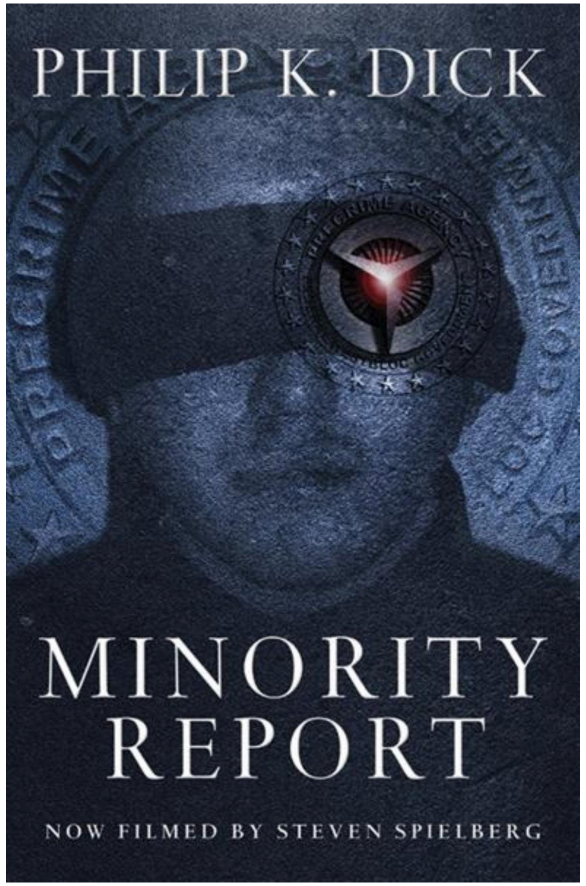
Mitigating Discrimination in Machine Learning

Introducing the Problem

Fairness Issues in Machine Learning

Say we want to estimate the risk of violent crimes in given population



- This is obviously a very ethically sensitive task
- ...Which makes it a good test case to discuss fairness in data-driven methods

Fairness in Data-Driven Methods

Fairness in data-driven methods is very actual topic

- As data-driven systems become more pervasive
- They have the potential to significantly affect social groups

Once you deploy an AI model, performance is not enough

- You might have stellar accuracy and efficient inference
- ...And still end up causing all sort of havoc

This is so critical that the topic is starting to be regulated

- The EU has drafted Ethics Guidelines for Trustworthy AI
- ...And has reached provisional agreement on a big AI act
- In some fields, models that do not comply with some rules cannot be deployed

Inspecting the Dataset

We will run an experiment on a version of the "crime" UCI dataset

In [2]:
```python
display(data.head())
print(f'Number of rows: {len(data)}')
```

	communityname	state	fold	pop	race	pct12-21	pct12-29	pct16-24	pct65up	pctUrban	...	pctForeignBorn	pctBornStateF
1008	EastLampetertownship	PA	5	11999	0	0.1203	0.2544	0.1208	0.1302	0.5776	...	0.0288	0.8132
1271	EastProvidencecity	RI	6	50380	0	0.1171	0.2459	0.1159	0.1660	1.0000	...	0.1474	0.6561
1936	Betheltown	CT	9	17541	0	0.1356	0.2507	0.1138	0.0804	0.8514	...	0.0853	0.4878
1601	Crowleycity	LA	8	13983	0	0.1506	0.2587	0.1234	0.1302	0.0000	...	0.0029	0.9314
293	Pawtucketcity	RI	2	72644	0	0.1230	0.2725	0.1276	0.1464	1.0000	...	0.1771	0.6363

5 rows × 101 columns

```
Number of rows: 1993
```

- The target is "violentPerPop" (number of violent offenders per 100K people)
- All attributes are continuous, except for "race"

