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

Optimization methods and numerical algorithms (screening rules) that help speed up the estimation of sparse regression models.

Synopsis

Overarching Theme

Optimization methods and numerical algorithms (screening rules) that help speed up the estimation of sparse regression models.

Overview

I–III Screening rules for sparse regression

IV Benchmarking optimization methods

V Coordinate descent (an optimization method) for SLOPE

VI Normalization in penalized regression

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 Malgorzata 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.
- Fitting models to sets of millions (maybe billions) of features is computationally costly.
- Hyperparameter tuning increases costs further.
- In sparse regression, we can solve a reduced problem if we somehow knew which features were active.
- Can we somehow figure this out **before** fitting the model?

Screening Rules

General Idea

1. If $p \gg n$, the solution will be sparse.[2]
2. Estimate feature importance before fitting (correlations, etc) and solve a reduced problem.
3. Maybe not so bad to miss some features—just add them back and update the fit.

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

Screening Rules

General Idea

1. If $p \gg n$, the solution will be sparse.[2]
2. Estimate feature importance before fitting (correlations, etc) and solve a reduced problem.
3. Maybe not so bad to miss some features—just add them back and update the fit.

Turn 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

$\boldsymbol{\beta}$ is optimal if

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

where $\partial h(\boldsymbol{\beta})$ 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

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

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

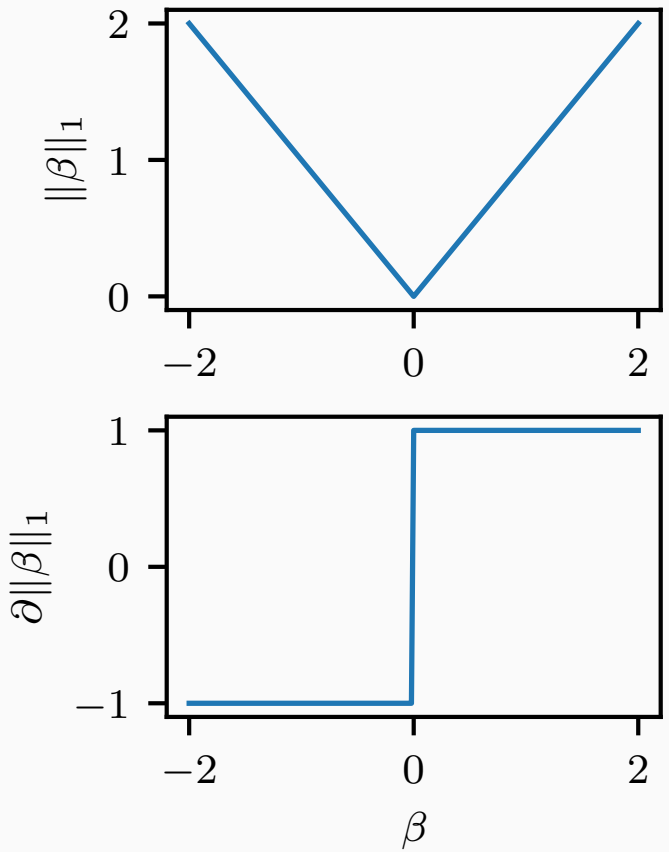


Figure 1: The lasso penalty ($h(\beta) = \lambda\|\beta\|_1$) and its subdifferential

Correlations and Coefficients

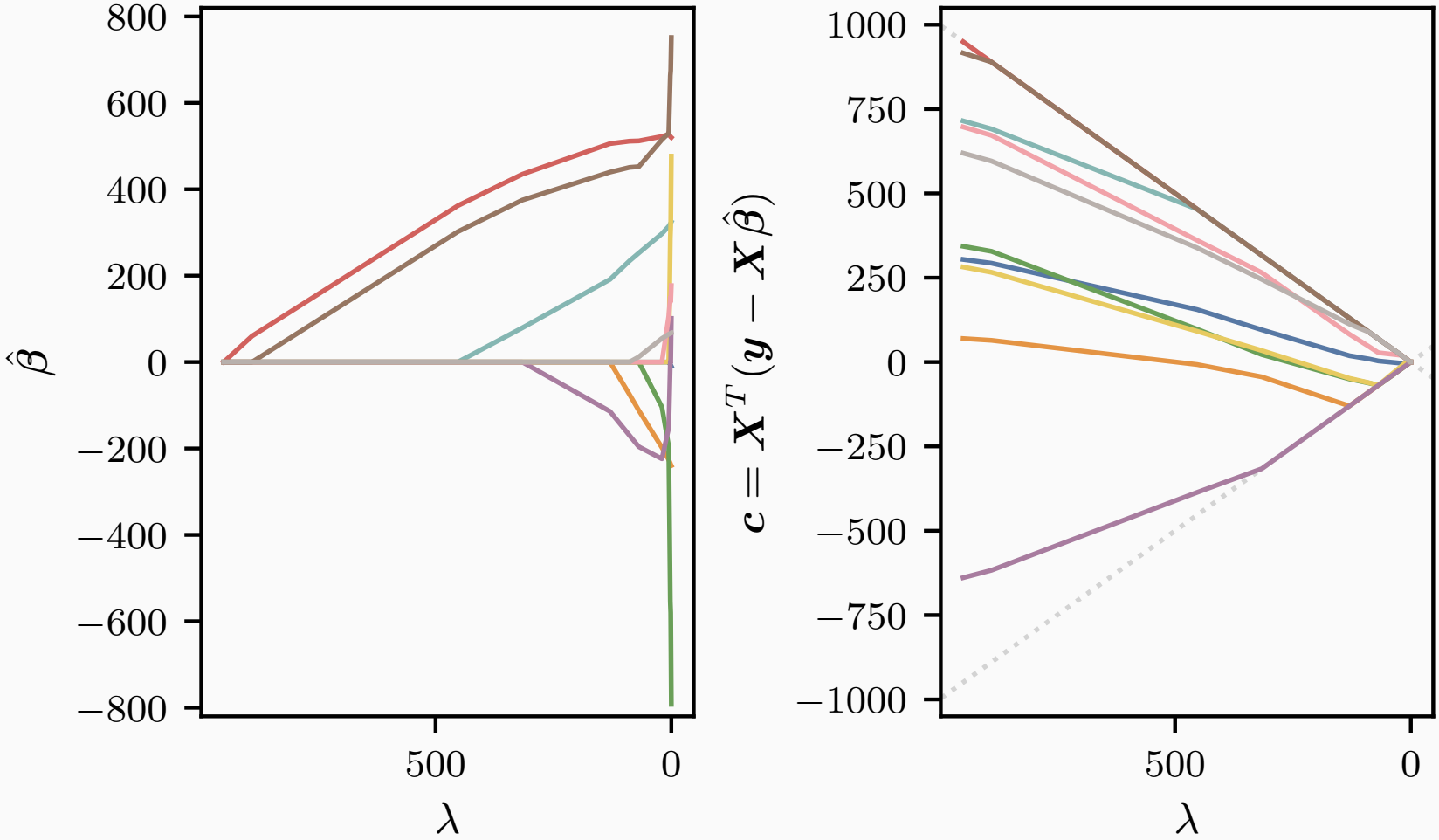


Figure 2: Coefficients and correlations along the lasso path

Strong Rule for the Lasso

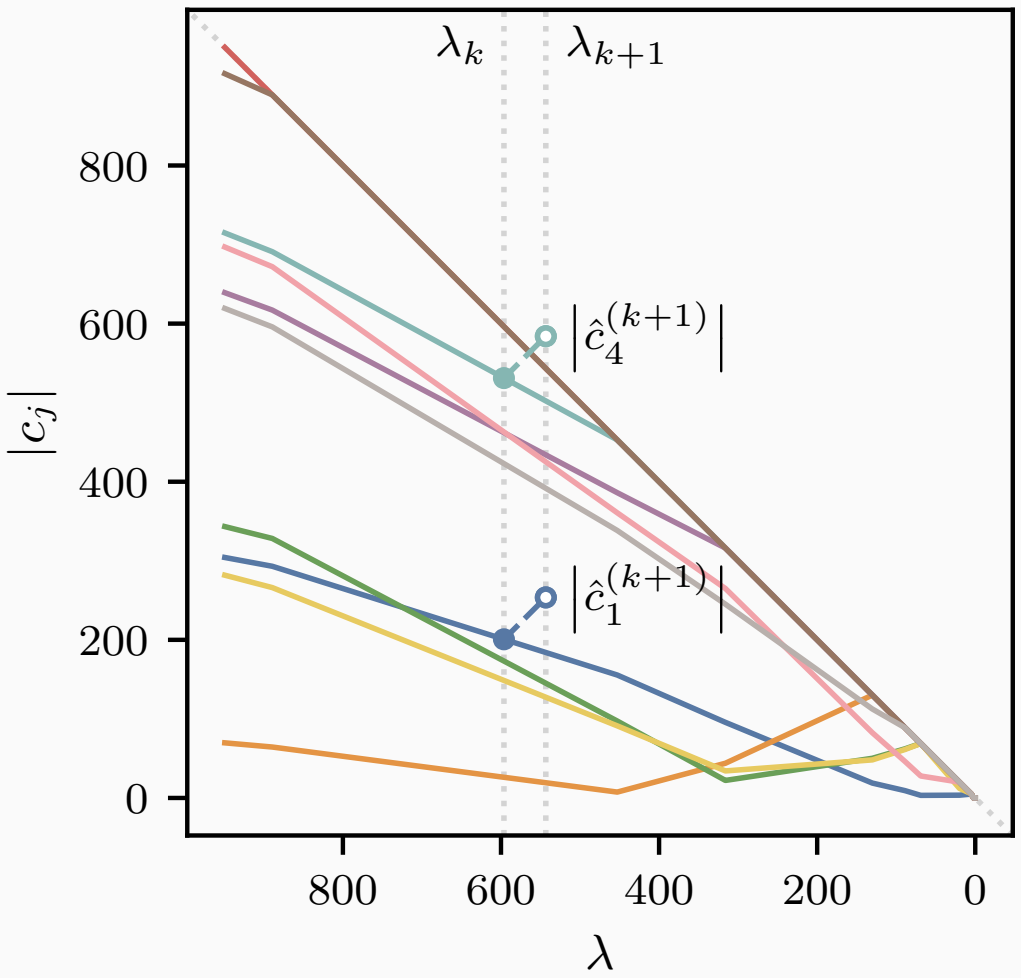


Figure 3: The strong rule for the lasso (Tibshirani et al. 2012)

