Py-Boost custom losses

Team 13

Gradient Boosting on Decision Trees

$$R^{(t)} \approx \sum_{i=1}^{m} \left[g_i h_t(\mathbf{x}_i) + \frac{1}{2} d_i h_t^2(\mathbf{x}_i) \right] + \Omega(h_t)$$

$$g_i = \partial_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right), \; d_i = \partial^2_{\widehat{y}^{(t-1)}} l\left(y_i, \widehat{y}^{(t-1)}\right)$$

The decision: $$\boxed{\omega_j^* = -\frac{G_j}{D_j + \lambda}}$$

$G_j = \sum_{i \in I_j} g_i,$

$D_j = \sum_{i \in I_j} d_i$

Problem statement

Median regression

$$MAE = \sum |y_{pred,i} - y_i|$$

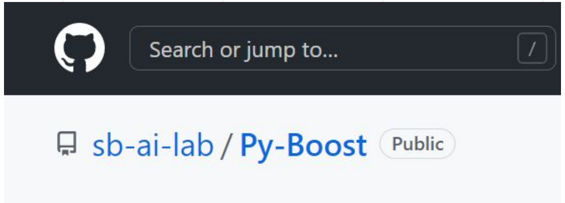
$$\nabla MAE = \mathbf{sgn}(y_{pred} - y)$$

$$\boxed{\nabla^2 MAE = \mathbf{diag}(0, \ldots, 0)}$$

What is a **good alternative loss function** for median regression?

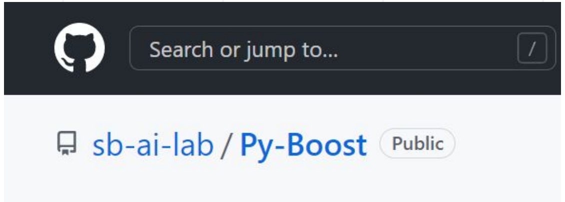
Py-Boost

- GB library, supports all common features and hyperparameters
- Computationally efficient, works only on GPU, uses Cupy and Numba
- Easy to customize sampling strategy, training control, losses and metrics



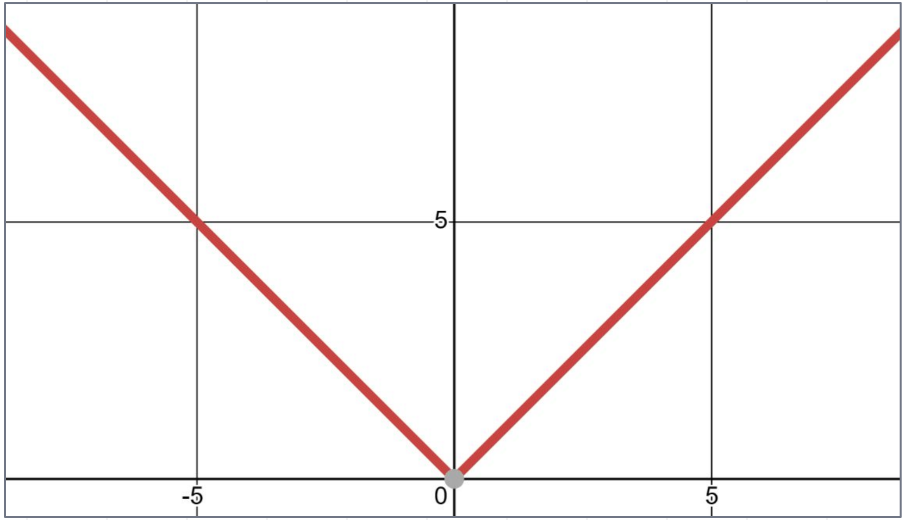
Py-Boost

- GB library, supports all common features and hyperparameters
- Computationally efficient, works only on GPU, uses Cupy and Numba
- Easy to customize sampling strategy, training control, losses and metrics



