Optimization and Algorithms in Sparse Regression

Optimization and Algorithms in Sparse Regression

Screening Rules, Coordinate Descent, and Normalization

Johan Larsson

LUND
UNIVERSITY

Thesis for the degree of Doctor of Philosophy

Thesis Advisors
Jonas Wallin and Małgorzata Bogdan

Faculty Opponent
Professor Mário A. T. Figueiredo (Instituto Superior Técnico, Lisbon, Portugal)

To be presented, with the permission of the Lund University School of Economics and Business Administration of Lund University, for public criticism in the Clark Kent lecture hall (Kentsalen) at the Department of Statistics on Sunday, the 34th of December 2024 at 24:00.

Organization
LUND UNIVERSITY
Department of Statistics
Box 7080
SE–220 07 Lund
Sweden

Document name
DOCTORAL DISSERTATION

Date of disputation
2024-06-28

Sponsoring organization

Author(s)
Johan Larsson

Title and subtitle
Optimization and Algorithms in Sparse Regression: Screening Rules, Coordinate Descent, and Normalization

Abstract
Datasets are growing in size and complexity, especially with respect to the number of features of the problems that we study, which now often number in the millions. This has lead to a surge in interest for sparse regression models, which help make sense of these datasets by modeling them efficiently whilst still retaining a notion of explainability. Because these datasets are so large, however, they have prompted a need for effective optimization methods with which to apply them—in this thesis, we present several contributions to this area of research.

In papers I–III, we focus on screening rules for the lasso and sorted ℓ_1 penalized regression (SLOPE)—two sparse regression methods. Screening rules are algorithms that discard a portion of the features in the model before optimization takes place, which means that we effectively get to solve smaller problems than the original ones, yet still recover the same solutions. For the lasso, there has been a large body of work on screening rules since they were first introduced in 2010. In the case of SLOPE, however, there did not exist any screening rule until our work in paper I, in which we introduce the first such rule: the strong screening rule for SLOPE.

In paper II, we continue our work on screening rules by introducing look-ahead screening rules for the lasso, which enable screening of features for a stretch of the lasso path, rather than just for the following step. In essence, this allows us save computation time by screening features only when it is necessary. In paper III, we then tackle the case of using screening rules with highly correlated features, which is a setting in which previous screening rules have struggled. We propose the Hessian screening rule, which uses second-order information about the problem in order to provide less conservative screening along the lasso path. In empirical studies we show that our screening rule leads to large improvements in performance.

In paper IV, we introduce benchopt: a framework for benchmarking optimization methods in a transparent, reproducible, and collaborative manner. The current field of research in optimization is overflowing with new algorithms, each time proclaimed by its authors to improve upon its predecessors. It is easy to find benchmarks that directly contradict one another, which is foten the result of variations in the factors that impact the performance of the optimization methods studied. Benchopt makes it easy to construct benchmarks that transparently and objectively compare these methods to one another.

One particularly effective optimization method for the lasso is coordinate descent. Unfortunately, we cannot directly use coordinate descent for SLOPE since the problem is not separable. In paper V, however, we present a hybrid method which circumvents this issue by incorporating proximal gradient descent steps to tackle the separability issue, whilst still enjoying the effectiveness of coordinate descent.

In the final paper, paper VI, we study the use of normalization for the lasso when the data is made up of binary features. Normalization is necessary in regularized regression to put features on the same scale, but its effects are generally not well-understood. In our paper we show that the solutions in the lasso and ridge regression depend strongly on the class balance of the binary features and that this effect depends on the type of normalization used.

Key words
high-dimensional statistics, sparse regression, lasso, SLOPE, optimization, screening rules, normalization

Classification system and/or index terms (if any)

Supplementary bibliographical information

Language
English

ISSN and key title

ISBN
978-91-8104-076-0 (print)
978-91-8104-077-7 (pdf)

Recipient's notes

Number of pages
224

Price

Security classification

Optimization and Algorithms in Sparse Regression

Screening Rules, Coordinate Descent, and Normalization

Johan Larsson

LUND
UNIVERSITY

Cover illustration front: The elastic net path for a dataset of diabetes patients

Lund University School of Economics and Management
The Department of Statistics
Box 743, SE-220 07
Lund, Sweden

ISBN: 978-91-8104-076-0 (print)
ISBN: 978-91-8104-077-7 (electronic)

Printed in Sweden by Media-Tryck, Lund University, Lund 2024

There's a point when you go with what you've got.
Or you don't go.

—Joan Didion

Contents

Acknowledgements

Abstract

Datasets are growing in size and complexity, especially with respect to the number of features of the problems that we study, which now often number in the millions. This has lead to a surge in interest for sparse regression models, which help make sense of these datasets by modeling them efficiently whilst still retaining a notion of explainability. Because these datasets are so large, however, they have prompted a need for effective optimization methods with which to apply them—in this thesis, we present several contributions to this area of research.

In papers I–III, we focus on screening rules for the lasso and sorted ℓ_1 penalized regression (SLOPE)—two sparse regression methods. Screening rules are algorithms that discard a portion of the features in the model before optimization takes place, which means that we effectively get to solve smaller problems than the original ones, yet still recover the same solutions. For the lasso, there has been a large body of work on screening rules since they were first introduced in 2010. In the case of SLOPE, however, there did not exist any screening rule until our work in paper I, in which we introduce the first such rule: the strong screening rule for SLOPE.

In paper II, we continue our work on screening rules by introducing look-ahead screening rules for the lasso, which enable screening of features for a stretch of the lasso path, rather than just for the following step. In essence, this allows us save computation time by screening features only when it is necessary. In paper III, we then tackle the case of using screening rules with highly correlated features, which is a setting in which previous screening rules have struggled. We propose the Hessian screening rule, which uses second-order information about the problem in order to provide less conservative screening along the lasso path. In empirical studies we show that our screening rule leads to large improvements in performance.

In paper IV, we introduce benchopt: a framework for benchmarking optimization methods in a transparent, reproducible, and collaborative manner. The current field of research in optimization is overflowing with new algorithms, each time proclaimed by its authors to improve upon its predecessors. It is easy to find benchmarks that

directly contradict one another, which is foten the result of variations in the factors that impact the performance of the optimization methods studied. Benchopt makes it easy to construct benchmarks that transparently and objectively compare these methods to one another.

One particularly effective optimization method for the lasso is coordinate descent. Unfortunately, we cannot directly use coordinate descent for SLOPE since the problem is not separable. In paper v, however, we present a hybrid method which circumvents this issue by incorporating proximal gradient descent steps to tackle the separability issue, whilst still enjoying the effectiveness of coordinate descent.

In the final paper, paper vi, we study the use of normalization for the lasso when the data is made up of binary features. Normalization is necessary in regularized regression to put features on the same scale, but its effects are generally not well-understood. In our paper we show that the solutions in the lasso and ridge regression depend strongly on the class balance of the binary features and that this effect depends on the type of normalization used.

Popular Science Summary

In this thesis we study the field of big data, where the sheer volume and complexity of information can be overwhelming. The focus is on sparse regression models, a type of statistical model that helps make sense of large datasets by simplifying them in the form of a sparse representation.

The first three papers of the thesis concentrate on screening rules for two types of sparse regression methods: the lasso and sorted ℓ_1 penalized regression (SLOPE). Screening rules are like filters that remove some of the less important features of the data before your computer is tasked with processing it. This makes the problem smaller and easier to solve, yet still gives the same results. We introduce the first-ever screening rule for SLOPE and develop look-ahead screening rules for the lasso, which save additional computation time when you do not know how simple you want your model to be. Finally, we also tackle the challenge of using screening rules when data features are highly correlated, proposing a new rule that improves performance in this situation.

The fourth paper introduces benchopt, a tool for comparing different optimization methods. With so many new algorithms being developed, it is hard to know which one is best. Benchopt provides a transparent and objective way to compare these methods.

The fifth paper presents a new optimization method for the lasso, a popular sparse regression model. In the method, we combine two existing methods to overcome a limitation of the lasso, making it more effective.

The final paper explores the impact of normalization—a process that puts all features on the same scale—on the lasso when dealing with binary features. We find that the balance of the binary features (the balance between ones and zeros) and the type of normalization used can significantly affect the results.