Strong Rule for SLOPE

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

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

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

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

Results: Effectiveness

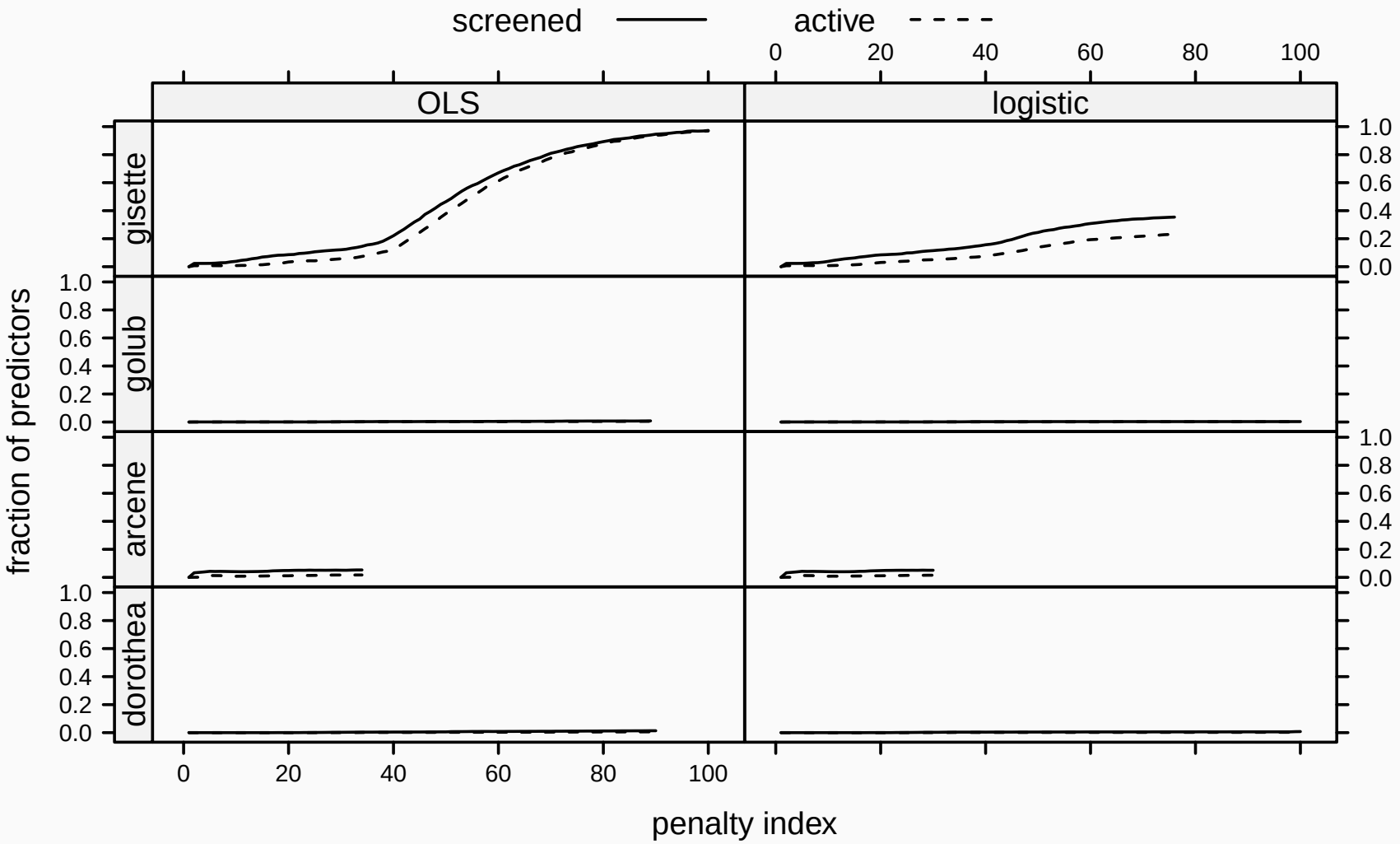


Figure 4: Effectiveness for some real data sets

Results: Speed

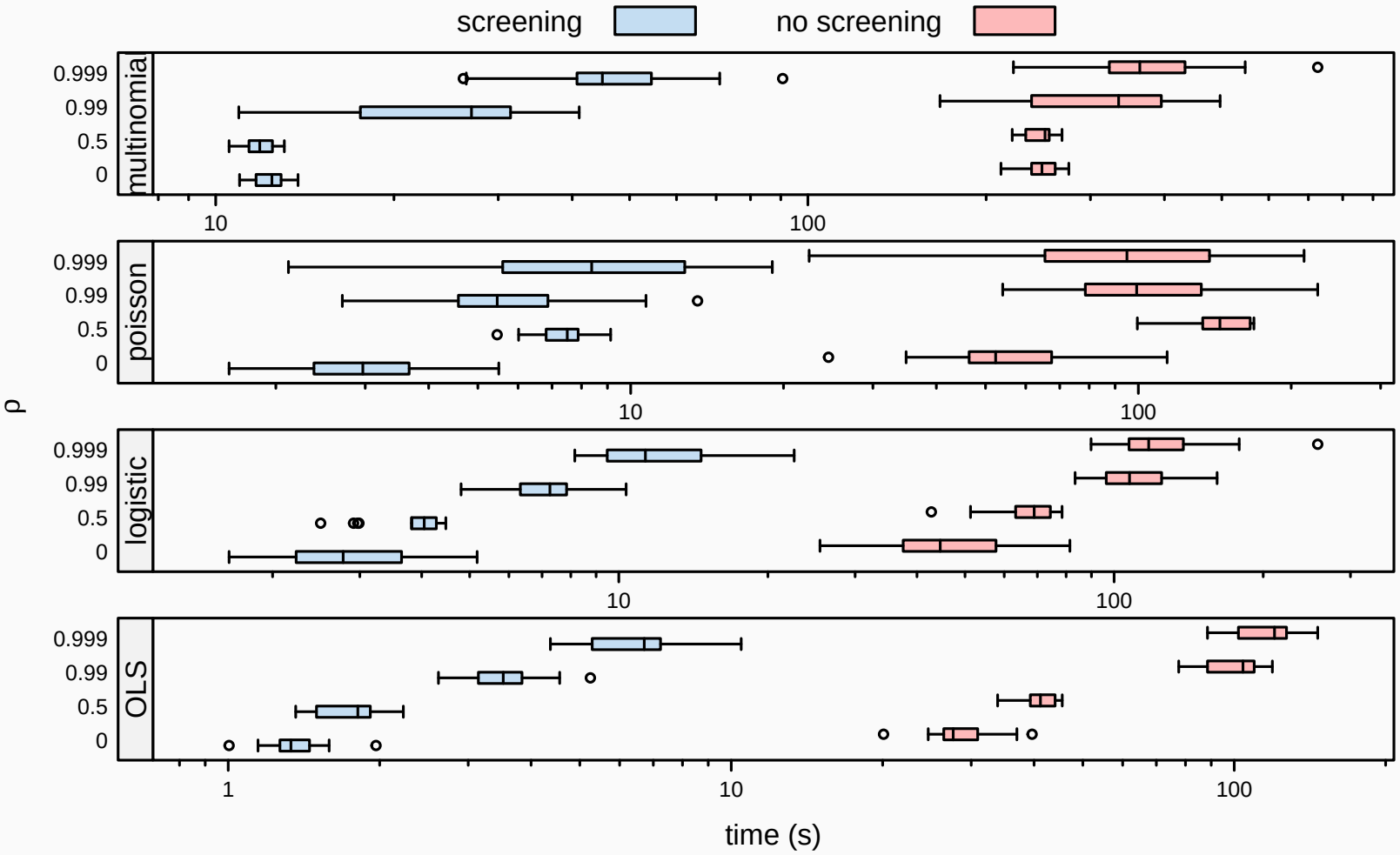


Figure 5: Time to fit a full SLOPE path with and without the strong rule

Heuristic Screening and Violations

- The rule is **heuristic**[3], which means that violations may occur.
- But they are so rare and checking for and refitting with them so cheap that it rarely matters.

[3] As opposed to *safe*.

Heuristic Screening and Violations

- The rule is **heuristic**[3], which means that violations may occur.
- But they are so rare and checking for and refitting with them so cheap that it rarely matters.

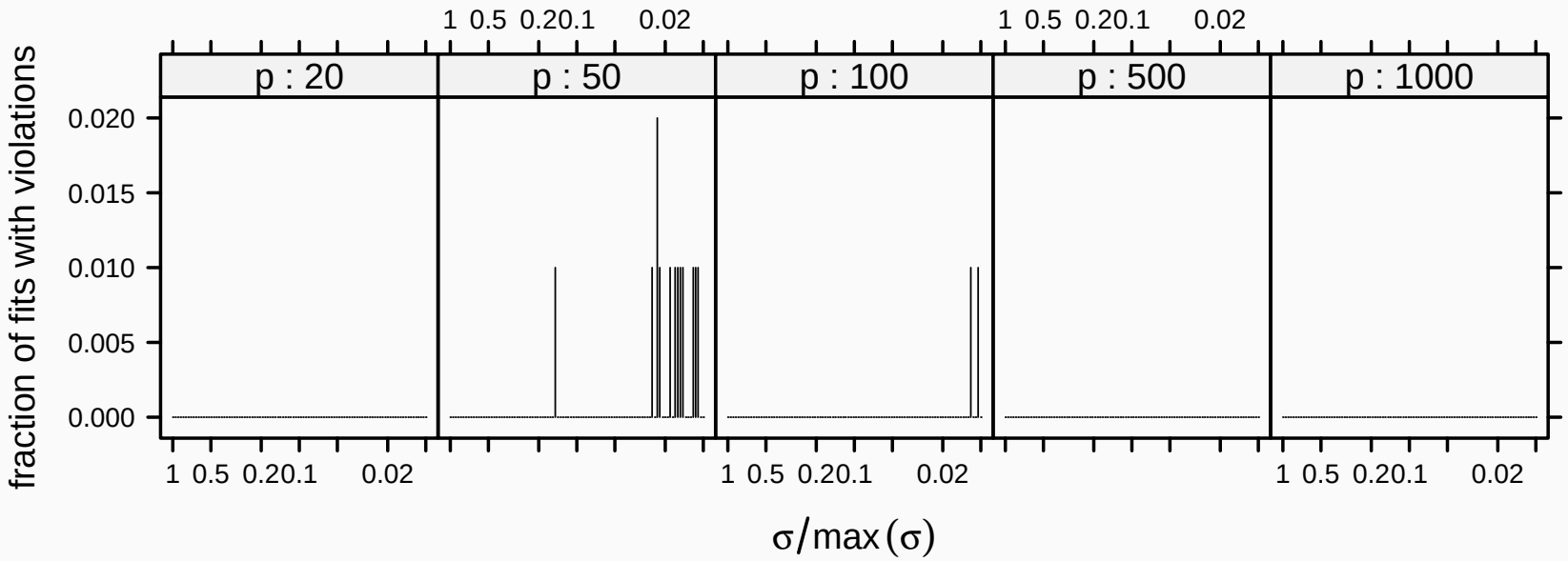


Figure 6: Violations of the strong rule for SLOPE

[3]As opposed to *safe*.

Summary

Contributions

- The first screening rule for SLOPE
- An efficient algorithm with which to apply it
- A new formulation of the subdifferential for SLOPE
- Implemented in the R package `SLOPE`[4]

[4] https://doi.org/10.32614/CRAN.package.SLOPE

Summary

Contributions

- The first screening rule for SLOPE
- An efficient algorithm with which to apply it
- A new formulation of the subdifferential for SLOPE
- Implemented in the R package `SLOPE`[4]

Limitations

- Somewhat conservative

[4] https://doi.org/10.32614/CRAN.package.SLOPE

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?