CuPy Numba

MAE



$$MAE = \sum |y_{pred,i} - y_i|$$

$$\nabla MAE = \mathbf{sgn}(y_{pred} - y)$$

$$\nabla^2 MAE = \mathbf{diag}(1, \ldots, 1)$$

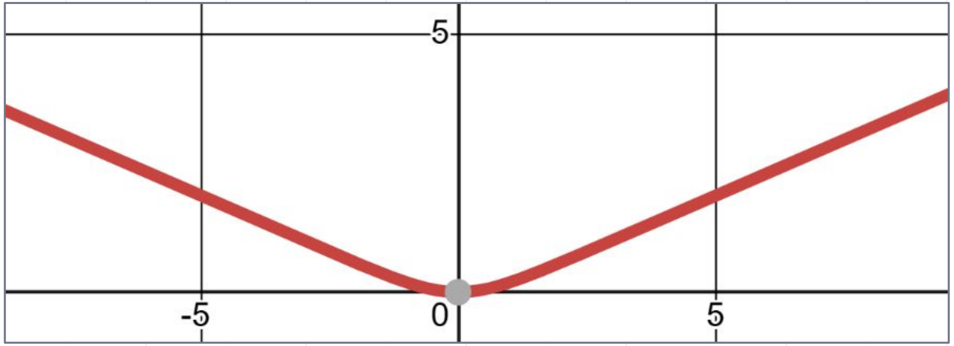
MSLE

$$MSLE = \sum (log(1 + y_{pred,i}) - log(1 + y_i))^2$$

$$log(1 + y) \rightarrow Y$$

$$MSLE = \sum (Y_{pred,i} - Y_i)^2$$

LogCosh

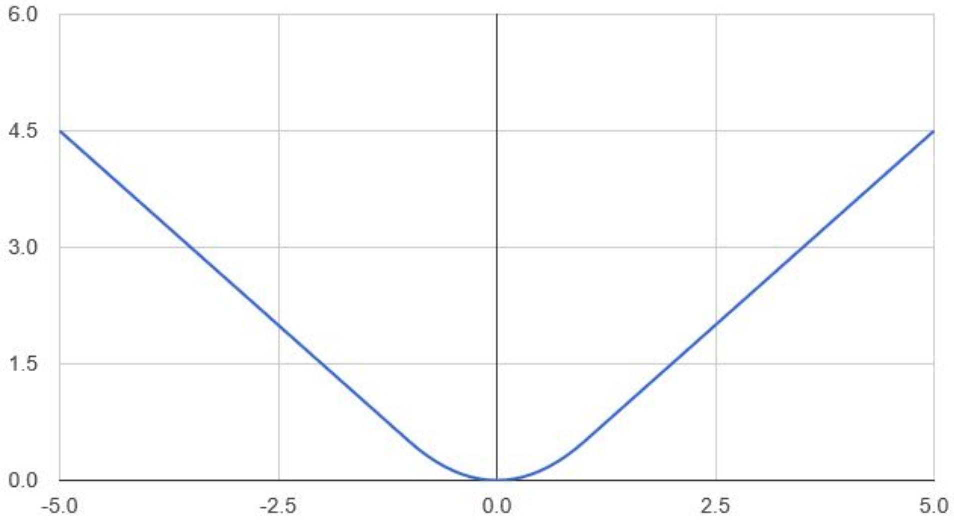


$$LogCosh = \sum log(cosh(y_{pred,i} - y_i))$$

