DoubleML for Python and R

Tutorial: A state-of-the-art framework for double machine learning

Online Causal Inference Seminar, Stanford (virtual)

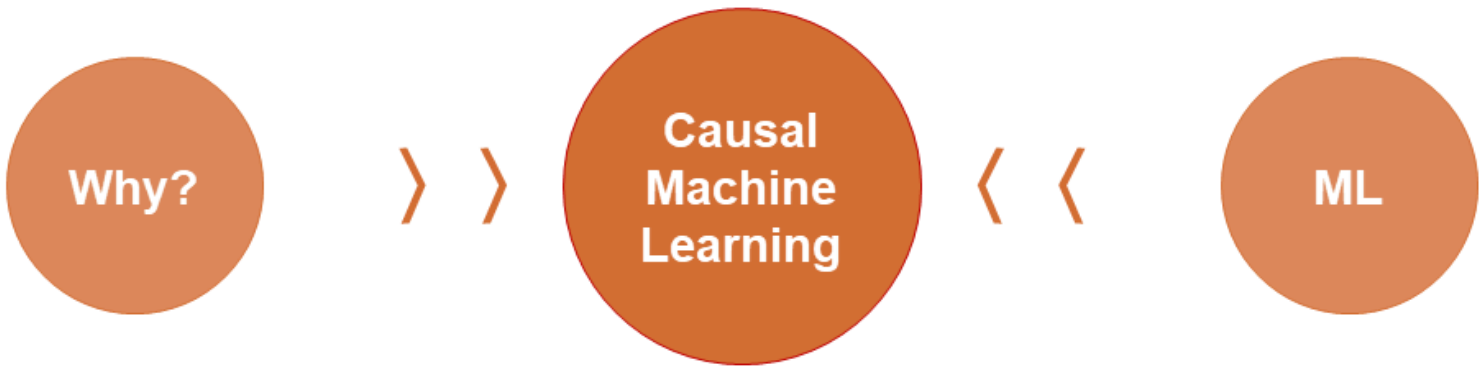
Philipp Bach[1], Victor Chernozhukov[2], Sven Klaassen[1,3], Malte Kurz[4], Martin Spindler[1,3]

[1]University of Hamburg, [2]MIT, [3]EconomicAI, [4]Technical Unversity of Munich

April 18, 2023

Motivation

Motivation for Causal ML



Causal Inference

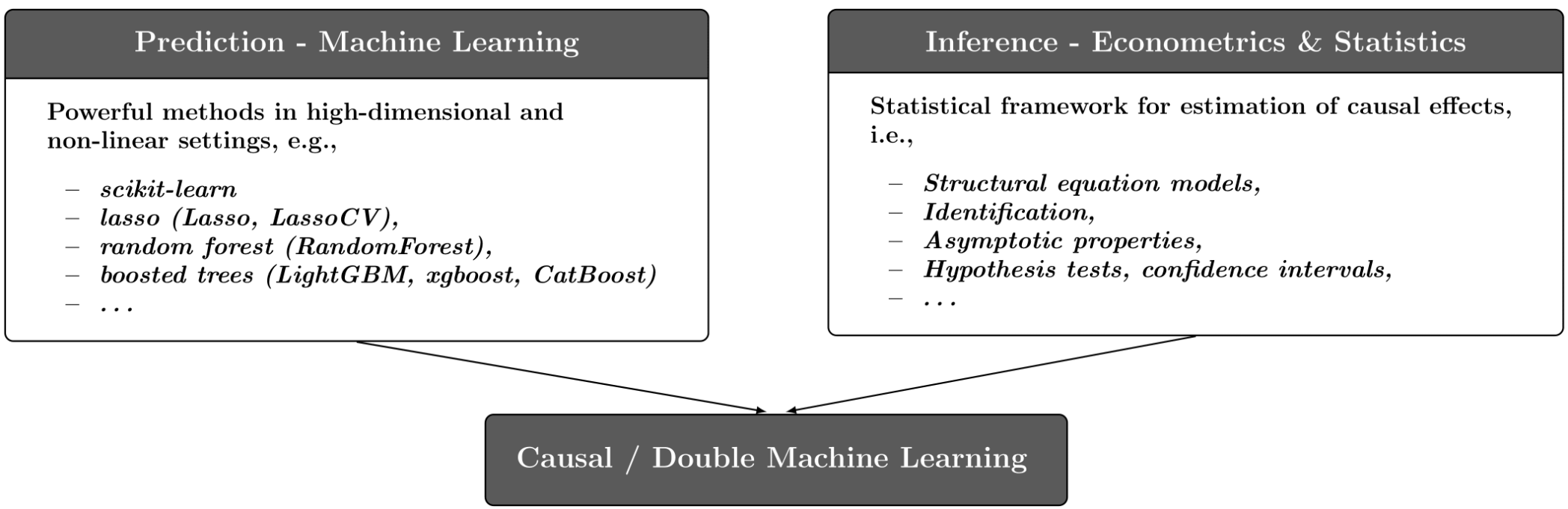
- Learning causal relationships
- Going beyond correlations
- Pioneers: Pearl, Rubin, Imbens (Nobel Prize 2021)

Machine Learning

- Learning complex patterns in data
- Based on association (correlation)
- Good at forecasting

What is Double/Debiased Machine Learning (DML)?

