Optimization and Algorithms in Sparse Regression

Screening Rules, Coordinate Descent, and Normalization

Johan Larsson

Department of Statistics, Lund University

June 28, 2024

Synopsis

Overarching Theme

Practical concerns in sparse regression: speed, accuracy, and reproducibility.

Synopsis

Overarching Theme

Practical concerns in sparse regression: speed, accuracy, and reproducibility.

Overview

I–III Screening rules

IV Benchmarking optimization methods

V Coordinate descent (an optimization method) for SLOPE

VI Normalization and regularization

The Strong Screening Rule for SLOPE (Paper I)

The Strong Screening Rule for SLOPE

Published and presented at NeurIPS 2020[1]

Co-written with supervisors Małgorzata Bogdan and Jonas Wallin.

[1] Johan Larsson, Małgorzata Bogdan, and Jonas Wallin (Dec. 6–12, 2020). "The Strong Screening Rule for SLOPE". In: *Advances in Neural Information Processing Systems 33*. 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Ed. by Hugo Larochelle et al. Vol. 33. Virtual: Curran Associates, Inc., pp. 14592–14603. ISBN: 978-1-71382-954-6

Motivation

- Data sets are growing in size.

Motivation

- Data sets are growing in size.
- Fitting models to sets of millions (maybe billions) of features is computationally costly.

Motivation

- Data sets are growing in size.
- Fitting models to sets of millions (maybe billions) of features is computationally costly.
- Hyperparameter tuning increases costs further.

Screening Rules

Basic Insight

- The support set is small in sparse regression, especially when $p \gg n$.[2]
- Can we somehow figure this support **before** fitting the model?

[2] The lasso can for instance at most select $\min(n, p)$ features.

Screening Rules

Basic Insight

- The support set is small in sparse regression, especially when $p \gg n$.[2]
- Can we somehow figure this support **before** fitting the model?

General Idea

1. Estimate how likely it is that a feature is in the support set.
2. If unlikely, discard it.
3. Fit a reduced model.
4. If we were wrong, just refit the model with missing features added.

[2] The lasso can for instance at most select $\min(n, p)$ features.

Screening Rules

Basic Insight

- The support set is small in sparse regression, especially when $p \gg n$.[2]
- Can we somehow figure this support **before** fitting the model?

General Idea

1. Estimate how likely it is that a feature is in the support set.
2. If unlikely, discard it.
3. Fit a reduced model.
4. If we were wrong, just refit the model with missing features added.

Turns out to be a pretty good idea!

[2] The lasso can for instance at most select $\min(n, p)$ features.

Optimality Conditions and the Gradient

Optimality Condition

β is optimal if

$$\mathbf{0} \in \nabla g(\beta) + \partial h(\beta),$$

where $\partial h(\beta)$ is the subdifferential of the penalty term h.

Optimality Conditions and the Gradient

Optimality Condition

β is optimal if

$$\mathbf{0} \in \nabla g(\beta) + \partial h(\beta),$$

where $\partial h(\beta)$ is the subdifferential of the penalty term h.

The Lasso

Here we have

$$\partial h(\beta)_j = \begin{cases} \{\lambda \operatorname{sign}(\beta_j)\} & \text{if } \beta_j \neq 0, \\ [-\lambda, \lambda] & \text{if } \beta_j = 0. \end{cases}$$

If $|\nabla g(\beta^*)_j| < \lambda$, then $\beta_j^* = 0$.

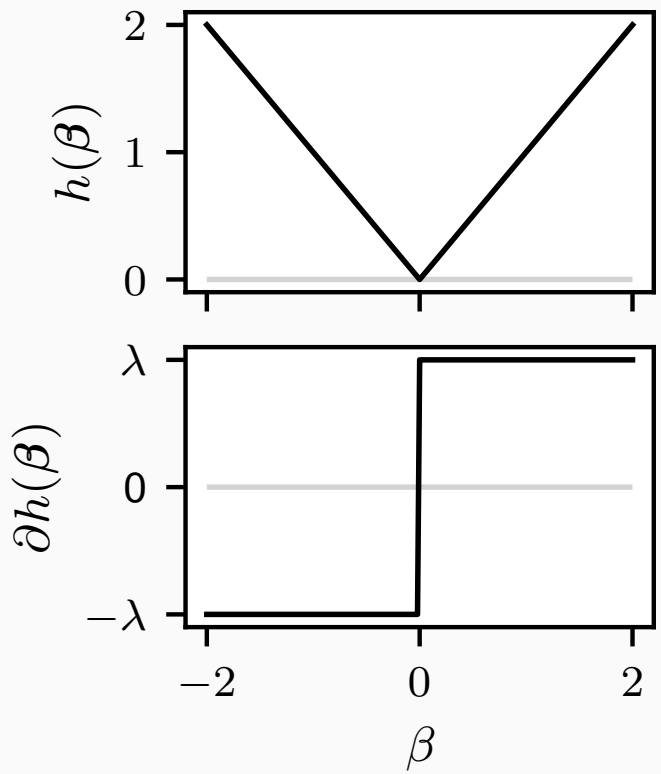


Figure 1: The lasso penalty $h(\beta) = \lambda\|\beta\|_1$ and its subdifferential at $\lambda = 1$.

The Lasso Path

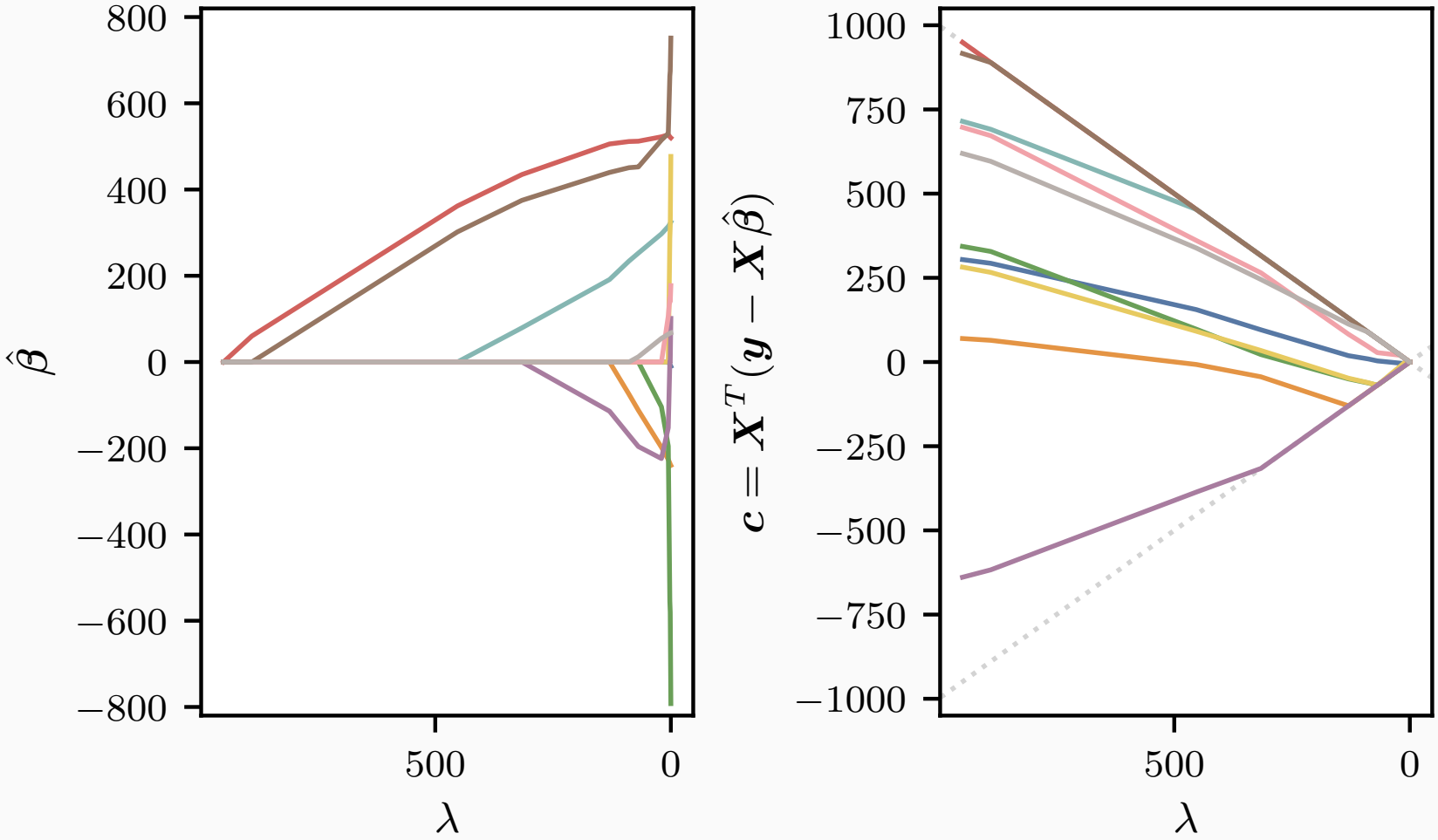


Figure 2: Coefficients and correlations (c) along the lasso path

Strong Rule for the Lasso

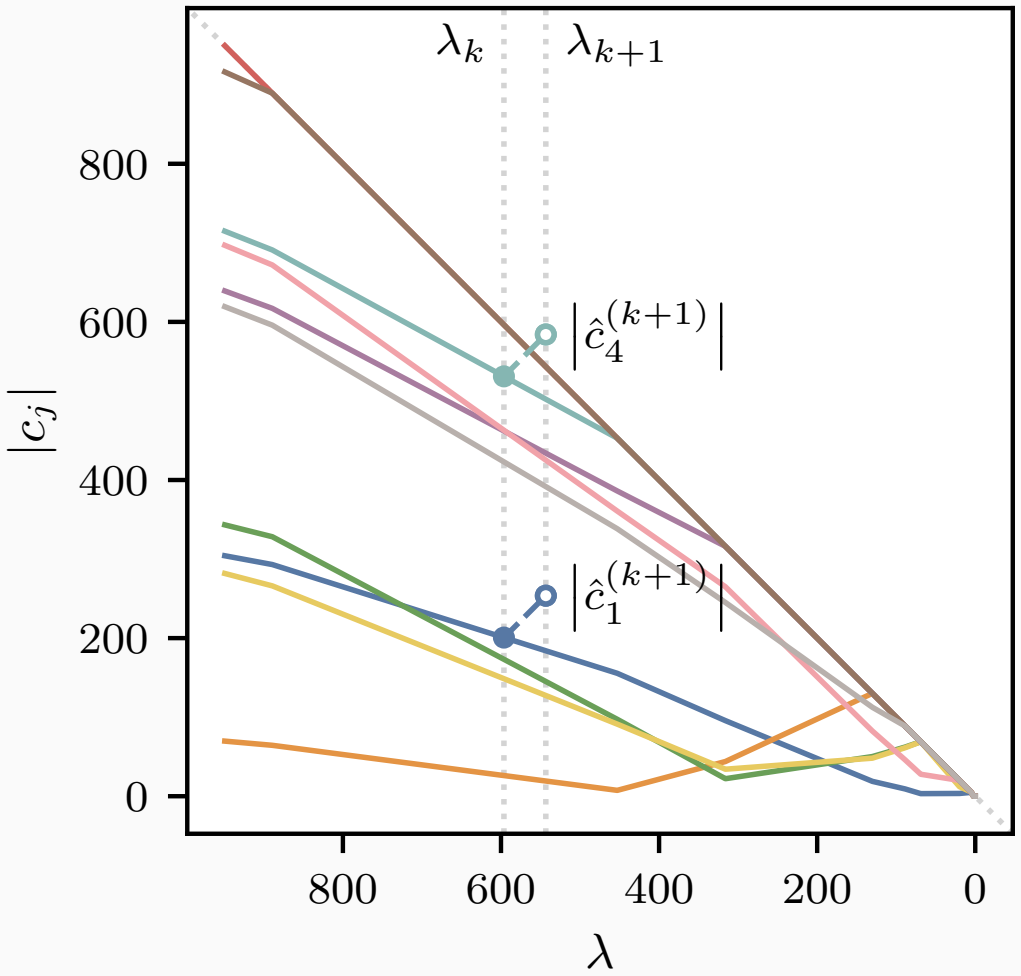


Figure 3: The strong rule for the lasso (Tibshirani et al. 2012). $|\hat{c}_j^{(k+1)}| = |c_j^{(k)}| + \lambda_k - \lambda_{k+1}$ is the absolute correlation estimate for feature j at step $k+1$.

Strong Rule for SLOPE

Same idea as for lasso strong rule! Just need the subdifferential.

SLOPE Subdifferential
The set of all $\boldsymbol{g} \in \mathbb{R}^p$ such that

$$\boldsymbol{g}_{\mathcal{A}_i} = \left\{ \boldsymbol{s} \in \mathbb{R}^{|\mathcal{A}_i|} \;\middle|\; \begin{cases} \operatorname{cumsum}(|\boldsymbol{s}|_\downarrow - \boldsymbol{\lambda}_{R(\boldsymbol{s})_{\mathcal{A}_i}}) \preceq \mathbf{0} & \text{if } \beta_{\mathcal{A}_i} = \mathbf{0}, \\ \operatorname{cumsum}(|\boldsymbol{s}|_\downarrow - \boldsymbol{\lambda}_{R(\boldsymbol{s})_{\mathcal{A}_i}}) \preceq \mathbf{0} & \\ \quad \text{and } \operatorname{sign}(\boldsymbol{\beta}_{\mathcal{A}_i}) = \operatorname{sign}(\boldsymbol{s}) & \\ \quad \text{and } \sum_{j \in \mathcal{A}_i} \left(|s_j| - \lambda_{R(\boldsymbol{s})_j} \right) = 0 & \text{otherwise.} \end{cases} \right\}$$

- $\mathcal{A}_i$ defines a cluster containing indices of coefficients equal in absolute value.
- $R(\boldsymbol{x})$ returns ranks of absolute values of elements in $\boldsymbol{x}$.
- $|\boldsymbol{x}|_\downarrow$ returns the absolute values of x sorted in non-increasing order.

Results: Effectiveness

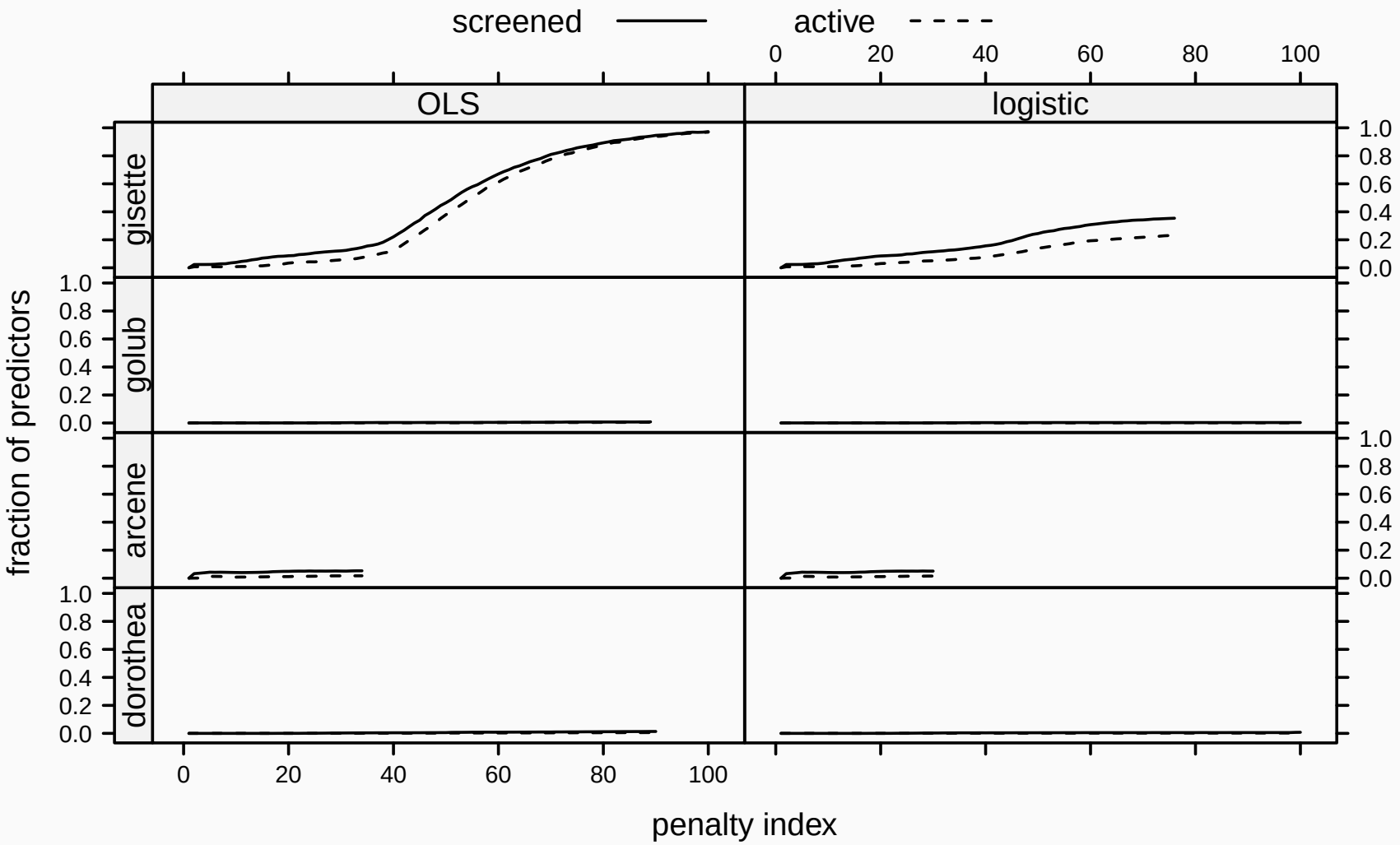


Figure 4: Effectiveness for some real data sets

Results: Speed

Table 1: Time to fit lasso paths using either the strong screening rule or no rule.

				Time (s)	
Dataset	Model	n	p	No Screening	Screening
dorothea	Logistic	800	88 119	914	14
e2006-tfidf	Least squares	3308	150 358	43 353	4944
news20	Multinomial	1000	62 061	5485	517
physician	Poisson	4406	25	34	34

Summary

Contributions

- The first screening rule for SLOPE
- A new formulation of the subdifferential for SLOPE
- Implementation in the R package `SLOPE`[3]

[3] https://doi.org/10.32614/CRAN.package.SLOPE
[4] As opposed to *safe*.

Summary

Contributions

- The first screening rule for SLOPE
- A new formulation of the subdifferential for SLOPE
- Implementation in the R package `SLOPE`[3]

Limitations

- Somewhat conservative
- Heuristic[4] rule, so violations may occur. But are not problematic in practice.

[3] https://doi.org/10.32614/CRAN.package.SLOPE
[4] As opposed to *safe*.

Look-Ahead Screening Rules for the Lasso (Paper II)

Look-Ahead Screening Rules for the Lasso

Published in EYSM 2021[5]

[5] Johan Larsson (Sept. 6, 2021). "Look-Ahead Screening Rules for the Lasso". In: *22nd European Young Statisticians Meeting - Proceedings*. 22nd European Young Statisticians Meeting. Ed. by Andreas Makridis et al. Athens, Greece: Panteion university of social and political sciences, pp. 61–65. ISBN: 978-960-7943-23-1

Motivation

- Even if screening rules improve the speed remarkably, they still involve costs (e.g. gradient computations).

Motivation

- Even if screening rules improve the speed remarkably, they still involve costs (e.g. gradient computations).
- Previous screening rules just look one-step ahead.

Motivation

- Even if screening rules improve the speed remarkably, they still involve costs (e.g. gradient computations).
- Previous screening rules just look one-step ahead.
- But information from the current step is useful for more than the next one, so why not look further ahead?