List of Publications

This thesis is based on the following publications.

I **The Strong Screening Rule for SLOPE**
Johan Larsson, Małgorzata Bogdan, and Jonas Wallin (Dec. 6–12, 2020). 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

II **Look-Ahead Screening Rules for the Lasso**
Johan Larsson (Sept. 6, 2021). In: *22nd European Young Statisticians Meeting – Proceedings*. 22nd European Young Statisticians Meeting. Ed. by Andreas Makridis, Fotios S. Milienos, Panagiotis Papastamoulis, Christina Parpoula, and Athanasios Rakitzis. Athens, Greece: Panteion university of social and political sciences, pp. 61–65. ISBN: 978-960- 7943-23-1

III **The Hessian Screening Rule**
Johan Larsson and Jonas Wallin (Nov. 28–Dec. 9, 2022). In: *Advances in Neural Information Processing Systems 35*. 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Ed. by Sanmi Koyejo, Sidahmed Mohamed, Alekh Agarwal, Danielle Belgrave, Kyunghyun Cho, and Alice Oh. Vol. 35. New Orleans, USA: Curran Associates, Inc., pp. 15823–15835. ISBN: 978-1-71387-108-8

IV **Benchopt: Reproducible, Efficient and Collaborative Optimization Benchmarks**
Thomas Moreau, Mathurin Massias, Alexandre Gramfort, Pierre Ablin, Pierre-Antoine Bannier, Benjamin Charlier, Mathieu Dagréou, Tom Dupré la Tour, Ghislain Durif, Cassio F. Dantas, Quentin Klopfenstein, Johan Larsson, En Lai, Tanguy Lefort, Benoit Malézieux, Badr Moufad, Binh T. Nguyen, Alain

Rakotomamonjy, Zaccharie Ramzi, Joseph Salmon, and Samuel Vaiter (Nov. 28–Dec. 9, 2022). In: *Advances in Neural Information Processing Systems 35*. 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Ed. by Sanmi Koyejo, Sidahmed Mohamed, Alekh Agarwal, Danielle Belgrave, Kyunghyun Cho, and Alice Oh. Vol. 35. New Orleans, USA: Curran Associates, Inc., pp. 25404–25421. ISBN: 978-1-71387-108-8

v **Coordinate Descent for SLOPE**

Johan Larsson, Quentin Klopfenstein, Mathurin Massias, and Jonas Wallin (Apr. 25–27, 2023). 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

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

Introduction

If we have data, let's look at data. If all we have are opinions, let's go with mine.

—Jim Barksdale

1 Background

With modern advances in science and technology, statistical models and the data on which they are fit are becoming increasingly complex. Datasets are expanding in size, often both in terms of their numbers of variables (features) as well as observations. In some fields, this growth in complexity has been paralleled with more effective methods with which to collect observations, as in crowd science and recommender systems. But in other areas, collecting data still amounts to a costly endeavor. In bioinformatics, for example, ethical concerns and rising requirements on the quality of data have only served to *raise* the costs of data collection. As a result, data collected in these fields is becoming *wider*: the ratio between the number of variables (features) and the number of observations is increasing (Table 1).

The growth in the number of observations (taller data) is mostly a luxury problem since more observations typically mean more accurate models. But an expansion in the number of features (wider data) is a more delicate issue. The problem is that if all the features that we have collected are important then we are out of luck as far as understanding our data goes. Instead, we have to more or less hope that there is a *sparse* representation of our data that, with some acceptable loss of information, allows us to make sense of the problem we are studying.

We can call this hope the *sparsity assumption*, which can be motivated through the *bet-on-sparsity principle*: assume that the underlying signal is sparse (Figure 1) and use a sparse method to model it. If the assumption is correct, then our method has a chance of doing well. But if the assumption is incorrect, then our method will not work—but

Table 1: Tall and wide data. Each row is an observation, for instance the measurement on a person in a study, and each column a feature, which represents all the measurements on a variable for all the observations.

(a) Tall data

Feature 1	Feature 2	Feature 3
0	0.32	1
1	−1.2	−1
⋮	⋮	⋮

(b) Wide data

Feature 1	Feature 2	Feature 3	⋯
0	0.32	1	⋯
1	−1.2	−1	⋯

no other method would (Hastie, Robert Tibshirani, and Friedman 2009). Thankfully, many problems in the real world exhibit this type of sparsity.

The success of neural networks and other complex models that model high-dimensional data well yet do not enforce sparsity does raise questions as to the validity of this principle. But in our setting, which, loosely speaking, is *explainable* methods for regression, it still bears relevance.

Technically speaking, we are interested in datasets that are made up of a $n \times p$ matrix of features $\boldsymbol{X}$ and a response vector of length n, $\boldsymbol{y}$:

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0.32 & \cdots & x_{1,p} \\ 1 & -1.2 & \cdots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} 0.2 \\ -0.9 \\ \vdots \\ y_n \end{bmatrix},$$

where we have inserted some arbitrary values for the sake of illustration. The data

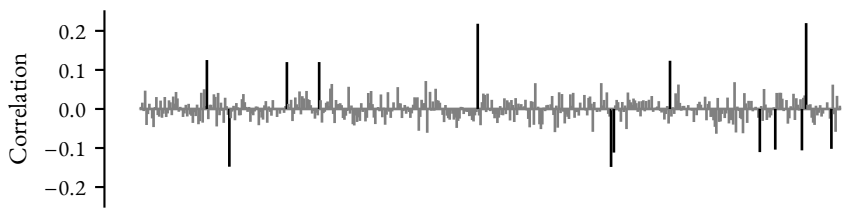


Figure 1: A relatively sparse signal. The plot shows correlations between the response vector $\boldsymbol{y}$ and each feature in the `madelon` dataset (Guyon et al. 2004). Correlations above 0.1 have been colored in black, the rest in gray.

presented in Table 1 corresponds to $\boldsymbol{X}$ here (each row is an observation, each column a feature).

In the simplest case, we assume that $\boldsymbol{y}$ is a linear combination of the features in $\boldsymbol{X}$ plus some noise ($\boldsymbol{\epsilon}$), for instance measurement noise, which we write mathematically as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \beta_0 + \boldsymbol{\epsilon},$$

where $\boldsymbol{\beta}$ is a vector of coefficients and β_0 the *intercept*. In this representation of the data, the coefficients $\boldsymbol{\beta}$ are the parameters that we are interested in estimating and represent the effect each feature has on the response vector $\boldsymbol{y}$. Assuming that this is in fact the true relationship between $\boldsymbol{\beta}$, $\boldsymbol{X}$, and $\boldsymbol{y}$, a natural choice of model to fit this data with is *linear regression*.[1]

In the presence of noise, however, there generally exists no β_0 and $\boldsymbol{\beta}$ that will fit the data perfectly, and we must therefore accept that the linear regression model is only an approximation. The natural follow-up question is then: what *is* a good approximation? To answer this question, we need to define some measure of error. The most common measure, at least as far as linear regression models go, is the sum of squared errors between the predicted response vector

$$\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}$$

and the true response vector $\boldsymbol{y}$, that is,

$$\|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2.$$

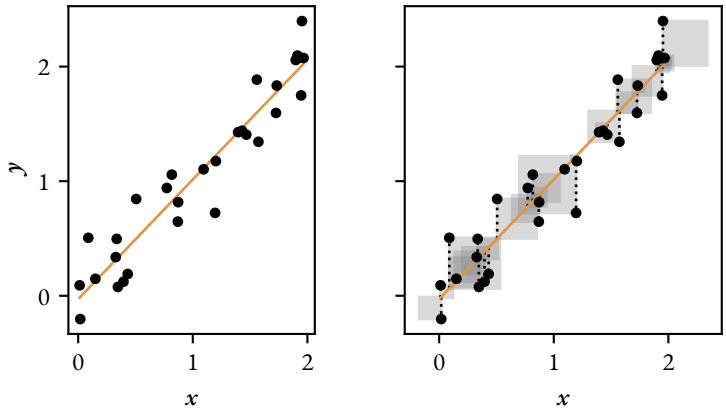
The smaller this measure—the better the fit, which means that finding a vector $\boldsymbol{\beta}$ that minimizes this error can be posed as the following optimization problem:

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2. \tag{1}$$

We let $\boldsymbol{\beta}^*$ to be the solution to this problem—the optimum, and will use $\hat{\boldsymbol{\beta}}$ to refer to the estimate that we obtain from some algorithm. Generally, we will assume that the algorithm has converged to the optimum so that $\boldsymbol{\beta}^* = \hat{\boldsymbol{\beta}}$, but in actuality we almost always have some amount of *suboptimality* so that $|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}| > 0$. The factor $1/2$ in Problem (1) is included for convenience, for reasons that will become clear later on.