- DML is a general framework for **causal inference** and estimation of causal parameters based on **machine learning**
- Summarized in Chernozhukov et al. (2018)
- Combines the strengths of **machine learning** and **econometrics**



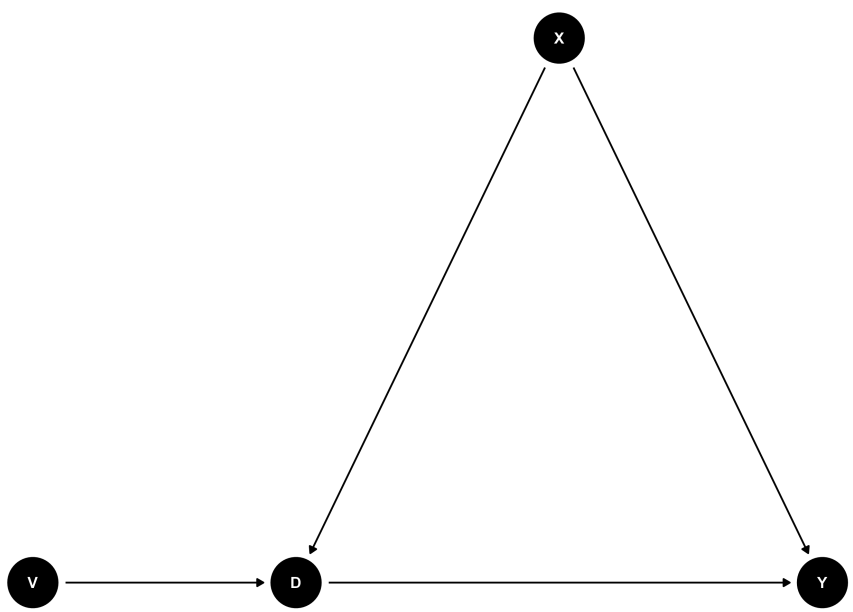
Motivating Example

Partially linear regression model (PLR)

$$
\begin{aligned}
Y &= D\theta_0 + g_0(X) + \zeta, & \mathbb{E}[\zeta|D, X] &= 0, \\
D &= m_0(X) + V, & \mathbb{E}[V|X] &= 0,
\end{aligned}
$$

with

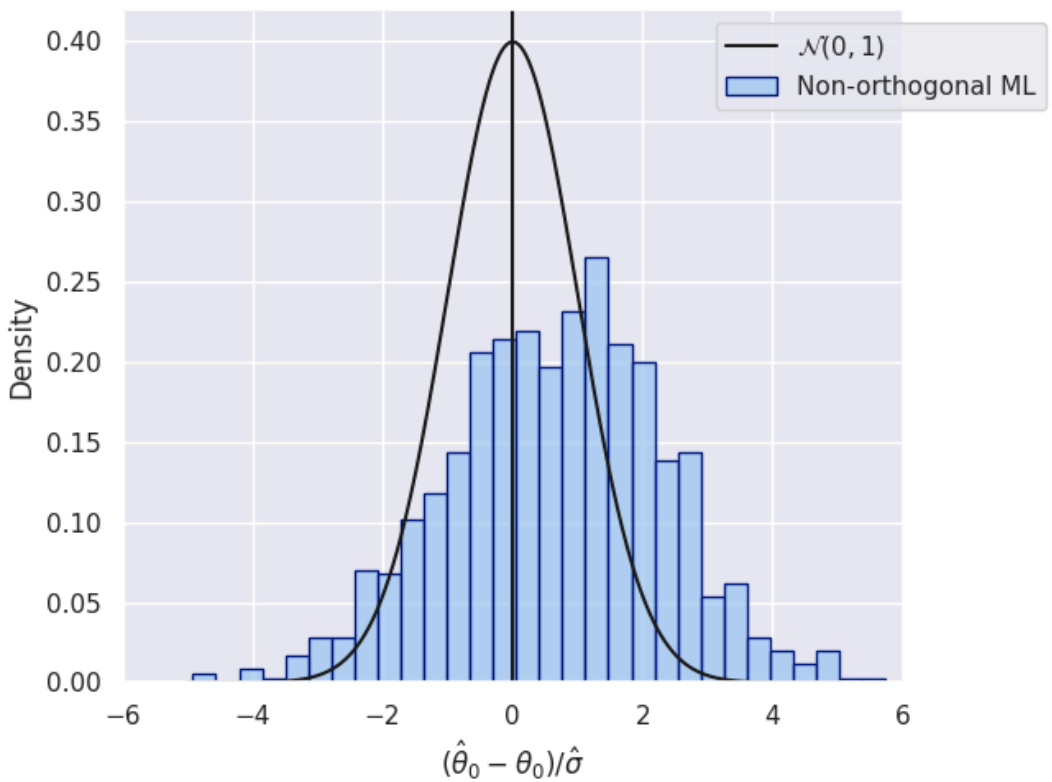
- Outcome variable Y
- Policy or treatment variable of interest D
- High-dimensional vector of confounding covariates $X = (X_1, \ldots, X_p)$
- Stochastic errors ζ and V



Motivating Example

Failure of naive approach

- What if we simply plug-in ML predictions $\hat{g}_0(X)$ for $g_0(X)$ into $Y = D\theta_0 + g_0(X) + \zeta$?
- See this example based on Chernozhukov et al. (2018)