Gap Safe Screening

Gap Safe screening (Fercoq, Gramfort, and Salmon 2015) uses the **duality gap** as a basis for a safe screening rule, and discards the jth predictor if

$$|\boldsymbol{x}_j^\intercal \boldsymbol{\theta}_\lambda| + \|\boldsymbol{x}_j\|_2 \sqrt{\frac{1}{\lambda_+^2}\big(\mathcal{P}(\boldsymbol{\beta}_\lambda; \lambda_+) - \mathcal{D}(\boldsymbol{\theta}_\lambda; \lambda_+)\big)} < 1 \tag{1}$$

Gap Safe Screening

Gap Safe screening (Fercoq, Gramfort, and Salmon 2015) uses the **duality gap** as a basis for a safe screening rule, and discards the jth predictor if

$$|\boldsymbol{x}_j^\intercal \boldsymbol{\theta}_\lambda| + \|\boldsymbol{x}_j\|_2 \sqrt{\frac{1}{\lambda_+^2}\big(\mathcal{P}(\boldsymbol{\beta}_\lambda; \lambda_+) - \mathcal{D}(\boldsymbol{\theta}_\lambda; \lambda_+)\big)} < 1 \tag{1}$$

where $\boldsymbol{\theta}_\lambda$ is the scaled dual variable,

$$\boldsymbol{\theta}_\lambda = \frac{\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_\lambda}{\max\big(\|\boldsymbol{X}^\intercal(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_\lambda)\|_\infty, \lambda\big)}.$$

Gap Safe Screening

Gap Safe screening (Fercoq, Gramfort, and Salmon 2015) uses the **duality gap** as a basis for a safe screening rule, and discards the jth predictor if

$$|\boldsymbol{x}_j^\intercal \boldsymbol{\theta}_\lambda| + \|\boldsymbol{x}_j\|_2 \sqrt{\frac{1}{\lambda_+^2}\big(\mathcal{P}(\boldsymbol{\beta}_\lambda; \lambda_+) - \mathcal{D}(\boldsymbol{\theta}_\lambda; \lambda_+)\big)} < 1 \tag{1}$$

where $\boldsymbol{\theta}_\lambda$ is the scaled dual variable,

$$\boldsymbol{\theta}_\lambda = \frac{\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_\lambda}{\max\big(\|\boldsymbol{X}^\intercal(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_\lambda)\|_\infty, \lambda\big)}.$$

Equation (1) is quadratic in λ, so we can use it to find the next critical value for λ and screen for all steps ahead.

Simple Example

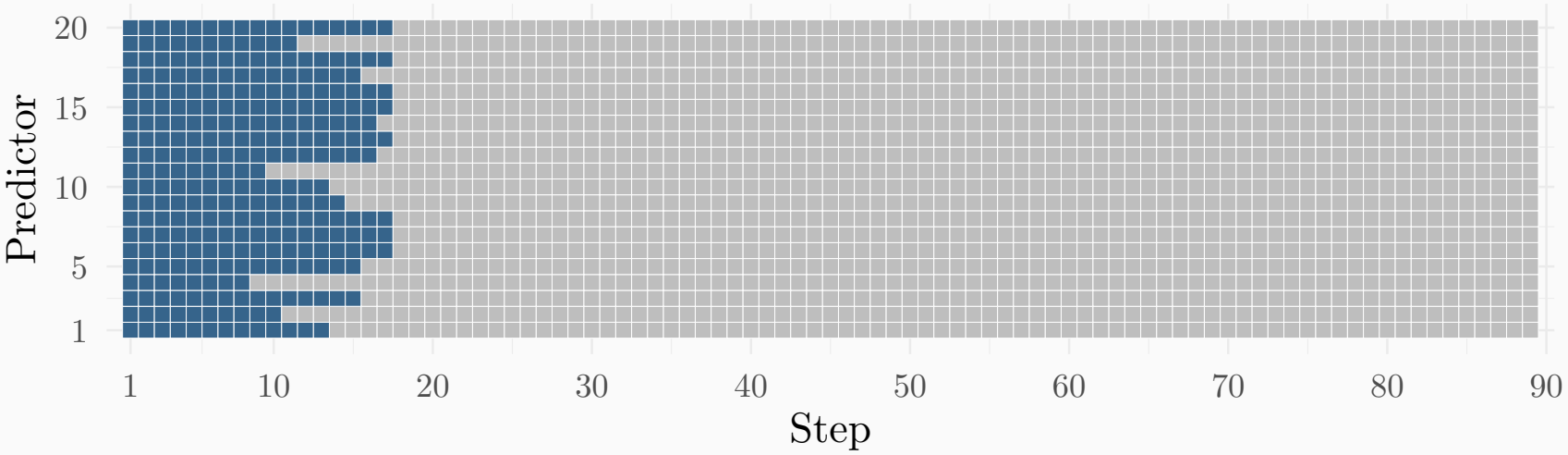


Figure 5: The lasso path for a sample of 20 features from the leukemia data set. The squares show for which steps the feature can be discarded.

Benchmarks

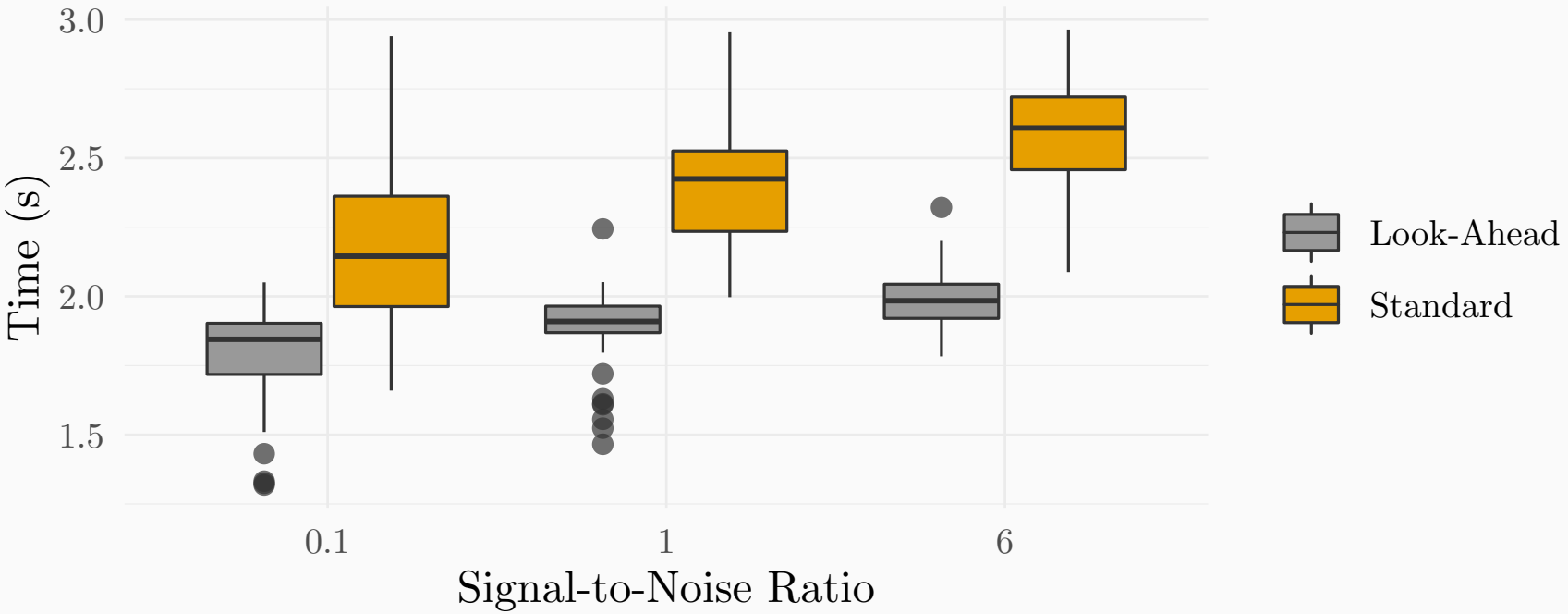


Figure 6: Benchmarks of time to fit the full lasso path with or without look-ahead screening.

Summary

Contributions

- Simple, safe, and essentially free screening rule for the lasso (and related problems)
- Moderate improvements in speed

Summary

Contributions

- Simple, safe, and essentially free screening rule for the lasso (and related problems)
- Moderate improvements in speed

Limitations

- Needs a safe rule

The Hessian Screening Rule (Paper III)

The Hessian Screening Rule

Published and presented at NeurIPS 2022[6]

Co-authored with Jonas

[6] Johan Larsson and Jonas Wallin (Nov. 28–Dec. 9, 2022). "The Hessian Screening Rule". In: *Advances in Neural Information Processing Systems 35*. 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Ed. by Sanmi Koyejo et al. Vol. 35. New Orleans, USA: Curran Associates, Inc., pp. 15823–15835. ISBN: 978-1-71387-108-8

Motivation

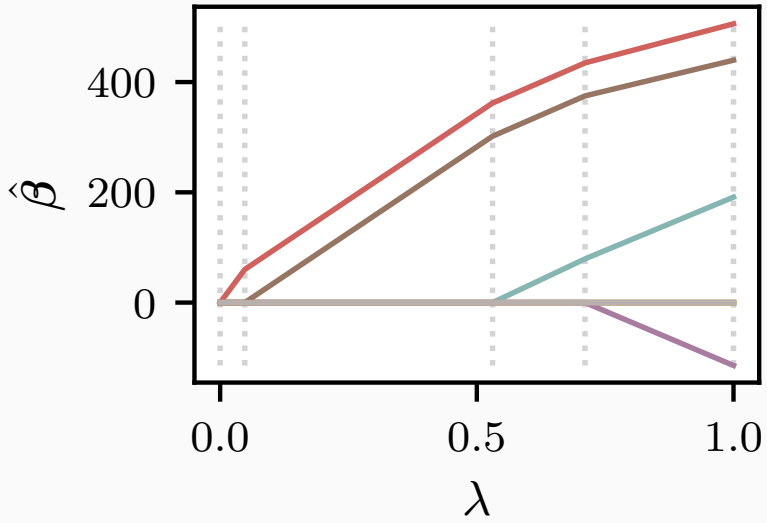
- Previous screening rules, like the strong rule, struggle with highly correlated features.
- Previous rules do not use information about the correlations along the lasso path.

The Hessian Screening Rule

For the ordinary lasso,

$$g(\beta) = \frac{1}{2}\|y - X\beta\|_2^2,$$
$$\nabla g(\beta) = X^\intercal(X\beta - y),$$

solution is piecewise linear.

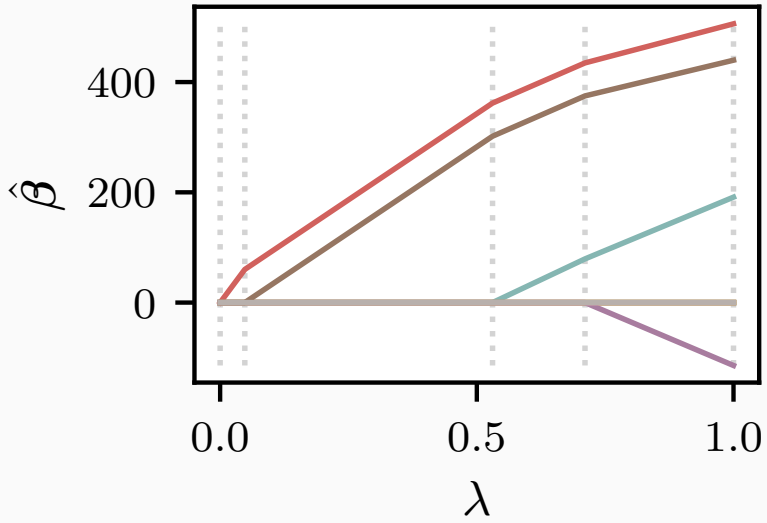


The Hessian Screening Rule

For the ordinary lasso,

$$g(\beta) = \frac{1}{2}\|y - X\beta\|_2^2,$$
$$\nabla g(\beta) = X^\intercal(X\beta - y),$$

solution is piecewise linear.



If active set is constant in $[\lambda_k, \lambda_{k+1}]$, we can express the solution as

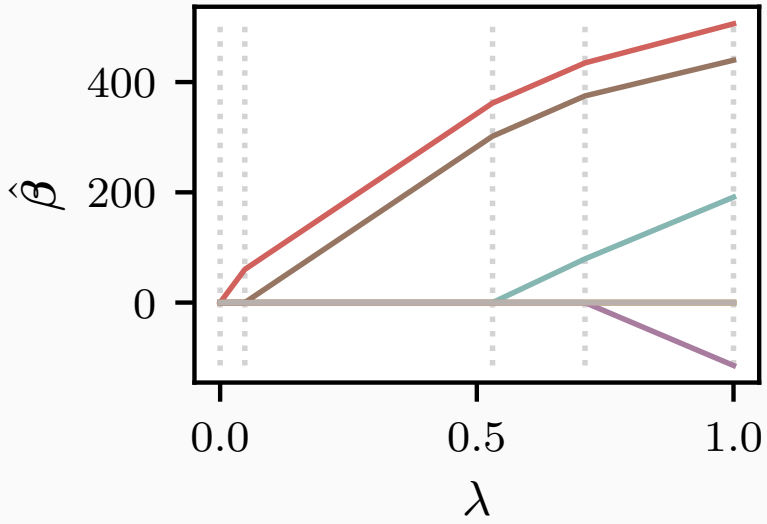
$$\hat{\beta}(\lambda_{k+1})_{\mathcal{A}} = \hat{\beta}(\lambda_k)_{\mathcal{A}} - (\lambda_k - \lambda_{k+1})\left(X_{\mathcal{A}}{}^\intercal X_{\mathcal{A}}\right)^{-1} \operatorname{sign}\left(\hat{\beta}(\lambda_k)_{\mathcal{A}}\right).$$

The Hessian Screening Rule

For the ordinary lasso,

$$g(\beta) = \frac{1}{2}\|y - X\beta\|_2^2,$$
$$\nabla g(\beta) = X^\intercal(X\beta - y),$$

solution is piecewise linear.



If active set is constant in $[\lambda_k, \lambda_{k+1}]$, we can express the solution as

$$\hat\beta(\lambda_{k+1})_\mathcal{A} = \hat\beta(\lambda_k)_\mathcal{A} - (\lambda_k - \lambda_{k+1})\left(X_\mathcal{A}^\intercal X_\mathcal{A}\right)^{-1} \operatorname{sign}\left(\hat\beta(\lambda_k)_\mathcal{A}\right).$$

Hessian Screening Rule (Basic Form)

Plug $\hat\beta$ into the gradient at step $k+1$:

$$\begin{aligned}\tilde{c}^H(\lambda_{k+1}) &= -\nabla g\left(\hat\beta(\lambda_{k+1})_\mathcal{A}\right) \\ &= c(\lambda_k) + (\lambda_{k+1} - \lambda_k) X^\intercal X_\mathcal{A}\left(X_\mathcal{A}^\intercal X_\mathcal{A}\right)^{-1} \operatorname{sign}\left(\hat\beta(\lambda_k)_\mathcal{A}\right).\end{aligned}$$