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

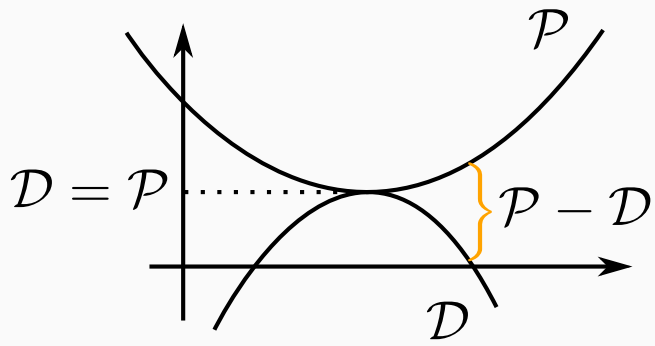
Complementary to the primal problem. For the lasso, it is

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

Duality Gap

Difference between the primal and dual objectives and is tight at the optimum, that is

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



The Dual

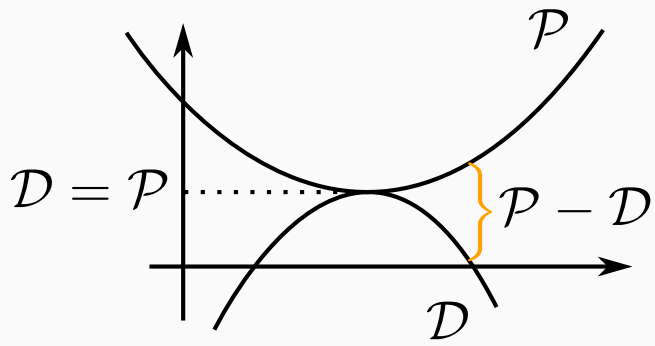
Complementary to the primal problem. For the lasso, it is

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

Duality Gap

Difference between the primal and dual objectives and is tight at the optimum, that is

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



Dual–Primal Relationship

The two problems are related via

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}(\lambda) + \lambda \boldsymbol{\theta}(\lambda).$$

Gap Safe Screening

Recall the optimality conditions for the lasso:

$$\text{if} \quad |\nabla g(\boldsymbol{\beta}^*)_j| < \lambda \iff |\boldsymbol{x}_j^\intercal \boldsymbol{\theta}^*| < 1 \quad \text{then} \quad \beta_j^* = 0.$$

Gap Safe Screening

Recall the optimality conditions for the lasso:

$$\text{if} \quad |\nabla g(\boldsymbol{\beta}^*)_j| < \lambda \iff |\boldsymbol{x}_j^\intercal \boldsymbol{\theta}^*| < 1 \quad \text{then} \quad \beta_j^* = 0.$$

Idea

Replace the inequality $|\boldsymbol{x}_j^\intercal \boldsymbol{\theta}^*| < 1$ with an inequality that offers same conclusion, but doesn't involve $\boldsymbol{\theta}^*$.

Gap Safe Screening

Recall the optimality conditions for the lasso:

$$\text{if} \quad |\nabla g(\boldsymbol{\beta}^*)_j| < \lambda \iff |\boldsymbol{x}_j^\intercal \boldsymbol{\theta}^*| < 1 \quad \text{then} \quad \beta_j^* = 0.$$

Idea

Replace the inequality $|\boldsymbol{x}_j^\intercal \boldsymbol{\theta}^*| < 1$ with an inequality that offers same conclusion, but doesn't involve $\boldsymbol{\theta}^*$.

Gap Safe screening (Fercoq, Gramfort, and Salmon 2015) uses the **duality gap** to define such a region, and discards the jth predictor if

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

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

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

Look-Ahead Screening

Inequality on last slide is quadratic, 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 on last slide is quadratic, which means we can find the next critical point easily:

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

where

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

This allows us to screen predictors for **all** upcoming steps by simply checking the inequality for all $\lambda_{k+1}, \lambda_{k+2}, \ldots$

Simple Example

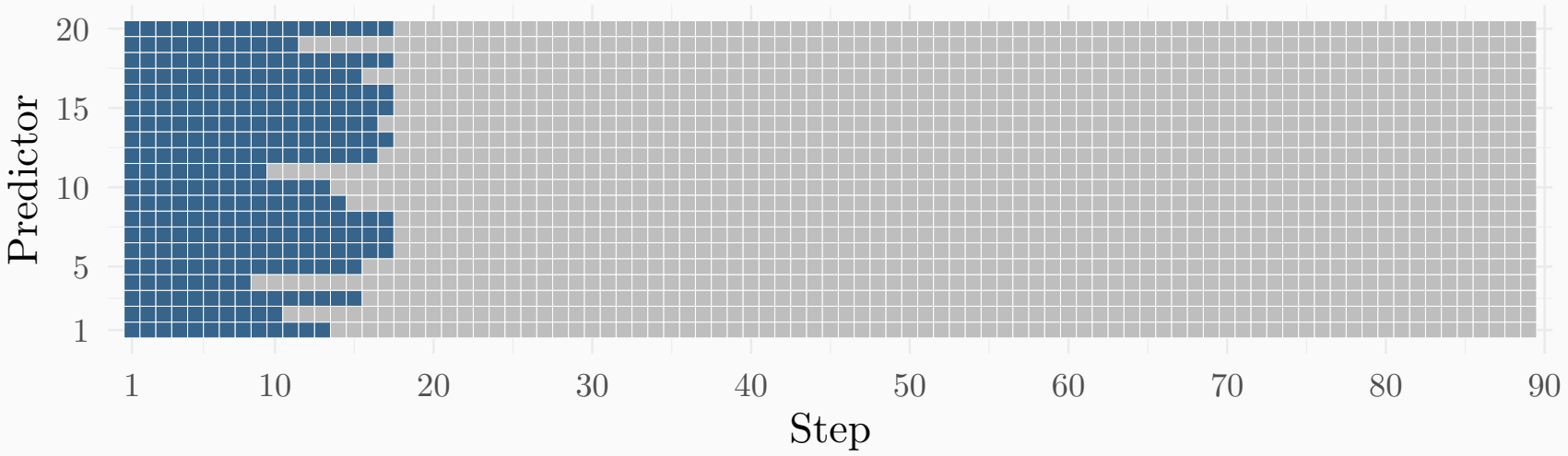


Figure 7: The lasso path for a sample of 20 features from the leukemia data set. The squares show for which steps the feature can be discarded.

Benchmarks

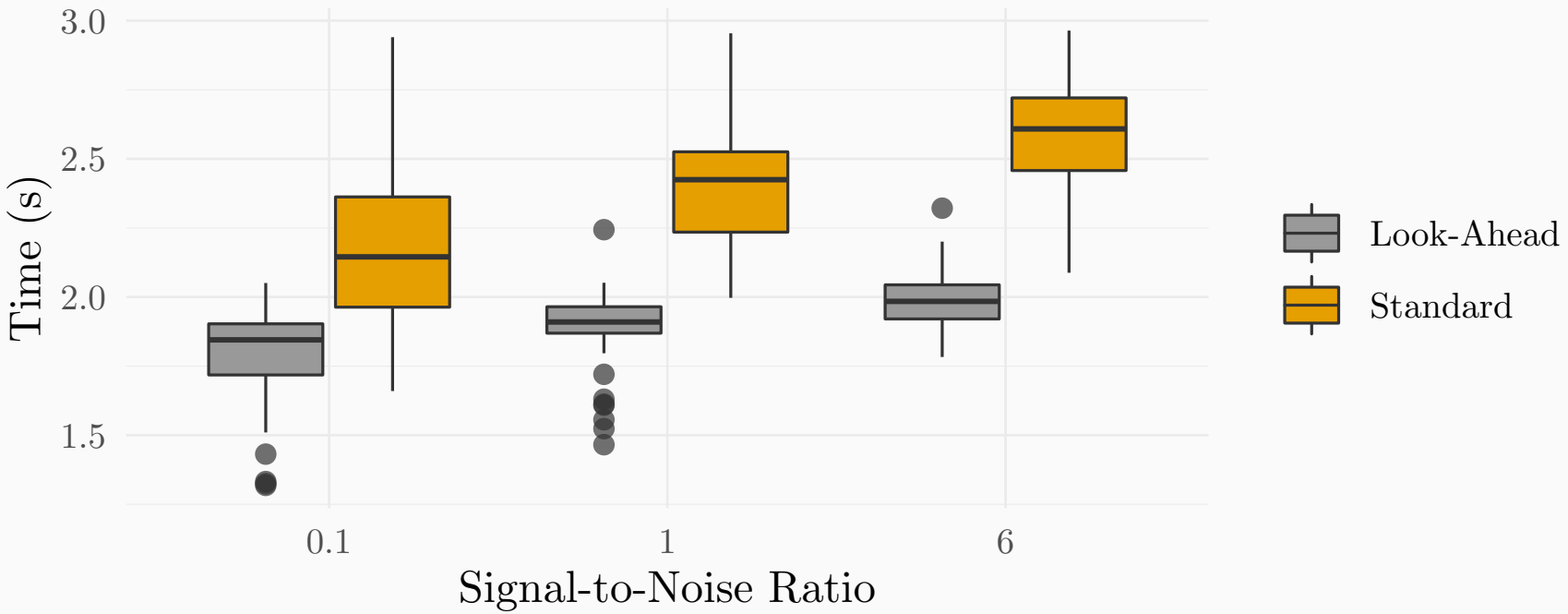


Figure 8: Benchmarks of time to fit the full lasso path with or without look-ahead screening.

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
- Most effective at start of path

The Hessian Screening Rule (Paper III)

The Hessian Screening Rule

Published and presented at NeurIPS 2022[6]

[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.
- Using second order information, we might both be able to improve screening, but also improve solving the problem.

Screening Rules Seen As Gradient Estimates

Let $\boldsymbol{c}(\lambda) = -\nabla g(\boldsymbol{\beta}(\lambda))$ be the correlation (negative gradient).

Lasso Screening Rule Template

$\mathbf{0} \in \nabla g(\boldsymbol{\beta}) + \lambda \partial$ suggests simple rule:

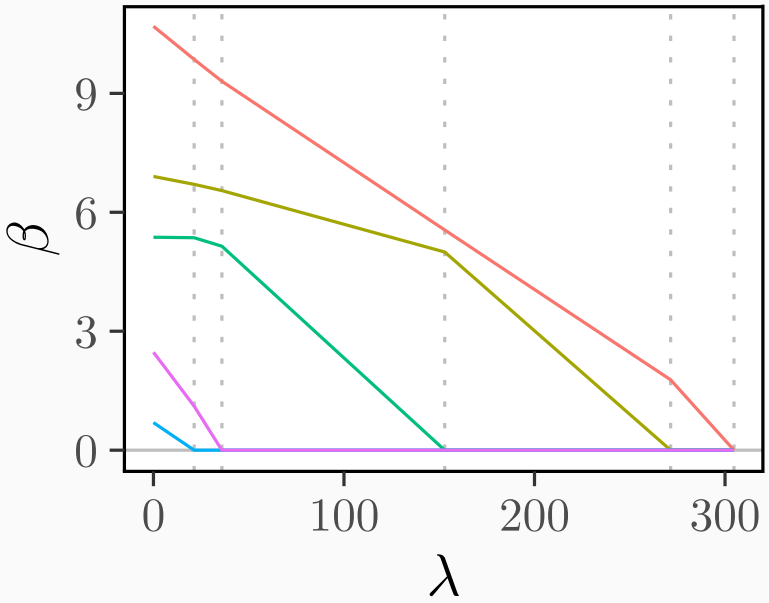
1. Replace $\boldsymbol{c}$ with an estimate $\tilde{\boldsymbol{c}}$.
2. If $|\tilde{\boldsymbol{c}}_j| < \lambda$, discard feature j.

If $\tilde{\boldsymbol{c}}$ is accurate and not too conservative, we have a useful rule.