Motivating Example

Solution to regularization bias: Orthogonalization

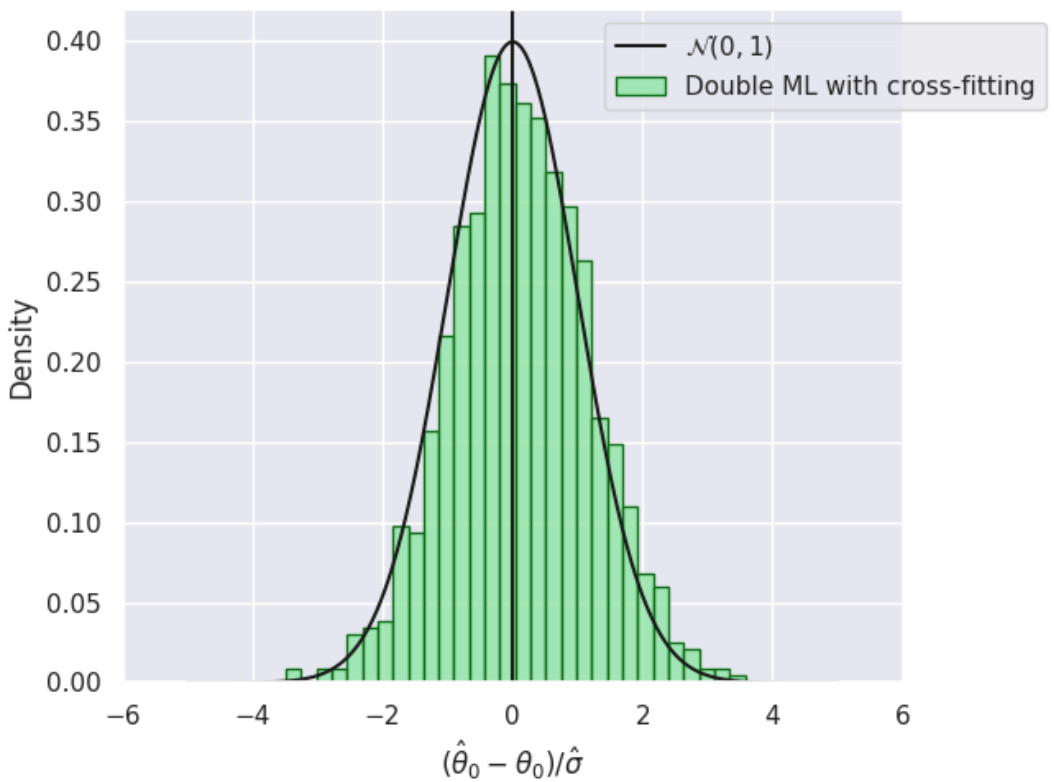
- Remember the *Frisch-Waugh-Lovell* (FWL) Theorem in a **linear regression model**

$$Y = D\theta_0 + X'\beta + \varepsilon$$

- θ_0 can be consistently estimated by **partialling out** X, i.e,
  1. OLS regression of Y on X: $\tilde{\beta} = (X'X)^{-1}X'Y \rightarrow$ Residuals $\hat{\varepsilon}$
  2. OLS regression of D on X: $\tilde{\gamma} = (X'X)^{-1}X'D \rightarrow$ Residuals $\hat{\zeta}$
  3. Final OLS regression of $\hat{\varepsilon}$ on $\hat{\zeta}$
- Orthogonalization: The idea of the FWL Theorem can be generalized to using ML estimators instead of OLS

Motivating Example

- Using an orthogonal score leads to an asymptotically normal estimator $\hat{\theta}_0$
- See this example based on Chernozhukov et al. (2018)



Neyman Orthogonality

Naive approach

$$\psi(W, \theta_0, \eta) = (Y - D\theta_0 - g_0(X))D$$

Regression adjustment score

$$\eta = g(X),$$
$$\eta_0 = g_0(X),$$

FWL partialling out

$$\psi(W, \theta_0, \eta_0) = ((Y - E[Y|X]) - (D - E[$$
$$(D - E[D|X])$$

Neyman-orthogonal score (Frisch-Waugh-Lovell)

$$\eta = (\ell(X), m(X)),$$
$$\eta_0 = (\ell_0(X), m_0(X)) = (\mathbb{E}[Y \mid X], \mathbb{E}[D \mid$$

Introduction to Double Machine Learning

DML Key Ingredients

1. Neyman Orthogonality

- Inference is based on a moment condition that satisfies the **Neyman orthogonality condition** $\psi(W; \theta, \eta)$

$$E[\psi(W; \theta_0, \eta_0)] = 0,$$

- where $W := (Y, D, X, Z)$ and with θ_0 being the unique solution that obeys the **Neyman orthogonality condition**

$$\partial_\eta \mathbb{E}[\psi(W; \theta_0, \eta)]|_{\eta=\eta_0} = 0.$$

- ∂_η denotes the pathwise (Gateaux) derivative operator

DML Key Ingredients

1. Neyman Orthogonality

- **Neyman orthogonality** ensures that the **moment condition** identifying θ_0 is **insensitive to small pertubations** of the nuisance function η around η_0
- Using a Neyman-orthogonal score **eliminates the first order biases** arising from the replacement of η_0 with a ML estimator $\hat{\eta}_0$
- PLR example: Partialling-out score function

