong to the protected group.

Step 8: compute fairness metrics on original training data

After defining `female` as protected attribute with a privileged and an unprivileged value, we use AIF360 to spot bias in the hiring data by computing the statistical parity difference. We arrive at this metric by subtracting the percentage of favourable results of the privileged group from that of its unprivileged complement. A negative difference indicates that the unprivileged group is less likely to achieve a favourable outcome than the complement. To double-check, we compute the disparate impact as well. This metric expresses the same information not as a difference however, but as a ratio. A value below one indicates that the unprivileged group is at a disadvantage. To compute the two metrics, we apply the `statistical_parity_difference()` and `disparate_impact()` methods from the `BinaryLabelDatasetMetric` class. The code below triggers those calculations and displays the output, showing a statistical parity difference of -0.243882 and a disparate impact of 0.391562.

```
metric_orig_train = BinaryLabelDatasetMetric(dataset_orig_train, unprivileged_
groups=unprivileged_groups,\

                                          privileged_groups=privileged_groups)

print("Difference in mean outcomes (statistical parity difference) between
unprivileged and\

privileged groups = %f" % metric_orig_train.statistical_parity_difference())


print("Disparate impact of unprivileged to privileged groups = %f" % metric_
orig_train.disparate_impact())
```

Output

```
Difference in mean outcomes (statistical parity difference) between unprivileged
and privileged groups = −0.243882

Disparate impact of unprivileged to privileged groups = 0.391562
```

A statistical parity difference of -24% indicates that the probability of getting invited to an interview given a female applicant is by 24% less than that given a male applicant:

$$\Pr(interview = 1|female = 1) = \Pr(interview = 1|female = 0) - 24\%$$

By disparate impact, women are only 39 percent as likely to be invited to an interview as men, which is a strong indicator of recruiters discriminating against female applicants.

$$\Pr(interview = 1|female = 1) = 39\% \times \Pr(interview = 1|female = 0)$$

In the next step, we apply a mitigation algorithm to neutralize bias in the training data.

Step 9: apply bias mitigation algorithm, compute fairness metrics on transformed data

In step 8 we saw that our privileged group is more than twice as likely to obtain a positive outcome. To minimize that bias, we apply a mitigation algorithm to the original training data, expecting that training a model on balanced data will balance the model's performance.
AIF360 offers a choice of preprocessing mitigation algorithms. We opt for the reweighing algorithm from the pertinent class in the ***aif360.algorithms.preprocessing*** package. The algorithm transforms the dataset for a more even distribution of positive outcomes among privileged and unprivileged groups as defined by a protected attribute. The transformation consists in resampling the dataset by adjusting the samples' weights such that the label becomes statistically independent of the protected attribute without changing attribute or label values. To generate a training set based on that transformation, we call the ***fit_transform()*** method. The new dataset is named ***dataset_transf_train***.

We compute the same two fairness metrics (statistical parity difference, disparate impact) on the transformed dataset. The statistical parity difference now is zero, the disparate impact has risen to 100 percent. Both metrics indicate that the new dataset's weighting neutralizes the statistical effect of the protected attribute on the outcome.

```
RW = reweighing(unprivileged_groups=unprivileged_groups,
                privileged_groups=privileged_groups)
dataset_transf_train = RW.fit_transform(dataset_orig_train)
metric_transf_train = BinaryLabelDatasetMetric(dataset_transf_train,
unprivileged_groups=unprivileged_groups, privileged_groups=privileged_groups)
print("Difference in mean outcomes (statistical parity difference) between
unprivileged \
and privileged groups = %f" % metric_transf_train.statistical_parity_diffe-
rence())
print("Disparate impact of unprivileged to privileged groups = %f" % metric_
transf_train.disparate_impact())
```

Output

```
Difference in mean outcomes (statistical parity difference) between unprivileged
and privileged groups = 0.000000
Disparate impact of unprivileged to privileged groups = 1.000000
```

In steps 10 and 11, we are going to train two versions of our ML model. The first version will be learning from the original hiring dataset, the second from the transformed training set. The model is a logistic regression imported from `sklearn.linear_model.` To train it, we use the `fit()` method. After the training, we have both models calculate their predictions, and measure their accuracy. In steps 12 and 13, we compute the fairness metrics based on those predictions. Step 14 recaps the results.