Screening Rule Comparison

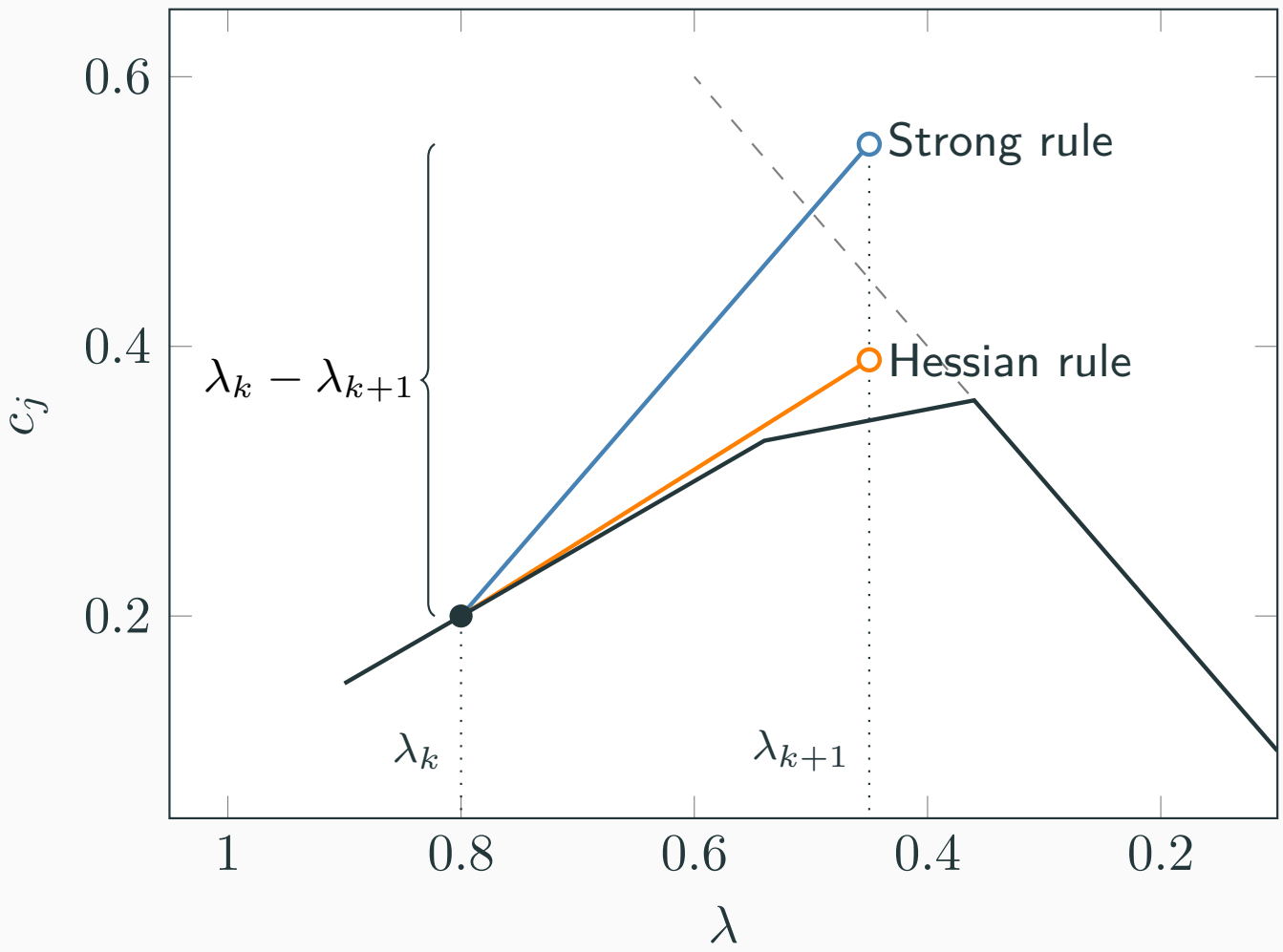


Figure 7: The strong and Hessian screening rules in action

Results: Effectiveness

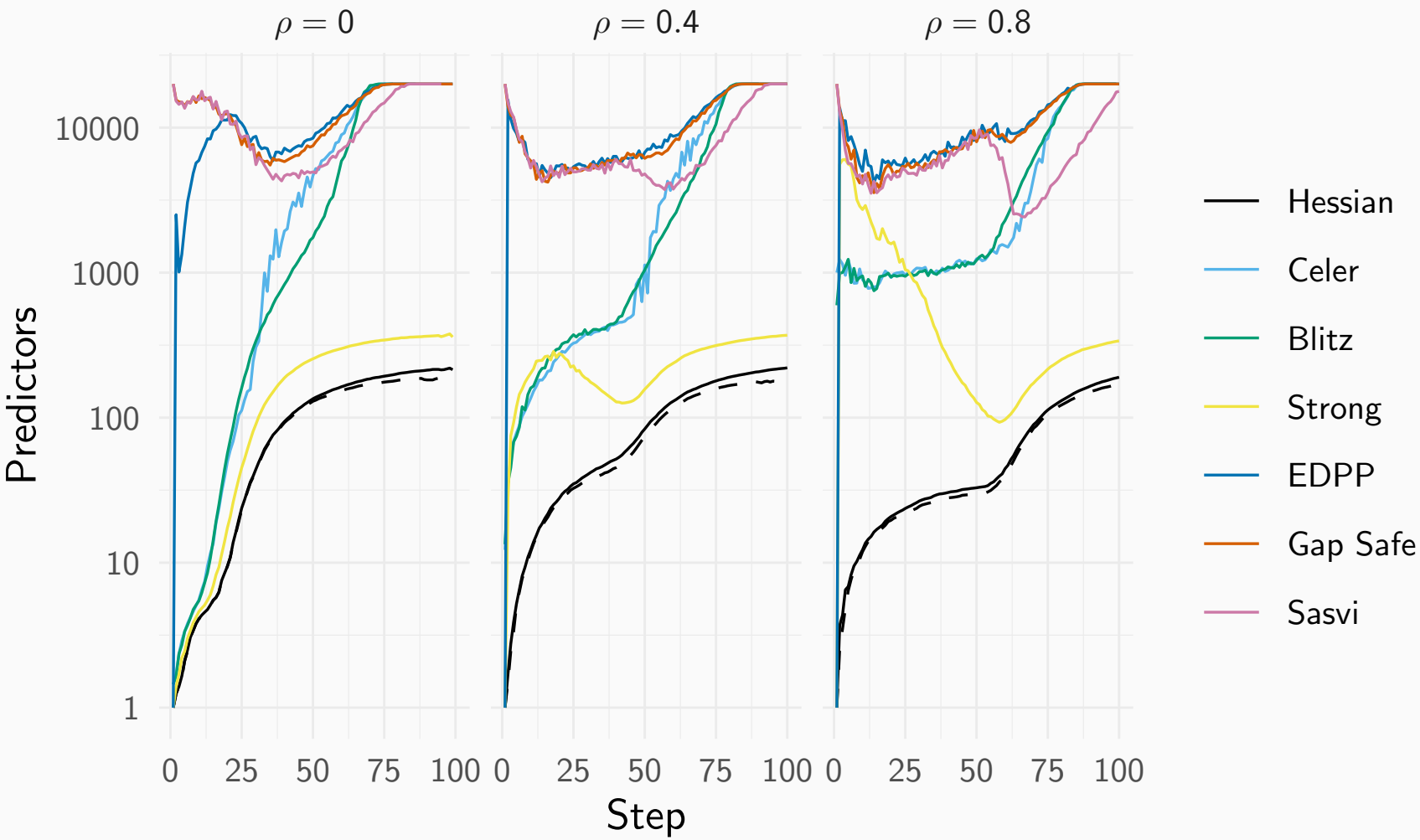


Figure 8: Features screened for a lasso path for ℓ_1-regularized least-squares to a design with varying correlation (ρ), $n = 200$, and $p = 20\,000$.

Results: Simulated Data

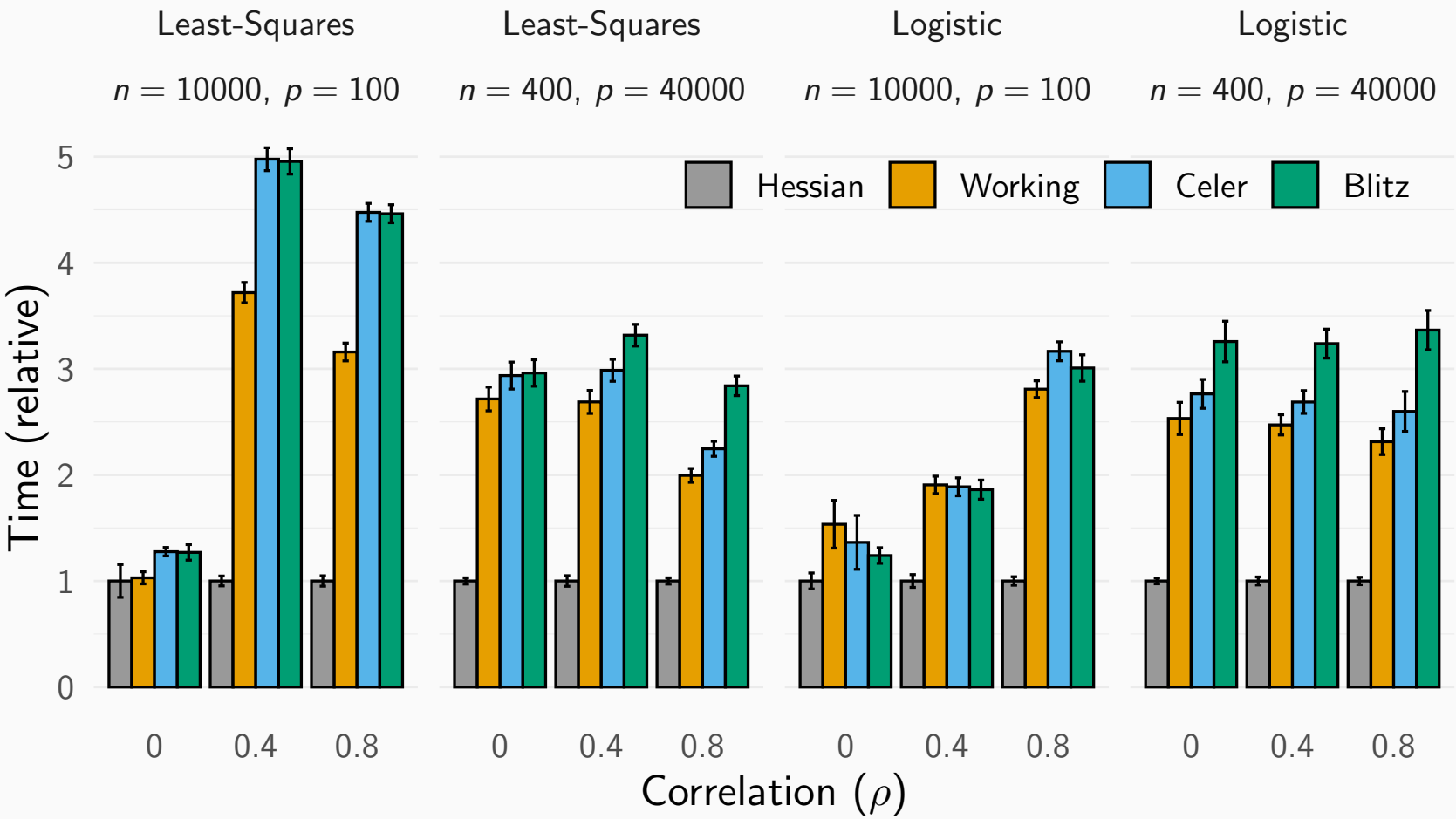


Figure 9: Time to fit a full path of ℓ_1-regularized least-squares and logistic regression to a design with n observations, p features, and pairwise correlation between features of ρ.

Summary

Contributions

- New screening rule that performs better in general and particularly for highly correlated features
- Works well for both least-squares and logistic regression

Summary

Contributions

- New screening rule that performs better in general and particularly for highly correlated features
- Works well for both least-squares and logistic regression

Limitations

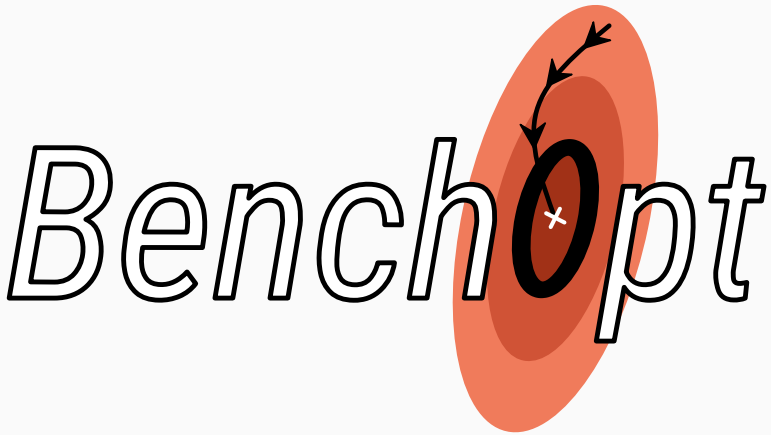
- Not optimal for problems with more complex Hessians
- Implementation is a bit more involved.

Benchopt: Reproducible, Efficient and Collaborative Optimization Benchmarks (Paper IV)

Benchopt

Published and presented at NeurIPS 2022[7]

Too many authors (20) to list here!



[7] Thomas Moreau et al. (Nov. 28–Dec. 9, 2022). "Benchopt: Reproducible, Efficient and Collaborative Optimization Benchmarks". In: *Advances in Neural Information Processing Systems 35*. 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Ed. by S. Koyejo et al. Vol. 35. New Orleans, USA: Curran Associates, Inc., pp. 25404–25421. ISBN: 978-1-71387-108-8

Motivation

- Surging number of optimization methods

A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AccAdaGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamPSGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Figure 10: Methods benchmarked in Schmidt, Schneider, and Hennig (2021)

Motivation

- Surging number of optimization methods
- Optimal choice depends on problem

A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AccAdaGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamPSGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Figure 10: Methods benchmarked in Schmidt, Schneider, and Hennig (2021)

Motivation

- Surging number of optimization methods
- Optimal choice depends on problem
  - Data set (dimensions, sparsity, conditioning, normalization)

A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AccAdaGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamPSGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Figure 10: Methods benchmarked in Schmidt, Schneider, and Hennig (2021)

Motivation

- Surging number of optimization methods
- Optimal choice depends on problem
  - Data set (dimensions, sparsity, conditioning, normalization)
  - Hyper-parameters (regularization)

A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	LAdam/LAMomentum	(Roliček & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamPSGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Figure 10: Methods benchmarked in Schmidt, Schneider, and Hennig (2021)

Motivation

- Surging number of optimization methods
- Optimal choice depends on problem
  - Data set (dimensions, sparsity, conditioning, normalization)
  - Hyper-parameters (regularization)
  - Hardware (GPU acceleration)

A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	LAdam/LAMomentum	(Roliček & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamPSGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Figure 10: Methods benchmarked in Schmidt, Schneider, and Hennig (2021)

Motivation

- Surging number of optimization methods
- Optimal choice depends on problem
  - Data set (dimensions, sparsity, conditioning, normalization)
  - Hyper-parameters (regularization)
  - Hardware (GPU acceleration)
- Reproducibility

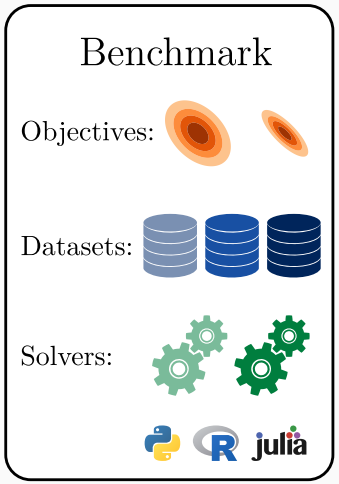
A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

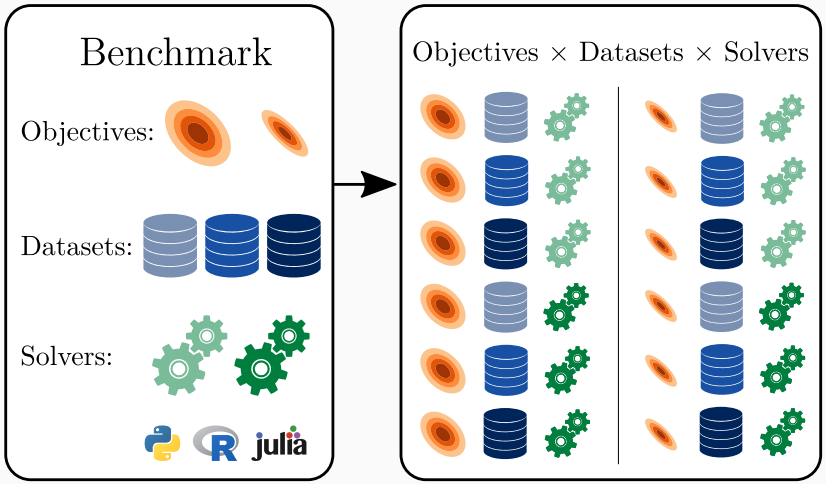
Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	LAdam/LAMomentum	(Roliček & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamPSGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zaleh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

Figure 10: Methods benchmarked in Schmidt, Schneider, and Hennig (2021)

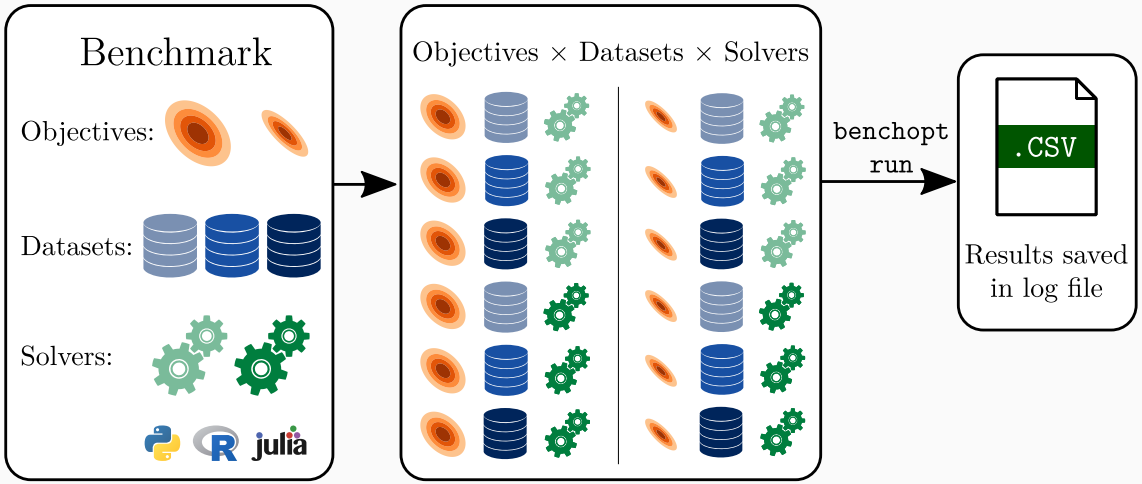
Benchopt Tries to Solve This!



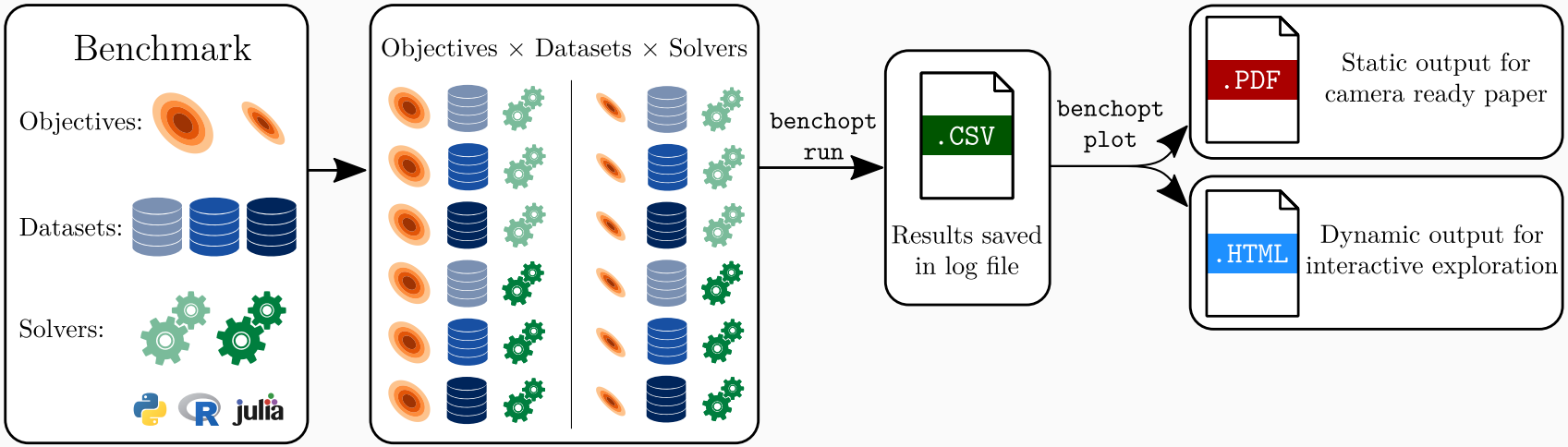
Benchopt Tries to Solve This!



Benchopt Tries to Solve This!



Benchopt Tries to Solve This!



Benchopt Tries to Solve This!

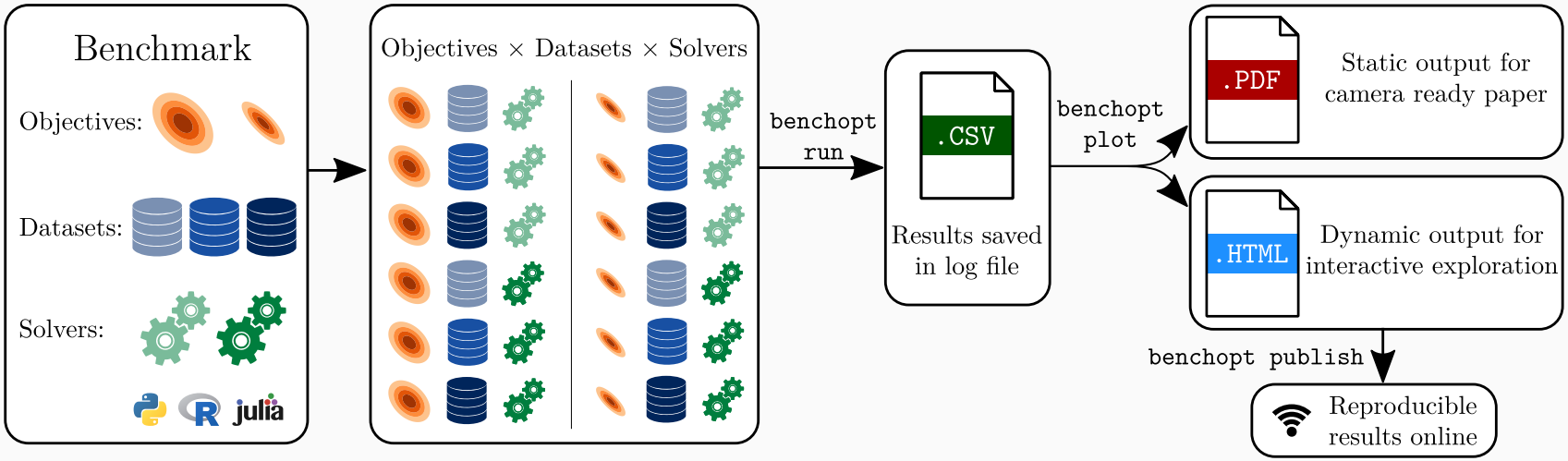


Figure 11: Schematic over how benchopt works

Example: SLOPE

Install benchopt in an isolated environment.

```
conda create -n benchenv python
conda activate benchenv
pip install -U benchopt
```

Example: SLOPE

Install benchopt in an isolated environment.

```
conda create -n benchenv python
conda activate benchenv
pip install -U benchopt
```

Install the benchmark (solvers and data sets).

```
benchopt install benchmark_slope -s rSLOPE -s PGD
```

Example: SLOPE

Install benchopt in an isolated environment.

```
conda create -n benchenv python
conda activate benchenv
pip install -U benchopt
```

Install the benchmark (solvers and data sets).

```
benchopt install benchmark_slope -s rSLOPE -s PGD
```

