Data Projection on the Fairness Use Case

Fingers Crossed

Projection Problem Formulation

The projection problem for the fairness case study is given by:

$$\underset{\hat{y}}{\operatorname{argmin}} \|y - z\|_2$$

$$\begin{aligned}
\text{subject to: } \bar{y} &= \frac{1}{m} 1^T z \\
\bar{y}_v &= \frac{1}{\|X_v\|_1} X_v z && \forall v \in D \\
d_v &\geq \bar{y} - \bar{y}_v && \forall v \in D \\
d_v &\geq -(\bar{y} - \bar{y}_v) && \forall v \in D \\
1^T d_v &\leq \varepsilon
\end{aligned}$$

- We want to tackle that via the OSQP solver
- ...Which means we need to define its parameters $\boldsymbol{P}, \boldsymbol{q}, \boldsymbol{A}, \boldsymbol{l}, \boldsymbol{u}$

Deriving the Matrices

We start by factoring out the $\bar{y}$ and $\bar{y}_v$ variables

$$\operatorname*{argmin}_z \frac{1}{2} z^T I z - y^T z$$

$$\text{subject to: } \left(\frac{1}{m} - \frac{1}{\|X_v\|_1} X_v\right)^T z - d_v \leq 0 \qquad \forall v \in D$$

$$-\left(\frac{1}{m} - \frac{1}{\|X_v\|_1} X_v\right)^T z - d_v \leq 0 \qquad \forall v \in D$$

$$1^T d \leq \varepsilon$$

- We no longer have equality constraints
- Which means our problem is in the correct form

Now, we only need to work out its matrix notation

Deriving the Matrices

Our problem features two types of variables

- The projected targets z
- The individual deviation terms d

Hence, in matrix form the problem is defined as:

$$\operatorname*{argmin}_z \frac{1}{2} \left(z \quad d \right)^T P \left(z \quad d \right) - q^T \left(z \quad d \right)$$

$$\text{subject to: } A \left(z \quad d \right) \leq u$$

- By comparison with the detailed formulation
- ...We can derive the matrix structure

Deriving the Matrices

For the objective we have:

$$\begin{pmatrix} z & d \end{pmatrix}^T P \begin{pmatrix} z & d \end{pmatrix} - q^T \begin{pmatrix} z & d \end{pmatrix}$$

Which maps in the detailed formulation to:

$$\frac{1}{2} z^T I z - y^T z$$

Hence, we have:

$$P = \begin{pmatrix} I & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad q = \begin{pmatrix} y \\ 0 \end{pmatrix}$$

Deriving the Matrices

Following the same process for the constraints leads to:

$$A = \begin{pmatrix} \frac{1}{m} - \frac{1}{\|X\|_1} \odot X & -I \\ -\frac{1}{m} + \frac{1}{\|X\|_1} \odot X & -I \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad u = \begin{pmatrix} 0 \\ 0 \\ \varepsilon \end{pmatrix}$$

- Where $\|X\|_1$ refers to the column vector $\{\|X_v\|_1\}_{v\in D}$
- ...And $\odot$ to the (broadcasted) element wise product

Deriving the Matrices

Following the same process for the constraints leads to:

$$A = \begin{pmatrix} \frac{1}{m} - \frac{1}{\|X\|_1} \odot X & -I \\ -\frac{1}{m} + \frac{1}{\|X\|_1} \odot X & -I \\ 0 & 1 \end{pmatrix} \quad \text{and} \quad u = \begin{pmatrix} 0 \\ 0 \\ \varepsilon \end{pmatrix}$$

- Where $\|X\|_1$ refers to the column vector $\{\|X_v\|_1\}_{v \in D}$
- ...And $\odot$ to the (broadcasted) element wise product

Deriving these matrices is a convoluted process

- Having a modeling library would greatly simplify it
- ...But sadly no good candidates appear to be available for OSQP

Solving the Projection Problem

We can now solve the projection problem

```
In [6]: tr_prj = util.project_fairness(tr['race'], tr[target], thr=0.13)
```

```
-----------------------------------------------------------------
           OSQP v0.6.3  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2021
-----------------------------------------------------------------
problem:  variables n = 1596, constraints m = 5
          nnz(P) + nnz(A) = 7976
settings: linear system solver = qdldl,
          eps_abs = 1.0e-03, eps_rel = 1.0e-03,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 4000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: off, time_limit: off

iter   objective    pri res    dua res    rho        time
   1  -1.4944e+01   4.05e-01   5.99e-01   1.00e-01   7.47e-04s
 125  -2.2899e+01   1.34e-06   4.41e-05   2.99e+01   3.11e-03s

status:               solved
number of iterations: 125
optimal objective:    -22.8992
run time:             3.14e-03s
optimal rho estimate: 1.04e+01
```