$$\psi(\cdot) = (Y - E[Y|X] - \theta(D - E[D|X]))(D - E[D|X])$$

DML Key Ingredients

2. High-Quality Machine Learning Estimators

- The nuisance parameters are estimated with high-quality (fast-enough converging) machine learning methods.
- Different structural assumptions on η_0 lead to the use of different machine-learning tools for estimating η_0 Chernozhukov et al. (2018) (Section 3)
- Rate requirements depend on the causal model and orthogonal score, e.g. (see Chernozhukov et al. (2018)),
  - PLR, partialling out:
    $\|\hat{m}_0 - m_0\|_{P,2} \times \left(\|\hat{m}_0 - m_0\|_{P,2} + \|\hat{\ell}_0 - \ell_0\|_{P,2}\right) \leq \delta_N N^{-1/2}$
  - IRM/DR score, ATE: $\|\hat{m}_0 - m_0\|_{P,2} \times \|\hat{\ell}_0 - \ell_0\|_{P,2} \leq \delta_N N^{-1/2}$

DML Key Ingredients

3. Sample Splitting

- To avoid the biases arising from overfitting, a form of **sample splitting** is used at the stage of producing the estimator of the main parameter θ_0.
- Efficiency gains by using cross-fitting (swapping roles of samples for train / hold-out)

DML Key Ingredients

Main result in Chernozhukov et al. (2018)

> There exist regularity conditions, such that the DML estimator $\tilde{\theta}_0$ concentrates in a $1/\sqrt{N}$-neighborhood of θ_0 and the sampling error is approximately
>
> $$\sqrt{N}(\tilde{\theta}_0 - \theta_0) \sim N(0, \sigma^2),$$
>
> with
>
> $$\sigma^2 := J_0^{-2}\mathbb{E}(\psi^2(W; \theta_0, \eta_0)),$$
> $$J_0 = \mathbb{E}(\psi_a(W; \eta_0)).$$

- See this example based on Chernozhukov et al. (2018)

DoubleML - Implementation in Python and R

Main Dependencies

Python

ML Learners

- `scikit-learn` and similar interfaces, for example `XGBoost`, `LightGBM`, custom learners

Other dependencies

- `pandas`, `NumPy`, `SciPy`, `statsmodels`, `joblib`

R

ML Learners

- `mlr3`, `mlr3learners` and similar interfaces, for example `mlr3extralearners`, custom learners

Other dependencies

- `R6`, `data.table`, `mlr3tuning`, extensions[1] of `mlr3` (`mlr3pipelines`, ...)

Installation

Python

- Latest PyPi release

```
pip install -U DoubleML
```

- Latest conda-forge release

```
conda install -c conda-forge doubleml
```

- **Development version** from GitHub

```
git clone git@github.com:DoubleML/doubleml-for-
cd doubleml-for-py
pip install -editable .
```

R

- Latest CRAN release

```
install.packages("DoubleML")
```

- **Development version** from GitHub

```
remotes::install_github("DoubleML/doubleml-for-
```