The Ordinary Lasso

We now focus on the ordinary lasso, ℓ_1-regularized least squares:

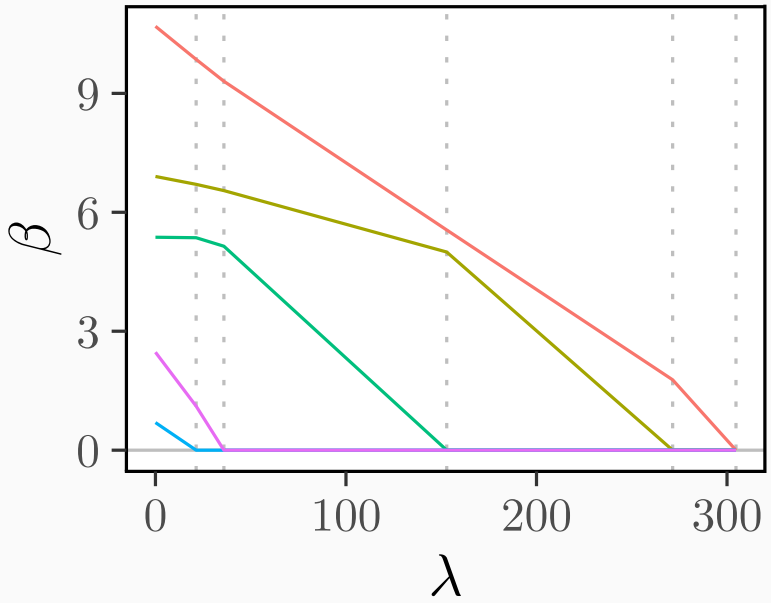
$$g(\boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2,$$
$$\nabla g(\boldsymbol{\beta}) = \boldsymbol{X}^\intercal(\boldsymbol{X}\boldsymbol{\beta} - \boldsymbol{y}).$$



The Ordinary Lasso

We now focus on the ordinary lasso, ℓ_1-regularized least squares:

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



It turns out that we can express the solution as a function of λ:

$$\hat{\boldsymbol{\beta}}(\lambda) = \left(\boldsymbol{X}_{\mathcal{A}}{}^\intercal \boldsymbol{X}_{\mathcal{A}}\right)^{-1}\left(\boldsymbol{X}_{\mathcal{A}}{}^\intercal \boldsymbol{y} - \lambda \operatorname{sign}(\hat{\boldsymbol{\beta}}_{\mathcal{A}})\right).$$

Holds for $[\lambda_k, \lambda_{k+1}]$ if active set remains constant, and so

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

The Hessian Screening Rule

Basic Form of the Rule

Take expression for $\hat{\beta}$ on last slide and stick it into the gradient at step $k+1$:

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

The Hessian Screening Rule

Basic Form of the Rule

Take expression for $\hat{\beta}$ on last slide and stick it into the gradient at step $k+1$:

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

- Exact expression for $\boldsymbol{c}$ at $k+1$ if active set is constant.
- Heuristic rule, so needs checks for violations.

Screening Rule Comparison

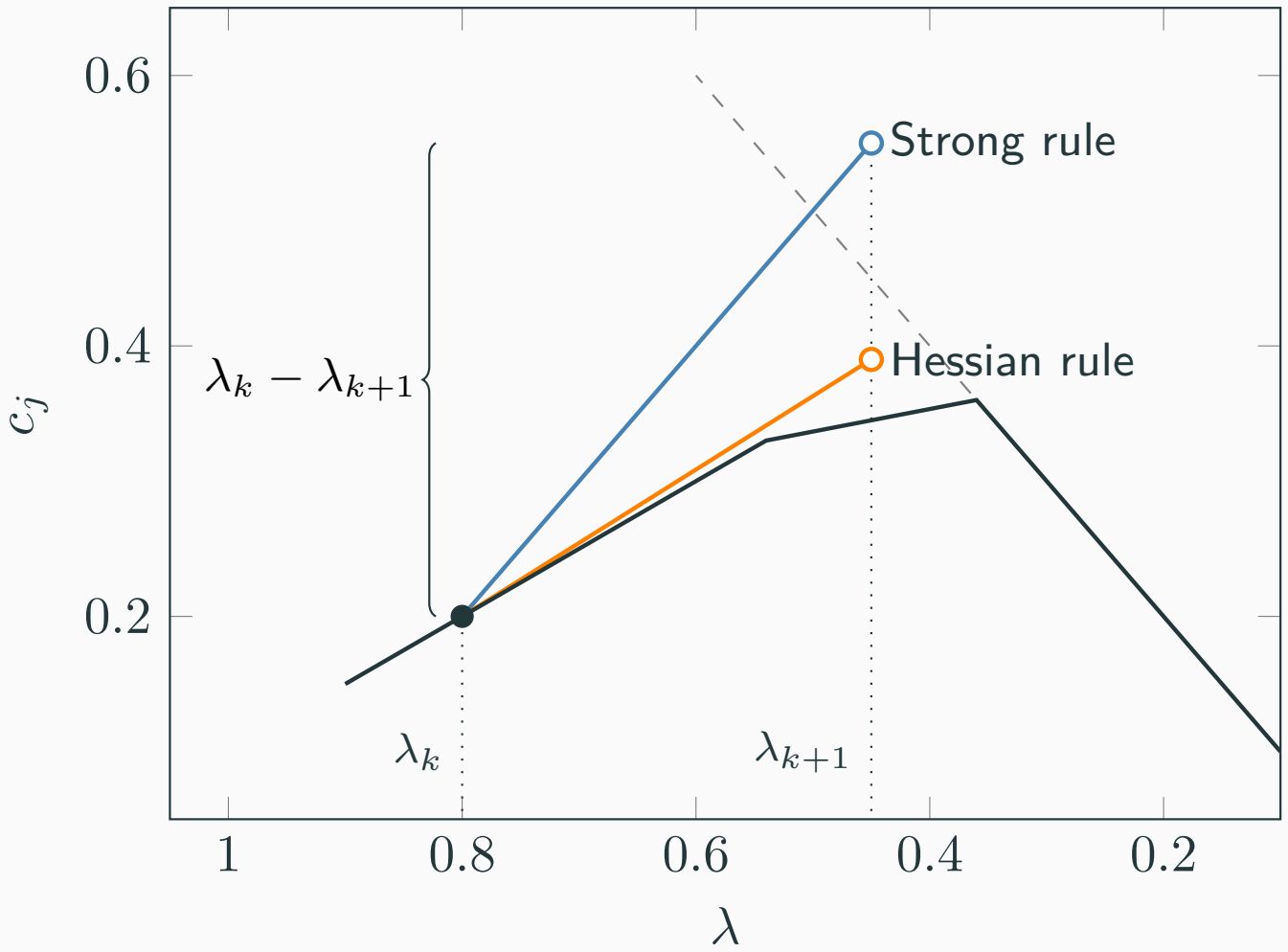


Figure 9: The strong and Hessian screening rules in action

Results: Effectiveness

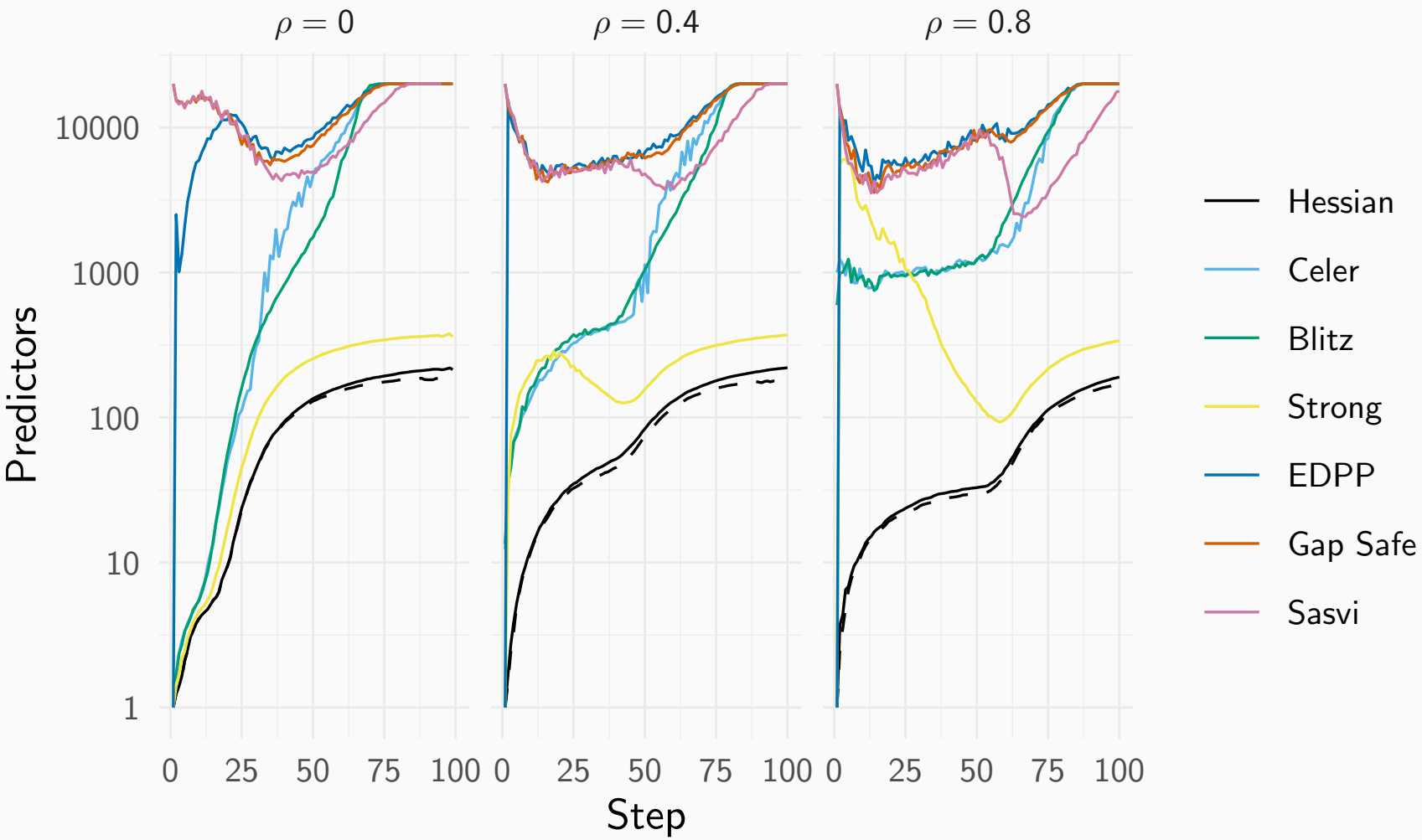


Figure 10: Features screened for a lasso path for ℓ_1-regularized least-squares to a design with varying correlation (ρ), $n = 200$, and $p = 20000$.

