1 Tuning with DoubleML

1.1 Simulation Study: Tuning with DoubleML

In DoubleML, it is not yet clear, how the parameters should be tuned for the learners. There are basically three to four different "strategies". The simulation study is run to gain insights into and evidence for the different approaches. The performance of the learners depends on the choice of parameters.

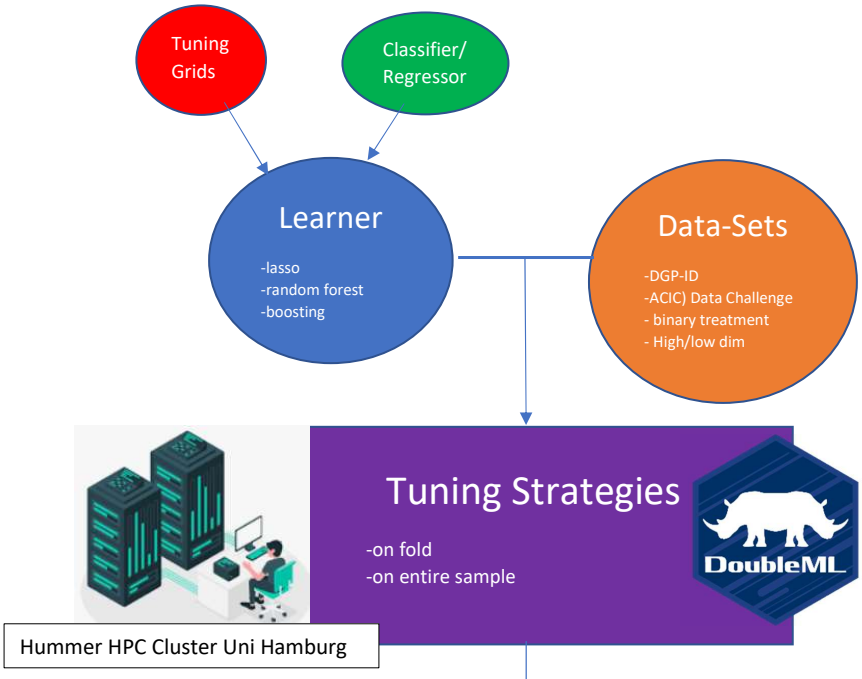
The open question is how the parameter tuning interacts with the sample splitting involved in the double machine learning approach. In the original papers, it's just written, that the model need to be trained on a training sample and predictions should be generated on a hold-out/test sample. Hence, it is important to distinguish between two different "types" of sample splits. The first type being split by folds according to the double machine learning algorithm / cross-fitting from Chernozhukov et al. (2018). The second, splitting into train/test/validation samples for parameter tuning. These folds are not necessarily identical. For example, the entire sample could be split into 5 folds, perform the tuning using cross-validation and then pass the obtained parameters to DoubleML. The sample splits used in cross-fitting and tuning are not necessarily the same. Thus, it's interesting to see, how the following strategies perform relatively to each other.

First, no tuning at all - this can't be a good idea because there will never be a one-size-fits-all parametrization. However, that's a helpful benchmark to illustrate the need to tune parameters. Second, internal tuning on the entire sample. This will split the data into K folds, which are then used for the parameter tuning. The cross-fitting is performed independently from the tuning. Third, internal tuning on folds, will split the sample into the folds that are eventually used for cross-fitting. This procedure is computationally more intense. The fourt strategy is manual tuning on hold-out sample. Split the data set into two different parts - one is used for the cross-fitting part (call it sample A) and the other part only used for cross-validated parameter search (sample B). Different values for the sample splits, e.g., 30%, 50% and 70% will be tried.

Different learners are tried with the above explained strategies. Further the data setting of the data-sets varies (low-dimensional vs high-dimensional) The goal is to implement this first on a small data set and then on a greater number of data sets (100; 250; 500). Because of the large number of data sets the simulation needs to be run on a HPC server. The data for the simulation study is the data used in the 2019 ACIC challenge, because the data is quite realistic in some regards.

Current Progress of the Project: The tuning and fitting framework for six learners is implemented in python. A process is implemented, which chooses the best learner, based on the error score of the nuisance components. Preliminary results in terms of plots are presented for one data set, where bias, coverage and squared error is shown. Below is a flow-chart that visualises the process. The next steps of the project include creating parameter grids for the learners, setting up the different strategies, running the project on the HPC (High Performance Computing) server and comparing results. Finally the goal is running the script for many (100+) data sets from the 2019 ACIC data challenge and comparing results accross data sets and data generating processes.

Simulation Study: Hyperparameter Tuning with DoubleML Package



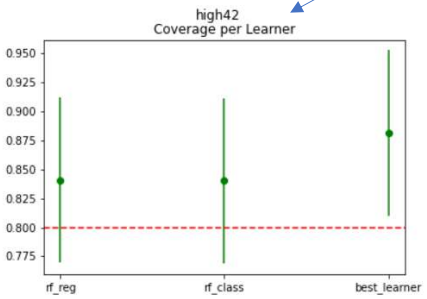
Results

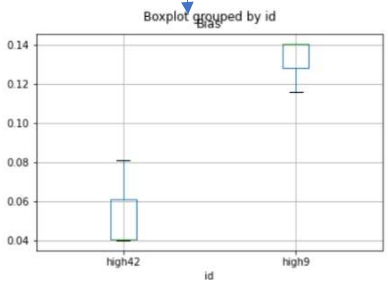
Coefficient	Confidence interval	True ATE	Date set ID	ml_g_tune	ml_m_tune	msr_type	Learner
0.84	0.7-0.9	0.8	High42	1.5	0.4	Acc/mse	lasso