And run it!

```
benchopt run benchmark_slope \
  -o SLOPE[reg=0.5,fit_intercept=False,q=0.1] \
  -s PGD[prox=prox_isotonic] -s rSLOPE \
  -d Simulated[n_features=500,n_samples=200]
```

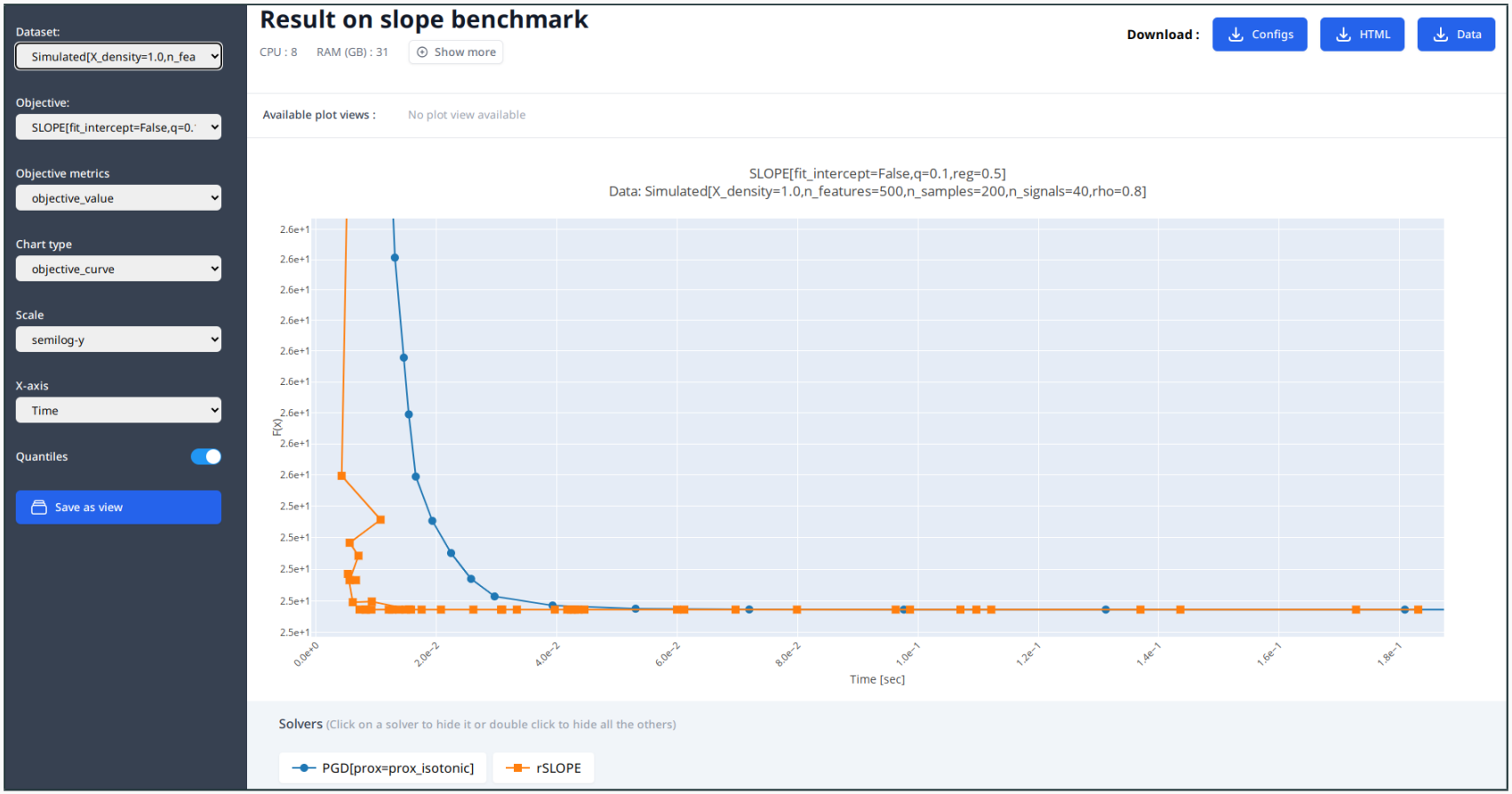


Figure 12: Benchopt benchmark results for SLOPE

Benchmark Structure

```
benchmark_slope
├── datasets
│   ├── dorothea.py
│   ├── simulated.py
│   └── ...
├── solvers
│   ├── pgd.py
│   ├── rSLOPE.py
│   ├── my_new_solver.py
│   └── ...
├── objective.py
└── README.rst
```

Summary

Contributions

A framework that simplifies benchmarking of optimization methods:

- Automatic installation of dependencies
- Caching
- Interactive visualizations
- Easy to publish results
- Support for multiple programming languages

Summary

Contributions

A framework that simplifies benchmarking of optimization methods:

- Automatic installation of dependencies
- Caching
- Interactive visualizations
- Easy to publish results
- Support for multiple programming languages

Limitations

- Dealing with dependencies for 20+ solvers is still challenging.

Coordinate Descent for SLOPE (Paper V)

Coordinate Descent for SLOPE

Published and presented at AISTATS 2023[8]



Co-authored with Mathurin Massias, Quentin Bertrand, and Jonas Wallin.

[8] Johan Larsson, Quentin Klopfenstein, et al. (Apr. 25–27, 2023). "Coordinate Descent for SLOPE". In: *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*. AISTATS 2023. Ed. by Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent. Vol. 206. Proceedings of Machine Learning Research. Valencia, Spain: PMLR, pp. 4802–4821

Motivation

- SLOPE has many appealing properties but the best algorithms for solving SLOPE are slow (relative to the lasso)

Motivation

- SLOPE has many appealing properties but the best algorithms for solving SLOPE are slow (relative to the lasso)
- Lasso solvers are fast because they use coordinate descent

Motivation

- SLOPE has many appealing properties but the best algorithms for solving SLOPE are slow (relative to the lasso)
- Lasso solvers are fast because they use coordinate descent
- Unfortunately cannot use basic coordinate descent for SLOPE

Coordinate Descent

Simple Method!

At each iteration, update a single coordinate (coefficient).

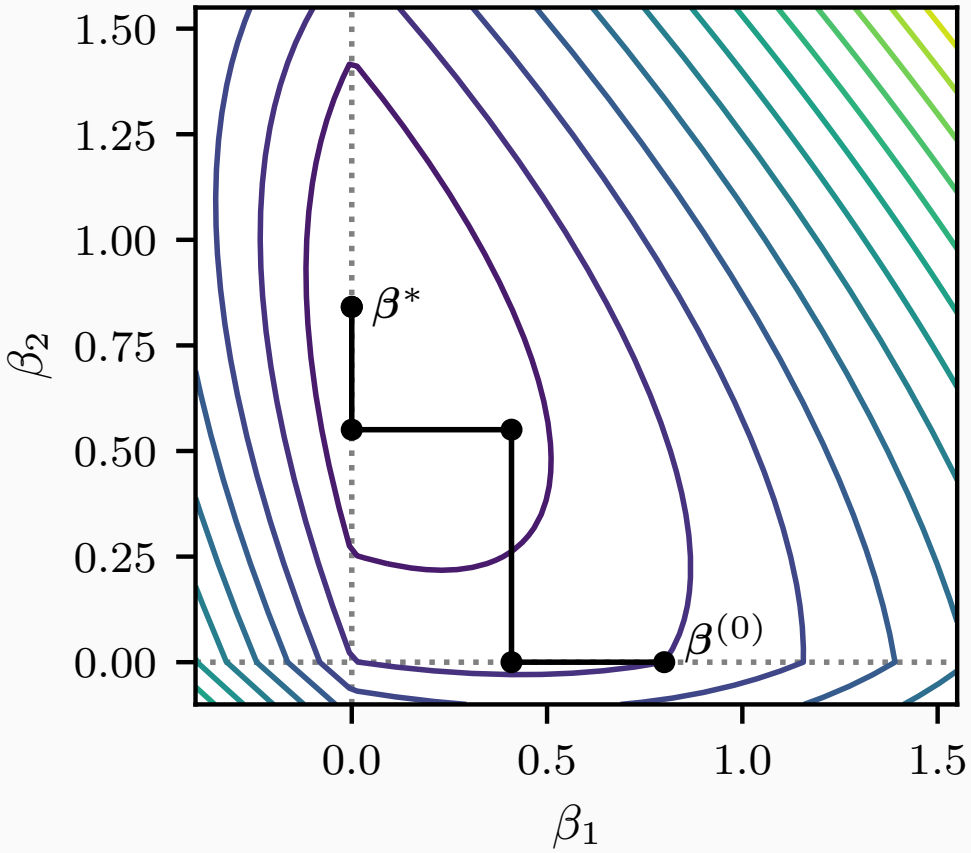


Figure 13: A basic example of coordinate descent in two dimensions

Coordinate Descent Performance

Performs (surprisingly) well!

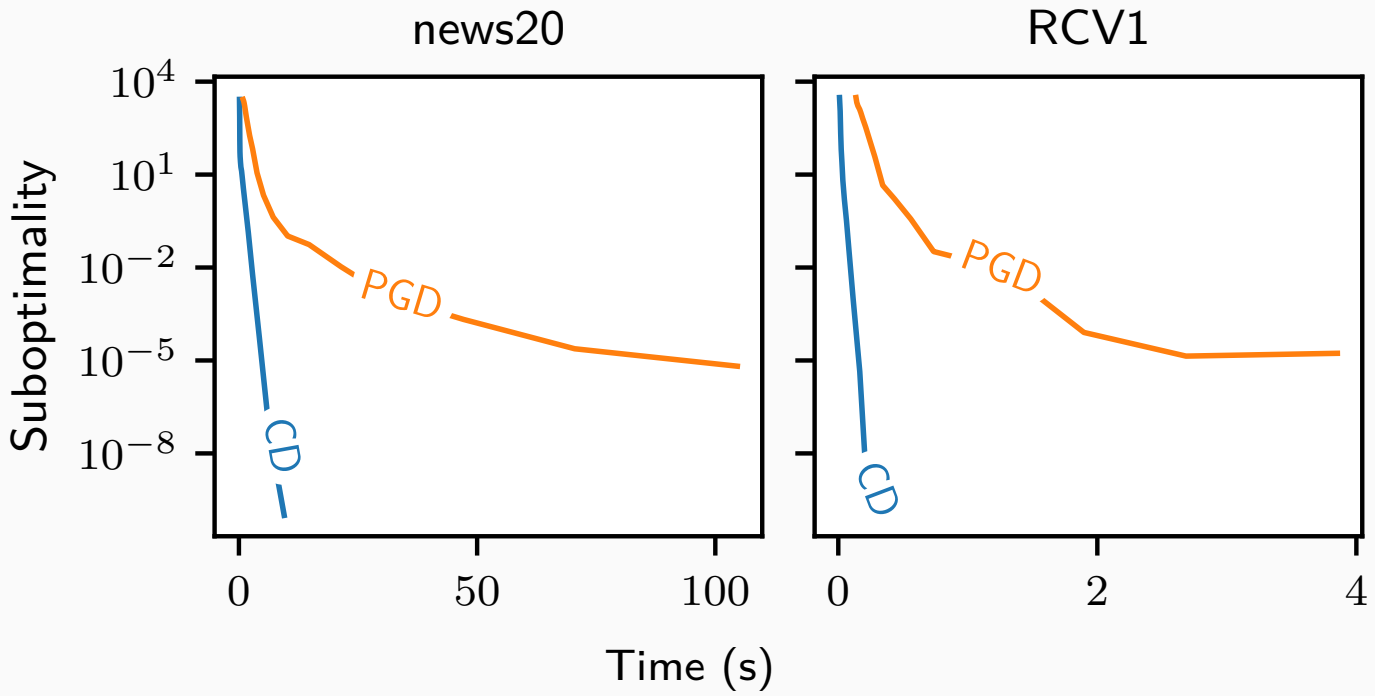


Figure 14: Coordinate descent versus proximal gradient descent for the lasso.

Coordinate Descent and Separability

Cannot use basic coordinate descent for SLOPE since the sorted ℓ_1 norm is inseparable:

$$h(\boldsymbol{\beta}) = \sum_{j=1}^{p} \lambda_j |\beta_{(j)}|.$$

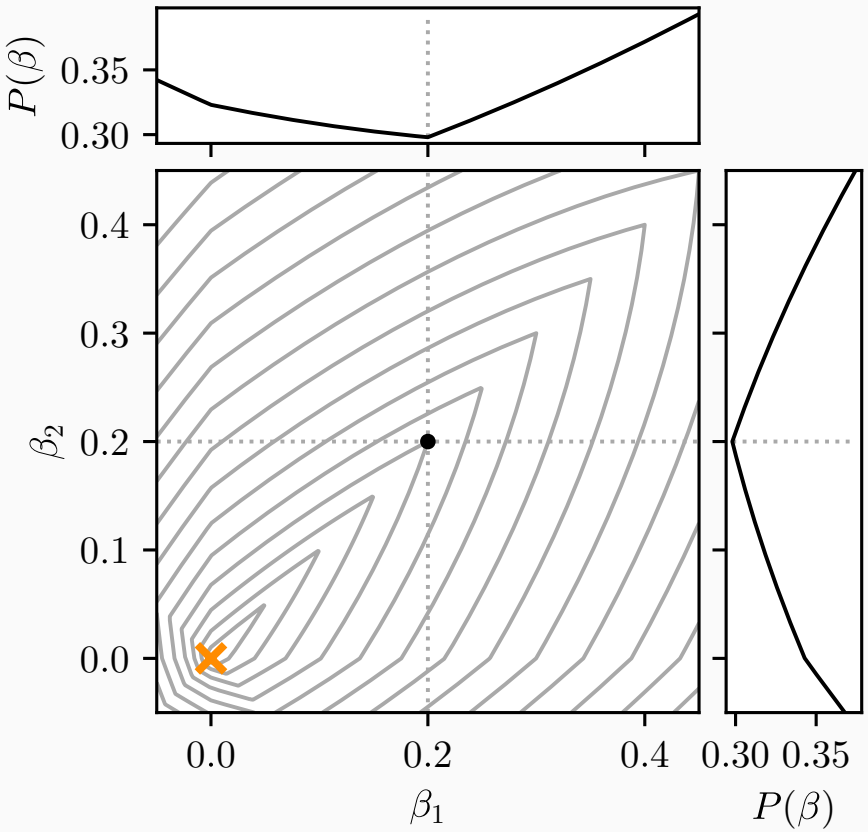


Figure 15: A *naive* coordinate descent algorithm cannot advance from the current iterate (●) to reach the optimum (✖).

Coordinate Descent and Separability

Cannot use basic coordinate descent for SLOPE since the sorted ℓ_1 norm is inseparable:

$$h(\boldsymbol{\beta}) = \sum_{j=1}^{p} \lambda_j |\beta_{(j)}|.$$

But if we could fix the clusters, we have separability!

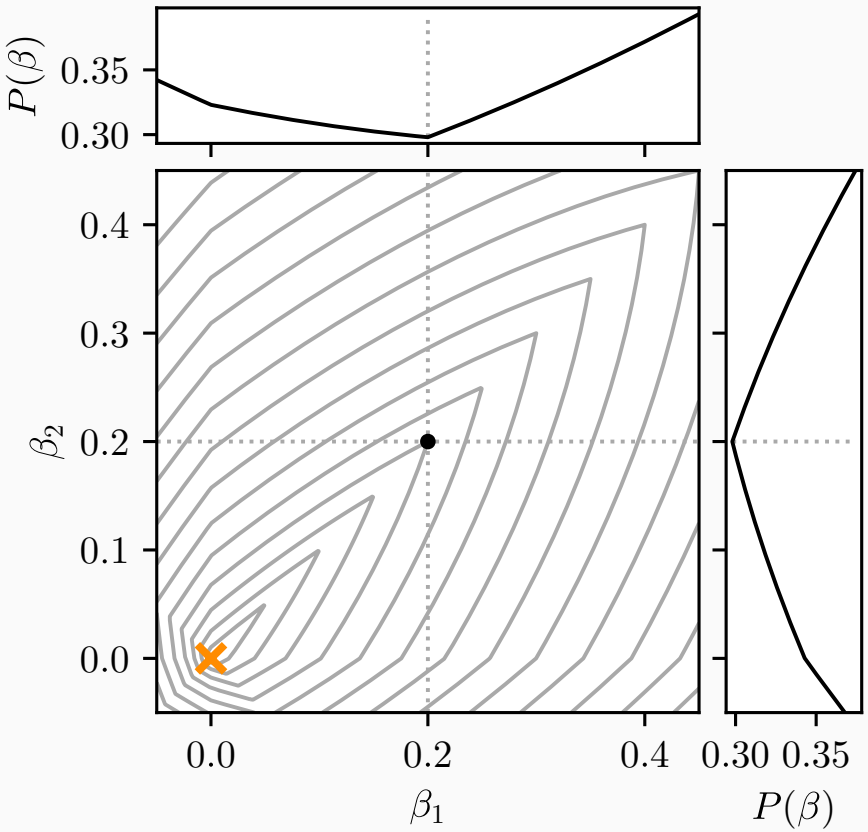


Figure 15: A *naive* coordinate descent algorithm cannot advance from the current iterate (●) to reach the optimum (✖).

Coordinate Descent and Separability

Cannot use basic coordinate descent for SLOPE since the sorted ℓ_1 norm is inseparable:

$$h(\boldsymbol{\beta}) = \sum_{j=1}^{p} \lambda_j |\beta_{(j)}|.$$

But if we could fix the clusters, we have separability!

Idea

Alternate between gradient descent and coordinate descent on the clusters.

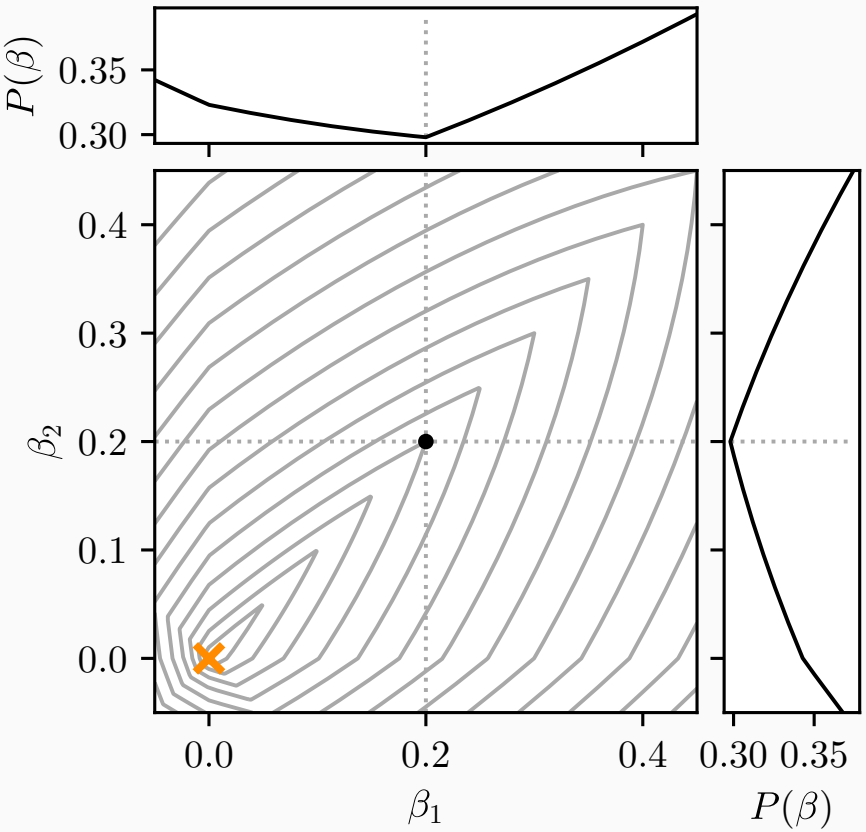


Figure 15: A *naive* coordinate descent algorithm cannot advance from the current iterate (●) to reach the optimum (✖).