A First Attempt at Mitigating Discrimination

The "race" attribute seems like one that could easily lead to discrimination

...So we'll attempt to keep it out

```
In [4]: data[attributes_nr].head()
```

Out[4]:

	pop	pct12-21	pct12-29	pct16-24	pct65up	pctUrban	medIncome	pctWwage	pctWfarm	pctWdiv	...	persHomeless	pctForeig
1008	11999	0.1203	0.2544	0.1208	0.1302	0.5776	34720	0.8275	0.0376	0.5482	...	0.000000	0.0288
1271	50380	0.1171	0.2459	0.1159	0.1660	1.0000	31007	0.7400	0.0059	0.4359	...	0.000000	0.1474
1936	17541	0.1356	0.2507	0.1138	0.0804	0.8514	53761	0.8562	0.0050	0.5863	...	0.000000	0.0853
1601	13983	0.1506	0.2587	0.1234	0.1302	0.0000	13804	0.6245	0.0242	0.2248	...	0.000000	0.0029
293	72644	0.1230	0.2725	0.1276	0.1464	1.0000	26541	0.7526	0.0038	0.3694	...	1.376576	0.1771

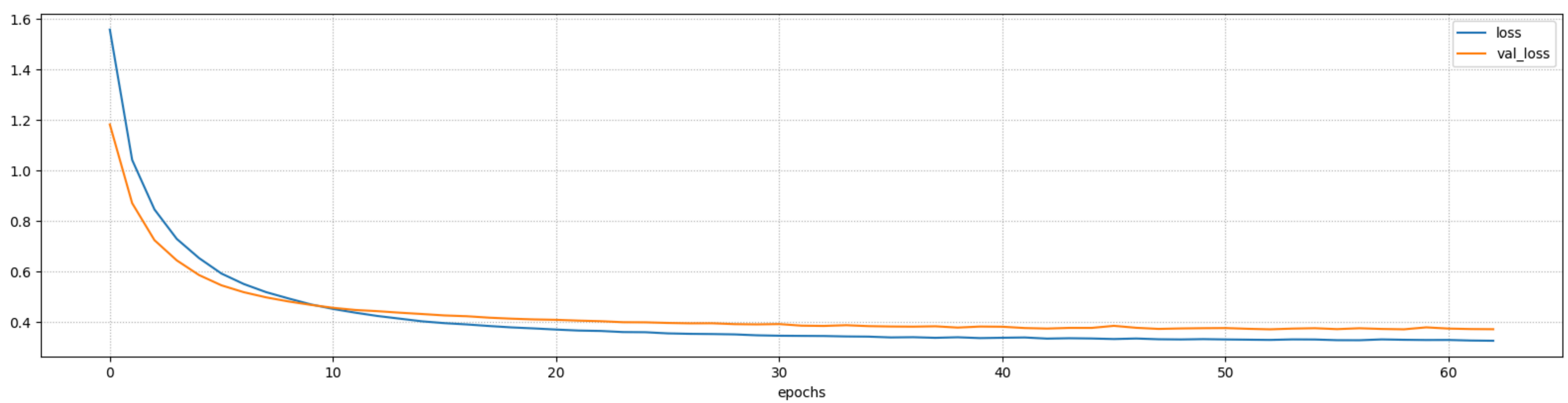
5 rows × 96 columns

- This is one of the first solutions that typically come to mind to mitigate unwanted bias
- We've also removed some columns that are not useful as model inputs/outputs

Baseline

Let's establish a baseline by tackling the task via Linear Regression

```
In [5]: nn = util.build_nn_model(input_shape=(len(attributes_nr),), output_shape=1, hidden=[], outpu
        history = util.train_nn_model(nn, tr[attributes_nr], tr[target], loss='mse', batch_size=32,
        util.plot_training_history(history, figsize=figsize)
```




```
Final loss: 0.3255 (training), 0.3712 (validation)
```

Baseline Evaluation

...And let's check the results

```
In [6]: tr_pred = nn.predict(tr[attributes_nr], verbose=0)
        r2_tr, mae_tr = r2_score(tr[target], tr_pred), mean_absolute_error(tr[target], tr_pred)
        ts_pred = nn.predict(ts[attributes_nr], verbose=0)
        r2_ts, mae_ts = r2_score(ts[target], ts_pred), mean_absolute_error(ts[target], ts_pred)
        print(f'R2 score: {r2_tr:.2f} (training), {r2_ts:.2f} (test)')
        print(f'MAE: {mae_tr:.2f} (training), {mae_ts:.2f} (test)')

        R2 score: 0.66 (training), 0.60 (test)
        MAE: 0.39 (training), 0.45 (test)
```