Solving Problem (1) is equivalent to fitting the ordinary least-squares (OLS) regression model, which, for a simple case of a single feature, we have illustrated in Figure 2. Picking a different measure of error would lead to a different method (and often a different linear regression line), but in this thesis we will focus on the method of least squares.

[1] In general, we also need additional assumptions on the noise term $\boldsymbol{\epsilon}$.



(a) The slope of the orange line is β. The point where the line intersects the y-axis is the intercept β_0.

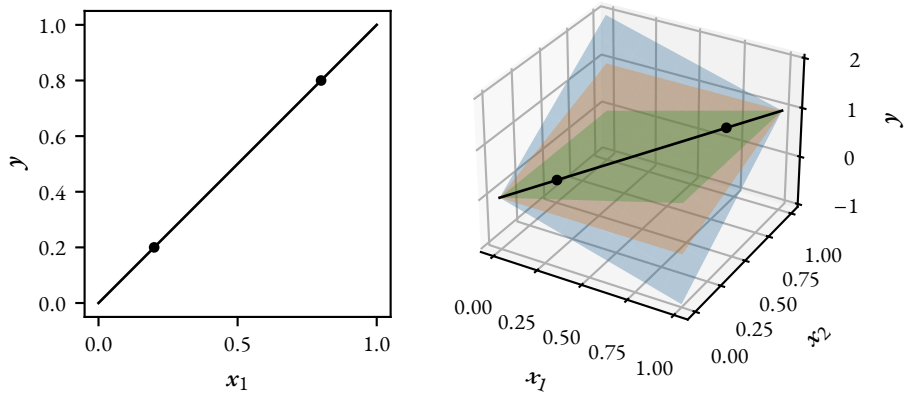
(b) The dotted lines are the residuals and the grey squares are the squared errors. The orange line is that which minimizes the total surface of all of these squares summed together.

Figure 2: Simple ordinary least-squares linear regression for a one-feature problem

If we have many more observations than features ($n \gg p$), then our linear regression model might stand a decent chance of recovering the true model (coefficients). But if the tables were turned and the features instead outnumbered the observations ($p \gg n$), the model would in fact break down.

The problem is that our linear regression model will be able to fit our particular dataset perfectly but typically not generalize well to new data, even if it comes from the same underlying data-generating mechanism. This is called *overfitting* and it happens because we have more parameters (regression coefficients) than unknowns (observations). This means that there are now many different regression models that will fit the data equally well and that the solution, therefore, is not unique. In Figure 3, we illustrate what overfitting looks like for the linear regression model in one and two parameters.

In principle, the problem of overfitting in linear regression is the same as that in polynomial regression: as we increase the number of parameters (degree of the polynomial in this case), the model eventually becomes too flexible and overfits (Figure 4). Note that in this example overfitting occurs even when the number of parameters is smaller than the number of observations. This is the case with linear regression too, for which the problem may occur in cases when n is larger than p.



(a) With one feature, the simple ordinary least-squares regression line fits the data perfectly.

(b) With two features, an infinite number of regression planes fits the data perfectly.

Figure 3: A linear regression problem with two observations. The example is of course artifical, but demonstrating overfitting in more dimensions than this would require more than the faculties of our limited human minds are capable of.

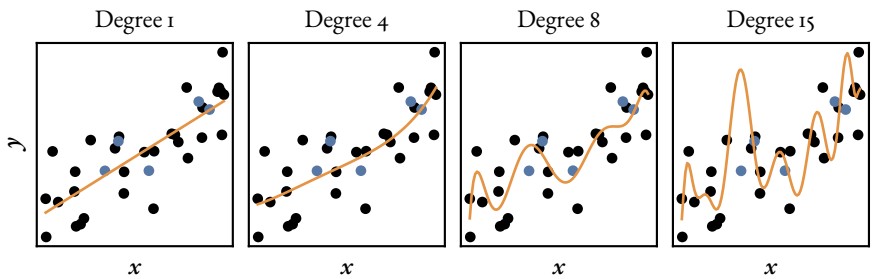


Figure 4: Polynomial regression for a one-feature problem. The data is generated from the simple linear model $= 2x_i + \varepsilon_i$, where $\varepsilon \sim \text{Normal}(0, 0.5^2)$ identically and independently. The line fits the data increasingly well as the degree of the polynomial increases, but when new data arrives (the blue points), we see that the model generalizes poorly.

One way to deal with overfitting is to decrease the complexity of the model by restricting the parameters in some way, for instance by allowing only a subset of the regression coefficients to be non-zero—in other words, make the model sparse. This procedure is called *regularization*.

2 Regularization

When we regularize a model, we introduce a budget on $\boldsymbol{\beta}$, allowing only some of its elements to be non-zero or restricting the values they can take. The simplest type of regularization is called *best-subset selection*, in which we simply limit the number of coefficients we allow to be non-zero to a constant k, which is equivalent to selecting k of the features in $\boldsymbol{X}$. We call the indices of the selected features the model's *support*. If features one and three are selected, for instance, then the support is $\{1, 3\}$. If all of the features are in the model, then the support is $\{1, 2, \ldots, p\}$ (as in the case of OLS regression). Finally, if no features are selected, then the support is the empty set ($\emptyset$).

In best-subset selection, we examine all $\binom{p}{k}$ possible combinations of features and pick the combination that fits our data best. Formally, the method can be posed as the following optimization problem:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2, \\ &\text{subject to} && \|\boldsymbol{\beta}\|_0 \leq k, \end{aligned}$$

where $\|\cdot\|_0$ is the ℓ_0 norm:[2] the number of non-zero elements in vector.

In other words, if $p = 3$ and $k = 2$, for instance, the following models would satisfy our constraints:

$$\boldsymbol{\beta} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \qquad \text{and} \qquad \boldsymbol{\beta} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

But the following model would not:

$$\boldsymbol{\beta} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

The primary problem with this method is that it is computationally infeasible when p is large since the number of possible models grows exponentially with p. With

[2] Technically speaking, the ℓ_0 norm is not actually a true norm.

$p = 100$ and $k = k$, for instance, there are

$$\binom{100}{5} = 75\,287\,520$$

possible models to consider. And this problem is further exacerbated by the fact that we typically has to consider a variety of values for k.

In an interesting turn of events, however, Bertsimas, A. King, and Mazumder (2016) has shown that the best subset selection problem can actually be written as a mixed-integer optimization problem. This enables the use of modern optimization software, which can then handle best subset problems of dimensions that previously were thought unattainable. Yet, although this result has offered a considerable improvement in run-time performance for the algorithm, it is still the case that it struggles in high dimensions. Solving a problem with $n = 500$ and $p = 100$ for $k \in \{1, 2, \ldots, 50\}$, for instance, still comes down to a hefty 76.8 hours of computation (Hastie, Robert Tibshirani, and Ryan Tibshirani 2020). In contrast, the time taken to fit the model that we will consider next, the lasso, amounts to 0.014 seconds for the equivalent problem.