Evaluation

Best learner

Plots





id	bias	squared error	coverage
high42	0.054105	0.001640	1.000000
high9	0.132542	0.019832	0.333333

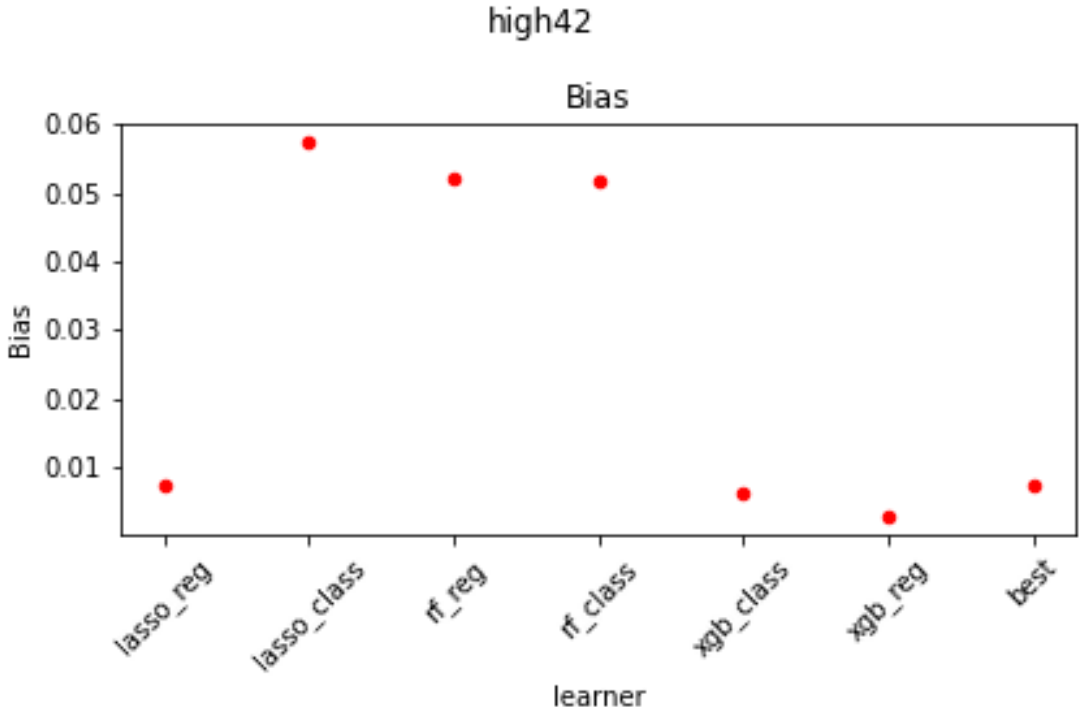
- DGP:
  - Based on ACIC 2019, see DGP description, 'id = "high42"

1.2 Results: Bias, Coverage, Squared Error

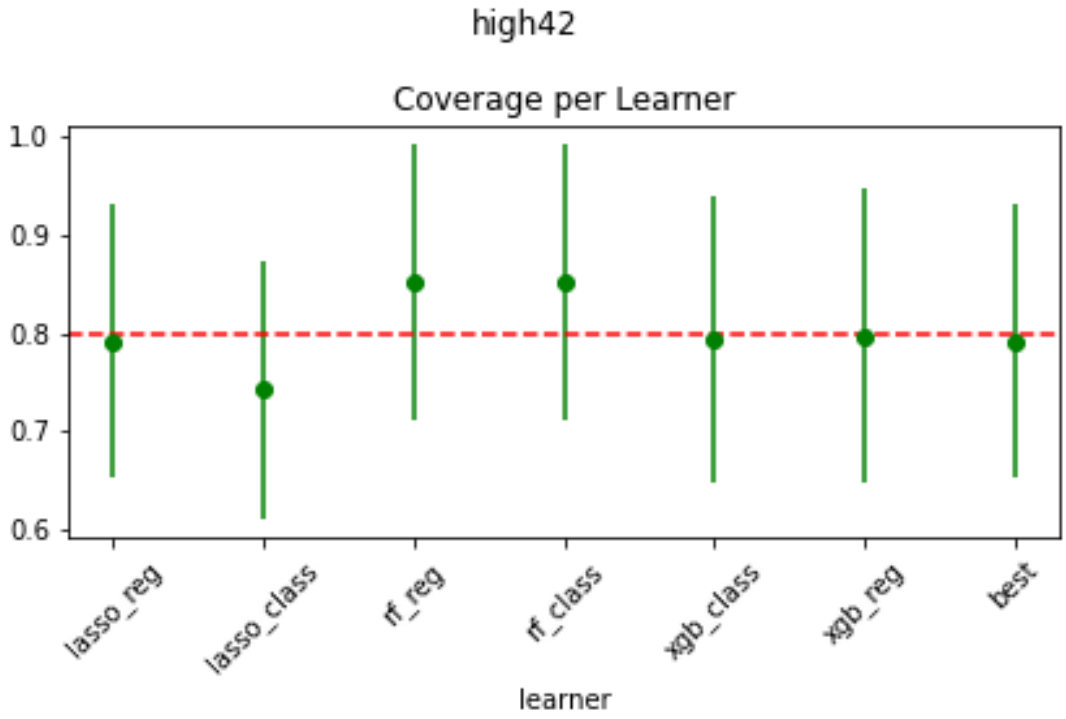
1.2.1 Coefficient estimates & confidence intervals

Example Data-Set

1.3 Bias



1.4 Coverage


high42
Coverage per Learner
1.0
0.9
0.8
0.7
0.6
lasso_reg
lasso_class
rf_reg
rf_class
xgb_class
xgb_reg
best
learner

1.4.1 Squared Error