Results: Simulated Data

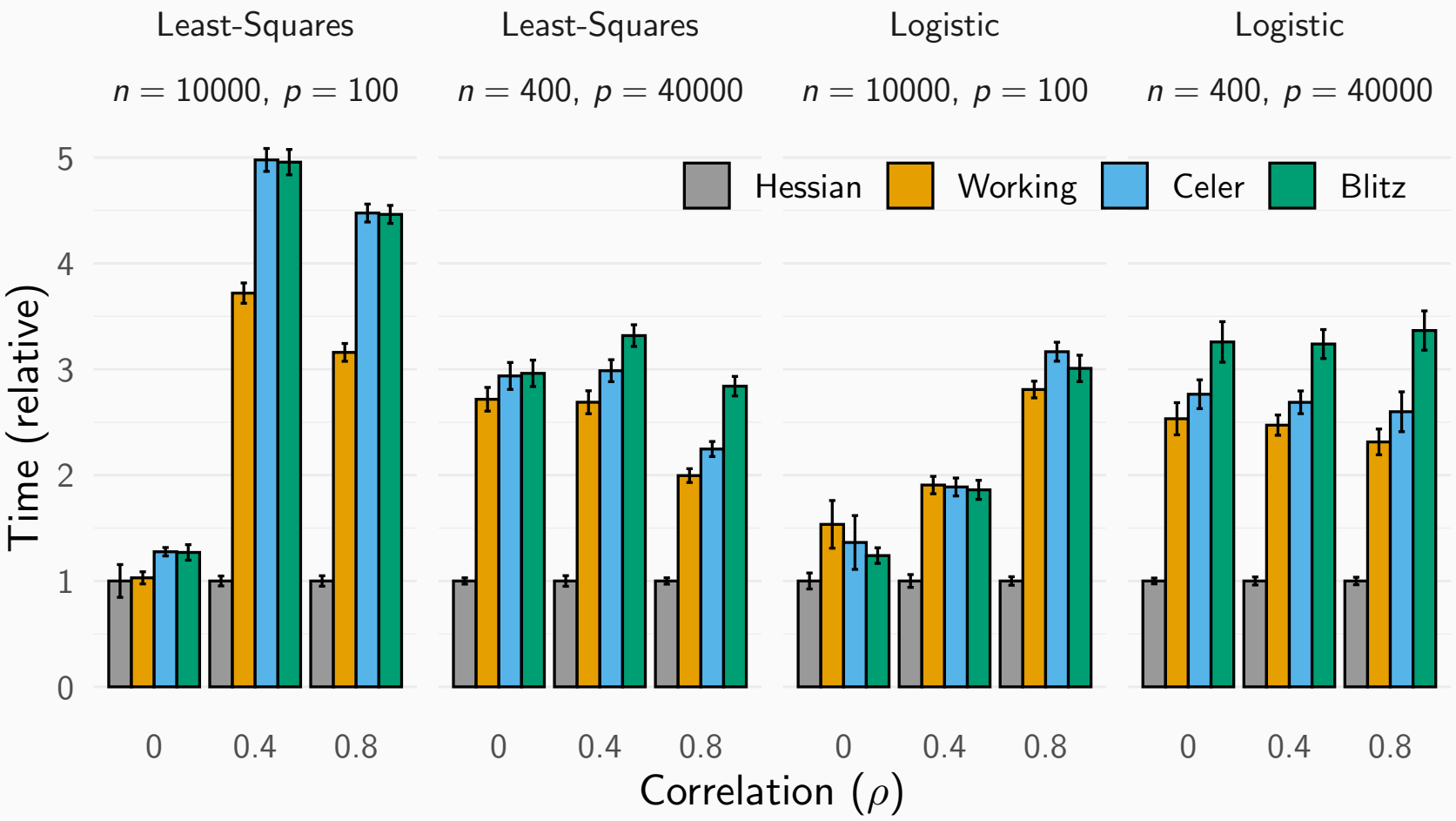


Figure 11: Time to fit a full path of ℓ_1-regularized least-squares and logistic regression to a design with n observations, p features, and pairwise correlation between features of ρ.

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.
- Can be used in combination with approximate homotopy.

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.
- Can be used in combination with approximate homotopy.

Limitations

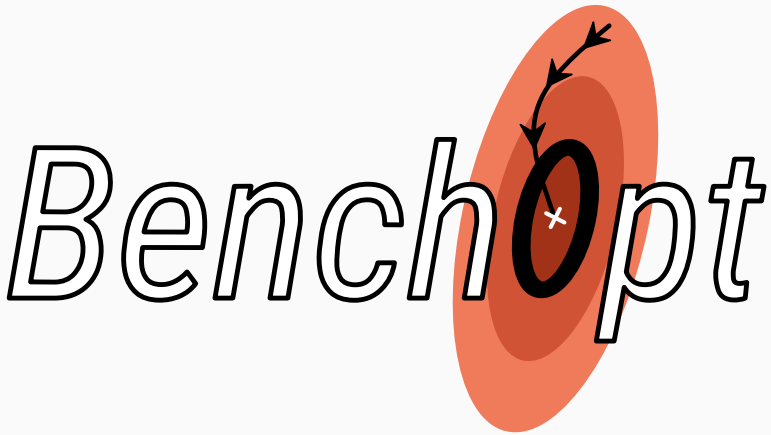
- Not well-tailored to problems with more complex Hessians.
- Implementation is a bit more involved.
- Although the rule works well with high correlation, speed-ups are not commensurate since our warm starts are worse.

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
- Best choice often depends on problem:
  - data set (dimensions, sparsity, conditioning, normalization)
  - hyper-parameters (regularization)
  - implementation (language)
- Can we trust researchers to publish *fair* comparisons?

Benchopt attempts to help with all of this!

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)	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	(Hao 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 12: Methods benchmarked in Schmidt, Schneider, and Hennig (2021)

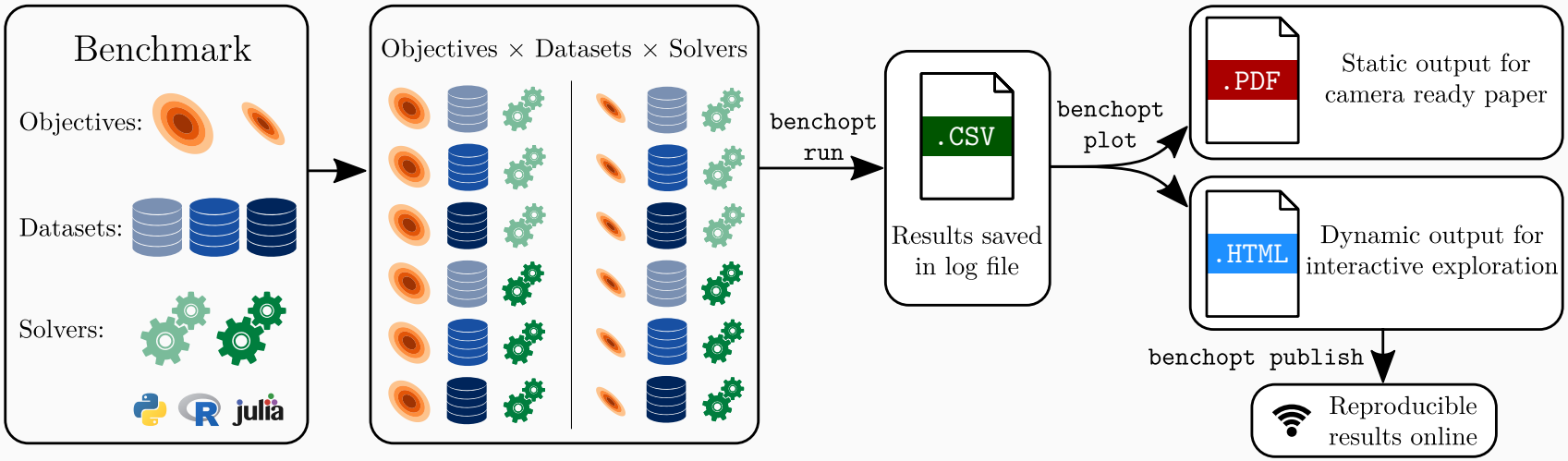


Figure 13: Schematic over how benchopt works

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]
```

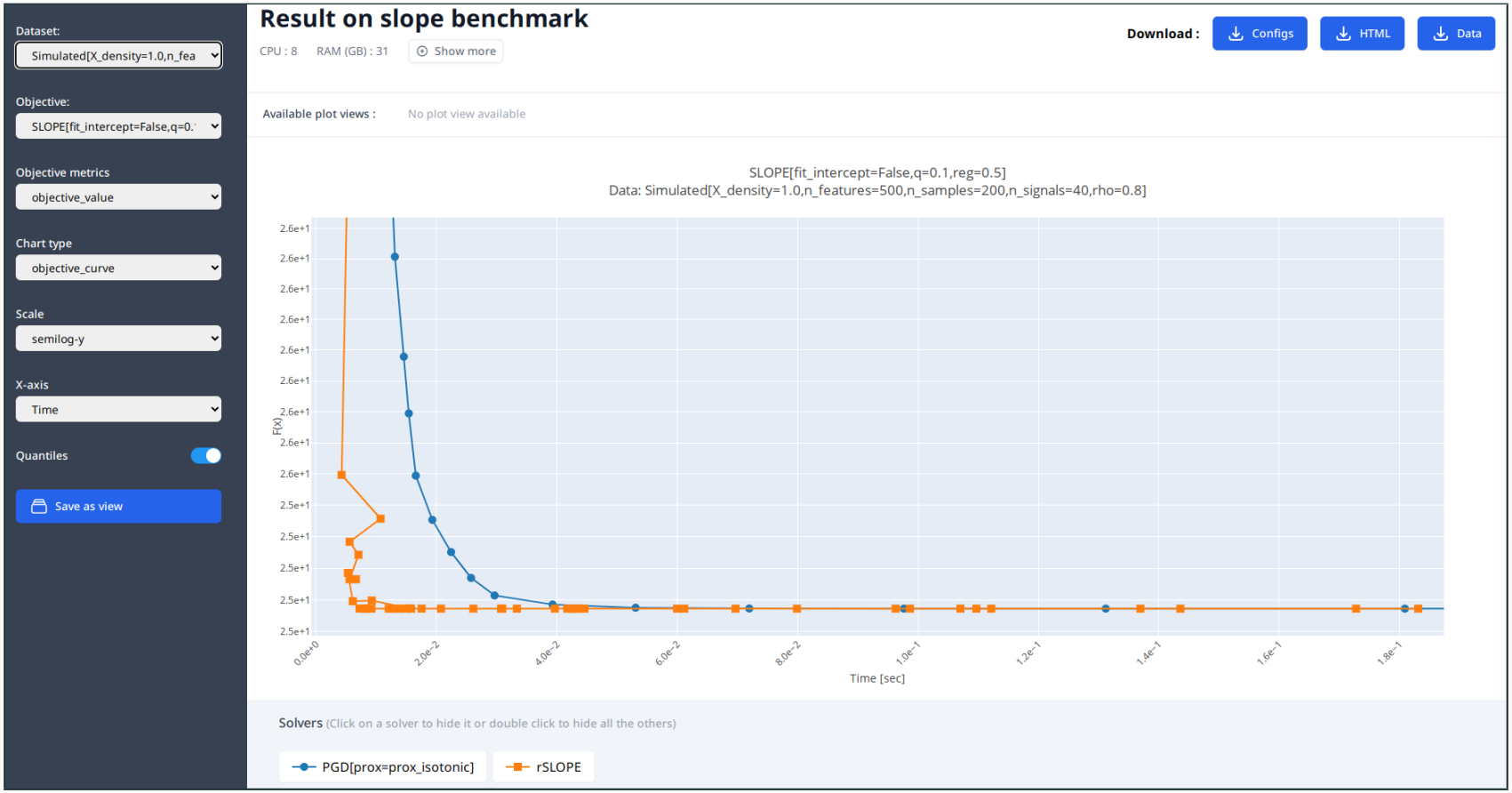


Figure 14: Benchopt benchmark results for SLOPE

Benchmark Structure

A benchopt benchmark is a directory with

- an `objective.py` file with an `Objective`,
- a directory `solvers` with one file per solver, and
- a directory `datasets` with dataset generators/fetchers.