$$\nabla LogCosh = \mathbf{tanh}(y_{pred} - y)$$

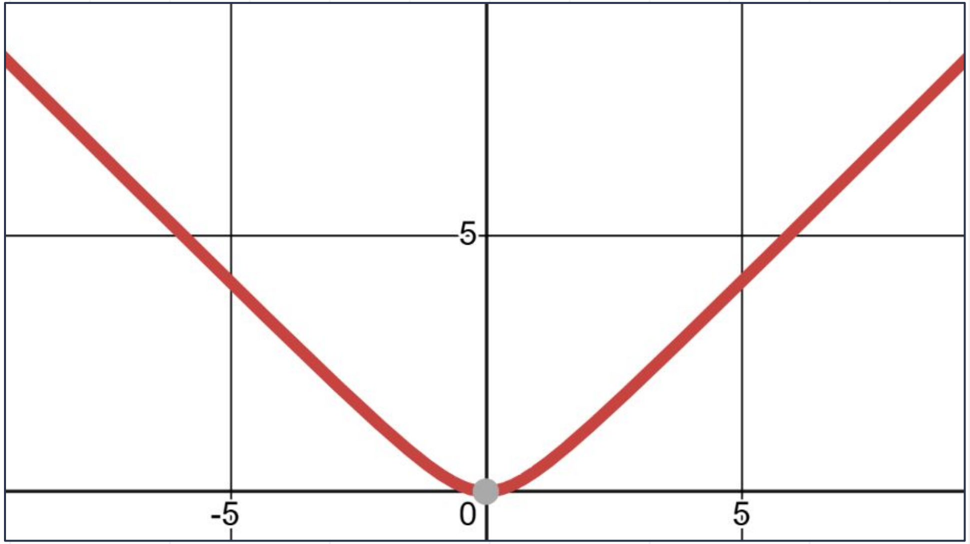
$$\nabla^2 LogCosh = \mathbf{diag}(\mathrm{sech}^2(y_{pred,i} - y_i))$$

Huber



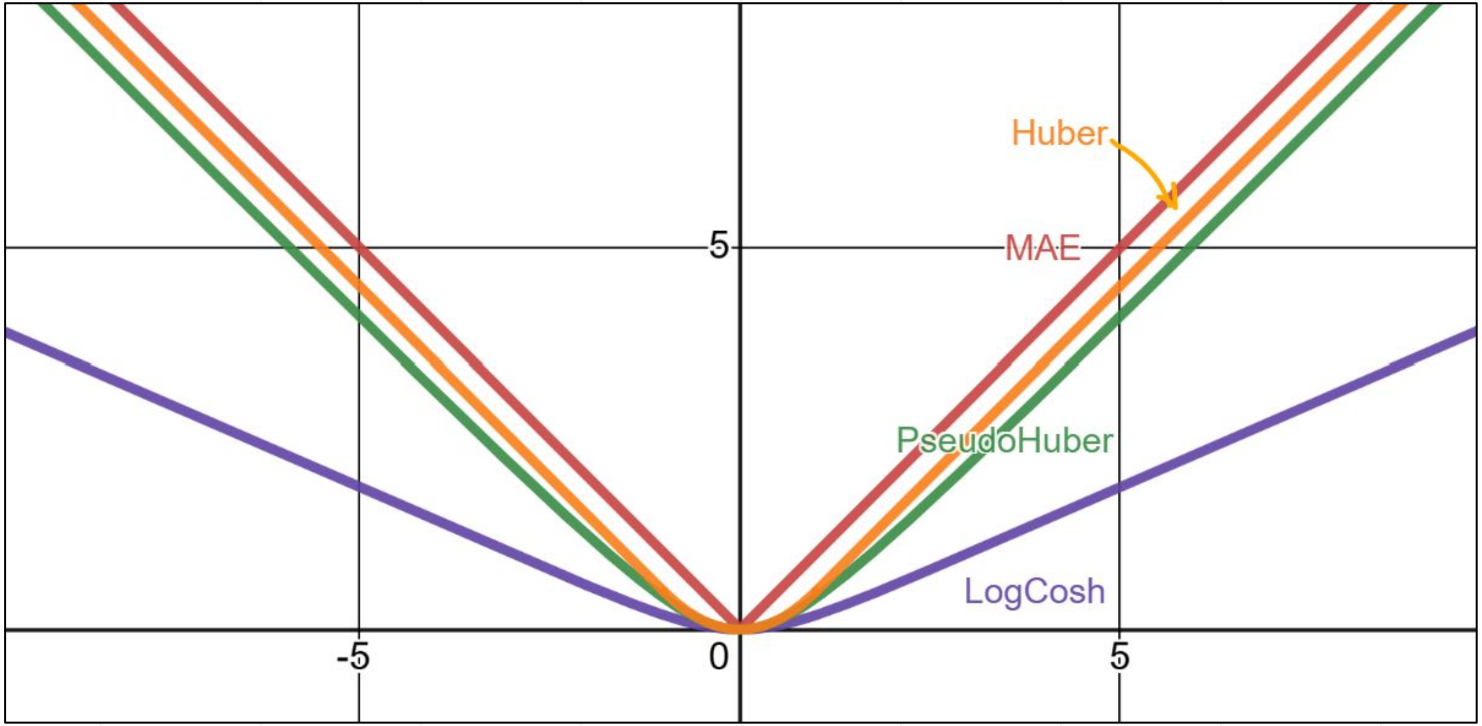
$$\text{Huber}_{\delta}(x) = \sum \left\{ |x| \le \delta : \frac{1}{2} x^{2}, \quad |x| > \delta : \delta \left(|x| - \frac{1}{2} \delta \right) \right\}$$