Step 10: train a classifier on the original training dataset

The code below splits the feature vectors from the label vector both in the training and in the test dataset. After that, it trains a logistic regression on the original hiring data.

```
x_train=dataset_orig_train.features #input vectors of training instances
y_train=dataset_orig_train.labels.reshape(-1) #real output of training instances
x_test=dataset_orig_test.features #input vectors of test instances
y_test=dataset_orig_test.labels.reshape(-1) #real output of test instances
#Train a logistic regression model based on original hiring data.
lr=LogisticRegression(solver='lbfgs')
lr.fit(x_train,y_train,sample_weight=dataset_orig_train.instance_weights)
'''
NOTE: All samples of the original dataset (dataset_orig_train) have the same
instance weight equalling 1. Entering the (sample_weight) parameter into the fit
method is optional.
'''
LogisticRegression(C=1.0, class_weight=none, dual=false, fit_intercept=true,
     intercept_scaling=1, l1_ratio=none, max_iter=100,
     multi_class='warn', n_jobs=none, penalty='l2',
     random_state=none, solver='lbfgs', tol=0.0001, verbose=0,
     warm_start=false)
#Compute the model's predictions by feeding it with test instances.
predictions=lr.predict(x_test)
#Compute the model's accuracy by comparing predictions with real output of test
instances.
accuracy=np.mean(predictions==y_test)
#Print the value of the accuracy measure.
accuracy
```

Output

```
0.8221476510067114
```

Step 11: train a classifier on the transformed training dataset

By analogy, we train the second version of our logistic regression model. This version learns its parameters from the transformed training dataset.

```
x_train_tr=dataset_transf_train.features

y_train_tr=dataset_transf_train.labels.reshape(-1)

'''

Train a logistic regression model on unbiased data by feeding the fit method with
the instance weights of the transformed training set.

'''

lr_tr=LogisticRegression(solver='lbfgs')

lr_tr.fit(x_train_tr,y_train_tr,sample_weight=dataset_transf_train.instance_
weights)

predictions_tr=lr_tr.predict(x_test) #Compute predictions of model trained on
unbiased data.

accuracy_tr=np.mean(predictions_tr==y_test) #Compute accuracy of model trained
on unbiased data.

#Print the value of the accuracy measure.

accuracy_tr
```

Output

```
0.8238255033557047
```

Step 12: compute fairness metrics on the performance of the model trained via original training set

Now, we assess the discrimination implicit in the prediction of the model trained on the original data. To compute the fairness metrics, we distinguish the prediction on the protected group (women) from that of the complement (men). The SPD is computed by subtracting the rate of favourable outcomes of the privileged group from that of the protected group. The disparate impact is computed as the ratio of those two percentages.

```
prediction_female=[]

prediction_male=[]

for i,j in zip((x_test[:,-1]==1.0).astype(int),predictions):

     if i ==1:

            prediction_female.append(j)

     else:

            prediction_male.append(j)

SP_female=np.mean(prediction_female)

SP_male=np.mean(prediction_male)

print("Rate of favourable outcomes for group:female = %f" %SP_female)
print("Rate of favourable outcomes for group:male = %f" %SP_male)

print("Statistical parity difference = %f" %(SP_female-SP_male))

print("Disparate impact of unprivileged to privileged groups = %f" %(SP_female/
SP_male))
```

Output

```
Rate of favourable outcomes for group:female = 0.120521

Rate of favourable outcomes for group:male = 0.384083


Statistical parity difference = -0.263562

Disparate impact of unprivileged to privileged groups = 0.313789
```

Step 13: compute fairness metrics on the performance of the model trained via transformed training set

By analogy with step 12, we compute the fairness metrics of the prediction made by the model trained on balanced data.

```
predictions_tr_female=[]

predictions_tr_male=[]

for i,j in zip((x_test[:,-1]==1.0).astype(int),predictions_tr):

  if i ==1:

     prediction_tr_female.append(j)

  else:

     prediction_tr_male.append(j)

SP_female_tr=np.mean(prediction_tr_female)

SP_male_tr=np.mean(prediction_tr_male)


print("Rate of favourable outcomes for group:female = %f" %SP_female_tr)

print("Rate of favourable outcomes for group:male = %f" %SP_male_tr)

print("Statistical parity difference = %f" %(SP_female_tr-SP_male_tr))

print("Disparate impact of unprivileged to privileged groups = %f" %(SP_fema-
le_tr/SP_male_tr))
```

Output

```
Rate of favourable outcomes for group:female = 0.198697Rate of favourable
outcomes for group:male = 0.339100

Statistical parity difference = -0.140403

Disparate impact of unprivileged to privileged groups = 0.585954
```

Step 14: comparison of fairness metrics before and after bias mitigation in the data

Finally, we compare the results yielded by our two models. The first model’s rate of favourable outcomes is 26 percent lower for the protected group than for the complement. After mitigating bias, the parity difference of the model trained on the balanced data falls to 14 percent. On disparate impact, results were as follows. When we trained the model on biased data, the rate of favourable outcomes for women equalled less than a third of that for men. After mitigating bias and training the model on balanced data, the ratio has almost doubled. Now, the former rate levels at almost two thirds of the latter.

```
result=pd.DataFrame([[SP_female,SP_female_tr],[SP_male,SP_
male_tr],[SP_female-SP_male,SP_female_tr-SP_male_tr],\

  [SP_female/SP_male,SP_female_tr/SP_male_tr]],index=
  ['rate of favourable outcomes: female','rate of favourable
  outcomes: male',\

  'statistical parity difference','disparate impact'],\

            columns=['before mitigation','after mitigation'])

result
```

Output

	Before bias mitigation	After bias mitigation
rate of favorable outcomes-Female	0.120521	0.198697
rate of favorable outcomes-Male	0.384083	0.339100
Statistical Parity Difference	−0.263562	−0.140403
Disparate Impact	0.313789	0.585954

Conclusion

Across industries, businesses are increasingly relying on machine learning and other artificial intelligence models to overcome human limitations in high-stakes decision-making. More recently, researchers have voiced concern about big-data-driven ML models boosting or even generating socioeconomic inequality. Intended or not, discrimination resulting from algorithmic business decisions can ensue ethical or legal issues. To manage the inherent reputational risk and avoid claims for damages, organizations need to ensure that their data-driven decision making remains fair.