```
my_benchmark/
├── README.rst
├── datasets
│   ├── simulated.py  # some dataset
│   └── real.py  # some dataset
├── objective.py  # contains the definition of the objective
└── solvers
    ├── solver1.py  # some solver
    └── solver2.py  # some solver
```

Summary

Contributions

A framework that simplifies benchmarking of optimization methods:

- Automatic installation of dependencies
- Parametrized datasets, objectives and solvers
- Caching
- Interactive visualizations and easy publishing of results
- Mostly language agnostic

Summary

Contributions

A framework that simplifies benchmarking of optimization methods:

- Automatic installation of dependencies
- Parametrized datasets, objectives and solvers
- Caching
- Interactive visualizations and easy publishing of results
- Mostly language agnostic

Limitations

- Doesn't directly solve the problem of varying implementations.
- 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)
- Lasso solvers are fast because they use coordinate descent
- Unfortunately, we cannot use coordinate descent directly for SLOPE
- But what if we could combine coordinate descent with another method?

Coordinate Descent

Simple Method!

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

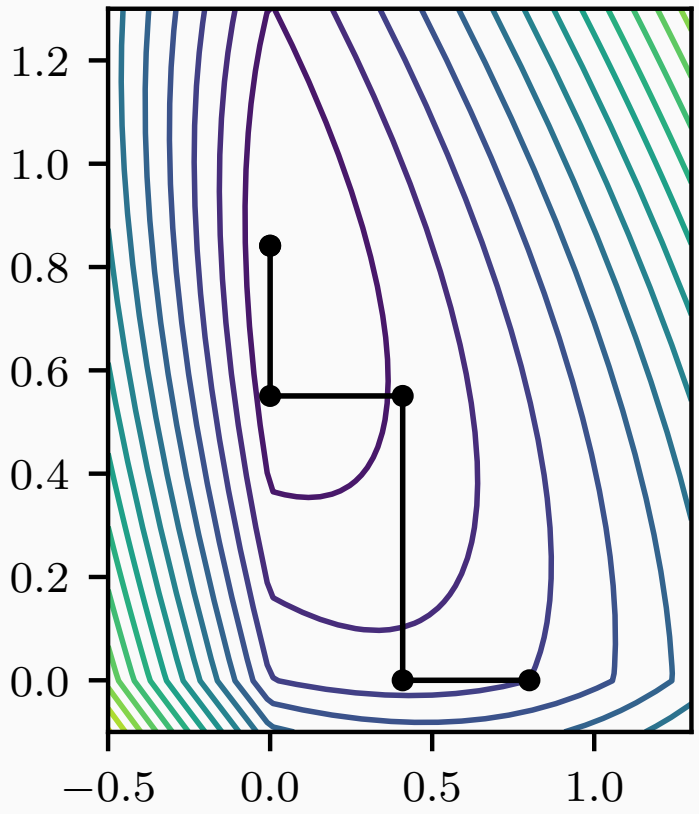


Figure 15: A basic example of coordinate descent in two dimensions

Coordinate Descent Performance

Performs (surprisingly) well!

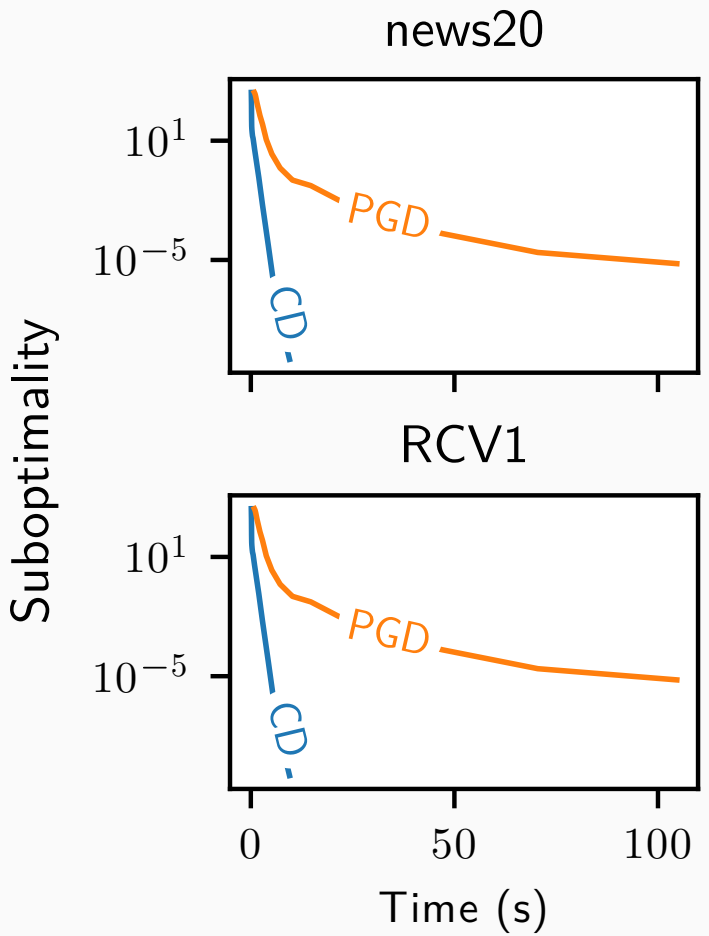


Figure 16: Coordinate descent versus proximal gradient descent for the lasso.

Coordinate Descent and Seperability

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

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

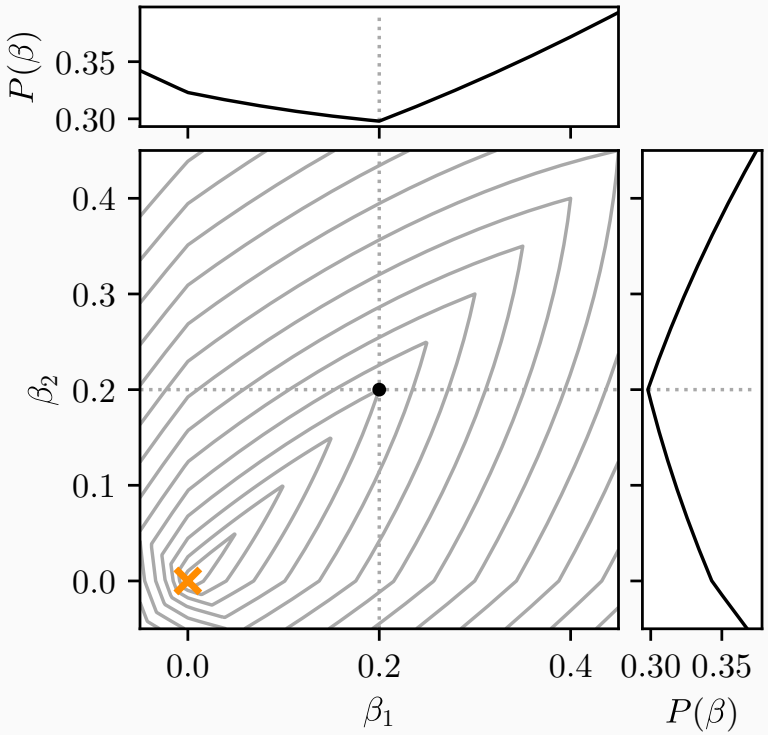


Figure 17: A *naive* coordinate descent algorithm cannot advance from the current iterate (●) to reach the optimum (✖).

Coordinate Descent and Seperability

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

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

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

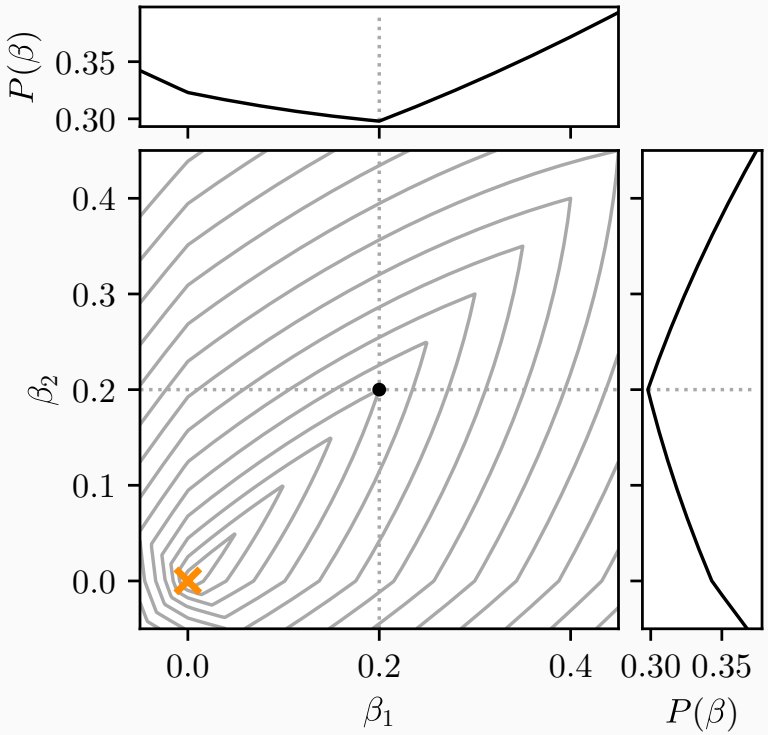


Figure 17: A *naive* coordinate descent algorithm cannot advance from the current iterate (●) to reach the optimum (✖).

Coordinate Descent and Seperability

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

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

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

Idea

Alternate between gradient descent and coordinate descent on the clusters.

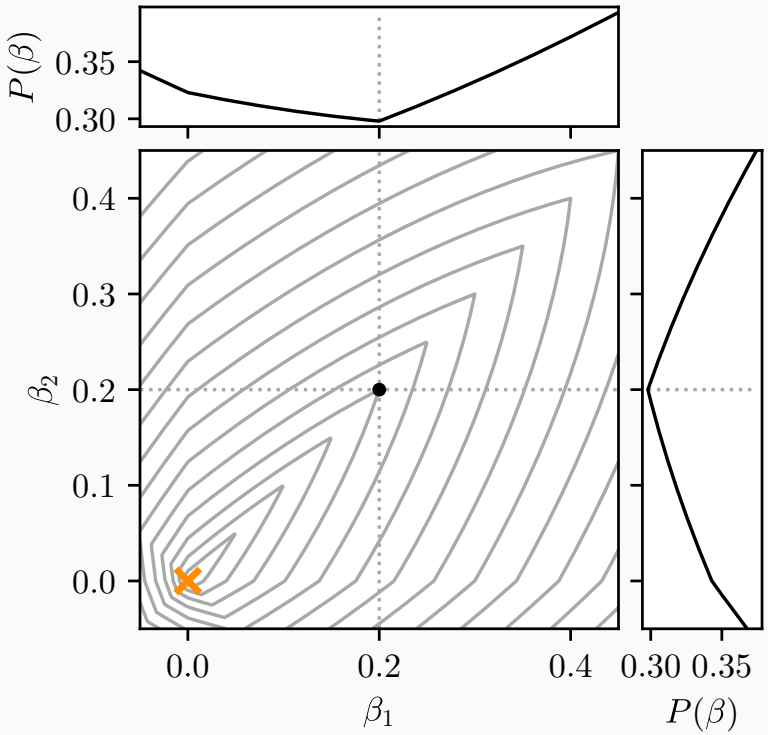


Figure 17: A *naive* coordinate descent algorithm cannot advance from the current iterate (●) to reach the optimum (✖).

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.

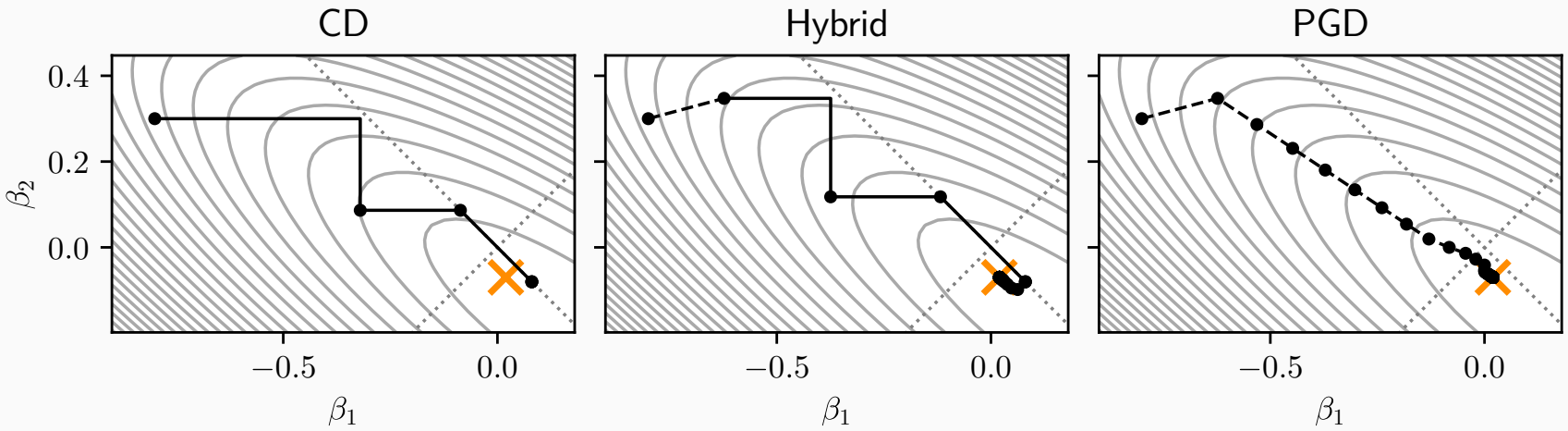


Figure 18: Our algorithm (hybrid) is a combination of CD and PGD.

Experiments: Simulated Data

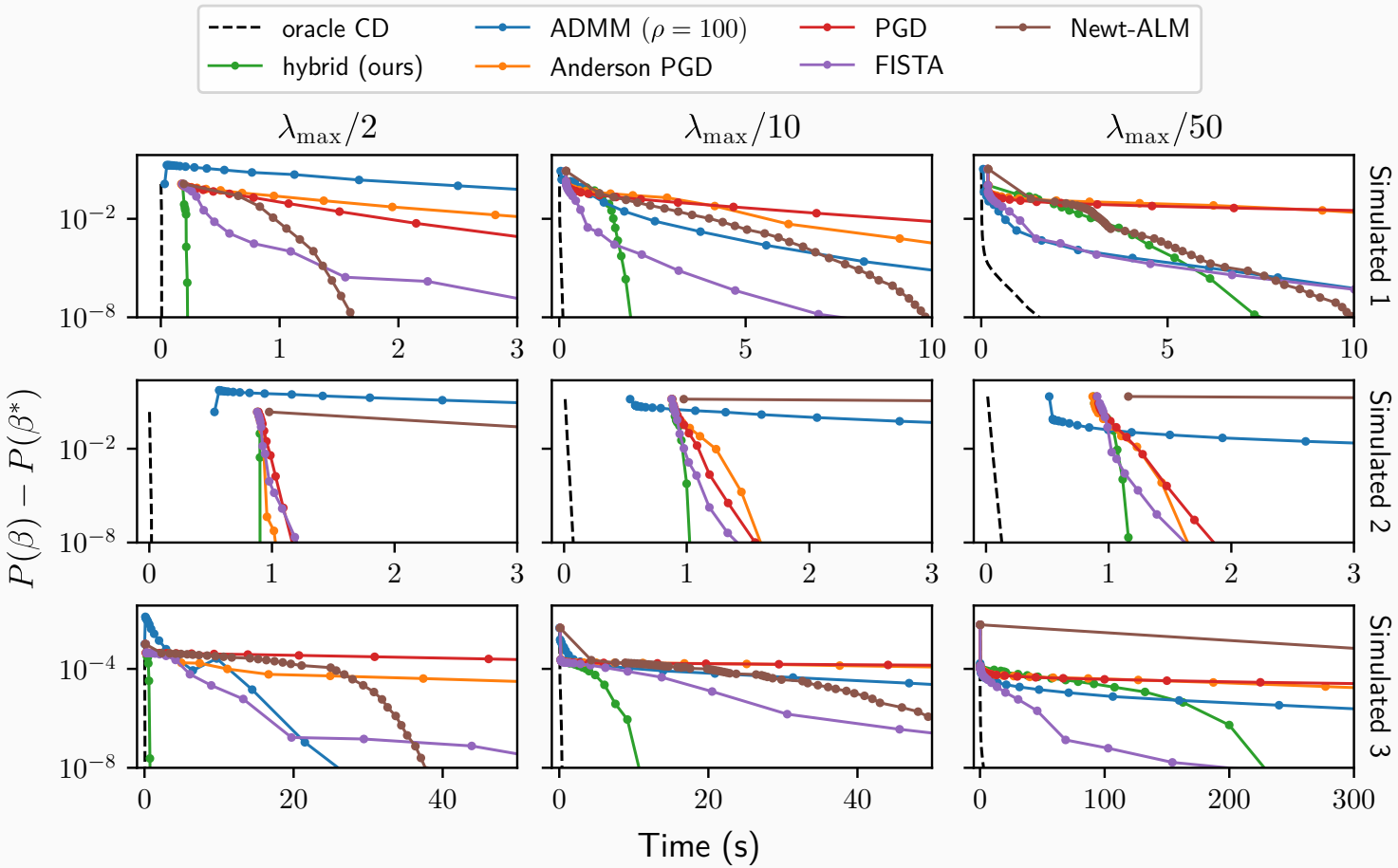