Checking the Results

We should evaluate two qualities in our projected targets

- Their accuracy w.r.t. the original targets
- Their DIDI value (to make sure that everything went right)

In [4]:
```
print(f'Projection R2: {r2_score(tr[target], tr_prj):.2f} (train)')
print(f'Projection DIDI: {util.DIDI_r(tr, tr_prj, protected):.2f} (train)')
```

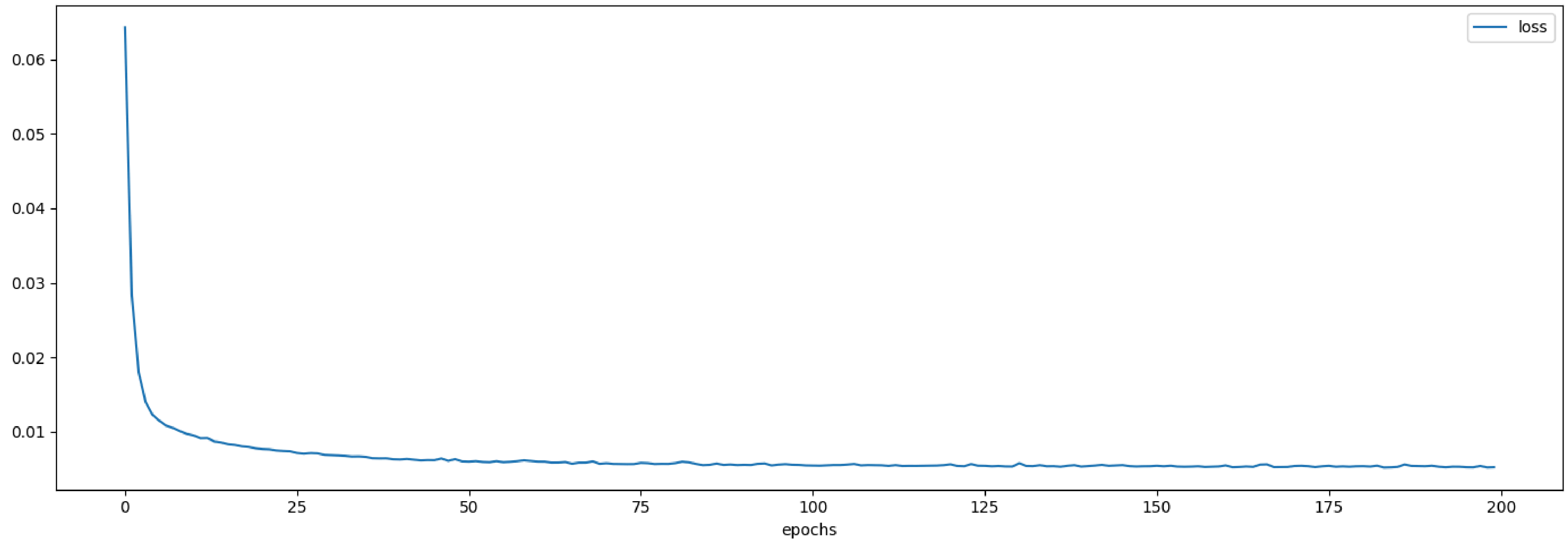
```
Projection R2: 0.96 (train)
Projection DIDI: 0.13 (train)
```

- We using the R^2 coefficient to measure accuracy
- The DIDI value is equal to the threshold
- ...Which makes sense considering that lower value come with reduced accuracy

Training a Model

We can now train a (simple) model on the projected targets

```
In [7]: nn_f = util.build_ml_model(input_size=len(attributes), output_size=1, hidden=[])
        history = util.train_ml_model(nn_f, tr[attributes], tr_prj, validation_split=0., epochs=200
        util.plot_training_history(history, figsize=figsize)
```




```
Model loss: 0.0052 (training)
```

Evaluating the Results

Finally, we can evaluate the results

In [8]:
```
tr_pred_f = nn_f.predict(tr[attributes], verbose=0)
r2_tr_f = r2_score(tr[target], tr_pred_f)
ts_pred_f = nn_f.predict(ts[attributes], verbose=0)
r2_ts_f = r2_score(ts[target], ts_pred_f)

print(f'R2 score: {r2_tr_f:.2f} (training), {r2_ts_f:.2f} (test)')
tr_DIDI_f = util.DIDI_r(tr, tr_pred_f, protected)
ts_DIDI_f = util.DIDI_r(ts, ts_pred_f, protected)
print(f'DIDI: {tr_DIDI_f:.2f} (training), {ts_DIDI_f:.2f} (test)')
```

```
R2 score: 0.63 (training), 0.57 (test)
DIDI: 0.13 (training), 0.14 (test)
```

The results should be pretty good!

- There a bit overfitting
- ...Which can lead to a modest constraint violation on the test data