Hybrid Algorithm

- Every vth iteration, take a full proximal gradient step. This allows clusters to split (or merge).
- At all other iterations, take coordinate descent steps on the clusters.

Hybrid Algorithm

- Every vth iteration, take a full proximal gradient step. This allows clusters to split (or merge).
- At all other iterations, take coordinate descent steps on the clusters.

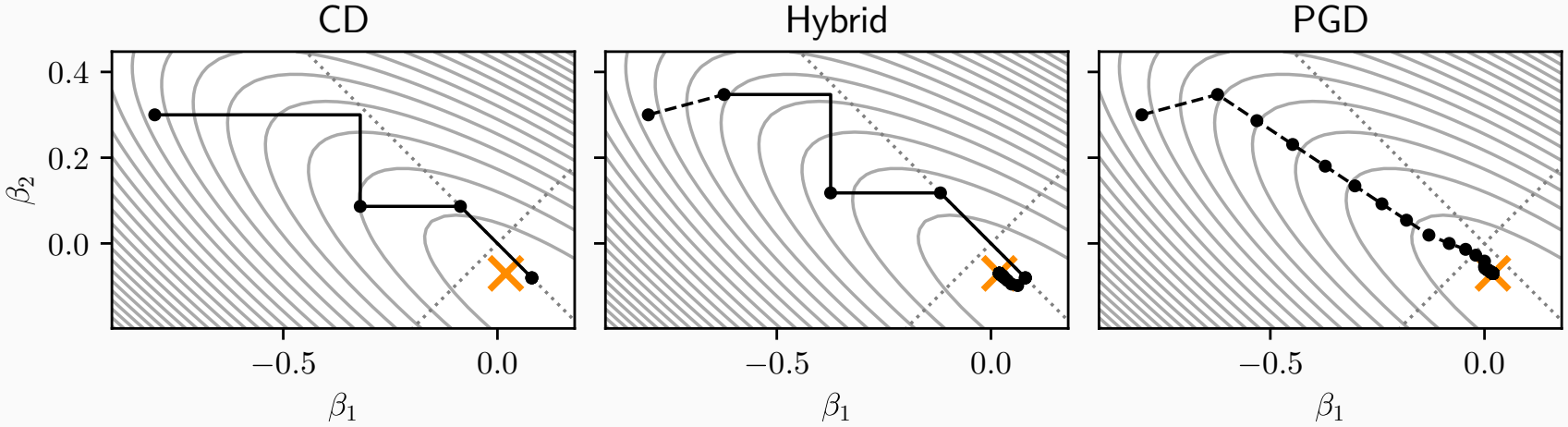


Figure 16: Our algorithm (hybrid) is a combination of CD and PGD.

Experiments: Simulated Data

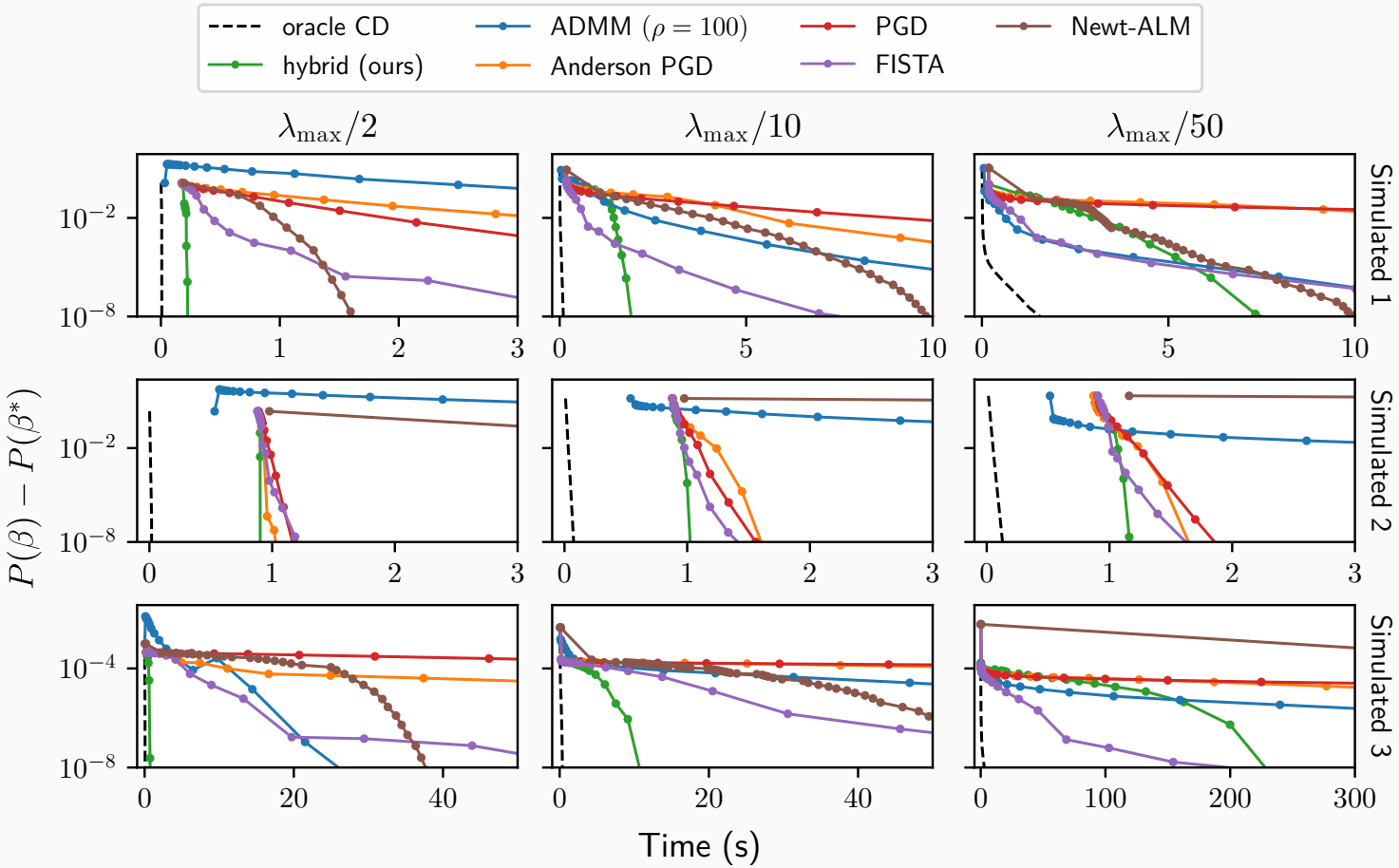


Figure 17: Benchmarks on simulated data. Scenario 1: $n = 200$ and $p = 20\,000$, X. Scenario 2: $n = 20\,000$ and $p = 200$. Scenario 3: $n = 200$, $p = 200\,000$, and sparse X.

Summary

Contributions

- A hybrid algorithm for SLOPE that brings the speed of coordinate descent to SLOPE
- Faster than existing methods
- Implemented in Python package `sortedl1`[9]

[9] https://pypi.org/project/sortedl1/

Summary

Contributions

- A hybrid algorithm for SLOPE that brings the speed of coordinate descent to SLOPE
- Faster than existing methods
- Implemented in Python package `sortedl1`[9]

Limitations

- Somewhat tricky to implement
- Not quite as fast as coordinate descent solvers for the lasso

[9] https://pypi.org/project/sortedl1/

The Lasso and Ridge Regression Yield Biased Estimates of Imbalanced Binary Features (Paper VI)

The Lasso and Ridge Regression Yield Biased Estimates of Imbalanced Binary Features

Not yet published, but will be soon™

Co-authored with Jonas.

Motivation

- Most regularized methods are scale-sensitive, so have to normalize.

Motivation

- Most regularized methods are scale-sensitive, so have to normalize.
- Straightforward normalization when everything is normal, but what about features that have other distributions (binary features)?

Motivation

- Most regularized methods are scale-sensitive, so have to normalize.
- Straightforward normalization when everything is normal, but what about features that have other distributions (binary features)?
- No literature on the effects of different normalization strategies

The Elastic Net

$$\beta^* = \underset{\beta \in \mathbb{R}^p}{\text{minimize}} \left(\frac{1}{2} \|y - X\beta\|_2^2 + \underbrace{\lambda_1 \|\beta\|_1}_{\text{lasso}} + \underbrace{\frac{\lambda_2}{2} \|\beta\|_2^2}_{\text{ridge}} \right)$$

The Elastic Net

$$\beta^* = \underset{\beta \in \mathbb{R}^p}{\text{minimize}} \left(\frac{1}{2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \underbrace{\lambda_1 \|\boldsymbol{\beta}\|_1}_{\text{lasso}} + \underbrace{\frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2}_{\text{ridge}} \right)$$

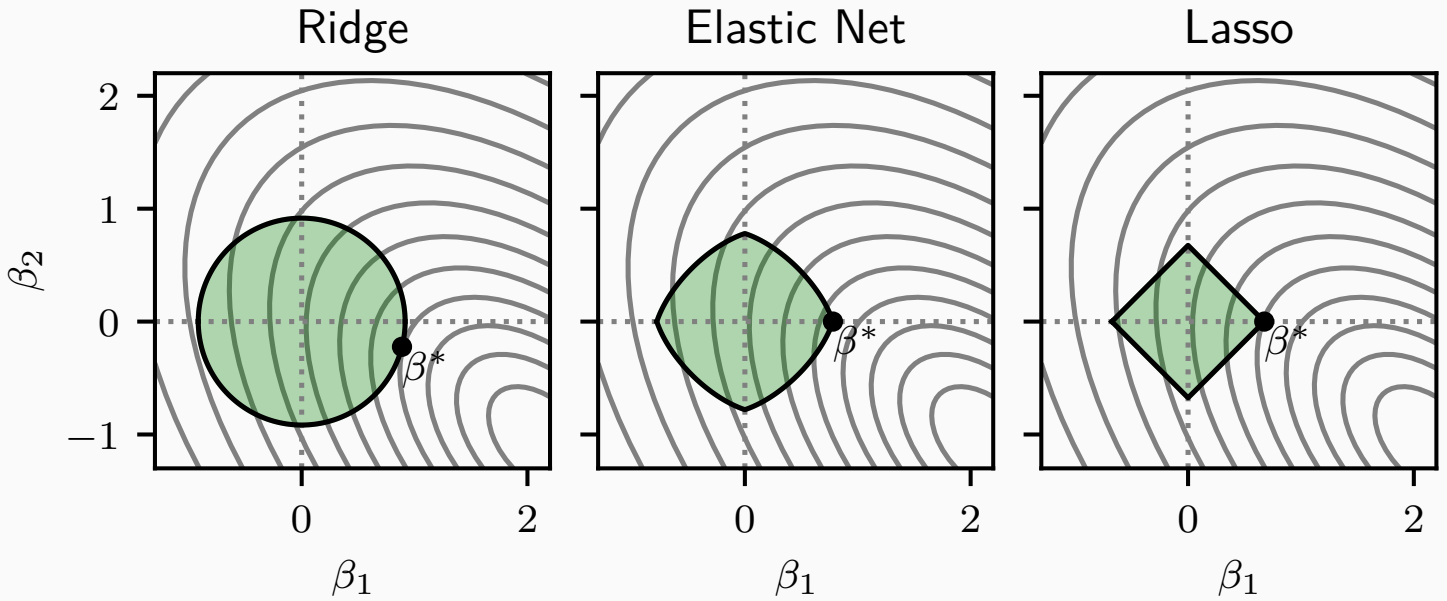


Figure 18: The elastic net penalty is a combination of the lasso and ridge penalties. Here shown as a constrained problem.

Sensitivity to Scale

Lasso and ridge penalties are **norms**—feature scales matter!

Sensitivity to Scale

Lasso and ridge penalties are **norms**—feature scales matter!

Example

Assume

$$\boldsymbol{X} \sim \text{Normal}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}\right), \qquad \boldsymbol{\beta}^* = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix}.$$

Sensitivity to Scale

Lasso and ridge penalties are **norms**—feature scales matter!

Example

Assume

$$\boldsymbol{X} \sim \text{Normal}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}4 & 0\\0 & 1\end{bmatrix}\right), \qquad \boldsymbol{\beta}^* = \begin{bmatrix}\frac{1}{2}\\1\end{bmatrix}.$$

Model	$\hat{\boldsymbol{\beta}}$	$\hat{\boldsymbol{\beta}}_{\text{std}}$
OLS	$\begin{bmatrix}0.50 & 1.00\end{bmatrix}^\intercal$	$\begin{bmatrix}1.00 & 1.00\end{bmatrix}^\intercal$

Sensitivity to Scale

Lasso and ridge penalties are **norms**—feature scales matter!

Example

Assume

$$\boldsymbol{X} \sim \text{Normal}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}4 & 0\\0 & 1\end{bmatrix}\right), \qquad \boldsymbol{\beta}^* = \begin{bmatrix}\frac{1}{2}\\1\end{bmatrix}.$$

Model	$\hat{\boldsymbol{\beta}}$	$\hat{\boldsymbol{\beta}}_{\text{std}}$
OLS	$\begin{bmatrix}0.50 & 1.00\end{bmatrix}^\intercal$	$\begin{bmatrix}1.00 & 1.00\end{bmatrix}^\intercal$
Lasso	$\begin{bmatrix}0.38 & 0.50\end{bmatrix}^\intercal$	$\begin{bmatrix}0.74 & 0.50\end{bmatrix}^\intercal$

Sensitivity to Scale

Lasso and ridge penalties are **norms**—feature scales matter!

Example

Assume

$$\boldsymbol{X} \sim \text{Normal}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}4 & 0\\0 & 1\end{bmatrix}\right), \qquad \boldsymbol{\beta}^* = \begin{bmatrix}\frac{1}{2}\\1\end{bmatrix}.$$

Model	$\hat{\boldsymbol{\beta}}$	$\hat{\boldsymbol{\beta}}_{\text{std}}$
OLS	$\begin{bmatrix}0.50 & 1.00\end{bmatrix}^\intercal$	$\begin{bmatrix}1.00 & 1.00\end{bmatrix}^\intercal$
Lasso	$\begin{bmatrix}0.38 & 0.50\end{bmatrix}^\intercal$	$\begin{bmatrix}0.74 & 0.50\end{bmatrix}^\intercal$
Ridge	$\begin{bmatrix}0.37 & 0.41\end{bmatrix}^\intercal$	$\begin{bmatrix}0.74 & 0.41\end{bmatrix}^\intercal$

Sensitivity to Scale

Lasso and ridge penalties are **norms**—feature scales matter!

Example

Assume

$$\boldsymbol{X} \sim \text{Normal}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}4 & 0\\0 & 1\end{bmatrix}\right), \qquad \boldsymbol{\beta}^* = \begin{bmatrix}\frac{1}{2}\\1\end{bmatrix}.$$

Model	$\hat{\boldsymbol{\beta}}$	$\hat{\boldsymbol{\beta}}_{\text{std}}$
OLS	$\begin{bmatrix}0.50 & 1.00\end{bmatrix}^\intercal$	$\begin{bmatrix}1.00 & 1.00\end{bmatrix}^\intercal$
Lasso	$\begin{bmatrix}0.38 & 0.50\end{bmatrix}^\intercal$	$\begin{bmatrix}0.74 & 0.50\end{bmatrix}^\intercal$
Ridge	$\begin{bmatrix}0.37 & 0.41\end{bmatrix}^\intercal$	$\begin{bmatrix}0.74 & 0.41\end{bmatrix}^\intercal$

Large scale means less penalization because the size of β_j can be smaller for an equivalent effect (on $\boldsymbol{y}$).

Normalization

- Scale sensitivity can be mitigated by normalizing the features.

Normalization

- Scale sensitivity can be mitigated by normalizing the features.
- Let $\tilde{\boldsymbol{X}}$ be the normalized feature matrix, with elements

$$\tilde{x}_{ij} = \frac{x_{ij} - c_j}{s_j}.$$

Normalization

- Scale sensitivity can be mitigated by normalizing the features.
- Let $\tilde{\boldsymbol{X}}$ be the normalized feature matrix, with elements

$$\tilde{x}_{ij} = \frac{x_{ij} - c_j}{s_j}.$$

- After fitting, we transform the coefficients back to their original scale via

$$\hat{\beta}_j = \frac{\hat{\beta}_j^{(n)}}{s_j} \quad \text{for} \quad j = 1, 2, \ldots, p.$$

Table 2: Common ways to normalize $\boldsymbol{X}$

Normalization	Centering (c_j)	Scaling (s_j)
Standardization	$\frac{1}{n}\sum_{i=1}^{n} x_{ij}$	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij}-\bar{x}_j)^2}$
Min–Max	$\min_i(x_{ij})$	$\max_i(x_{ij}) - \min_i(x_{ij})$
Unit Vector (L2)	0	$\sqrt{\sum_{i=1}^{n} x_{ij}^2}$
Max–Abs	0	$\max_i(\lvert x_{ij}\rvert)$
Adaptive Lasso	0	β_j^{OLS}

The Type of Normalization Matters

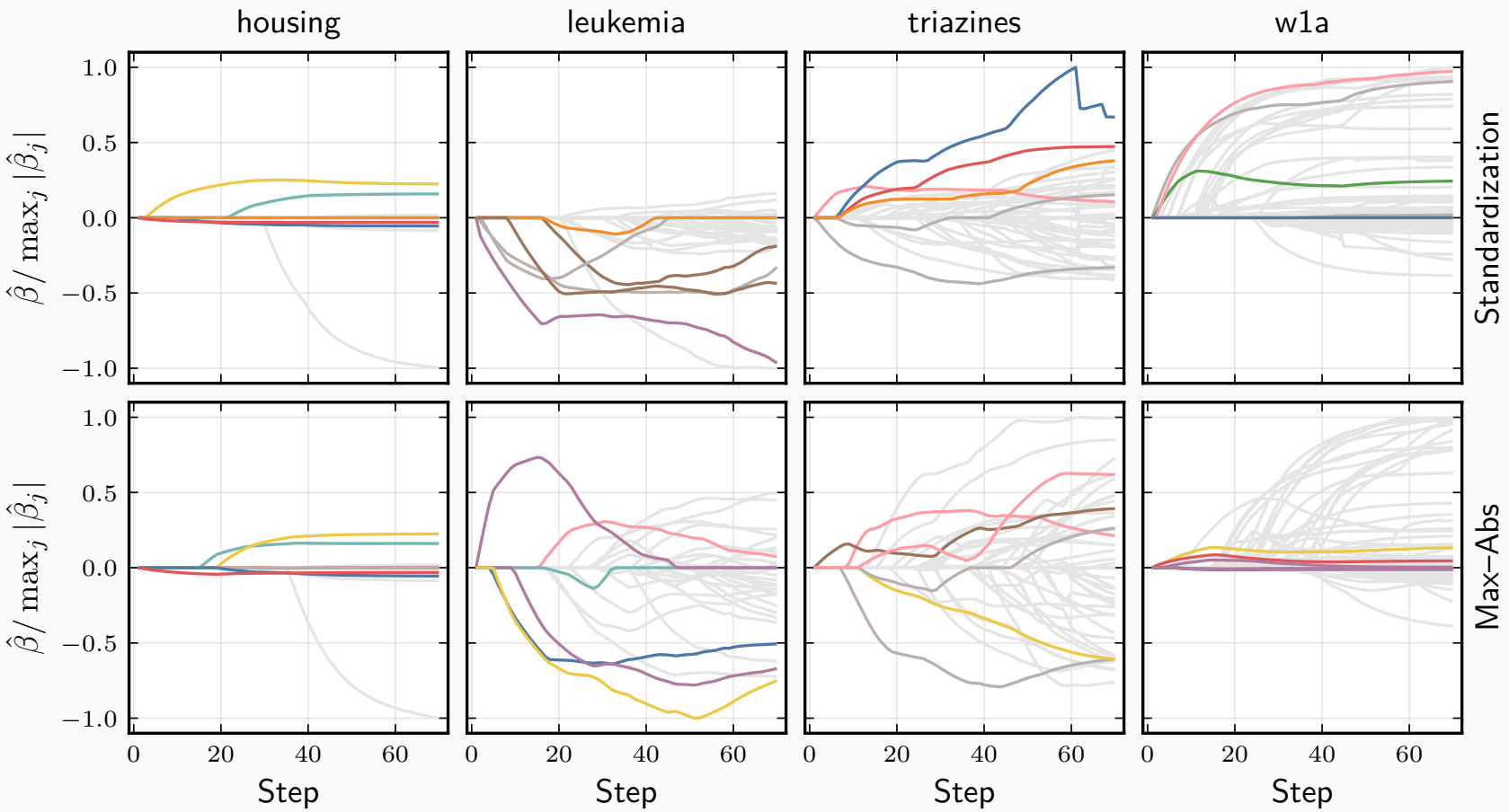


Figure 19: Lasso paths under two different types of normalization (standardization and max–abs normalization). The union of the first five features selected in any of the schemes are colored.