Papers, User Guide, Resources

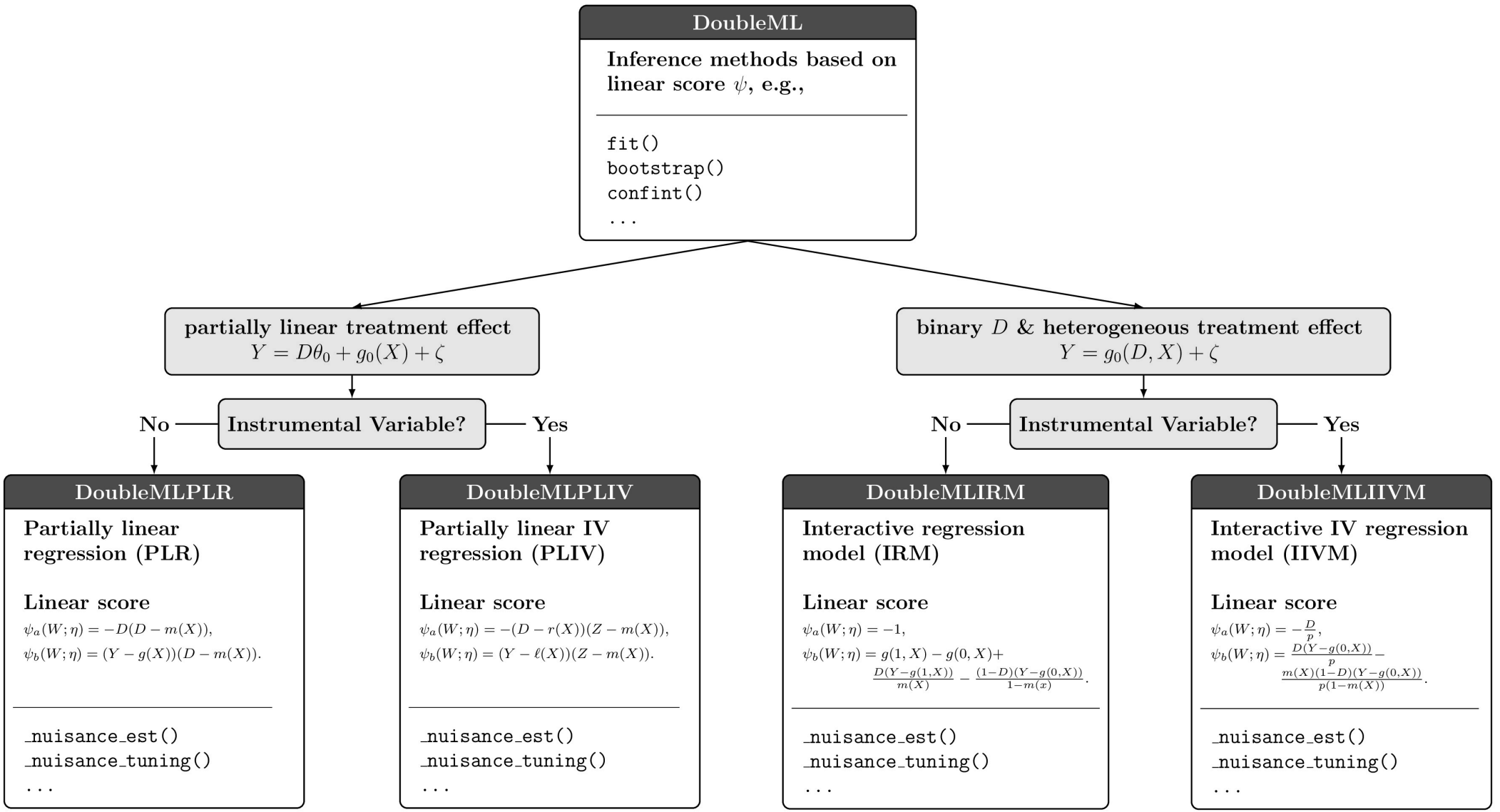
Papers

- R package- with a nontechnical introduction to DML: Bach et al. (2021)
- Python package: Bach et al. (2022)

Software implementation:

- https://github.com/DoubleML/doubleml-for-py
- https://github.com/DoubleML/doubleml-for-r
- Docu, user guide and examples: docs.doubleml.org and example gallery

Class Structure and Causal Models



Advantages of Object Orientation

- DoubleML gives the user a **high flexibility** with regard to specifications of DML models
  - Choice of ML learners for approximation of nuisance parameters
  - Different resampling schemes
  - DML algorithms (DML1, DML2)
  - Different Neyman-orthogonal score functions
- DoubleML can be **easily extended**
  - New model classes with appropriate Neyman-orthogonal score functions can be inherited from abstract base class DoubleML
  - Score functions can be provided as `callables` (`functions` in R)
  - Resampling schemes are customizable in a flexible way

Getting Started with DoubleML

DoubleML Workflow Example

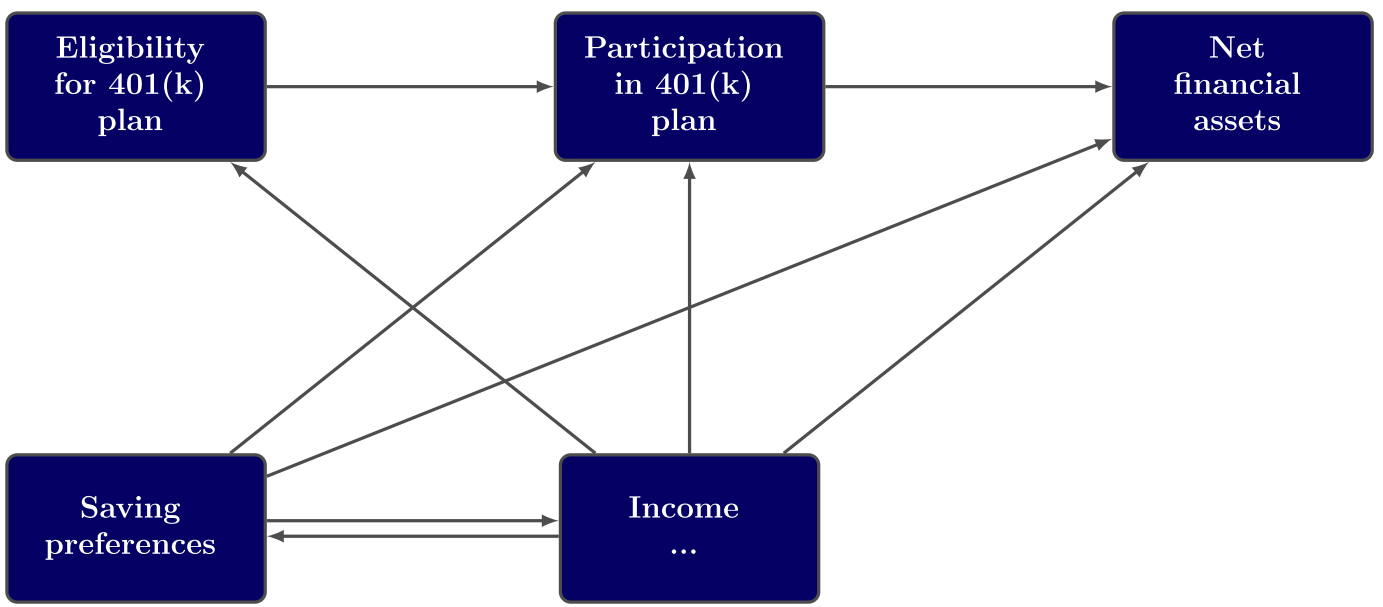
Workflow

0. Problem Formulation

1. Data-Backend
2. Causal Model
3. ML Methods
4. DML Specification
5. Estimation
6. Inference

0. Problem Formulation

- **401(k) Example**[1]
- Goal: Estimate ATE of eligibility in 401(k) pension plans on employees' net financial assets