Pseudo Huber



$$\mathrm{PHuber}_{\delta}(x) = \delta^2 \left(\sqrt{1 + (x/\delta)^2} - 1 \right)$$

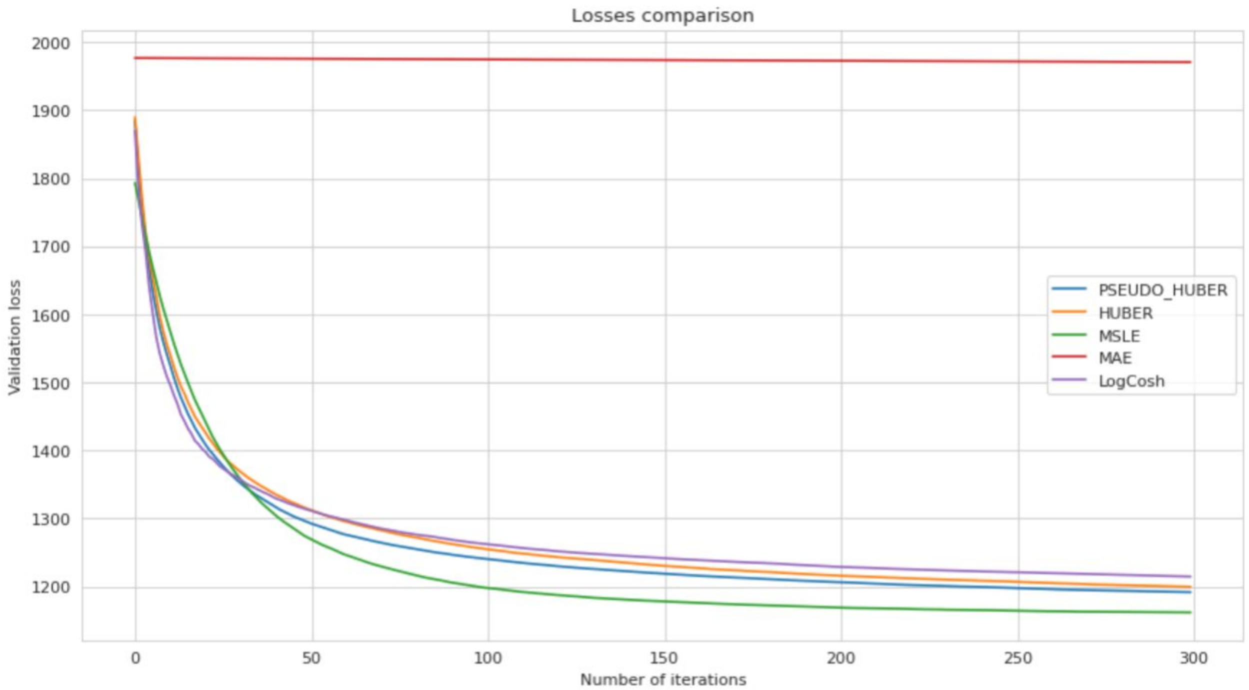
Losses



Hyperparameter tuning

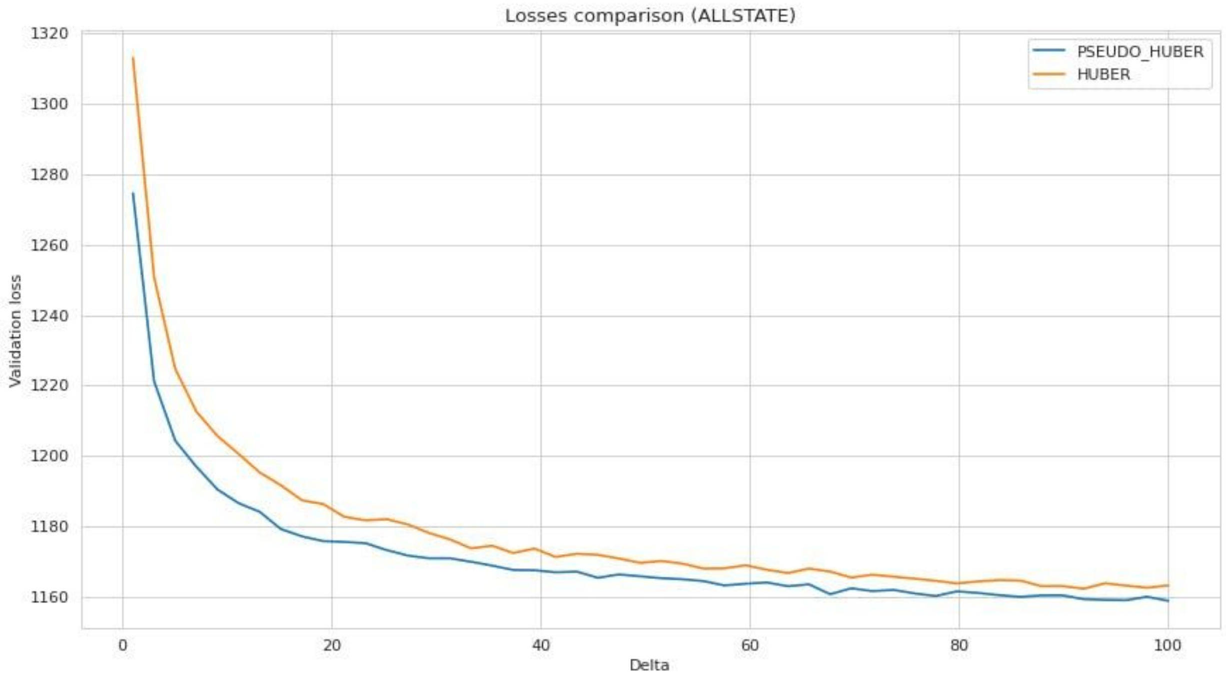
```python
def objective(trial):
    model = GradientBoosting(
        loss=CustomHuberLoss(delta=trial.suggest_float("delta", 1, 20)),
        metric=CustomMAEMetric(),
        lr=trial.suggest_float("lr", 1e-2, 1e-1, log=True),
        lambda_l2 = trial.suggest_float("lambda_l2", 1., 20.),
        max_depth=trial.suggest_int("max_depth", 3, 10),
        verbose = 1000
    )
    model.fit(X, y, eval_sets=[{'X': X_val, 'y': y_val},])
    y_pred = model.predict(X_val)

    return (np.abs(y_val - y_pred)).mean()
```