- Some improvements (not much) can be obtained with a Deeper model

Baseline Evaluation

...And let's check the results

In [6]:
```python
tr_pred = nn.predict(tr[attributes_nr], verbose=0)
r2_tr, mae_tr = r2_score(tr[target], tr_pred), mean_absolute_error(tr[target], tr_pred)
ts_pred = nn.predict(ts[attributes_nr], verbose=0)
r2_ts, mae_ts = r2_score(ts[target], ts_pred), mean_absolute_error(ts[target], ts_pred)
print(f'R2 score: {r2_tr:.2f} (training), {r2_ts:.2f} (test)')
print(f'MAE: {mae_tr:.2f} (training), {mae_ts:.2f} (test)')
```

```
R2 score: 0.66 (training), 0.60 (test)
MAE: 0.39 (training), 0.45 (test)
```

- Some improvements (not much) can be obtained with a Deeper model

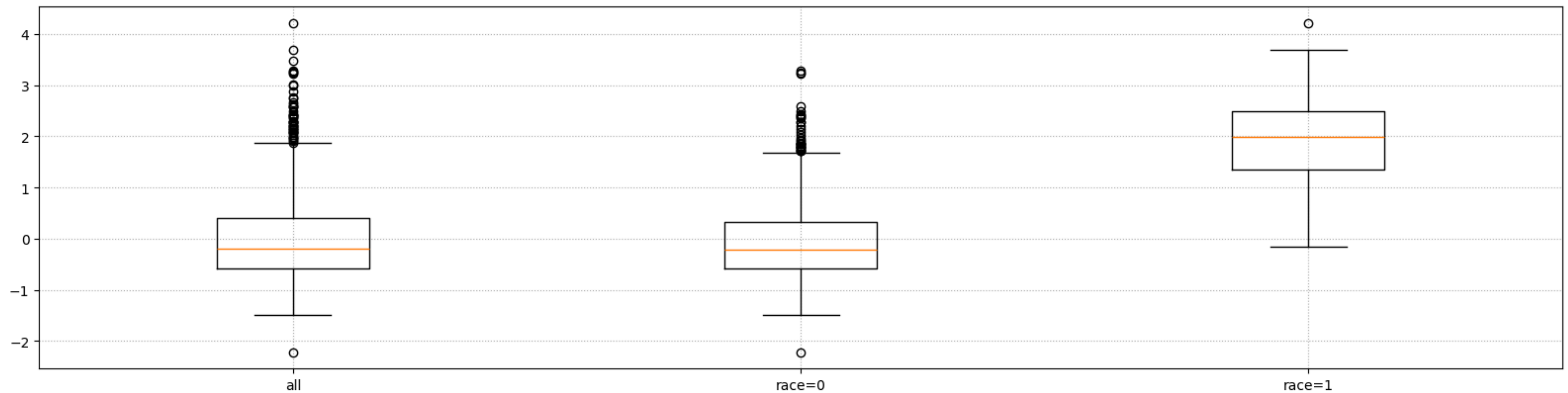
Without the "race" attribute, we might think that no discrminiation can occur

....But is it true?

Disparate Treatment

Indeed, our model treats the groups differently

```
In [7]: protected={'race': (0, 1)}
        util.plot_pred_by_protected(tr, tr_pred, protected={'race': (0, 1)}, figsize=figsize)
```



- Here we inspecte the distribution of predictions for the "race = 0" and "race = 1" groups
- ...And the two are indeed significantly different

Mitigating Discrimination in Machine Learning

Fairness Metrics

Fairness Metrics

Evaluating fairness is complicated

- Ethical topics are almost always quite nuanced
- ...And they do not lend themselves to clear, unambiguous definitions

From an algorithmic purpose, however...