2.1 The Lasso

One solution to the complexity problem of best-subset selection is to relax the constraint to one that makes the problem easier to solve, yet still retains the sparsity-enforcing property of the ℓ_0 norm. A natural candidate for this is the ℓ_1 norm, which leads to the following problem,

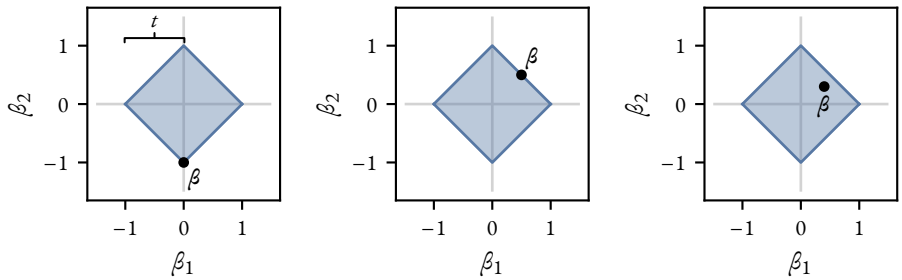
$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2, \\ &\text{subject to} && \|\boldsymbol{\beta}\|_1 \leq t, \end{aligned} \tag{2}$$

where all we did was replace the ℓ_0 norm with the ℓ_1 norm,

$$\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{p} |\beta_j|$$

As a consequence, we have replaced the integer-valued k with a real-valued (but positive) t. This problem is known as ℓ_1-regularized regression or, more commonly, as the *lasso*[3]. The lasso was introduced to the statistics community by Robert Tibshirani (1996) but actually stems from much earlier research done in the field of signal processing by

[3]The lasso is sometimes written as the acronym LASSO for *least absolute shrinkage and selection operator*, but we will stick with the lower-case version here, which the authors themselves use in their recent work.



(a) A sparse solution, $\beta_1 = 0$ and $\beta_2 = -1$.

(b) A dense solution, $\beta_1 = 0.5$ and $\beta_2 = 0.5$.

(c) A solution in which the constraint is inactive (t is too large to affect the solution), $\beta_1 = 0.4$ and $\beta_2 = 0.3$.

Figure 5: The ℓ_1 norm ball in $\mathbb{R}^2$ with some possible solutions indicated by $\boldsymbol{\beta}$. The ℓ_0 ball (for best-subset selection) and $k = 1$ would be lines of infinite length along both of the axes.

Santosa and Symes (1986). Donoho and Johnstone (1994, 1995) subsequently introduced the concept of the *basis pursuit* problem, which is closely related to the lasso, and developed much of the theoretical framework for the lasso.

If you have encountered the lasso previously, it is likely that you have seen it formulated as the following unconstrained optimization problem:

$$\text{minimize} \quad \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1.$$

In this section, however, we will prefer the constrained formulation of the problem as it is given in Problem (2), which we think is intuitive and easier to understand. The two formulations are equivalent, however, and will lead to exactly the same solution for a suitable choice of t and λ. Later, in Section 3.2, we will make this connection clear.

We saw previously that the ℓ_0 constraint in best-subset selection puts a budget on the number of features allowed in the model. The ℓ_1 norm, in contrast, instead puts a budget on the *size* of the coefficients. This enforces not only sparsity but also shrinkage in the solution. In Figure 5, we have visualized how this constraint affects the solution of the least-squares objective.

There is an extensive body of work on the lasso and it has spawned a number of offshoots, such as the fused lasso (Robert Tibshirani, M. Saunders, et al. 2005), group lasso (Yuan and Lin 2005), adaptive lasso (Zou 2006), graphical lasso (Friedman, Hastie, and Robert Tibshirani 2008), and square-root lasso (Belloni, Chernozhukov, and Wang

2011). In this thesis, however, we focus on the standard lasso.

Note, also, that the lasso is not limited to just regularized *linear* regression but can in fact be used for the entire family of generalized linear models, such as logistic, Poisson, multinomial, and multivariate regression, as well as survival models such as Cox regression. The use of the ℓ_1-norm penalty has also found its way into many other areas of statistics as well as the fields of signal processing and machine learning, including matrix factorization, clustering, and deep learning.

An interesting property of the lasso is that it is possible (and computationally feasible) to exactly solve the lasso problem for all possible values of $t \in [0, \infty)$. This is called the *lasso path* (Figure 6). It begins at $t = 0$, for which the constraint region is a point, forcing all of the coefficients to be exactly zero. Then as t increases, the constraint region grows, allowing the coefficients to enter the model and grow. The reason for why it is possible to solve for the full path is that the solution vector β, as a function of t, is linear and continuous between the values of t for which features enter or leave the model.

One problem with the lasso, however, is that it does not deal with correlated features as intuition (at least that of the author) would suggest. If two features are correlated highly enough, for instance, the lasso will select one of them and drop the other (set its coefficient to zero).[4] This is not necessarily a problem for the prediction $\hat{\boldsymbol{y}}$, but it means that the estimated coefficients no longer provide trustworthy estimates of variable (feature) importance. This effect is the result of the behavior of the ℓ_1 norm, which penalizes the *size* of the coefficients. If two features provide the same, or nearly the same, information about the response, then the optimization problem can attain a lower value by setting one of the coefficients to zero.

This is a problem that the *elastic net*—the topic of the next section—is designed to overcome.

2.2 The Elastic Net

The elastic net is a combination of the lasso and ridge regression,[5] which can be written as the following optimization problem:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2, \\ &\text{subject to} && \alpha\|\boldsymbol{\beta}\|_1 + (1-\alpha)\|\boldsymbol{\beta}\|_2^2 \leq t. \end{aligned} \tag{3}$$

[4] In the case where the features are perfectly correlated, it may in fact be the optimization algorithm that decides which of them is picked.

[5] Ridge regression is also known as Tikhonov regression; and in deep learning, ℓ_1-regularization is typically called *weight decay*.

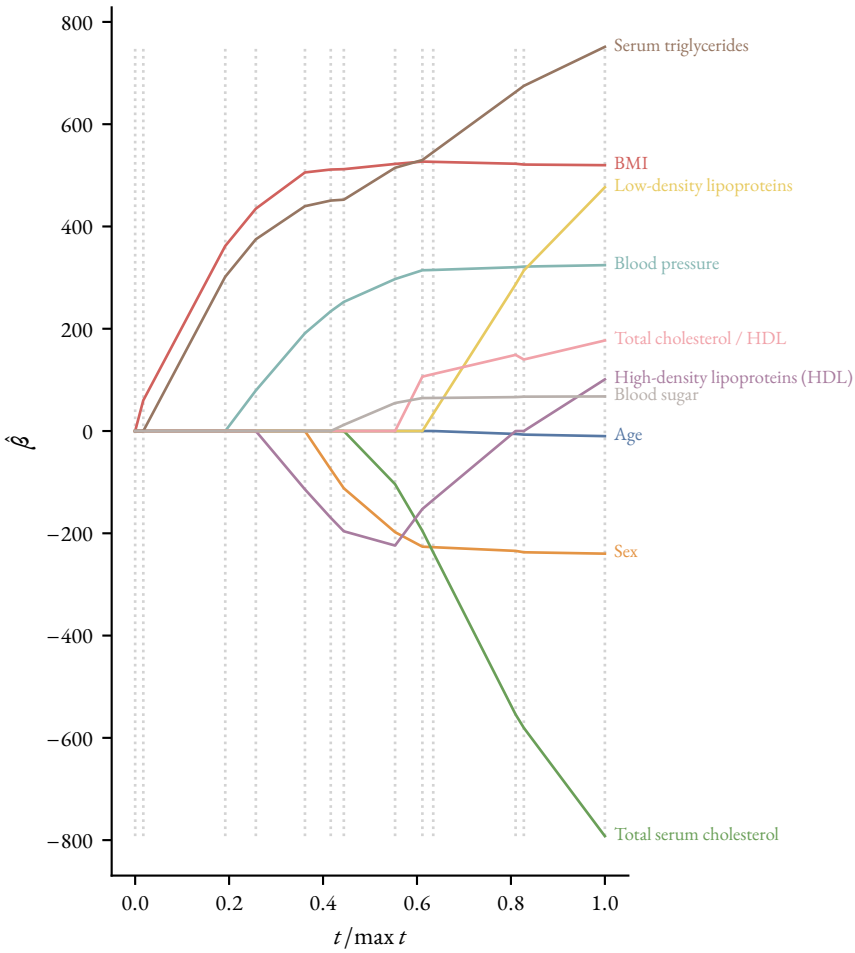
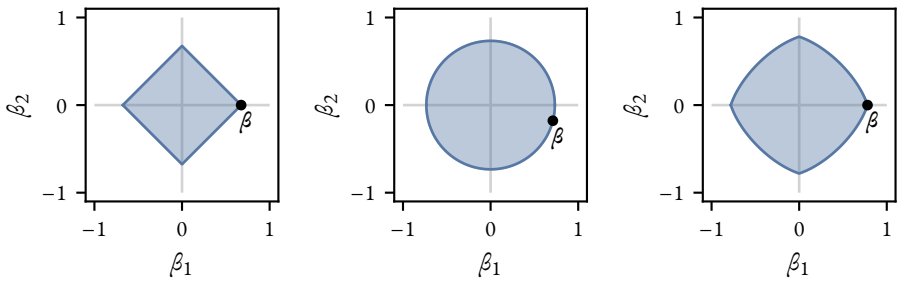