Results: Allstate dataset

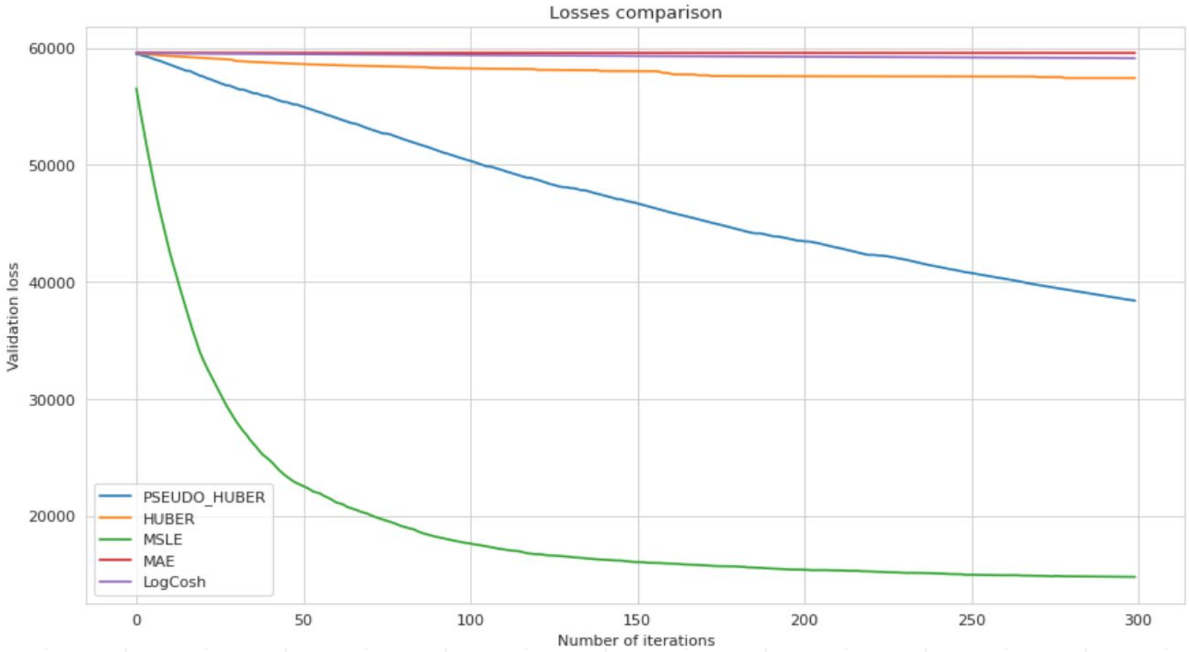


	Pseudo Huber	Huber	MSLE	MAE	LogCosh
Test loss	1139	1139	1140	1915	1152

Dependence on delta: Allstate dataset



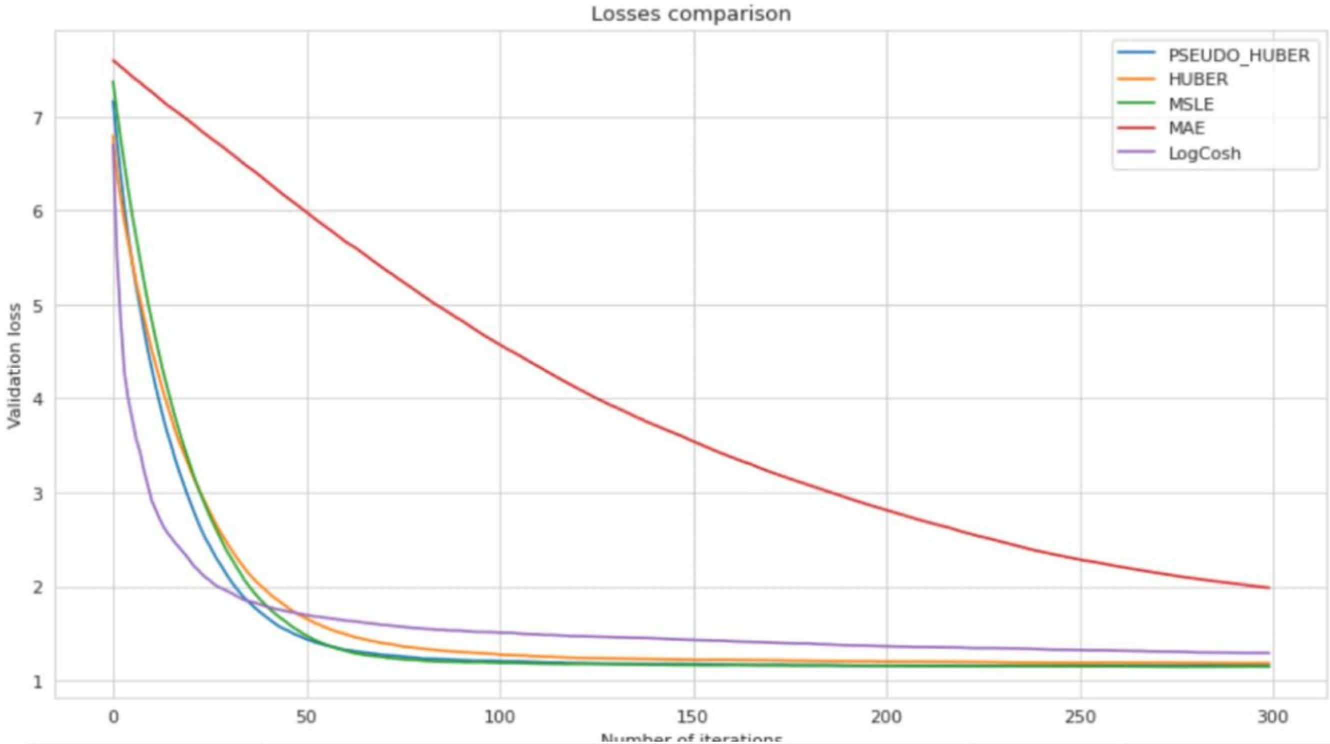
Results: House Prices dataset



	Pseudo Huber	Huber	MSLE	MAE	LogCosh
Test loss	22818	58577	**16914**	62387	60336

Dependence on delta: House Prices dataset

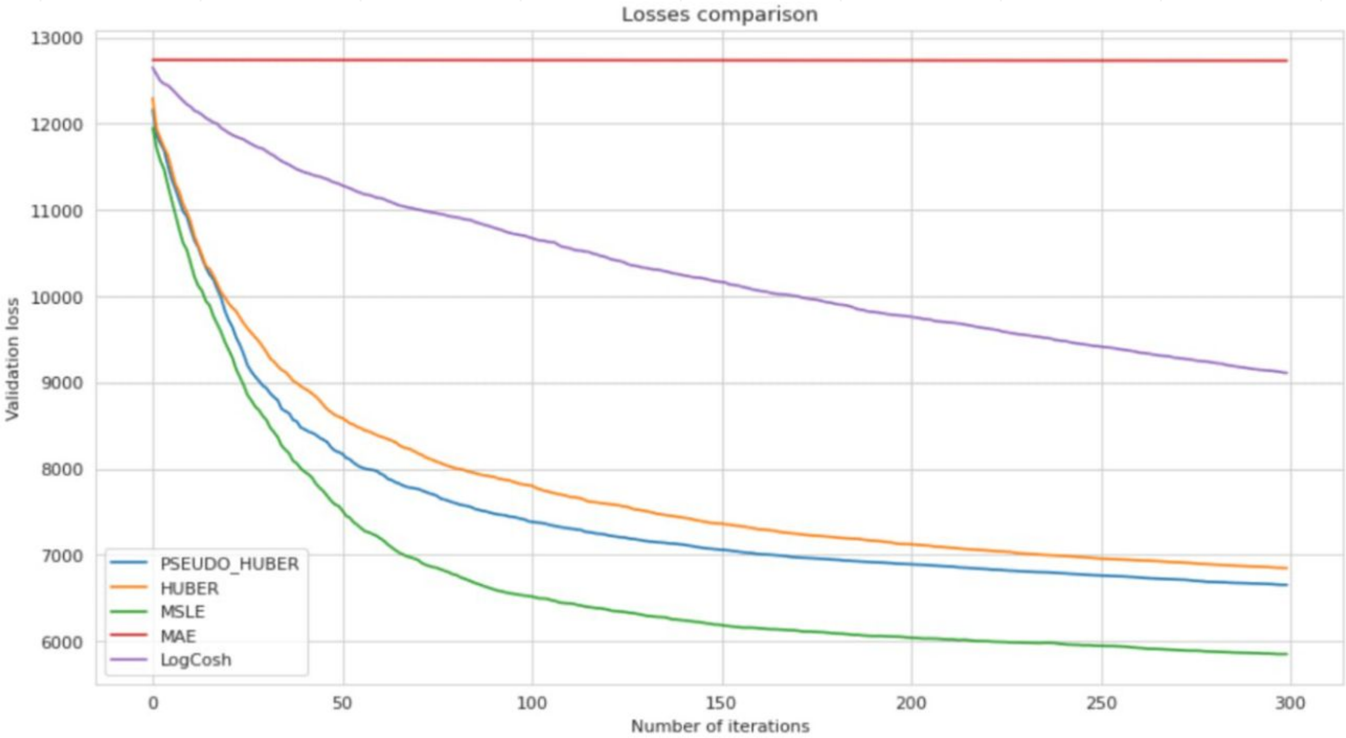
Results: Life Expectancy dataset