...One of the most manageable approach consists in relying on a fairness metric

- Even if any discrimination metric may indeed be questionable
- ...Measurable quantities can at least be manipulated with Maths

Several fairness metrics have been proposed

- Since defining a single, catch-all, metric seems unrealistic
- Having the ability to choose among multiple options is a good thing

Here we will focus on the idea of disparate treatment

The DIDI Indicator

In particular, we will use the indicatror from this paper

- Given a set of categorical protected attribute (indexes) J_p
- ...The Disparate Impact Discrimination Index (for regression) is given by:

$$\mathrm{DIDI}_r = \sum_{j \in J_p} \sum_{v \in D_j} \left| \frac{1}{m} \sum_{i=1}^{m} y_i - \frac{1}{|I_{j,v}|} \sum_{i \in I_{j,v}} y_i \right|$$

- Where D_j is the domain of attribute j
- ...And $I_{j,v}$ is the set of example such that attribute j has value v

The DIDI is a measure of discrepancy w.r.t. the average prediction

- It is obtained by computing the average prediction for every protected group
- ...And summing up their discrepancy w.r.t. the global average

Using the DIDI to Evaluate Our Model

For our Linear Regression model, we get

```
In [8]: tr_DIDI = util.DIDI_r(tr, tr_pred, protected)
        ts_DIDI = util.DIDI_r(ts, ts_pred, protected)
        print(f'DIDI: {tr_DIDI:.2f} (training), {ts_DIDI:.2f} (test)')

        DIDI: 2.00 (training), 2.13 (test)
```

- We wish to improve over this baseline, which is not an easy task:
- Discrimination is a form of bias in the training set, but bias is not necessarily bad

In fact, ML works because of bias

...I.e. because the training distribution contains information about the test one

- Improving fairness requires to get rid of the unwanted part of this bias
- ...Which will likely lead to some loss of accuracy (hopefully not too much)

Mitigating Discrimination in Machine Learning

Fairness Constraints

Fairness as a Constraint

Let's recap our goals:

We want to train an accurate regressor ($L =$ loss function):

$$\text{argmin}_\theta \, \mathbb{E}_{x,y \sim P(X,Y)} \left[L(y, \hat{f}(x, \theta)) \right]$$

We want to measure fairness via the DIDI:

$$\text{DIDI}(y) = \sum_{j \in J_p} \sum_{v \in D_j} \left| \frac{1}{m} \sum_{i=1}^{m} y_i - \frac{1}{|I_{j,v}|} \sum_{i \in I_{j,v}} y_i \right|$$

...And we want the DIDI to be low, e.g.:

$$\text{DIDI}(\hat{f}(x, \theta)) \leq \varepsilon$$

Fairness as a Constraint

We can use this information to re-state the training problem

$$\text{argmin}_\theta \left\{ \mathbb{E}_{x,y} \left[L(y, \hat{f}(x, \theta)) \right] \mid \text{DIDI}(\hat{f}(x, \theta)) \leq \varepsilon \right\}$$

- Training is now a constrained optimization problem
- We require the DIDI for ML output to be within acceptable levels

After training, the constraint will be distilled in the model parameters

We are requiring constraint satisfaction on the training set

...Meaning that we'll have no satisfaction guarantee on unseen examples

- This is suboptimal, but doing better is very difficult
- ...Since our constraint is defined (conceptually) on the whole distribution

We'll trust the model to generalize well enough

Constrained Machine Learning

There's more than one way to account for constraints while training

- The one we'll focus on is based on the idea to add a constrained based penalty
- In particular, we will modify our training problem as follows:

$$\text{argmin}_\theta \, \mathbb{E}_{x,y}\left[L(y, \hat{f}(x,\theta))\right] + \lambda \max\left(0, \text{DIDI}(\hat{f}(x,\theta)) - \varepsilon\right)$$

Without going into too much detail, the term $\max\left(0, \text{DIDI}(\hat{f}(x,\theta)) - \varepsilon\right)$:

- Is equal to 0 if the constraint $\text{DIDI}(\hat{f}(x,\theta)) \leq \varepsilon$ is satisfied
- Is > 0 otherwise, meaning it acts as a penalty in this case
- The penalty is scaled by a factor λ
- ...which can be increased until the constraint is satisfied

There are a lot of caveats, but will skip them

Building the Constrained Model

We can build a constrained version of our predictor

In [9]:
```
protected = {'race': (0, 1)}
didi_thr = 1.0
base_pred = util.build_nn_model(input_shape=(len(attributes),), output_shape=1, hidden=[])
nn = util.LagDualDIDIModel(base_pred, attributes, protected, thr=didi_thr)
```

We will try to roughly halve the "natural" DIDI of the model

- Since for our baseline we have $\text{DIDI}(y) \simeq 2$
- ...We decided to pick $\varepsilon = 1$