Figure 6: The lasso path for the `diabetes` dataset (Efron et al. 2004), which consists of $n = 442$ observations and $p = 10$ features. The path shows the coefficients as a function of the parameter t, which controls the size of the constraint region. The path is piecewise linear with kinks occurring only when features enter or exit the model. At $t = 0$, the model is completely sparse and the support is $\emptyset$—the empty set. At $t/\max t = 1$, the model is the ordinary least-squares model and the support is the full set of predictors, $\{1, 2, \ldots, p\}$.

The difference compared to the lasso is that we have transformed our constraint into a linear combination of the ℓ_1 and squared ℓ_2 norms. The parameter α controls the balance between these constraints. By setting $\alpha = 1$, we turn the problem into the lasso (Figure 7a). And at $\alpha = 0$, we instead have ridge regression (Figure 7b). Any value $\alpha \in (0, 1)$ yields a combination of the two (Figure 7c).



(a) When $\alpha = 1$, the constraint is the lasso (ℓ_1-norm) ball. And in this case the solution is sparse.

(b) When $\alpha = 0$, the constraint is the ridge (squared ℓ_2-norm) ball. Here, the solution is not sparse.

(c) When $\alpha \in (0, 1)$, the constraint region is a combination of the lasso and ridge balls. In this case we have $\alpha = 1/2$. Once again, the solution is sparse.

Figure 7: The constraint regions for the elastic net for different values of α

The elastic net was first proposed by Zou and Hastie (2005). In addition to dealing with the problems encountered in using the lasso for highly correlated features, the elastic net also yields improved predictive performance in many situations. The latter fact is perhaps not so surprising given that it is a combination of methods that essentially assume different structure in the data. The lasso works well when the true signal is sparse, while ridge regression handles the situation where there are weak signals better. And since the elastic net contains both these models as special cases, in addition to any combination thereof, it naturally extends to a wider range of problems.

2.3 SLOPE

Another way of dealing with the problem of correlated features is to use *Sorted L-One Penalized Estimation* (SLOPE) (Bogdan, Berg, Sabatti, et al. 2015; Bogdan, Berg, Su, et al. 2013; Zeng and Figueiredo 2014). SLOPE is a generalization of both the lasso and *the octagonal shrinkage and clustering algorithm for regression* (OSCAR) (Bondell and

Reich 2008). It is represented by the following optimization problem:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2, \\ &\text{subject to} && \sum_{j=1}^{p} \lambda_j |\beta|_{(j)} \leq t, \end{aligned} \tag{4}$$

where $\boldsymbol{\lambda}$ is a non-increasing and non-negative sequence of penalization weights and where we define the subscript operator (j) such that

$$|\beta|_{(1)} \geq |\beta|_{(2)} \geq \cdots \geq |\beta|_{(p)}.$$

The left-hand side of the constraint in SLOPE is, perhaps somewhat surprisingly, actually a norm: the *sorted* ℓ_1 *norm*.

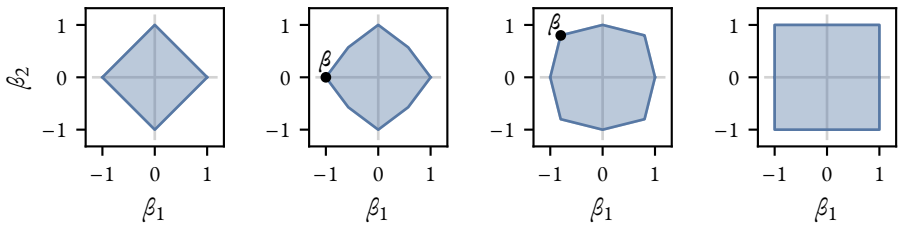
One of the most salient features of SLOPE is that it clusters coefficients (Figueiredo and Nowak 2014; Schneider and P. Tardivel 2022). This is a consequence of the sorted ℓ_1 norm and the choice of $\boldsymbol{\lambda}$—larger differences between adjacent elements increase the propensity for clustering. This property makes SLOPE well-adapted to the case when features are highly correlated, which is a situation that the lasso struggles with. In cases where the lasso might set one of the correlated features to zero, SLOPE will often instead set them to exactly the same value. This is a property that is not shared by the elastic net, which handles correlation (although not quite as delicately), but does not cluster coefficients.

But as we mentioned previously, SLOPE is actually a generalization of lasso and thus contains it as a special case (Figure 8a), which is attained by setting all of the elements of the penalization weight vector $\boldsymbol{\lambda}$ to the same value. On the opposite end, setting only the first element to a non-zero value and the remaining ones to zero yields the infinity norm (Figure 8d).

SLOPE also has other appealing properties, such as the ability to (under certain assumptions on the design) control the false discovery rate[6] (Bogdan, Berg, Sabatti, et al. 2015) and recover sparsity and ordering patterns in the solution (Bogdan, Dupuis, et al. 2022). Another key feature is that the problem is convex, which has implications that we will come to appreciate in Section 3. This latter fact also puts SLOPE apart from other competing penalization methods such as the minimax concave penalty (MCP) (Zhang 2010) and smoothly clipped absolute deviation (SCAD) (Fan and R. Li 2001).

The constraints in the lasso and elastic net are parameterized by one and two parameters, respectively: t in the case of the lasso, and (t, α) in the case of the elastic

[6] By false discovery rate, we mean the fraction of coefficients incorrectly identified as non-zero (false discoveries) as a proportion of the total number of non-zero coefficients (discoveries).



(a) When $\lambda_1 = \lambda_1 = 1$, the constraint region is the lasso ball, $\|\boldsymbol{\beta}\|_1$.

(b) Here we have $\lambda_1 = 1$ and $\lambda_2 = 3/4$, but no clustering.

(c) With $\lambda_1 = 1$ and $\lambda_2 = 1/4$, the solution is now clustered.

(d) When $\lambda_1 = 1$ and $\lambda_2 = 0$, the constraint is the infinity norm, $\|\boldsymbol{\beta}\|_\infty$.

Figure 8: SLOPE balls (the sorted ℓ_1 norm) for various choices of the penalization weight vector $\boldsymbol{\lambda}$. It is the kinks at the boundaries, occurring when $|\beta_1| = |\beta_2|$, that induce clustering. The larger the difference between adjacent values in $\boldsymbol{\lambda}$, the stronger the clustering effect becomes.

net. SLOPE, in contrast, is parameterized not only by t but also by the $\boldsymbol{\lambda}$ vector, which has p elements—one for each feature. Finding an optimal setting for all of those $p + 1$ parameters is a non-trivial task when p is large. As a result, we typically need to re-parameterize the problem. The most common form for this parameterization is the Benjamini–Hochberg sequence, which sets the penalization weights to

$$\lambda_i = \Phi^{-1}(1 - q_i), \qquad q_i = \frac{qi}{2p},$$

where Φ^{-1} is the quantile function of the standard normal distribution and $q \in [0, 1]$. It is this choice that gives SLOPE its false discovery rate control property (Bogdan, Berg, Sabatti, et al. 2015). Note that if we use a linearly decreasing sequence instead, then we would recover OSCAR. And if we use a constant sequence, we recover the lasso.

This reparameterization reduces the number of parameters to just two: (t, q)—the same number as the elastic net. And it means that it is tractable to find optimal settings for the parameters using the methods that we will introduce in the next section.