1. Data-Backend

- Declare the roles for the **treatment** variable, the **outcome** variable and **controls**

Python | R

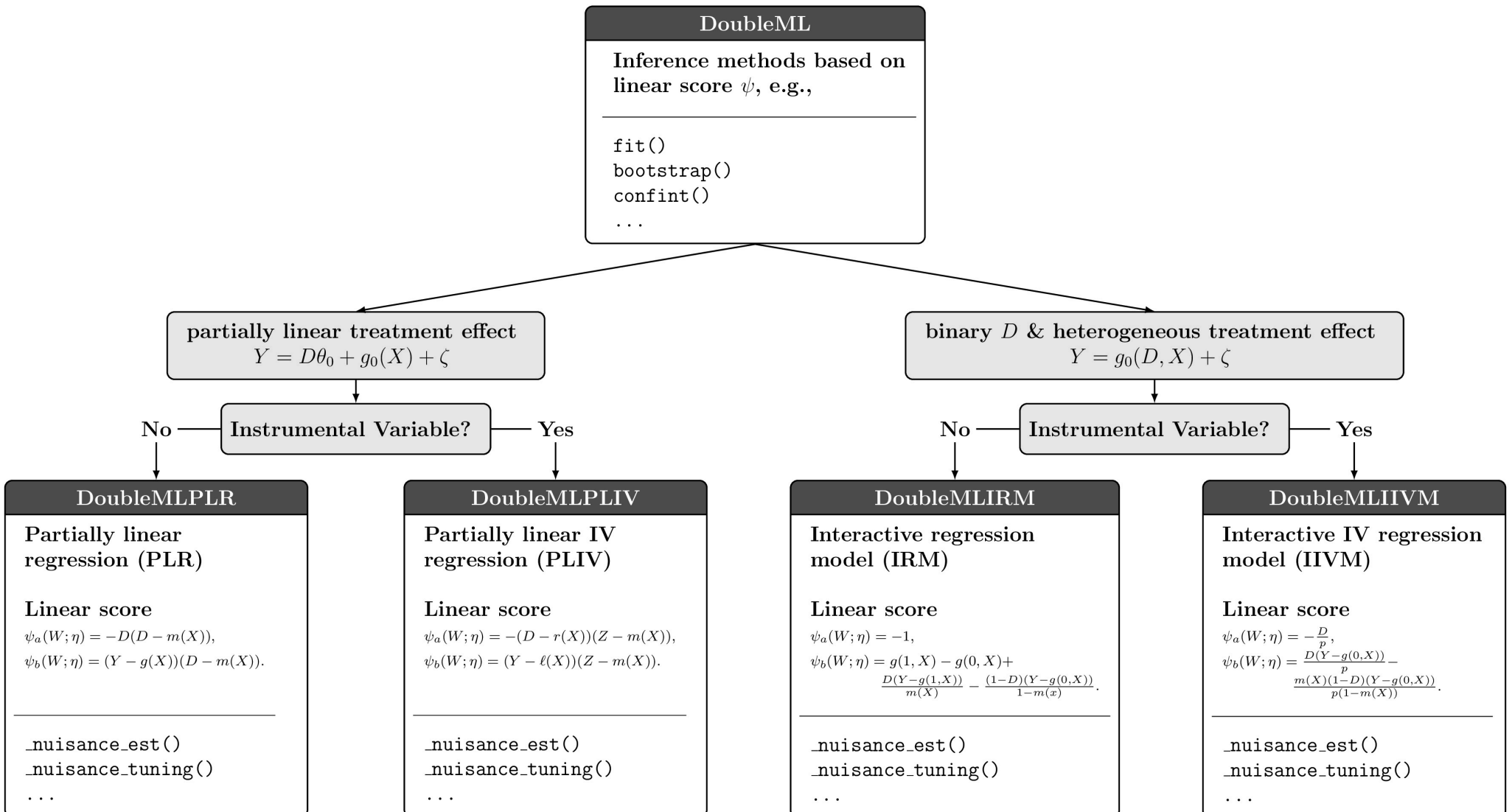
```python
from doubleml import DoubleMLData
from doubleml.datasets import fetch_401K

data = fetch_401K(return_type='DataFrame')

# Construct DoubleMLData object
dml_data = DoubleMLData(data,
                        y_col='net_tfa',
                        d_cols='e401',
                        x_cols=['age', 'inc', 'educ', 'fsize', 'marr',
                                'twoearn', 'db', 'pira', 'hown'])
```

2. Causal Model

- Choose your **DoubleML** model



3. ML Methods

- Initialize the **learners** with hyperparameters

Python | R

```python
# Random forest learners
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

ml_l_rf = RandomForestRegressor(n_estimators = 500, max_depth = 7,
                                max_features = 3, min_samples_leaf = 3)

ml_m_rf = RandomForestClassifier(n_estimators = 500, max_depth = 5,
                                max_features = 4, min_samples_leaf = 7)


# Xgboost learners
from xgboost import XGBClassifier, XGBRegressor

ml_l_xgb = XGBRegressor(objective = "reg:squarederror", eta = 0.1,
                        n_estimators =35)

ml_m_xgb = XGBClassifier(objective = "binary:logistic", eta = 0.1, n_estimators = 34,
                         use_label_encoder = False ,eval_metric = "logloss")
```