Binary Features

For binary features (values 0 and 1 only), we have for the noiseless case

$$\hat{\beta}_j = \frac{\mathrm{S}_{\lambda_1}\left(\frac{\beta_j^* n(q-q^2)}{s_j}\right)}{s_j\left(\frac{n(q-q^2)}{s_j^2} + \lambda_2\right)}.$$

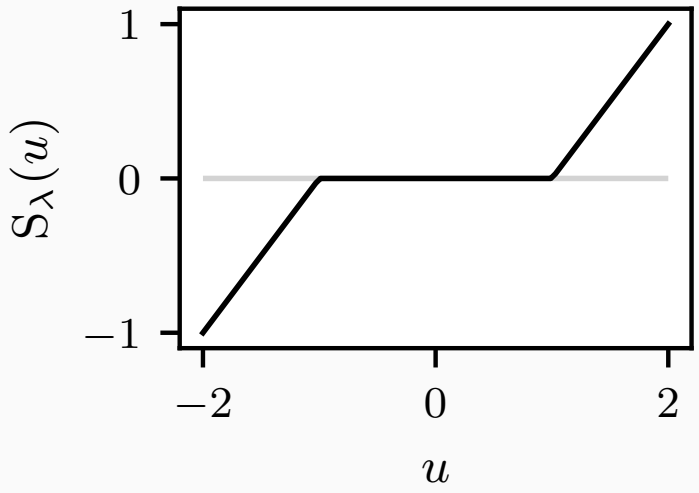


Figure 20: Soft-thresholding

Binary Features

For binary features (values 0 and 1 only), we have for the noiseless case

$$\hat{\beta}_j = \frac{\mathrm{S}_{\lambda_1}\left(\frac{\beta_j^* n(q-q^2)}{s_j}\right)}{s_j\left(\frac{n(q-q^2)}{s_j^2} + \lambda_2\right)}.$$

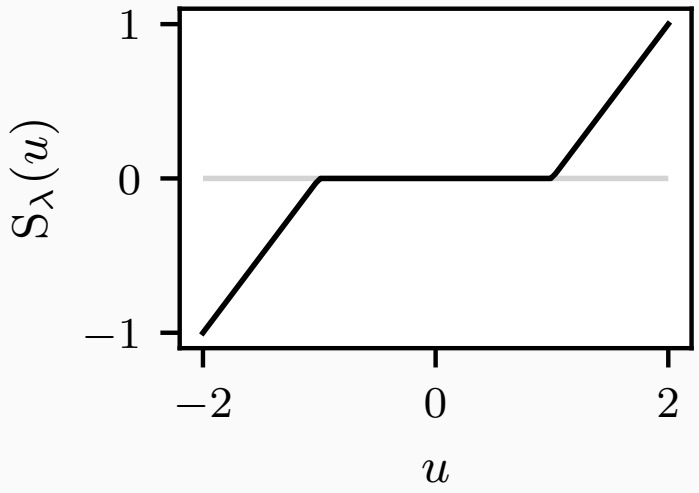


Figure 20: Soft-thresholding

- Means that the elastic net estimator depends on class balance (q).

Binary Features

For binary features (values 0 and 1 only), we have for the noiseless case

$$\hat{\beta}_j = \frac{\mathrm{S}_{\lambda_1}\left(\frac{\beta_j^* n(q-q^2)}{s_j}\right)}{s_j\left(\frac{n(q-q^2)}{s_j^2} + \lambda_2\right)}.$$

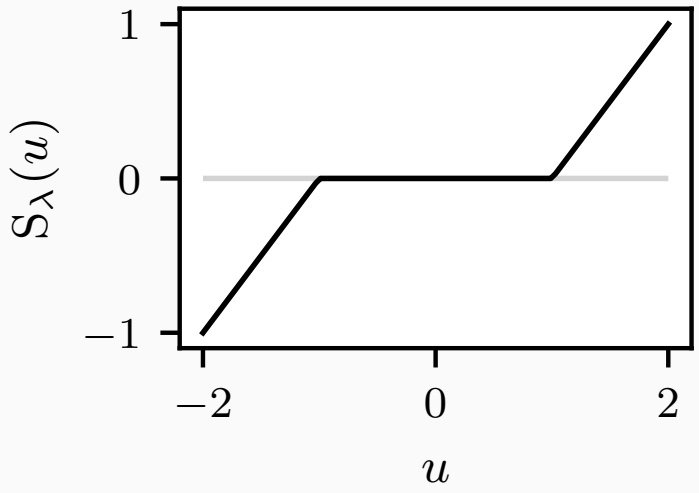


Figure 20: Soft-thresholding

- Means that the elastic net estimator depends on class balance (q).
- $s_j = q - q^2$ for lasso and $s_j = \sqrt{q - q^2}$ for ridge removes effect of q.

Binary Features

For binary features (values 0 and 1 only), we have for the noiseless case

$$\hat{\beta}_j = \frac{\mathrm{S}_{\lambda_1}\left(\frac{\beta_j^* n(q-q^2)}{s_j}\right)}{s_j\left(\frac{n(q-q^2)}{s_j^2} + \lambda_2\right)}.$$

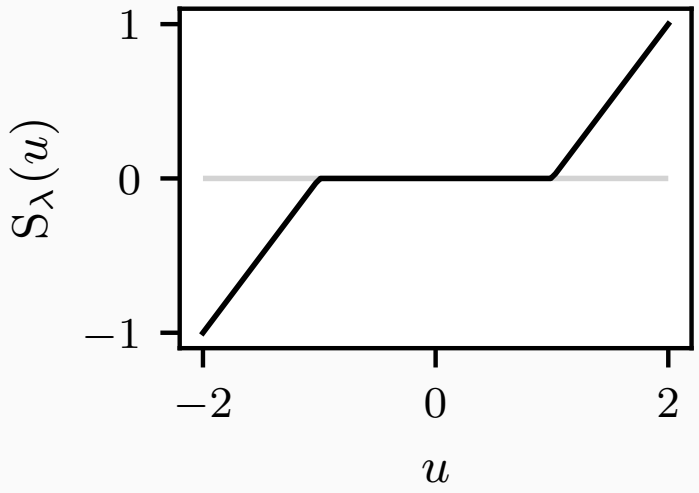


Figure 20: Soft-thresholding

- Means that the elastic net estimator depends on class balance (q).
- $s_j = q - q^2$ for lasso and $s_j = \sqrt{q - q^2}$ for ridge removes effect of q.
- Suggests the parametrization

$$s_j = (q - q^2)^\delta, \qquad \delta \geq 0.$$

Mixed Data

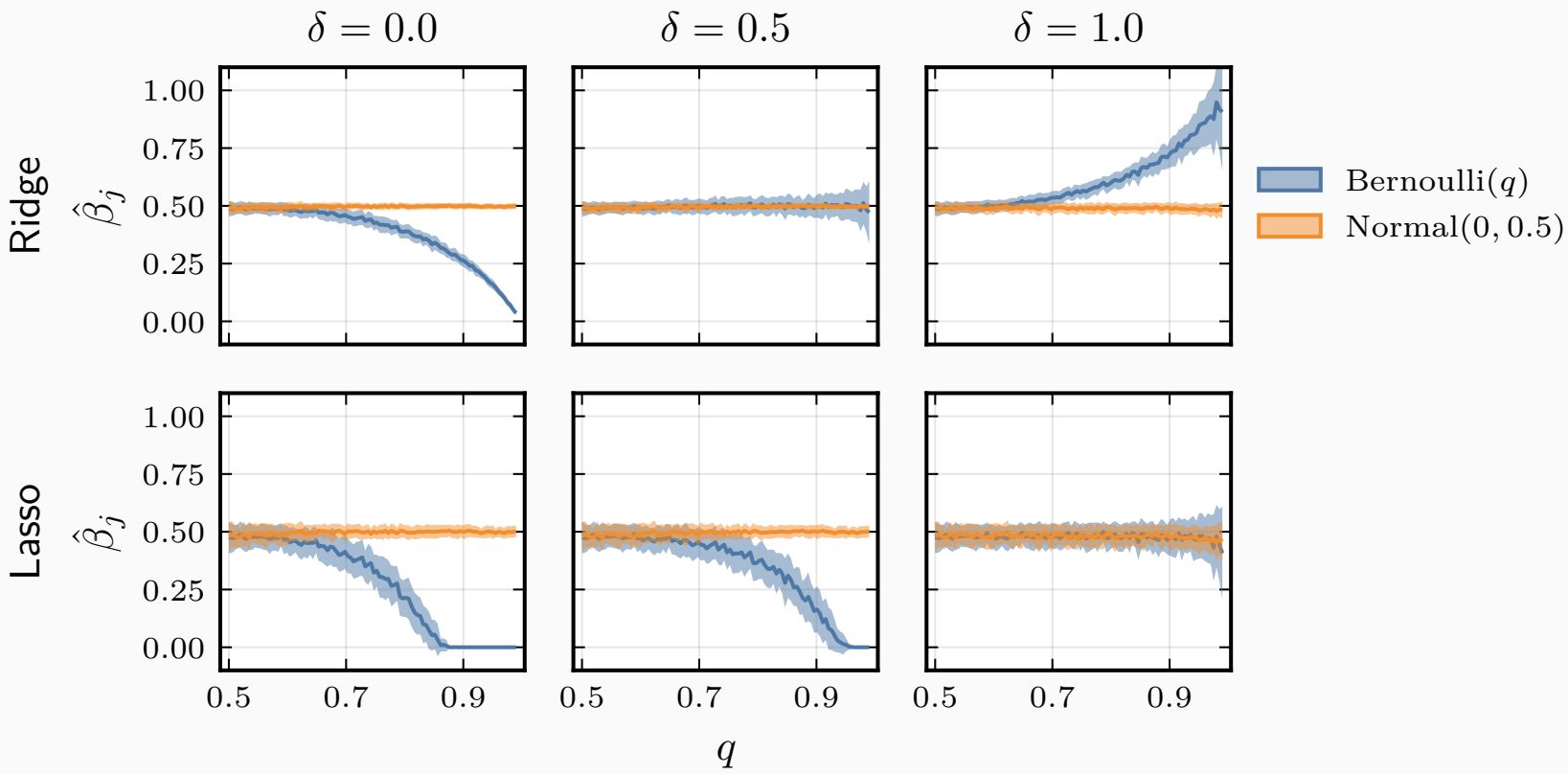


Figure 21: Comparison between lasso and ridge estimators for a data set with one binary and one quasi-normal feature.

Summary

Contributions

- As far as we know the first paper to investigate the interplay between normalization and regularization
- New scaling approach to deal with class-imbalanced binary features
- Discussion and suggestions for dealing with mixed data

Summary

Contributions

- As far as we know the first paper to investigate the interplay between normalization and regularization
- New scaling approach to deal with class-imbalanced binary features
- Discussion and suggestions for dealing with mixed data

Limitations

- So far only theoretical results for limited cases:
  - Fixed data ($\boldsymbol{X}$), normal noise
  - Orthogonal features
  - Normal and binary features

Recap

- Several screening rules that improve performance for lasso and SLOPE

Recap

- Several screening rules that improve performance for lasso and SLOPE
- Framework for benchmarking optimization methods

Recap

- Several screening rules that improve performance for lasso and SLOPE
- Framework for benchmarking optimization methods
- Fast optimization method for SLOPE

Recap

- Several screening rules that improve performance for lasso and SLOPE
- Framework for benchmarking optimization methods
- Fast optimization method for SLOPE
- Analysis of interplay between normalization and penalization

Thank you!

References i

 Fercoq, Olivier, Alexandre Gramfort, and Joseph Salmon (July 6–11, 2015). "Mind the Duality Gap: Safer Rules for the Lasso". In: *Proceedings of the 37th International Conference on Machine Learning*. ICML 2015. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, pp. 333–342.

 Larsson, Johan (Sept. 6, 2021). "Look-Ahead Screening Rules for the Lasso". In: *22nd European Young Statisticians Meeting - Proceedings*. 22nd European Young Statisticians Meeting. Ed. by Andreas Makridis et al. Athens, Greece: Panteion university of social and political sciences, pp. 61–65. ISBN: 978-960-7943-23-1.

 Larsson, Johan, Małgorzata Bogdan, and Jonas Wallin (Dec. 6–12, 2020). "The Strong Screening Rule for SLOPE". In: *Advances in Neural Information Processing Systems 33*. 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Ed. by Hugo Larochelle et al. Vol. 33. Virtual: Curran Associates, Inc., pp. 14592–14603. ISBN: 978-1-71382-954-6.

References ii

 Larsson, Johan, Quentin Klopfenstein, et al. (Apr. 25–27, 2023). "Coordinate Descent for SLOPE". In: *Proceedings of the 26th International Conference on Artificial Intelligence and Statistics*. AISTATS 2023. Ed. by Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent. Vol. 206. Proceedings of Machine Learning Research. Valencia, Spain: PMLR, pp. 4802–4821.

 Larsson, Johan and Jonas Wallin (Nov. 28–Dec. 9, 2022). "The Hessian Screening Rule". In: *Advances in Neural Information Processing Systems 35*. 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Ed. by Sanmi Koyejo et al. Vol. 35. New Orleans, USA: Curran Associates, Inc., pp. 15823–15835. ISBN: 978-1-71387-108-8.

References iii


Moreau, Thomas et al. (Nov. 28–Dec. 9, 2022). "Benchopt: Reproducible, Efficient and Collaborative Optimization Benchmarks". In: *Advances in Neural Information Processing Systems 35*. 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Ed. by S. Koyejo et al. Vol. 35. New Orleans, USA: Curran Associates, Inc., pp. 25404–25421. ISBN: 978-1-71387-108-8.


Schmidt, Robin M., Frank Schneider, and Philipp Hennig (July 18–24, 2021). "Descending through a Crowded Valley – Benchmarking Deep Learning Optimizers". In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. Virtual: PMLR, pp. 9367–9376.


Tibshirani, Robert et al. (Mar. 2012). "Strong Rules for Discarding Predictors in Lasso-Type Problems". In: *The Journal of the Royal Statistical Society, Series B (Statistical Methodology)* 74.2, pp. 245–266. ISSN: 1369-7412. DOI: 10/c4bb85.

Paper I Extras

Strong Rule Algorithm for SLOPE

Algorithm 1: Basic algorithm for the strong rule for SLOPE

Data: $\boldsymbol{c} \in \mathbb{R}^p$, $\boldsymbol{\lambda} \in \mathbb{R}^p$, where $\lambda_1 \geq \cdots \geq \lambda_p \geq 0$

$\mathcal{S}, \mathcal{B} \leftarrow \varnothing$;

for $i \leftarrow 1$ **to** p **do**

  $\mathcal{B} \leftarrow \mathcal{B} \cup \{i\}$;

  if $\sum_{j \in \mathcal{B}} \left(c_j - \lambda_j\right) \geq 0$ **then**

    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{B}$;

    $\mathcal{B} \leftarrow \varnothing$;

  end

end

return $\mathcal{S}$;

Strong Rule Algorithm for SLOPE

Algorithm 1: Basic algorithm for the strong rule for SLOPE

Data: $\boldsymbol{c} \in \mathbb{R}^p$, $\boldsymbol{\lambda} \in \mathbb{R}^p$, where $\lambda_1 \geq \cdots \geq \lambda_p \geq 0$
$\mathcal{S}, \mathcal{B} \leftarrow \varnothing$;
for $i \leftarrow 1$ **to** p **do**
  $\mathcal{B} \leftarrow \mathcal{B} \cup \{i\}$;
  if $\sum_{j \in \mathcal{B}} \left(c_j - \lambda_j\right) \geq 0$ **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{B}$;
    $\mathcal{B} \leftarrow \varnothing$;
  end
end
return $\mathcal{S}$;

Set

$$\boldsymbol{c} = |\nabla g(\hat{\boldsymbol{\beta}}) + \boldsymbol{\lambda}^{(k-1)} - \boldsymbol{\lambda}^{(k)}|_{\downarrow} \qquad \boldsymbol{\lambda} = \boldsymbol{\lambda}^{(k)},$$

and run the algorithm above; the result is the predicted support for $\hat{\boldsymbol{\beta}}(\boldsymbol{\lambda}^{(k)})$ (subject to a permutation).