2.4 Hyperparameter Optimization

An issue that we have so far largely ignored is how to pick good values for t in the case of the lasso, (t, α) in the case of the elastic net, and (t, q) in the case of SLOPE. We call these *hyperparameters* of the problems. Under strong assumptions on our data, in particular the error term $\boldsymbol{\epsilon}$, it is possible to derive optimal settings of some of

these hyperparameters. The problem, however, is that we typically do not know the distribution of the error term. And in high dimensions, estimating it is not easy either.

The alternative, which is the dominating procedure in practice, is to resort to hyperparameter optimization, in which we treat the problem of finding good hyperparameters as an upper-level optimization problem (on top of the optimization problem of finding β). This procedure is often also called *model validation*. This is most commonly done via a grid search in which we construct a grid across the hyperparameter space. In the lasso case, for instance, it is common to construct a linearly spaced sequence of t values.[7]

In the simplest case, typically called *hold-out validation*, we split the dataset into a training set and a validation set. The lasso is fit for the full t sequence on the training data; in other words, we fit a full lasso path. Afterwards, we measure the model's error on the validation data and pick the t value with the best score. Often, there is also a separate test dataset that is held-out before hyperparameter optimization; at the end, this test set is used to obtain a final unbiased goodness-of-fit measure for the model selected during hyper-optimization.

A more common, although slightly more involved, method is to use k-fold cross-validation (Figure 9), which is similar to hold-out validation, except the data is iteratively split into k folds; the method is run for as many iterations. In the kth iteration, the kth fold is held out as a validation set on which an error is computed after the model has been fit on the remaining $k - 1$ folds. After the last iteration, a cross-validation error is computed by averaging the validation error over all of the validation folds. Typical choices of k are 5 and 10. If $k = n$, then the method is called *leave-one-out cross-validation*. In a sense, cross-validation can be seen as a variance-reduction technique for hold-out validation. The downside, however, is that this comes at a price of increased bias since the validation sets are also used during training. There is also *repeated* k-fold cross validation, which simply repeats the cross-validation procedure for some number of times, each time with different splits.

For the lasso, it is typical to construct a grid of 100 t values, which means that we have to, for instance, fit $100k$ lasso models if we use cross-validation. This can easily become computationally expensive, especially if either or both of n and p are large. In the case of the elastic net and SLOPE, the problem is complicated further since we then have α or q to optimize over as well. As a result, it is vital that there are efficient methods for fitting these models, which is the topic of the following two sections. In Section 3, we discuss the various optimization methods that we can use to solve the lasso, the elastic net, and SLOPE. And in Section 4, we introduce the concept of *screening rules*, which have been a game changer in the high-dimensional context.

[7] If we were to use the unconstrained version of the lasso in which the model is parameterized by λ, the sequence is typically *geometrically* spaced instead.

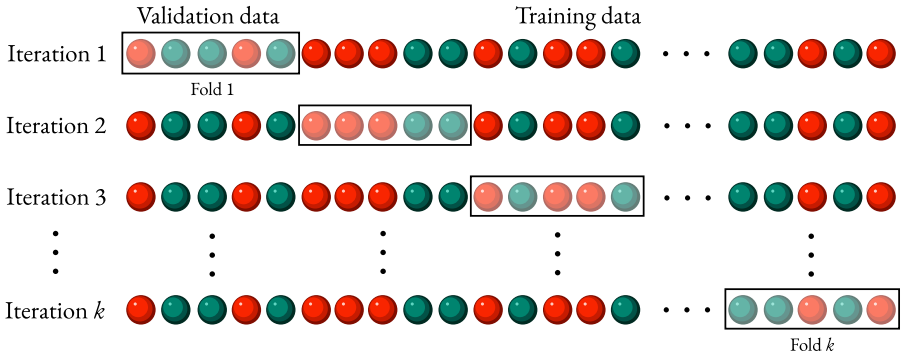


Figure 9: An illustration of k-fold cross-validation. The data is split into k folds (subsets). In the kth iteration, the model is fit on training data consisting of $k - 1$ folds and then applied to the held-out validation data in fold k, on which an error is computed. A final cross-validation error is then computed by averaging the error across all of the iterations. This figure is an edited version of an illustration by Gufosowa (via Wikimedia Commons, licensed under CC BY-SA 4.0).

3 Optimization

In the previous section we introduced the statistical models that this thesis will revolve around: the lasso, the elastic net, and SLOPE. They feature many interesting theoretical properties, which we have touched upon briefly, but it is actually not these properties that we will concern ourselves with in this thesis. Instead, we will be interested in the *numerical* aspects of these problems. That is: how do we actually solve them? And, moreover, how do we do this as efficiently as possible?

As we have seen, fitting these models to data is equivalent to solving optimization problems and we have so far assumed that this is possible and that we, as a result, can recover their optima. This assumption is by no means wrong: methods for fitting the lasso, elastic net, and SLOPE are readily and freely available in many programming languages and across all major operating systems. Installing and running them amounts to only a few lines of code. To fit the full lasso path to the diabetes data that we encountered previously (Figure 6), for instance, we need only to call the following `R` (R Core Team 2024) code, which loads the necessary `R` package `lars`, imports the dataset `diabetes`, and fits the lasso path to the data:

```
library(lars)
data(diabetes)
fit <- lars(diabetes$x, diabetes$y, type = "lasso")
```

Behind the scenes, however, the method invoked through these commands actually involve a complicated optimization algorithm into which considerable effort has been put in order to ensure that what you get in `fit` is reliable—and that you get it *fast*.

Throughout the following sections, we will discuss various optimization methods that can be used to solve the lasso, the elastic net, and SLOPE. For simplicity, we will generally focus on the lasso and, in particular, the ordinary lasso (ℓ_1-regularized least-squares regression). In general, however, the methods that we introduce here can be used to solve all of these problems, including the case when the objective that is regularized is part of the family of generalized linear models, which include, for instance, logistic , multinomial, and Poisson regression.

3.1 Direct Methods

The first optimization problem that we encountered in this text was ordinary least-squares regression, which we formally defined in Problem (1). Naively speaking, the solution to this problem is actually relatively straightforward. To start off, we let

$$f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2 \tag{5}$$

be the objective function that we want to minimize. Here and onwards we will omit the intercept term for simplicity but note that we can incorporate in any of the following methods. To minimize Equation (5), we simply set the gradient of it to zero:

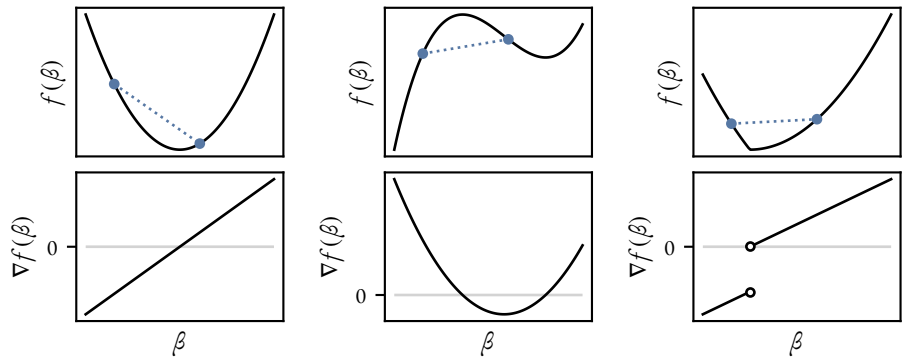
$$\begin{aligned} &\nabla f(\beta) = X^\intercal(X\beta - y) = \mathbf{0} \implies \\ &X^\intercal X\beta = X^\intercal y. \end{aligned}$$

This system[8] is called the *normal equations*. Solving the system in β yields the ordinary least squares estimate, which, for a one-dimensional problem is equivalent to locating the "bottom" of the function $f(\beta)$ in Figure 10a. It might be tempting to want to simply invert $X^\intercal X$ here and premultiply by both sides to yield an explicit solution of the form

$$\beta = (X^\intercal X)^{-1} X^\intercal y$$

but this is typically a bad idea since the inverse need not be numerically stable or even exist. A better option is to use a method such as QR factorization and solve the resulting system through forward or backwards elimination, which is both more stable and efficient; all modern software use some variation on this approach.