Figure 19: Benchmarks on simulated data. Scenario 1: $n = 200$ and $p = 20\,000$, X. Scenario 2: $n = 20\,000$ and $p = 200$. Scenario 3: $n = 200$, $p = 200\,000$, and sparse X.

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

- Implementation is a bit complex.
- 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 (e.g. SLOPE and lasso) are scale-sensitive

Motivation

- Most regularized methods (e.g. SLOPE and lasso) are scale-sensitive
- Straightforward when everything is normal, but about features that have other distributions, such as binary features?

Motivation

- Most regularized methods (e.g. SLOPE and lasso) are scale-sensitive
- Straightforward when everything is normal, but about features that have other distributions, such as binary features?
- No literature on the effects of different normalization strategies

The Elastic Net

Linear regression plus a combination of the ℓ_1 and ℓ_2 penalties:

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \underset{\beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^p}{\arg\min} \left(\frac{1}{2} \|\boldsymbol{y} - \beta_0 - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2 \right).$$

The Elastic Net

Linear regression plus a combination of the ℓ_1 and ℓ_2 penalties:

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \underset{\beta_0 \in \mathbb{R}, \beta \in \mathbb{R}^p}{\arg\min} \left(\frac{1}{2} \|\boldsymbol{y} - \beta_0 - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2 \right).$$

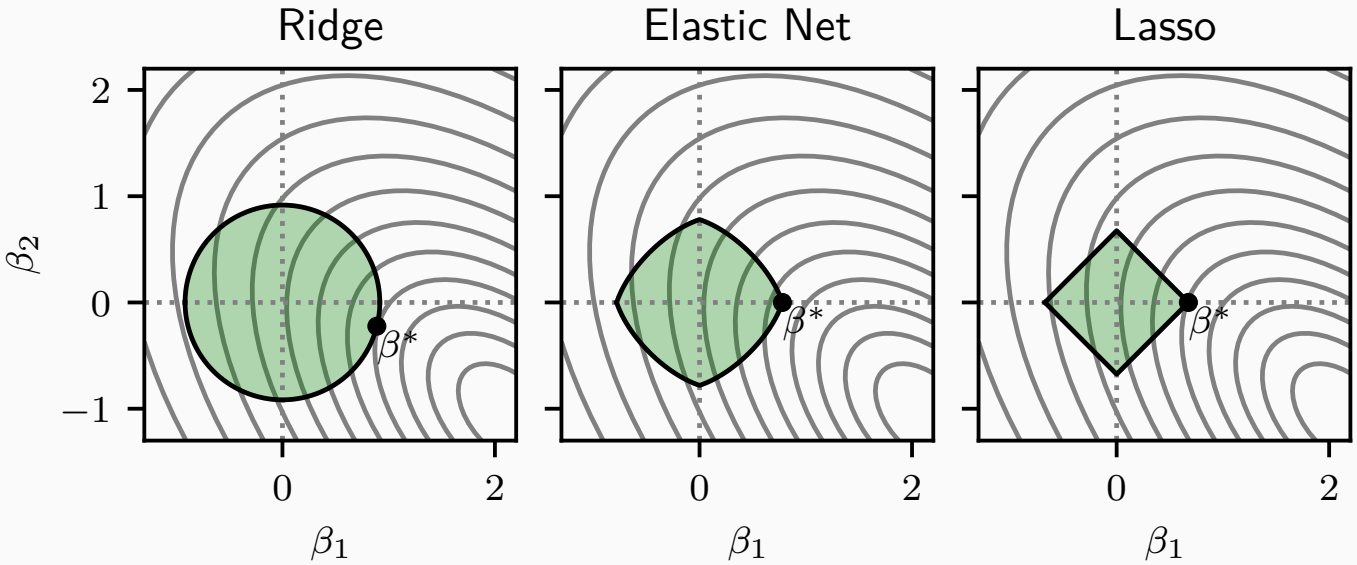


Figure 20: The elastic net penalty is a combination of the lasso and ridge penalties. Here shown as a constrained problem.

Sensitivity to Scale

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

Sensitivity to Scale

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

Example

Assume

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

Sensitivity to Scale

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

Example

Assume

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

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** of the coefficients—feature scales matter!

Example

Assume

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

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

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

where x_{ij} is an element of the (unnormalized) feature matrix and c_j and s_j are the *centering* and *scaling* factors respectively.

Normalization

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

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

  where x_{ij} is an element of the (unnormalized) feature matrix and c_j and s_j are the *centering* and *scaling* factors respectively.
- Usage of key terms ambiguous in literature.

Normalization

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

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