4. DML Specifications

- Initialize the model `DoubleML` object

Python | R

```python
import numpy as np
from doubleml import DoubleMLPLR
import numpy as np

np.random.seed(42)
# Default values
dml_plr_rf = DoubleMLPLR(dml_data,
                         ml_l = ml_l_rf,
                         ml_m = ml_m_rf)

np.random.seed(42)
# Parametrized by user
dml_plr_rf = DoubleMLPLR(dml_data,
                         ml_l = ml_l_rf,
                         ml_m = ml_m_rf,
                         n_folds = 3,
                         n_rep = 1,
                         score = 'partialling out',
                         dml_procedure = 'dml2')
```

5. Estimation

- Use the `fit()` method to estimate the model

Python | R

```python
# Estimation
dml_plr_rf.fit()
```

```python
# Coefficient estimate
dml_plr_rf.coef
```

```python
# Standard error
dml_plr_rf.se
```

```python
# Summary
dml_plr_rf.summary
```

5. Estimation

- For an overview on DoubleML objects use the `print()` method

Python | R

```
# Estimation
print(dml_plr_rf)
```

6. Inference

- For confidence intervals use the `confint()` method

Python | R

```python
# Summary
dml_plr_rf.summary
```

```python
# Confidence intervals
dml_plr_rf.confint(level=0.95)
```

```python
# Multiplier bootstrap (relevant in case with multiple treatment variables)
_ = dml_plr_rf.bootstrap()

dml_plr_rf.confint(joint = True)
```

Outlook: Extensions of DoubleML

Implemented Extensions

- Simultaneous Inference for Multiple Treatments
- Clustered Standard Errors
- Group Average Treatment Effects (GATEs)
- Conditional Average Treatment Effects (CATEs)
- (Local) Quantile Treatment Effects (QTEs)
- Effects on Conditional Value at Risk (CVaR)

Cluster Robust DoubleML (Chiang et al. 2022)

Python | R

▶ Code

```
# initializion from pandas.DataFrame
dml_cluster_data = DoubleMLClusterData(df, y_col='Y', d_cols='D',  z_cols='Z',
                                       cluster_cols=['cluster_var_i', 'cluster_var_j'])

dml_pliv_obj = DoubleMLPLIV(dml_cluster_data, ml_l=clone(learner), ml_m=clone(learner), ml_r=clone(le
_ = dml_pliv_obj.fit()
print(dml_pliv_obj)
```

GATEs and CATEs

- Build GATEs and CATEs on basic IRM models
- Estimation is based on the best linear predictor (Semenova and Chernozhukov 2021)

```
from doubleml import DoubleMLIRM

dml_irm = DoubleMLIRM(dml_data,
                      ml_g = ml_l_rf,
                      ml_m = ml_m_rf,
                      n_folds = 3,
                      n_rep = 1)

_ = dml_irm.fit()
dml_irm.summary
```

Group Average Treatment Effects (GATEs)

- Estimate GATEs for different income groups (above and below income median)

```
import pandas as pd
groups =  pd.DataFrame(columns=['Group'], index=range(dml_data.data["inc"].shape[0]), dtype=str)
for i, x_i in enumerate(dml_data.data["inc"]):
    if x_i <= dml_data.data["inc"].median():
         groups['Group'][i] = '1'
    else:
         groups['Group'][i] = '2'

gate = dml_irm.gate(groups=groups)
ci_gate = gate.confint()
print(ci_gate)
```

```
ci_gate_joint = gate.confint(joint=True)
print(ci_gate_joint)
```

Conditional Average Treatment Effects (CATEs)

- Use e.g. a spline dictionary to estimate CATE for based on age

```
import pandas as pd
import patsy
age_data = dml_data.data["age"]
design_matrix = patsy.dmatrix("bs(age, df=6, degree=3)", {"age":age_data})
spline_basis = pd.DataFrame(design_matrix)

cate = dml_irm.cate(spline_basis)
print(cate)
```

Conditional Average Treatment Effects (CATEs)

- Create confidence intervals based on a grid of values

```
# create a confidence band
new_data = {"age": np.linspace(np.quantile(age_data, 0.2), np.quantile(age_data, 0.8), 50)}
spline_grid = pd.DataFrame(patsy.build_design_matrices([design_matrix.design_info], new_data)[0])
df_cate = cate.confint(spline_grid, level=0.95, joint=True, n_rep_boot=2000)
print(df_cate.head(n=8))
```

Conditional Average Treatment Effects (CATEs)

▶ Code

Quantile Treatement Effects (QTEs)

- QTEs (Kallus, Mao, and Uehara 2019) are implemented via a separate class

```python
from doubleml import DoubleMLQTE
from lightgbm import LGBMClassifier, LGBMRegressor
from sklearn.base import clone

tau_vec = np.arange(0.1,0.95,0.2)
n_folds = 5

# Learners
class_learner = LGBMClassifier(n_estimators=300, learning_rate=0.05, num_leaves=10)

np.random.seed(42)
dml_QTE = DoubleMLQTE(dml_data, ml_g=clone(class_learner), ml_m=clone(class_learner),
                      quantiles=tau_vec, score='PQ', normalize_ipw=True)
_ = dml_QTE.fit()
print(dml_QTE)
```

Quantile Treatement Effects (QTEs)

- Create simultaneously valid confidence intervals

```
_ = dml_QTE.bootstrap(n_rep_boot=2000)
ci_QTE = dml_QTE.confint(level=0.95, joint=True)

print(ci_QTE)
```

Quantile Treatement Effects (QTEs)

- Code

Call for Collaboration

Future Extensions

- DoubleML for difference-in-differences models
- AutoDML
- Sensitivity analysis for omitted variable bias
- Support for unstructured data
- Copula models

Collaborators Welcome!