[8] We have ignored the intercept β_0 here for simplicity, but it could be incorporated easily by prepending a vector of ones to X.



(a) A convex and smooth function. The minimum ocurs when $\nabla f(\beta) = 0$.

(b) A non-convex but smooth function. The derivative offers no information about the global minimum (at $-\infty$).

(c) A convex but non-smooth function. There is a global minimum, but the derivative does not exist at this point.

Figure 10: Three different kinds of functions. We show the objective value $f(\beta)$ and the gradient $\nabla f(\beta)$ for each. In each case, our objective is to find the minimum of the function. Only the first and last are convex (the complete line segment between two points on the function lies above the function) and have a global minimum.

Regardless, however, OLS regression can be solved directly and with accuracy at machine precision. This property is shared by ridge regression in which we can attain a solution simply by adding a diagonal matrix[9] to $\boldsymbol{X}^\intercal \boldsymbol{X}$ and solving as before. The key reason for why this is the case is that OLS regression is a differentiable and quadratic problem, which means that it is *convex* and hence has a global solution (Figure 10a), unlike, for instance, the problem in Figure 10b, which is non-convex (actually a third-degree polynomial) and hence has a local minimum.

All the problems that we have covered so far: ordinary least-squares regression, the lasso, the elastic net, and SLOPE are all convex, which is the class of problems this thesis focuses on. Being convex, however, does not necessarily mean that the problem is easy to solve. The lasso, Problem (2), for instance, is a convex problem, but the involvement of the inequality constraint means that a direct solution is not readily available.

Somewhat remarkably, however, it actually *is* possible to solve the lasso directly, as long as we also solve the full lasso path up to the t that we want. The class of methods that makes this possible are called *homotopy algorithms* since they can be used to solve

[9] This procedure refers to the *unconstrained* form of ridge regression, which have not yet—but will soon—introduce.

the problem for all values they are parameterized by (in this case t). The first homotopy method for the lasso was introduced by Osborne, Presnell, and Turlach (2000a,b) but it is the LARS algorithm (Efron et al. 2004), that we already saw in action at the beginning of the section, which popularized the method for the lasso.

In essence, homotopy methods for the lasso are based on the idea that the model can be solved directly if we know the support of the solution (the identity of the non-zero coefficients). Based on this idea, we start with the empty support at $t = 0$. From this point, it is possible to say which features will become active first and at what t this happens, which makes it possible to then solve the problem (directly) a this value of t. The process is repeated until the entire path has been computed. We give a rough, but slightly more formalized, description of the method in Algorithm 1.

Algorithm 1: A rough outline of the homotopy method for the lasso path. The steps in lines 3 and 4 represent the critical aspect of the algorithm, but are omitted here for brevity. They are not, however, particularly demanding computationally. Instead, the primary costs come from the linear systems that need to be solved at each step of the path.

Input: $\boldsymbol{\beta}^{(0)} = \mathbf{0}$, $t \leftarrow 0$, $\mathcal{A} = \emptyset$, $i = 0$

1 **repeat**
2 $\quad i \leftarrow i + 1$;
3 $\quad t \leftarrow$ next value for which the support changes;
4 $\quad \mathcal{A} \leftarrow$ support at t;
5 $\quad \boldsymbol{\beta}^{(i)}_{\mathcal{A}} \leftarrow \arg\min_{\boldsymbol{\beta} \in \mathbb{R}^{|\mathcal{A}|}} f(\boldsymbol{\beta})$;
6 $\quad \boldsymbol{\beta}^{(i)}_{\mathcal{A}^C} \leftarrow \mathbf{0}$;
7 **until** $|\mathcal{A}| = p$;

At the time that these methods were introduced, they offered a remarkable boost in efficiency compared to the original algorithm used by Robert Tibshirani (1996), which consisted of an iterative method based on an algorithm by Lawson and Hanson (1995), which scales badly with p and is altogether inapplicable when $p > n$.

Since the elastic net can be recast as a lasso problem, it also means that the same homotopy methods can be used also in this case. In addition, there now also exists homotopy methods for SLOPE (Dupuis and P. J. C. Tardivel 2023; Nomura 2020), which comes from the SLOPE path sharing the piecewise-linear property of the lasso path (although the SLOPE path is typically more complicated and features more changes in support).

Even if these homotopy methods provide a much-wanted upgrade compared to the original method in the high-dimensional setting, it is nevertheless this domain that

they ultimately struggle to deal with. The root of this problem is that there are at least $\min(n, p)$ kinks (changes in support) along the full lasso path—and in the worst case as many as $(3^p + 1)/2$ such changes (Mairal and Yu 2012). The algorithm has to solve an equivalent number of OLS regression problems, albeit at a complexity much reduced from that of solving the full problem,[10] which, in the end, means that the method has found itself outperformed by iterative optimization methods (Friedman, Hastie, and Robert Tibshirani 2010), which we will introduce in the next section.

3.2 Iterative Optimization

In iterative optimization, we start with an initial guess of the solution (usually $\beta = \mathbf{0}$) and then update it step-by-step until we get "close enough" to the optimum.[11] This puts them apart from the direct methods we covered in the previous section, which typically involve solving a system of equations or a similar problem and yield a solution directly and at machine precision.

Gradient Descent

It is probably fair to say that *gradient descent* is the quintessential iterative optimization method. The basic idea is to update the optimization variable (β) by moving in the direction of the negative gradient of the objective function ($\nabla f(\beta)$). For a convex objective, the negative gradient points in the direction of the global minimum, which means that we eventually reach it given appropriate choices of our step sizes (how far we move in the opposite direction of the gradient).

Technically, the idea in gradient descent is to form a second-order Taylor expansion of the objective around the current iterate β',

$$f(\beta) \approx f(\beta') + \nabla f(\beta')^\intercal (\beta - \beta') + \frac{1}{2}(\beta - \beta')^\intercal \nabla^2 f(\beta')(\beta - \beta'),$$

replace the Hessian matrix $\nabla^2 f(\beta')$ with $\frac{1}{\tau}\boldsymbol{I}$ in this approximation, yielding

$$f'(\beta; \beta') = f(\beta') + \nabla f(\beta')^T (\beta - \beta') + \frac{1}{2\tau}\|\beta - \beta'\|_2^2,$$

and then, finally, solve for β, which gives the gradient descent update

$$\beta^+ = \underset{\beta \in \mathbb{R}^p}{\arg\min}\, f'(\beta; \beta') = \beta' - \tau \nabla f(\beta').$$

[10] LARS, for instance, incrementally update a Cholesky factorization of the Hessian matrix $\boldsymbol{X}^\intercal \boldsymbol{X}$.

[11] As you might expect, defining "close enough" is not at all a trivial matter.

The method, which is simple this step repeated in each iteration, is outlined in Algorithm 2.

Observe that the τ we used when we replaced the Hessian with a diagonal matrix is the step size in the gradient descent algorithm. This is the key parameter in the algorithm, which controls how far we move in the direction of the negative gradient. In Figure 11, we show how gradient descent works for simple problems in one and two features.

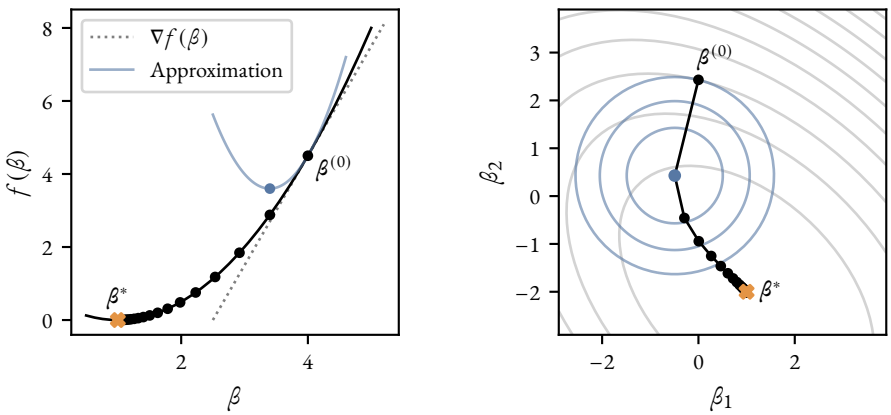
Algorithm 2: Gradient descent with fixed step size. An intercept can be added by either prepending a vector of ones to $\boldsymbol{X}$ or adding a separate update step where $\boldsymbol{\beta}$ is held fixed.