Results: Speed

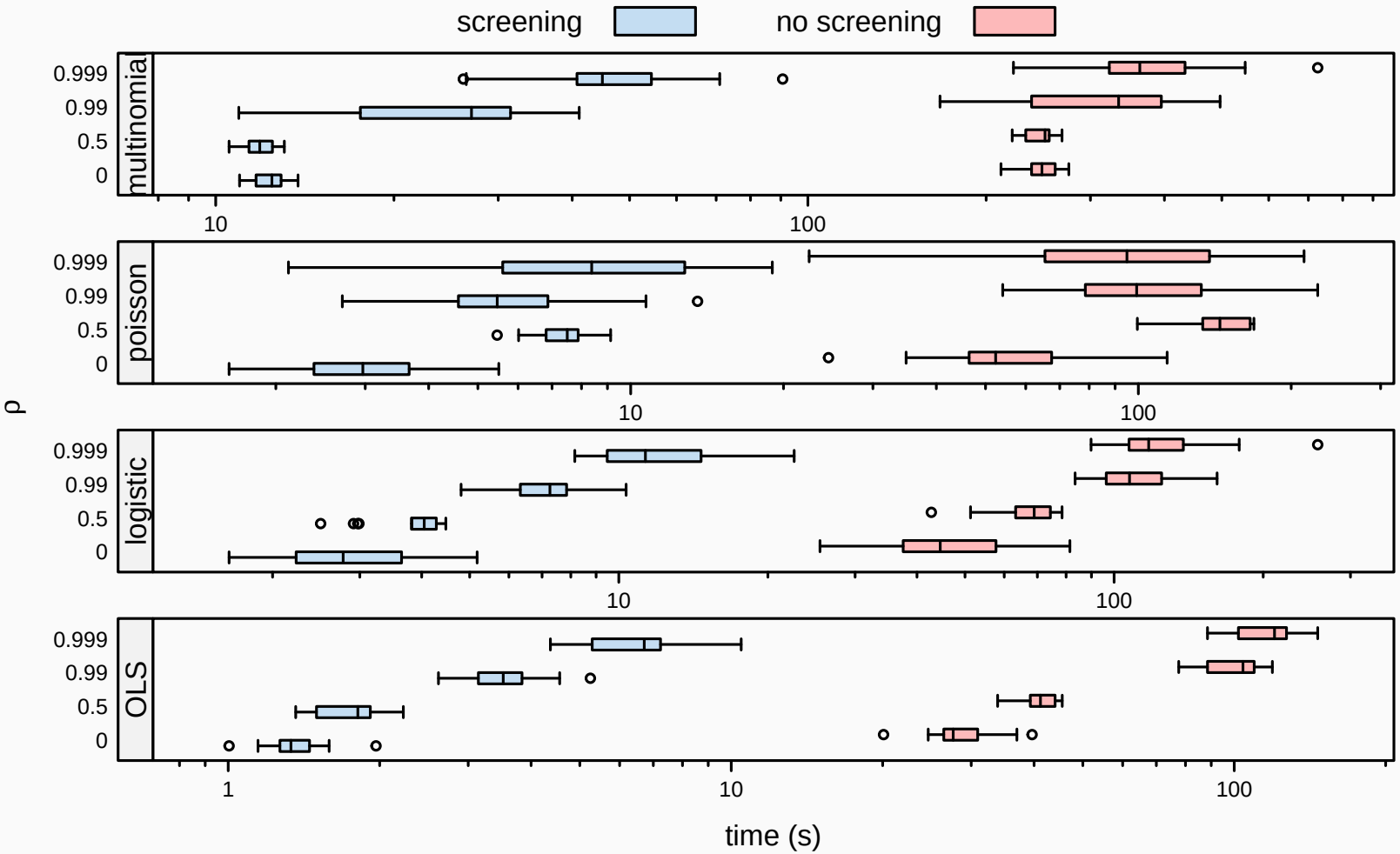


Figure 22: Time to fit a full SLOPE path with and without the strong rule

Paper II Extras

The Dual

Complementary to the primal problem. For the lasso, it is

$$\underset{\boldsymbol{\theta} \in \mathbb{R}^n}{\text{maximize}} \left(D(\boldsymbol{\theta}; \lambda) = \frac{1}{2} \boldsymbol{y}^\intercal \boldsymbol{y} - \frac{\lambda^2}{2} \left\| \boldsymbol{\theta} - \frac{\boldsymbol{y}}{\lambda} \right\|_2^2. \right)$$

The Dual

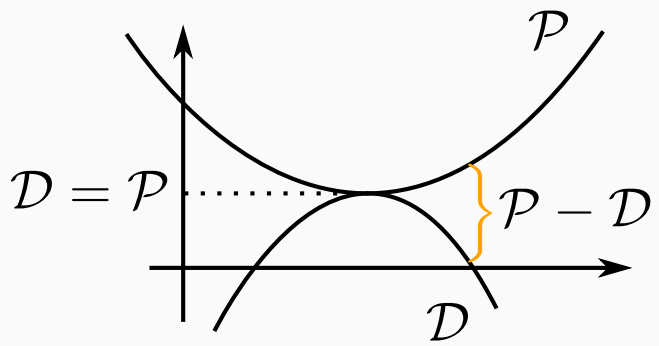
Complementary to the primal problem. For the lasso, it is

$$\underset{\boldsymbol{\theta} \in \mathbb{R}^n}{\text{maximize}} \left(D(\boldsymbol{\theta}; \lambda) = \frac{1}{2} \boldsymbol{y}^\intercal \boldsymbol{y} - \frac{\lambda^2}{2} \left\| \boldsymbol{\theta} - \frac{\boldsymbol{y}}{\lambda} \right\|_2^2. \right)$$

Duality Gap

Difference between the primal and dual objectives. Tight at the optimum (for the lasso):

$$P(\boldsymbol{\beta}^*; \lambda) - D(\boldsymbol{\theta}^*; \lambda) = 0.$$



The Dual

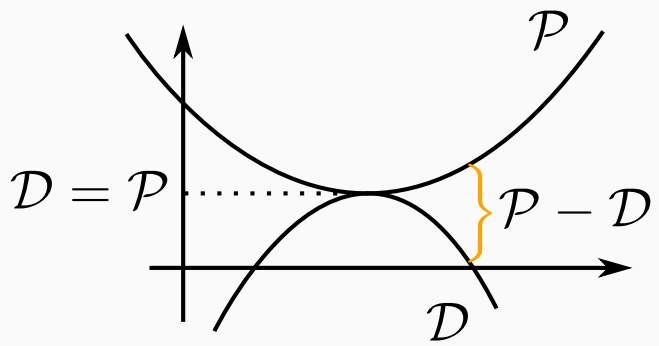
Complementary to the primal problem. For the lasso, it is

$$\underset{\boldsymbol{\theta} \in \mathbb{R}^n}{\text{maximize}} \left(D(\boldsymbol{\theta}; \lambda) = \frac{1}{2} \boldsymbol{y}^\intercal \boldsymbol{y} - \frac{\lambda^2}{2} \left\| \boldsymbol{\theta} - \frac{\boldsymbol{y}}{\lambda} \right\|_2^2. \right)$$

Duality Gap

Difference between the primal and dual objectives. Tight at the optimum (for the lasso):

$$P(\boldsymbol{\beta}^*; \lambda) - D(\boldsymbol{\theta}^*; \lambda) = 0.$$



Dual–Primal Relationship

The two problems are related via

$$\frac{\boldsymbol{y} - \boldsymbol{X}\beta}{\lambda} = \boldsymbol{\theta}.$$

Look-Ahead Screening

Inequality for Gaps Safe screening rule is quadratic in λ, which means we can find the next critical point easily:

$$\lambda_+ = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where

$$\begin{aligned} a &= \left(1 - |\boldsymbol{x}_j^\intercal \boldsymbol{\theta}_\lambda|\right)^2 - \frac{1}{2}\boldsymbol{\theta}_\lambda^\intercal \boldsymbol{\theta}_\lambda \|\boldsymbol{x}_j\|_2^2, \\ b &= \left(\boldsymbol{\theta}_\lambda^\intercal \boldsymbol{y} - \|\boldsymbol{\beta}_\lambda\|_1\right)\|\boldsymbol{x}_j\|_2^2, \\ c &= -\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_\lambda\|_2^2 \|\boldsymbol{x}_j\|_2^2. \end{aligned}$$

Look-Ahead Screening

Inequality for Gaps Safe screening rule is quadratic in λ, which means we can find the next critical point easily:

$$\lambda_+ = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where

$$\begin{aligned} a &= \left(1 - |\boldsymbol{x}_j^\intercal \boldsymbol{\theta}_\lambda|\right)^2 - \frac{1}{2}\boldsymbol{\theta}_\lambda^\intercal \boldsymbol{\theta}_\lambda \|\boldsymbol{x}_j\|_2^2, \\ b &= \left(\boldsymbol{\theta}_\lambda^\intercal \boldsymbol{y} - \|\boldsymbol{\beta}_\lambda\|_1\right)\|\boldsymbol{x}_j\|_2^2, \\ c &= -\frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}_\lambda\|_2^2 \|\boldsymbol{x}_j\|_2^2. \end{aligned}$$

Allows us to screen predictors for **all** upcoming steps

Paper III Extras

Warm Starts

The availability of the Hessian inverse enables a better warm start:

$$\hat{\beta}(\lambda_{k+1})_{\mathcal{A}} = \hat{\beta}(\lambda_k)_{\mathcal{A}} + (\lambda_k - \lambda_{k+1})\left({X_{\mathcal{A}}}^T X_{\mathcal{A}}\right)^{-1} \operatorname{sign}\left(\hat{\beta}(\lambda_k)_{\mathcal{A}}\right)$$

Warm Starts

The availability of the Hessian inverse enables a better warm start:

$$\hat{\beta}(\lambda_{k+1})_{\mathcal{A}} = \hat{\beta}(\lambda_k)_{\mathcal{A}} + (\lambda_k - \lambda_{k+1})\left(X_{\mathcal{A}}{}^T X_{\mathcal{A}}\right)^{-1} \operatorname{sign}\left(\hat{\beta}(\lambda_k)_{\mathcal{A}}\right)$$

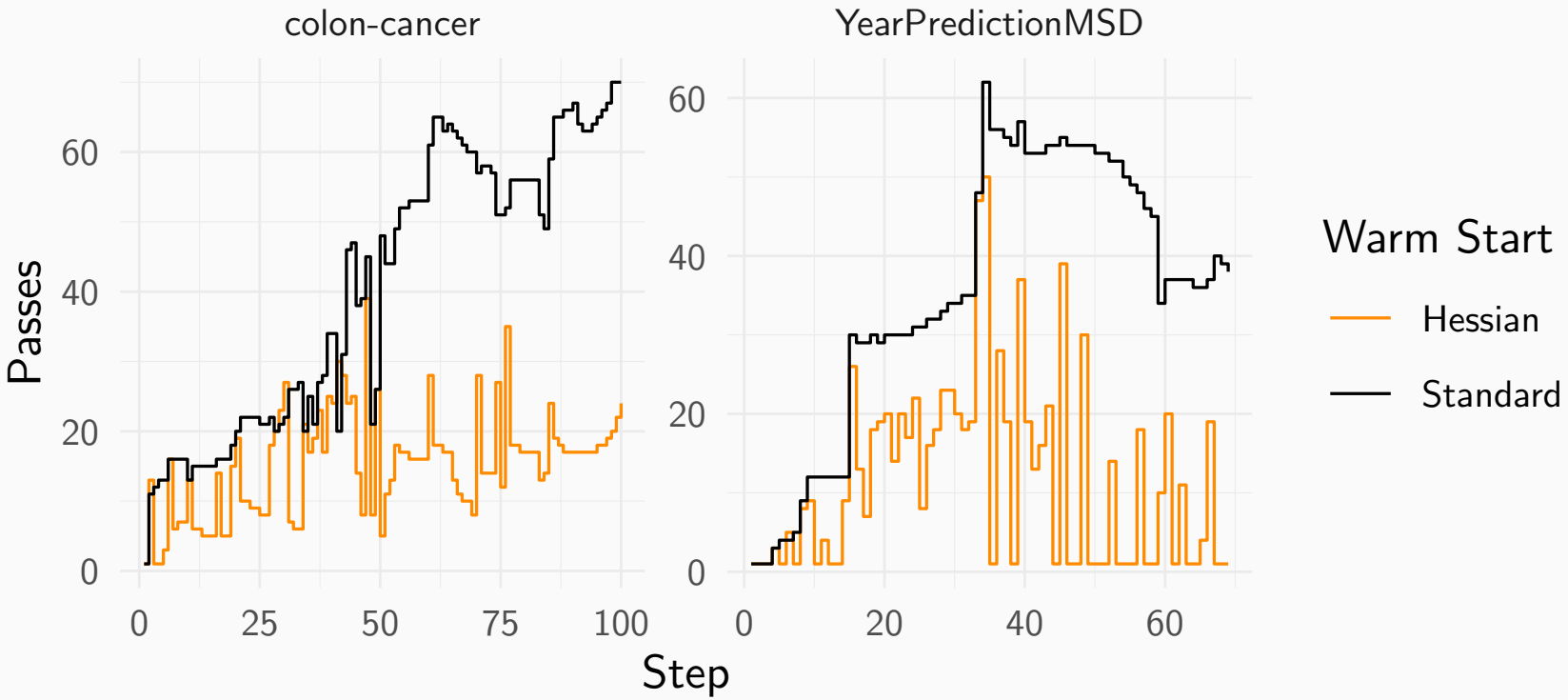


Figure 23: Number of passes of coordinate descent for two datasets using either Hessian warm starts or standard warm starts.

Updating the Hessian

Computing the Hessian and its inverse naively is expensive:
$\mathcal{O}(|\mathcal{A}|^3 + |\mathcal{A}|^2 n)$

Updating the Hessian

Computing the Hessian and its inverse naively is expensive:
$\mathcal{O}(|\mathcal{A}|^3 + |\mathcal{A}|^2 n)$

Fortunately, we can sweep columns of the Hessian and inverse in our out, yielding complexity

- $\mathcal{O}(|\mathcal{D}|^2 n + n|\mathcal{D}||\mathcal{E}| + |\mathcal{D}^2||\mathcal{E}| + |\mathcal{D}|^3)$ when augmenting the Hessian and
- $\mathcal{O}(|\mathcal{C}|^3 + |\mathcal{C}|^2|\mathcal{E}| + |\mathcal{C}||\mathcal{E}|^2)$ when reducing it,

where

- $\mathcal{C} = \mathcal{A}_{k-1} \setminus \mathcal{A}_k$ (to-be deactivated)
- $\mathcal{D} = \mathcal{A}_k \setminus \mathcal{A}_{k-1}$ (to-be activated)
- $\mathcal{E} = \mathcal{A}_k \cap \mathcal{A}_{k-1}$ (still activate)

Paper V Extras

How Do We Minimize Over One Cluster?

The optimality condition, using the directional derivative, is

$$\forall \delta \in \{-1, 1\}, \quad G'(z; \delta) \geq 0,$$

with

$$G'(z;\delta) = \delta \sum_{j \in \mathcal{C}_k} X_{:j}^\top (X\beta(z) - y) + H'(z;\delta).$$

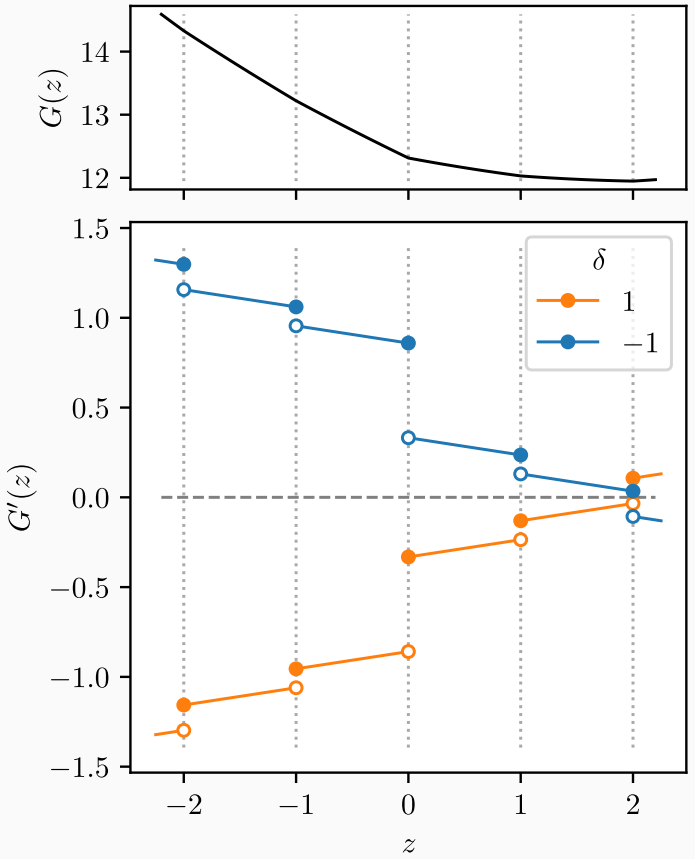


Figure 24: G and its directional derivative $G'(\cdot;\delta)$ for an example with $\beta = [-3, 1, 3, 2]^T$, $k = 1$, and consequently $c^{\setminus k} = \{1, 2\}$.

The SLOPE Thresholding Operator

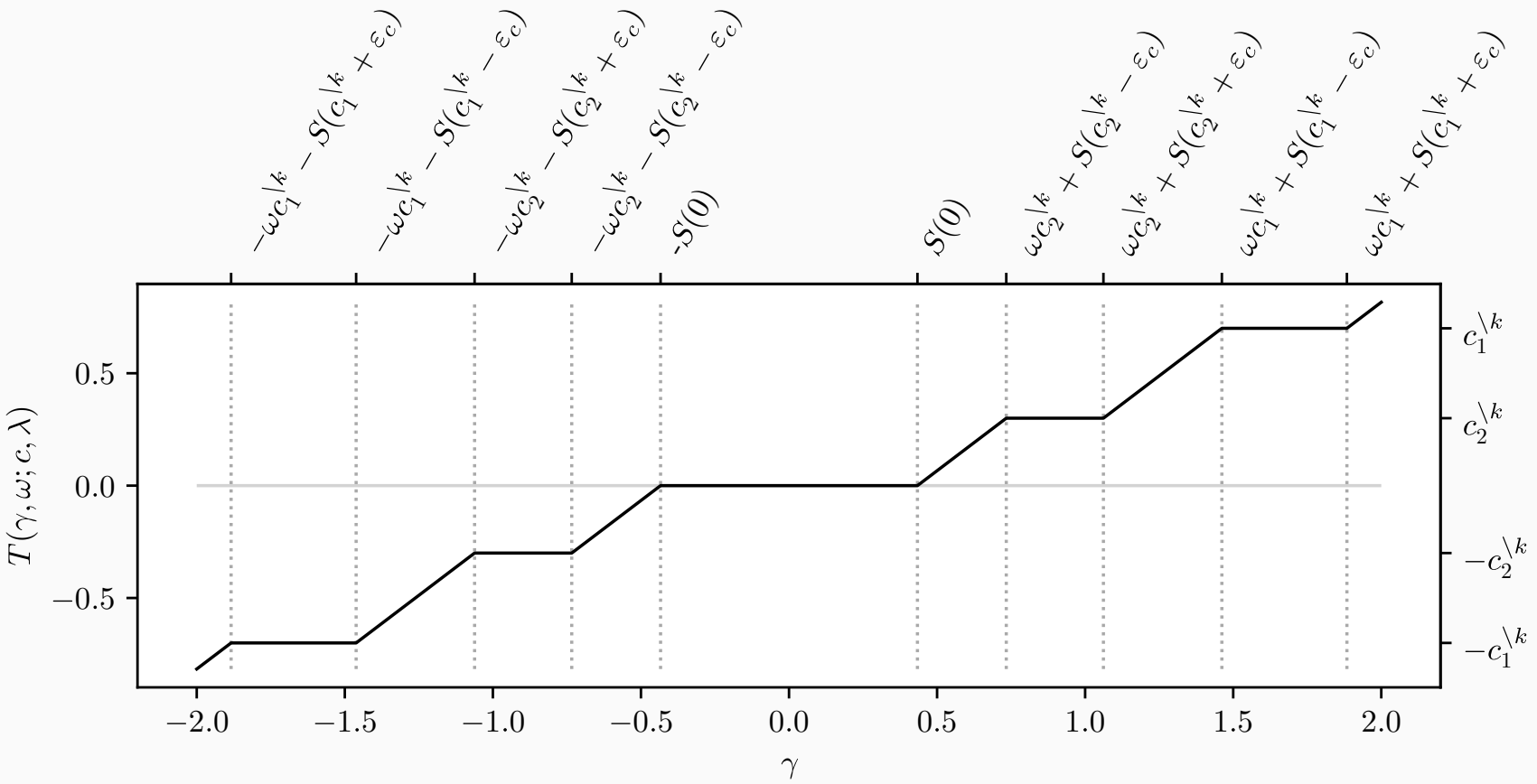


Figure 25: The SLOPE Thresholding Operator

Experiments: Real Data

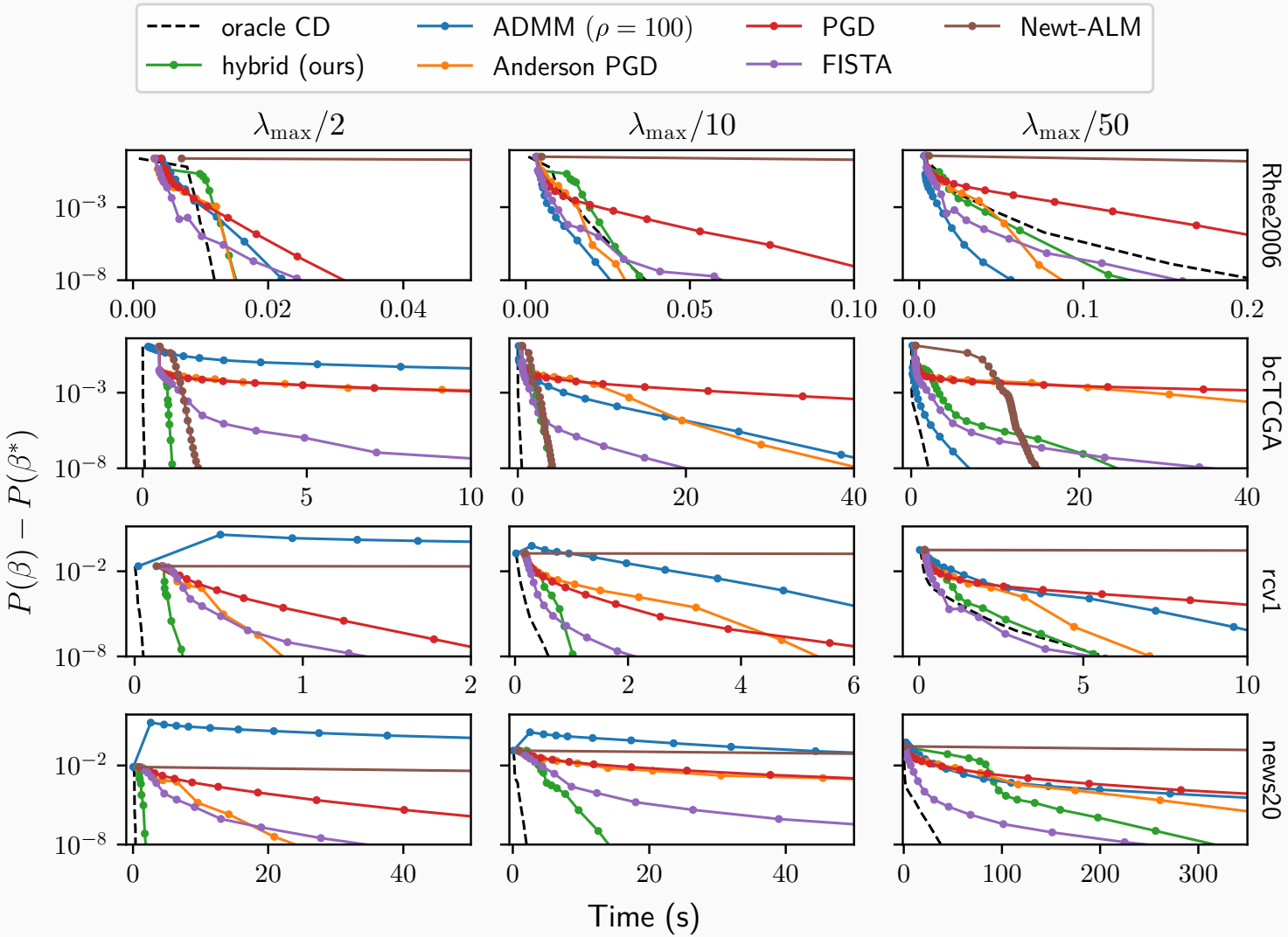


Figure 26: Benchmarks on real data

Paper VI Extras

Probability of Selection

Since $\boldsymbol{X}$ is fixed and $\boldsymbol{\varepsilon}$ is normal, it is straightforward to compute the probability of selection:

$$\Pr(\hat{\beta}_j \neq 0) = \Phi\left(\frac{\mu - \lambda_1}{\sigma}\right) + \Phi\left(\frac{-\mu - \lambda_1}{\sigma}\right).$$

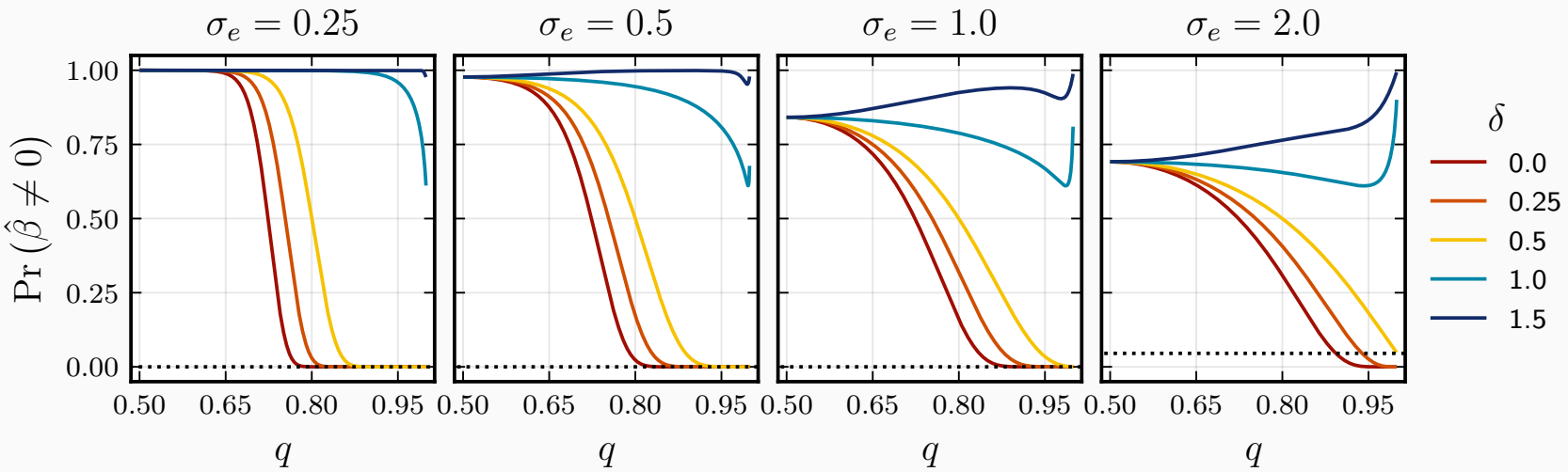


Figure 27: Probability that the elastic net selects a feature across different noise levels (σ_ε), types of normalization (δ), and class balance (q). The dashed line is asymptotic behavior for $\delta = 1/2$.

Asymptotic Results for Bias and Variance

Theorem

If $\boldsymbol{x}_j$ is a binary feature with class balance $q \in (0,1)$ and $\lambda_1, \lambda_2 \in (0,\infty)$, $\sigma_\varepsilon > 0$, and $s_j = (q - q^2)^\delta$, $\delta \geq 0$, then

$$\lim_{q \to 1^+} \mathrm{E}\, \hat{\beta}_j = \begin{cases} 0 & \text{if } 0 \leq \delta < \frac{1}{2}, \\ \frac{2n\beta_j^*}{n+\lambda_2} \Phi\left(-\frac{\lambda_1}{\sigma_\varepsilon \sqrt{n}}\right) & \text{if } \delta = \frac{1}{2}, \\ \beta_j^* & \text{if } \delta \geq \frac{1}{2}. \end{cases}$$

Asymptotic Results for Bias and Variance

Theorem

If $\boldsymbol{x}_j$ is a binary feature with class balance $q \in (0,1)$ and $\lambda_1, \lambda_2 \in (0,\infty)$, $\sigma_\varepsilon > 0$, and $s_j = (q - q^2)^\delta$, $\delta \geq 0$, then

$$\lim_{q \to 1^+} \mathrm{E}\, \hat{\beta}_j = \begin{cases} 0 & \text{if } 0 \leq \delta < \frac{1}{2}, \\ \frac{2n\beta_j^*}{n+\lambda_2} \Phi\left(-\frac{\lambda_1}{\sigma_\varepsilon \sqrt{n}}\right) & \text{if } \delta = \frac{1}{2}, \\ \beta_j^* & \text{if } \delta \geq \frac{1}{2}. \end{cases}$$

and

$$\lim_{q \to 1^+} \mathrm{Var}\, \hat{\beta}_j = \begin{cases} 0 & \text{if } 0 \leq \delta < \frac{1}{2}, \\ \infty & \text{if } \delta \geq \frac{1}{2}. \end{cases}$$

A Bias–Variance Tradeoff

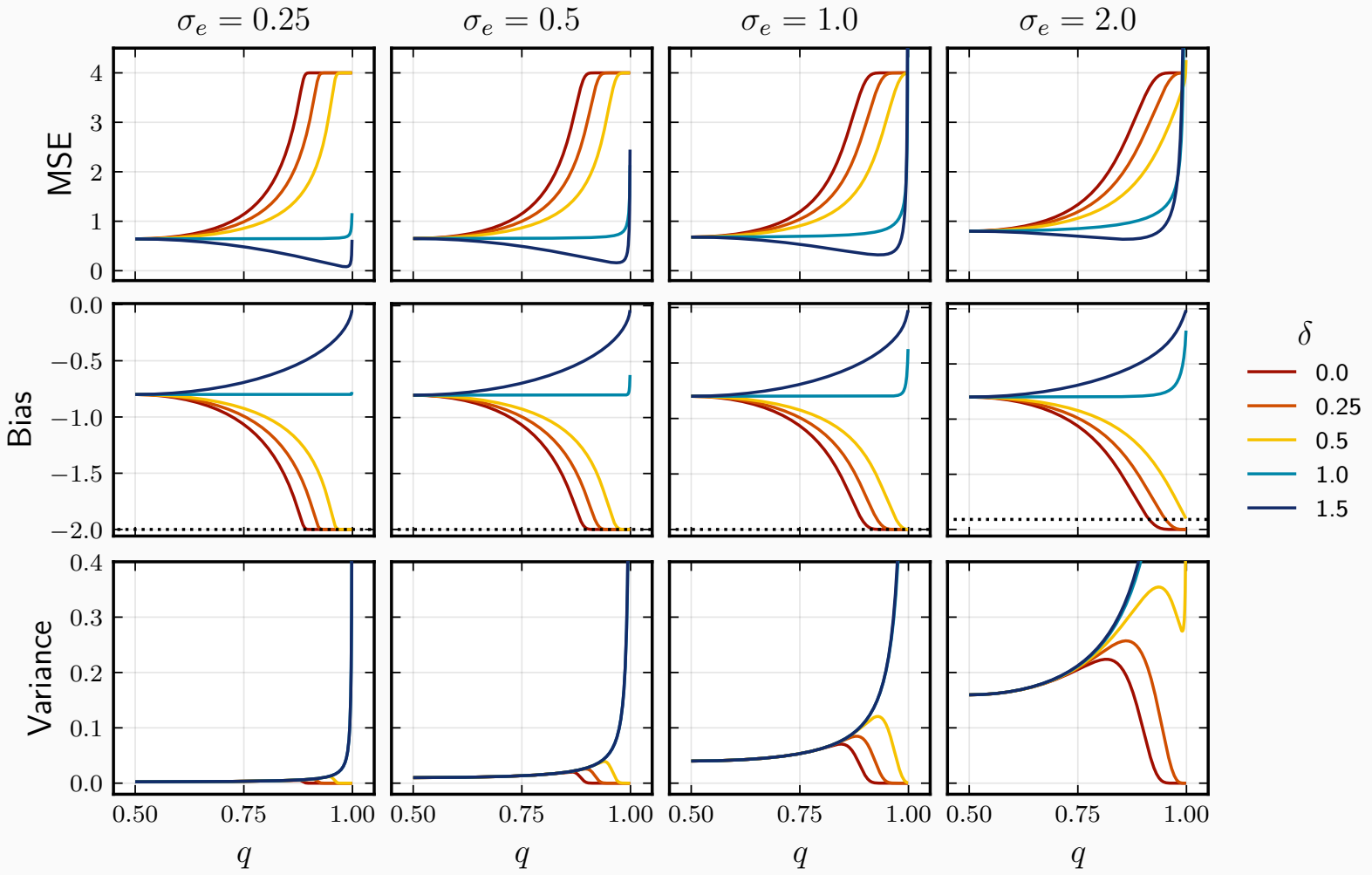
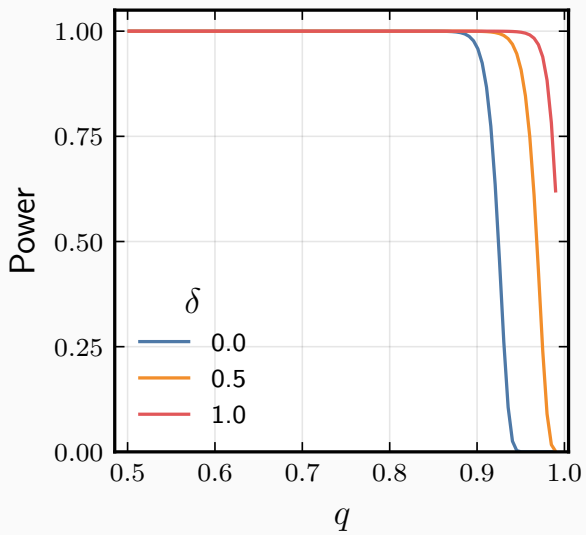


Figure 28: Bias, variance, and mean-squared error for a one-dimensional lasso problem. Theoretical result for orthogonal features. Dotted line is asymptotic result or $\delta = 1/2$.

Multiple Features: Power, FDR, and NMSE

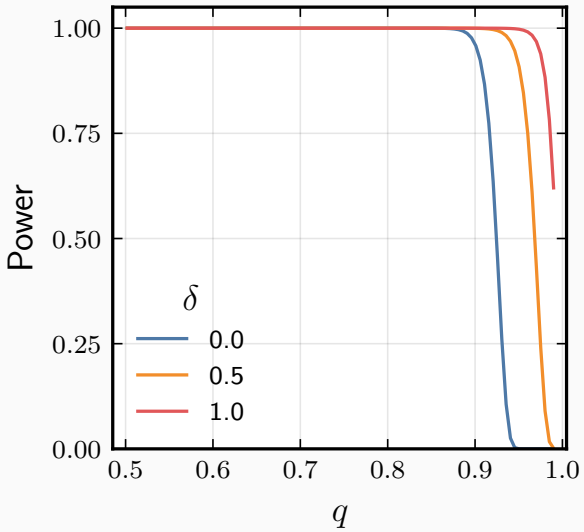
Lasso example with 10 true signals and varying q and p.



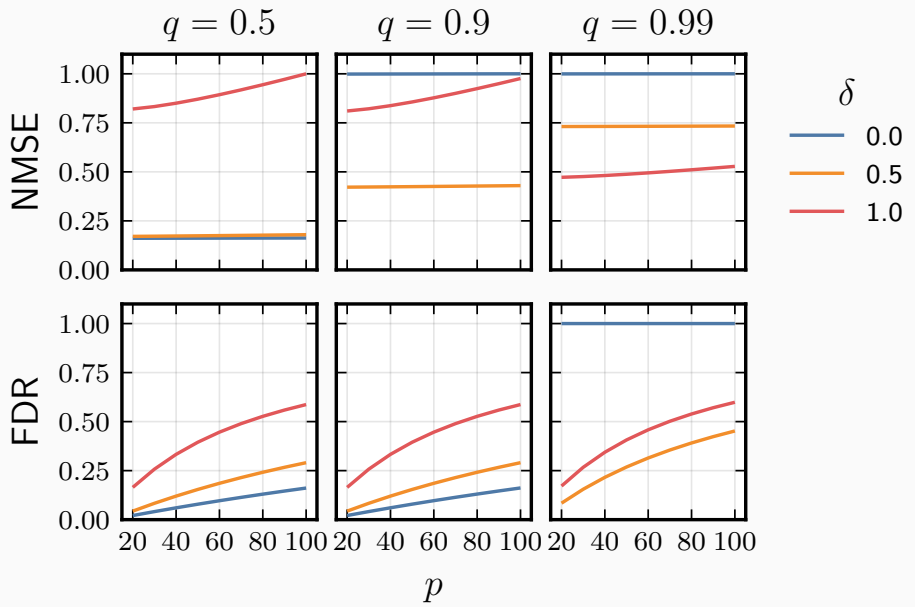
(a) Power in the sense of detecting all the true signals. Constant p.

Multiple Features: Power, FDR, and NMSE

Lasso example with 10 true signals and varying q and p.



(a) Power in the sense of detecting all the true signals. Constant p.



(b) False discovery rate (FDR) and normalized mean-squared error (NMSE).

Figure 29: Mean squared error (MSE), false discovery rate (FDR), and power.

Hyperparameter Optimization

Idea: The choice of δ affects the model, so let's optimize over it.

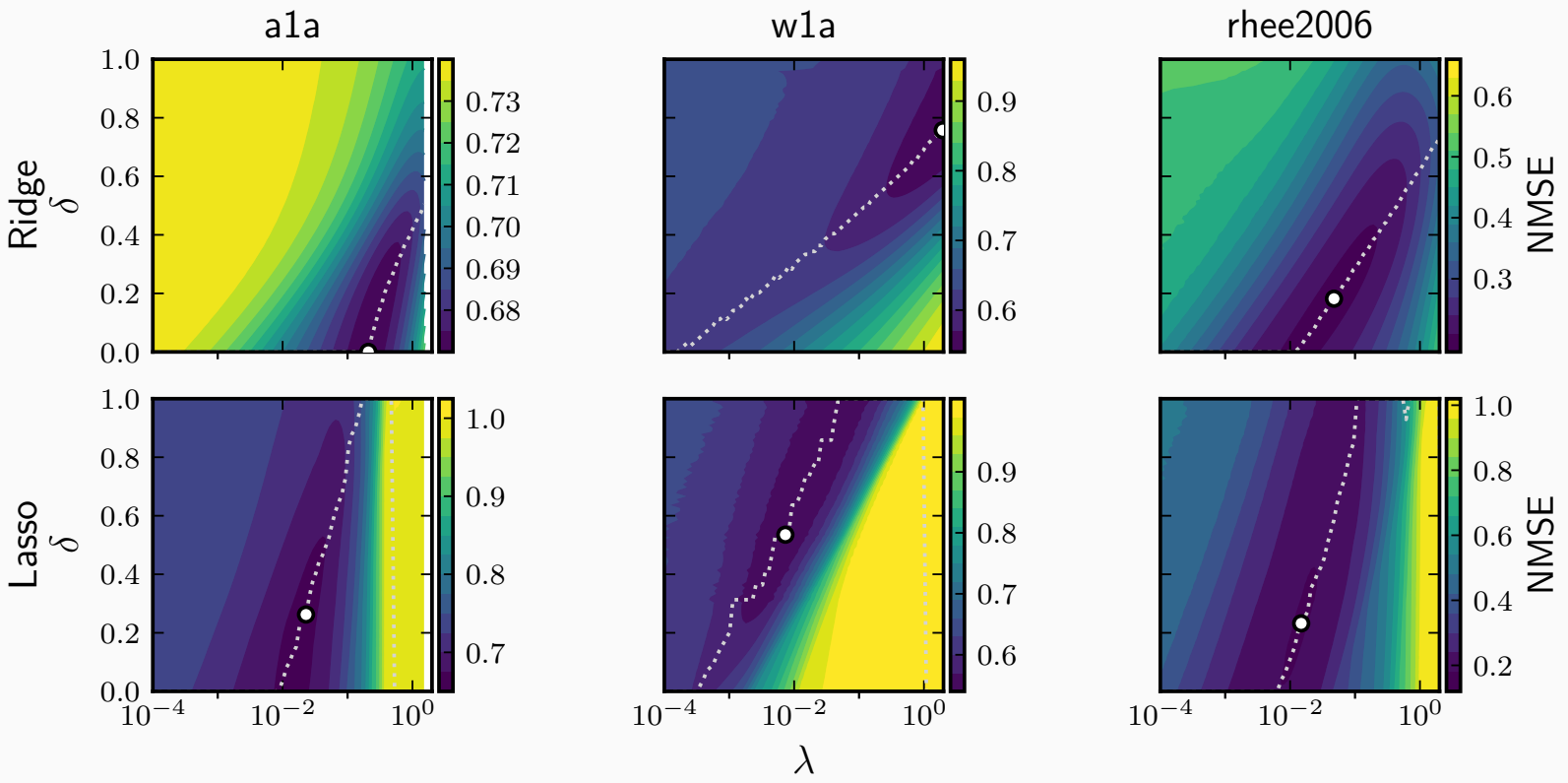


Figure 30: Contour plots of hold-out (validation set) error across a grid of δ and λ values for the lasso and ridge.