	Pseudo Huber	Huber	MSLE	MAE	LogCosh
Test loss	1.17	1.16	**1.13**	1.20	1.34

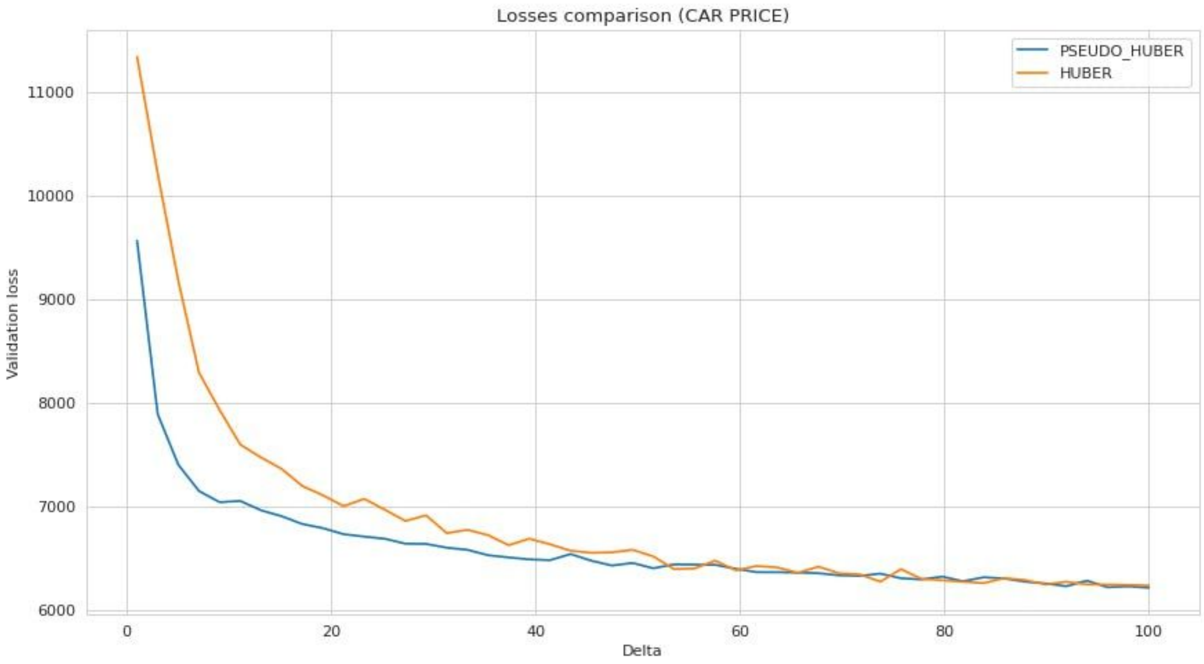
Dependence on delta: Life Expectancy dataset

Results: Car Price Prediction dataset



	Pseudo Huber	Huber	MSLE	MAE	LogCosh
Test loss	5872	5941	**5868**	12691	6892

Results: Car Price Prediction dataset



Conclusion

- As expected, MAE is always a bad choice, maybe with the exception of the Life Expectancy dataset (there the scale of the target is suitable for this loss)

- MSLE always performs the best - astonishing results on the House Prices dataset illustrate this unequivocally

- The losses for Huber/Pseudo Huber mostly tend to decrease with the larger values of delta - although the plots for House Prices and Life Expectancy show some nontrivial dependencies

THANKS!

Stas Pyatkin

Dmitriy Kornilov

Danil Ivanov

Danil Gusak

Bari Khairullin