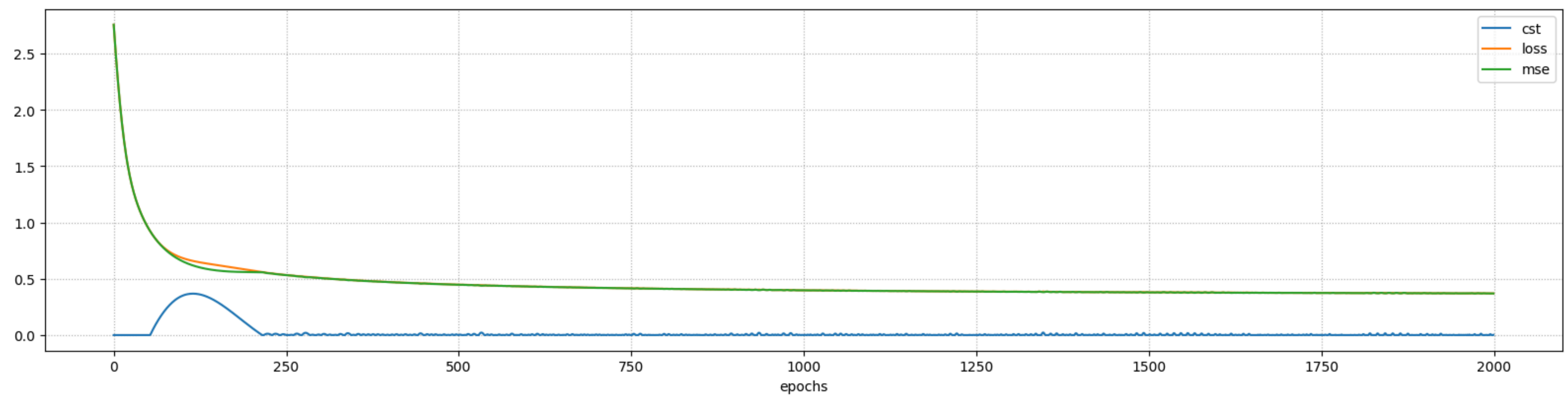
Even if we are focusing here on a DIDI constraint

- The approach can be employed for other fairness metrics
- ...And also for constraints not related to fairness (e.g. domain knowledge)

Training the Constrained Model

We can train the constrained model more or less as usual

In [10]:
```
base_pred = util.build_nn_model(input_shape=(len(attributes),), output_shape=1, hidden=[])
nn = util.LagDualDIDIModel(base_pred, attributes, protected, thr=didi_thr)
history = util.train_nn_model(nn, tr[attributes], tr[target], loss='mse', validation_split=(
util.plot_training_history(history, figsize=figsize)
```




```
Final loss: 0.3699 (training)
```

Constrained Model Evaluation

Let's check both the prediction quality and the DIDI

```
In [11]: tr_pred = nn.predict(tr[attributes], verbose=0)
         r2_tr = r2_score(tr[target], tr_pred)
         ts_pred = nn.predict(ts[attributes], verbose=0)
         r2_ts = r2_score(ts[target], ts_pred)
         tr_DIDI = util.DIDI_r(tr, tr_pred, protected)
         ts_DIDI = util.DIDI_r(ts, ts_pred, protected)

         print(f'R2 score: {r2_tr:.2f} (training), {r2_ts:.2f} (test)')
         print(f'DIDI: {tr_DIDI:.2f} (training), {ts_DIDI:.2f} (test)')

         R2 score: 0.63 (training), 0.55 (test)
         DIDI: 1.00 (training), 1.09 (test)
```

We lost some accuracy, but the DIDI has the desired value on the training data

- On the test data, the value is a bit larger than we wished
- This happened since we enforced the constraint only on the training data

Some Comments

This is not the only approach for constrained ML

- There approaches based on projection, pre-processing, iterative projection...
- ...And in some cases you can enforce constraints through the architecture itself

...But it is simple and flexible

- You just need your constraint to be differentiable
- ...And some good will to tweak the implementation

The approach can be used also for symbolic knowledge injection

- Perhaps domain experts can provide you some intuitive rule of thumbs
- You model those as constraints and take them into account at training time
- Just be careful with the weights, as in this case feasibility is not the goal