  where x_{ij} is an element of the (unnormalized) feature matrix and c_j and s_j are the *centering* and *scaling* factors respectively.
- Usage of key terms ambiguous in literature.
- 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 1: 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

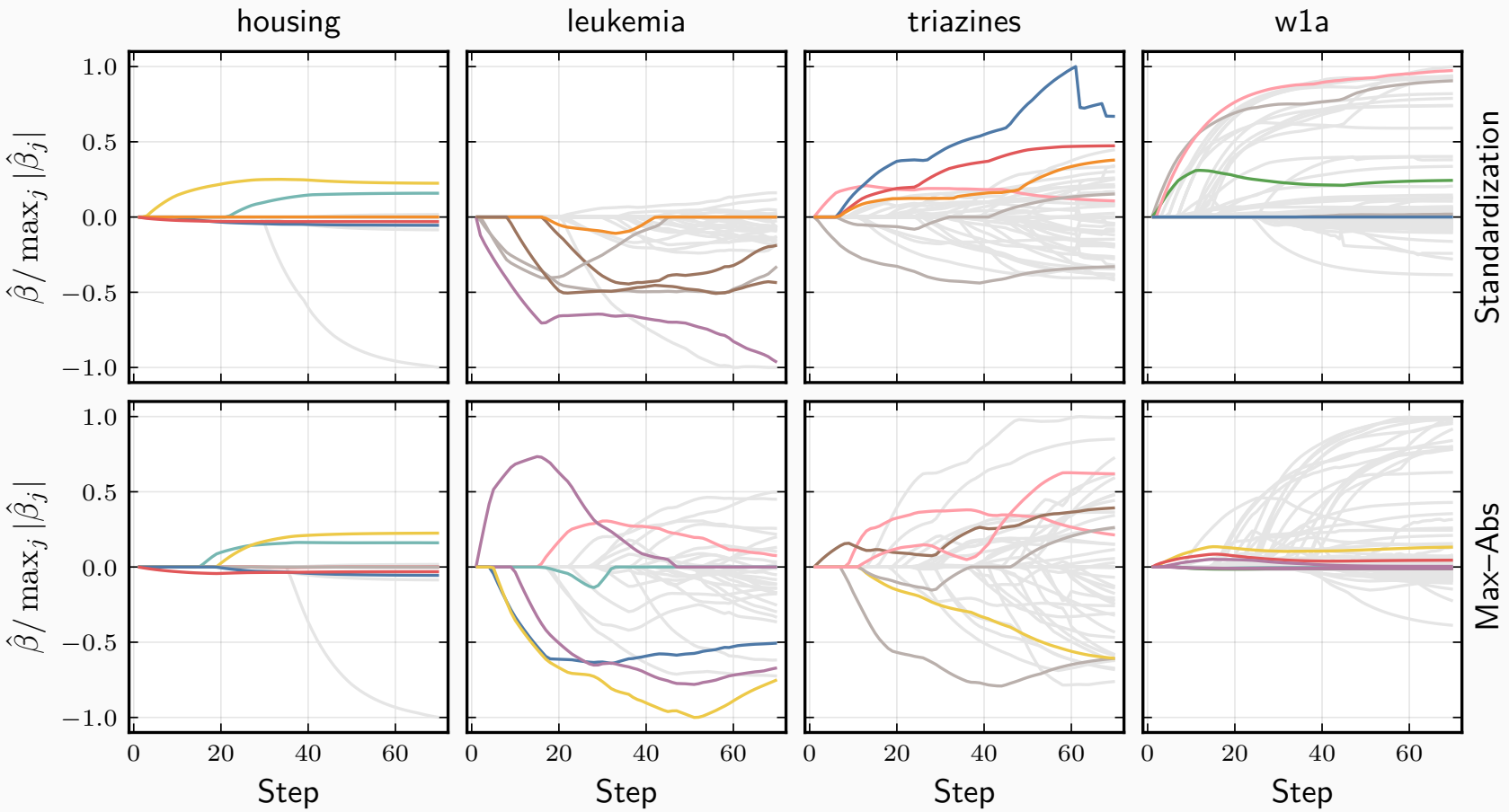


Figure 21: Lasso paths under two different types of normalization (standardization and max–abs normalization). The union of the first ten features selected in any of the schemes are colored.

Binary Features

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

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

Binary Features

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

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

- 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

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

- Means that the elastic net estimator depends on class balance (q).
- To remove the effect of q, an intuitive setting would be $s_j = q - q^2$ (the variance of $\boldsymbol{x}_j$) in the case of the lasso and $s_j = \sqrt{q - q^2}$ (the standard deviation of $\boldsymbol{x}_j$) in the case of the ridge.

Binary Features

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

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

- Means that the elastic net estimator depends on class balance (q).
- To remove the effect of q, an intuitive setting would be $s_j = q - q^2$ (the variance of $\boldsymbol{x}_j$) in the case of the lasso and $s_j = \sqrt{q - q^2}$ (the standard deviation of $\boldsymbol{x}_j$) in the case of the ridge.
- Suggests the parametrization

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

Binary Features

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

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

- Means that the elastic net estimator depends on class balance (q).
- To remove the effect of q, an intuitive setting would be $s_j = q - q^2$ (the variance of $\boldsymbol{x}_j$) in the case of the lasso and $s_j = \sqrt{q - q^2}$ (the standard deviation of $\boldsymbol{x}_j$) in the case of the ridge.
- Suggests the parametrization

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

- Indicates there might be no (simple) s_j that works for the elastic net. (Need to use weighted elastic net.)

Mixed Data

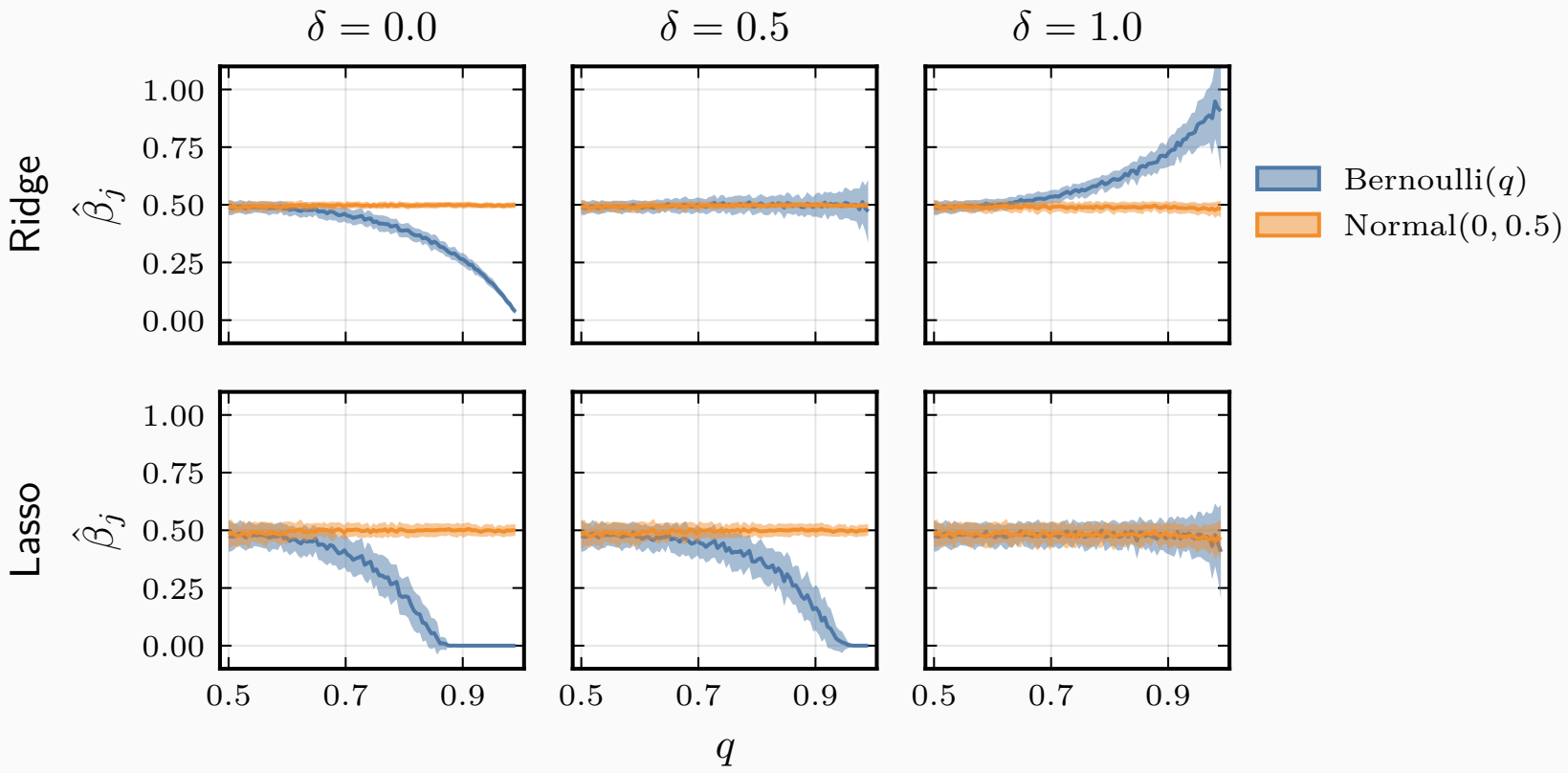


Figure 22: Comparison between lasso and ridge estimators for a data set with one binary and one quasi-normal feature.

Hyperparameter Optimization

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

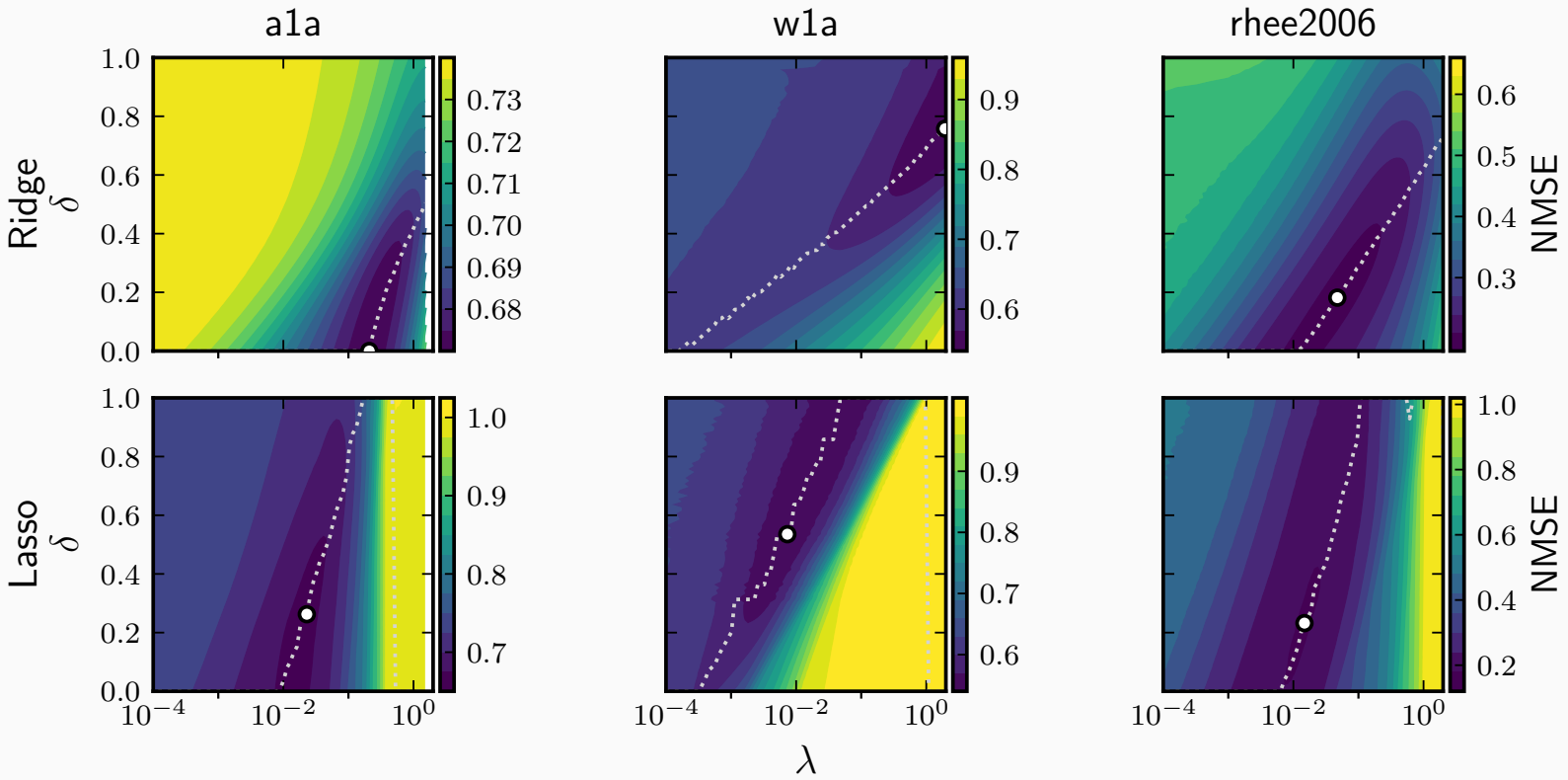


Figure 23: Contour plots of hold-out (validation set) error across a grid of δ and λ values for the lasso and ridge.

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
- Only experimental results for interactions

Recap

Thank you!