In this article, we define fairness for ML purposes and exemplify how biased data causes ML models to arrive at discriminatory predictions, and how bias mitigation algorithms can counter such effects. To quantify bias in data, we have introduced various fairness metrics and selected two that suited the definition of fairness applicable to our example. After an overview of bias mitigation algorithms, we have expanded on reweighing, an algorithm run on the data before using the latter as ML input.

As a use case, we chose an AI-model that decides whether or not to invite job applicants for an interview. To illustrate the case, we have provided a step-by-step demo including a Jupyter Notebook script. The demo shows how the preprocessing reweighing algorithm reduces bias in training data and thus in decisions based on that. To quantify the mitigation effect, we have compared the ML model trained on an unbiased dataset with a benchmark based on the original data.

References

AI Fairness 360. IBM Research Trusted AI. URL: https://aif360.mybluemix.net/

Bellamy, Rachel ; Dey, Kuntal ; Hind, Michael ; Hoffman, Samuel ; Houde, Stephanie ; Kannan, Kalapriya ; Lohia, Pranay ; Martino, Jacquelyn ; Mehta, Sameep ; Mojsilovic, Aleksandra ; Nagar, Seema ; Natesan Ramamurthy, Karthikeyan ; Richards, John ; Saha, Diptikalyan ; Sattigeri, Prasanna ; Singh, Moninder ; Kush, Ramazon ; Zhang, Yunfeng. (2018). AI Fairness 360: An Extensible Toolkit for Detecting, Understanding, and Mitigating Unwanted Algorithmic Bias. URL: https://arxiv.org/pdf/1810.01943.pdf

d'Alessandro, B., O'Neil, C., & LaGatta, T. (2017). Conscientious Classification: A Data Scientist's Guide to Discrimination-Aware Classification. Big Data, 5(2), 120–134. https://arxiv.org/ftp/arxiv/papers/1907/1907.09013.pdf

Fairness Tree. [Graph]. URL: http://aequitas.dssg.io/static/images/metrictree.png

Fairness: Types of Bias | Machine Learning Crash Course. (2021). Google Developers.
URL: https://developers.google.com/machine-learning/crash-course/fairness/types-of-bias

N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman and A. Galstyan, A survey on bias and fairness in machine learning, Sep. 2019, URL: https://arxiv.org/pdf/1908.09635.pdf

Strategeion Resume Skills. (2019, January 9). Kaggle.
URL: https://www.kaggle.com/vingkan/strategeion-resume-skills

T. Calders, F. Kamiran and M. Pechenizkiy, Building Classifiers with Independency Constraints, 2009 IEEE International Conference on Data Mining Workshops, 2009, pp. 13-18, doi: 10.1109/ICDMW.2009.83.
URL: https://www.win.tue.nl/~mpechen/publications/pubs/CaldersICDM09.pdf

T. Trusted-AI/AIF360. GitHub. URL: https://github.com/Trusted-AI/AIF360

Yadav, D. (2020, April 14). Weighted Logistic Regression for Imbalanced Dataset. Medium.
URL: https://towardsdatascience.com/weighted-logistic-regression-for-imbalanced-dataset-9a5cd88e68b

Fairness in AI Systems – How to Detect and Mitigate Bias in Machine Learning Models

About Consileon

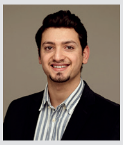
The Consileon Business Consultancy GmbH is a midsized German IT and management consultancy, located in the beautiful city of Karlsruhe, Germany. We are mostly working for clients in the DACH region (Germany, Austria, Switzerland) but are also internationally represented with a total of 15 offices across Europe. We assist our clients in many kinds of digitization initiatives with a clear focus on strategic decision making as well as translating strategies into operative action while providing the needed IT support. What makes us unique is that not only do we elaborate highly tailored concepts for the specific challenge our client is facing but we are also assisting the client with the implementation.

Publisher

Team Artificial Intelligence
Consileon Business Consultancy GmbH
www.consileon.ai

Contact

Consileon Business Consultancy GmbH
Maximilianstraße 5
76133 Karlsruhe
www.consileon.de



Ramiz Abusabbah
Consultant, member of AI-Team
ramiz.abusabbah@consileon.de

Dr. Andreas Alin
Head of AI-Team
andreas.alin@consileon.de