- Please contact us if you are interested in contributing to DoubleML
  - Issues,
  - Examples,
  - Extending model classes and learners
- Contributing guidelines for R and Python available online

Thank you!

Contact

In case you have questions or comments, feel free to contact us

philipp.bach@uni-hamburg.de

sven.klaassen@uni-hamburg.de

martin.spindler@uni-hamburg.de

Hexagon Stickers

👇 Order a *free* package sticker 👇

https://forms.gle/CWAHEh8RxQJi8V3m9

Acknowledgement

We gratefully acknowledge support by EconomicAI 🙏

EconomicAI - Causal ML for Business Applications.

References

References

Bach, Philipp, Victor Chernozhukov, Malte S Kurz, and Martin Spindler. 2021. "DoubleML – An Object-Oriented Implementation of Double Machine Learning in R." https://arxiv.org/abs/2103.09603.

———. 2022. "DoubleML-an Object-Oriented Implementation of Double Machine Learning in Python." *Journal of Machine Learning Research* 23: 53–51.

Becker, Marc, Michel Lang, Jakob Richter, Bernd Bischl, and Daniel Schalk. 2020. *Mlr3tuning: Tuning for 'Mlr3'*. https://CRAN.R-project.org/package=mlr3tuning.

Binder, Martin, Florian Pfisterer, Michel Lang, Lennart Schneider, Lars Kotthoff, and Bernd Bischl. 2021. "Mlr3pipelines - Flexible Machine Learning Pipelines in r." *Journal of Machine Learning Research* 22 (184): 1–7. http://jmlr.org/papers/v22/21-0281.html.

Chang, Winston. 2020. *R6: Encapsulated Classes with Reference Semantics*. https://CRAN.R-project.org/package=R6.

Chen, Tianqi, and Carlos Guestrin. 2016. "XGBoost: A Scalable Tree Boosting System." In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–94. KDD '16. San Francisco, California, USA.

Chernozhukov, Victor, Denis Chetverikov, Mert Demirer, Esther Duflo, Christian Hansen, Whitney Newey, and James Robins. 2018. "Double/Debiased Machine Learning for Treatment and Structural Parameters." *The Econometrics Journal* 21 (1): C1–68. https://onlinelibrary.wiley.com/doi/abs/10.1111/ectj.12097.

Chiang, Harold D, Kengo Kato, Yukun Ma, and Yuya Sasaki. 2022. "Multiway Cluster Robust Double/Debiased Machine Learning." *Journal of Business & Economic Statistics* 40 (3): 1046–56.

Dowle, Matt, and Arun Srinivasan. 2020. *Data.table: Extension of 'Data.frame'*. https://CRAN.R-project.org/package=data.table.

Harris, C. R., K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, et al. 2020. "Array Programming with NumPy." *Nature* 585 (7825): 357–62. https://doi.org/10.1038/s41586-020-2649-2.

Kallus, Nathan, Xiaojie Mao, and Masatoshi Uehara. 2019. "Localized Debiased Machine Learning: Efficient Inference on Quantile Treatment Effects and Beyond." *arXiv Preprint arXiv:1912.12945*.

Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. "Lightgbm: A Highly Efficient Gradient Boosting Decision Tree." *Advances in Neural Information Processing Systems* 30: 3146–54.

Lang, Michel, Quay Au, Stefan Coors, and Patrick Schratz. 2020. *Mlr3learners: Recommended Learners for 'Mlr3'*. https://CRAN.R-project.org/package=mlr3learners.

Lang, Michel, Martin Binder, Jakob Richter, Patrick Schratz, Florian Pfisterer, Stefan Coors, Quay Au, Giuseppe Casalicchio, Lars Kotthoff, and Bernd Bischl. 2019. "mlr3: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*. https://joss.theoj.org/papers/10.21105/joss.01903.

McKinney, W. 2010. "Data Structures for Statistical Computing in Python." In *Proceedings of the 9th Python in Science Conference*, 56–61. https://doi.org/10.25080/Majora-92bf1922-00a.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12 (85): 2825–30. http://jmlr.org/papers/v12/pedregosa11a.html.

Seabold, S., and J. Perktold. 2010. "Statsmodels: Econometric and Statistical Modeling with Python." In *Proceedings of the 9th Python in Science Conference*, 92–96. https://doi.org/10.25080/Majora-92bf1922-011.

Semenova, Vira, and Victor Chernozhukov. 2021. "Debiased Machine Learning of Conditional Average Treatment Effects and Other Causal Functions." *The Econometrics Journal* 24 (2): 264–89.

Sonabend, Raphael, and Patrick Schratz. 2020. *Mlr3extralearners: Extra Learners for Mlr3*.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, et al. 2020. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." *Nature Methods* 17: 261–72. https://doi.org/10.1038/s41592-019-0686-2.