Input: $\boldsymbol{\beta} = \mathbf{0}$, $\tau > 0$

1 **repeat**

2 $\quad \boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \tau \nabla f(\boldsymbol{\beta})$;

3 **until** *convergence*;



(a) A one-dimensional problem. The black curve shows the value of the objective $f(\boldsymbol{\beta})$.

(b) A two-dimensional problem. The grey curves are the level curves of the OLS regression objective $f(\boldsymbol{\beta})$.

Figure 11: Gradient descent in one and two dimensions. In each case the algorithm starts at $\boldsymbol{\beta}^{(0)}$ and then proceeds towards the optimum $\boldsymbol{\beta}^*$ (✖). The quadratic approximation of the first step is drawn in blue lines (and its optimum marked by a blue dot).

Projected Gradient Descent

The problem for us, however, is that the lasso, the elastic net, and SLOPE—the problems we are mainly concerned with—have inequality constraints, which gradient descent cannot handle directly. A natural alternative, however, exists in the form of *projected* gradient descent. The method consist of simply taking a gradient descent step, as in Algorithm 2, and then projecting the result onto the feasible region (if the update after the gradient step is infeasible). The update is then

$$\boldsymbol{\beta} \leftarrow \text{proj}_C\left(\boldsymbol{\beta} - \tau \nabla f(\boldsymbol{\beta})\right)$$

where

$$\text{proj}_C(\boldsymbol{u}) = \underset{v \in C}{\arg\min} \|\boldsymbol{v} - \boldsymbol{u}\|_2 \tag{6}$$

is the projection operator that projects $\boldsymbol{u}$ onto the feasible region C. For the lasso, for instance, C is the ℓ_1 norm ball (Figure 5), while for SLOPE it is the sorted ℓ_1 norm ball (Figure 8). The method is outlined in Algorithm 3 and is identical to the gradient descent algorithm (Algorithm 2), except that we have now wrapped a projection operator around the gradient descent update.

Algorithm 3: Projected gradient descent. The projection operator $\text{proj}_C(\cdot)$ is defined in Equation (6).

Input: $\boldsymbol{\beta} = \mathbf{0}$, $\tau > 0$

1 **repeat**

2 $\quad \boldsymbol{\beta} \leftarrow \text{proj}_C\left(\boldsymbol{\beta} - \tau \nabla f(\boldsymbol{\beta})\right)$;

3 **until** *convergence*;

Figure 12 shows the method in action for a two-dimensional lasso problem. Note how the gradient step (dotted lines) takes the algorithm outside the feasible region and that the projection step (solid lines) then moves the solution back to the feasible region.

The efficiency of projected gradient descent hinges on the efficiency with which the projection can be computed. Thankfully, there exists efficient projections for both the ℓ_1 (Duchi et al. 2008) and sorted ℓ_1 norms (Davis 2015; Q. Li and X. Li 2021; Perez et al. 2022; Zeng and Figueiredo 2015), which means that the method is quite efficient for both the lasso and SLOPE.

The Subgradient Method

Projected gradient descent attacks our optimization problems by tackling the inequality constraints directly; but the by far most popular approach for solving these problems

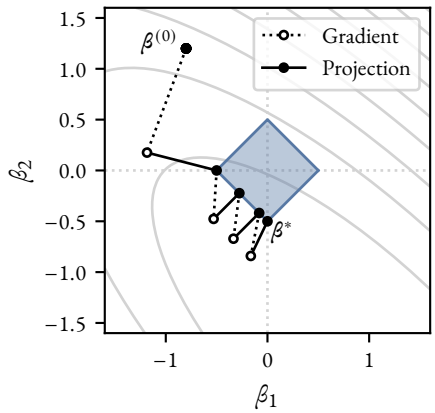


Figure 12: Projected gradient descent for a two-dimensional lasso problem. The gray curves are the level curves of the ordinary least-squares objective $f(\beta)$ and the blue diamond shape is the ℓ_1 norm constraint. The algorithm starts at β_0 and then proceeds towards the solution β^*. Each step consists of a gradient step (dashed lines) and a projection step (solid lines).

actually takes a different route by transforming the constrained problem into an unconstrained one. The idea is that we can achieve an equivalent regularization effect by, instead of constraining the solution directly, penalizing the coefficients via the objective function.

In other words, we turn the problem

$$\begin{aligned} &\text{minimize} && g(\beta), \\ &\text{subject to} && h(\beta) \leq t, \end{aligned}$$

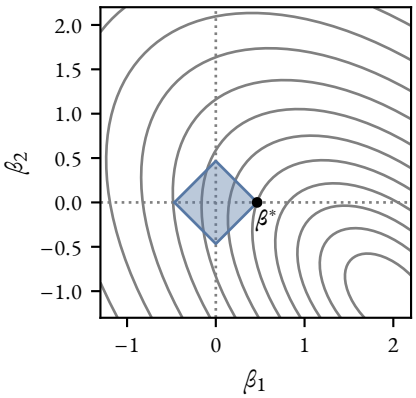
into

$$\text{minimize} \quad f(\beta) = g(\beta) + h(\beta).$$

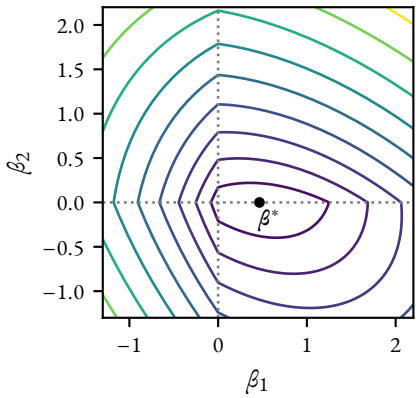
In the case of SLOPE, for instance, the unconstrained version of Problem (4) then becomes

$$\text{minimize} \quad \frac{1}{2}\|y - X\beta\| + \sum_{j=1}^{p} \lambda_j |\beta_{(j)}|.$$

You may wonder what good this did us, but the key is that the we have now gotten rid of the constraint, which means that we can focus on minimizing the objective



(a) The constrained form of the lasso. The level curves show the ordinary least-squares objective. The blue region is the ℓ_1-norm constraint



(b) The unconstrained form of the lasso. Here, the level curves show the unconstrained optimization objective.

Figure 13: Constrained and unconstrained versions of the lasso. The equivalence between the problems is obtained by setting $t = \|\beta\|_1$ after solving the unconstrained problem to convergence for a given λ.

directly. We can see how the two optimization problems compare for an equivalent setting of λ (in the case of the unconstrained problem) and t (in the case of the constrained problem) for the lasso in Figure 13.

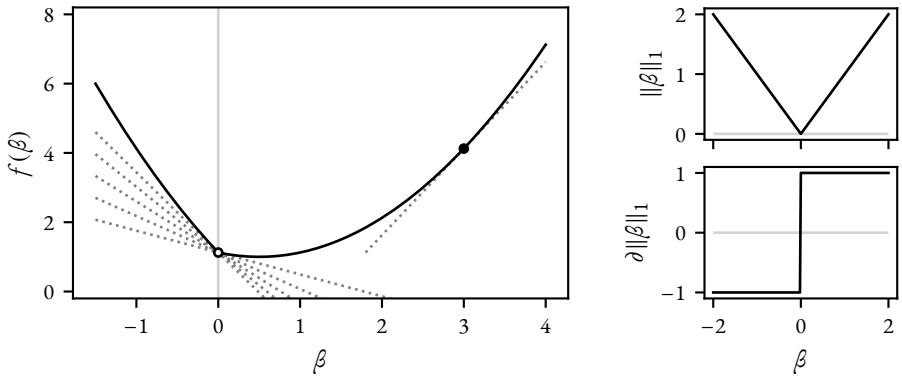
The bad news is that the new objective is no longer differentiable, which means that we cannot, for example, use gradient descent with the new formulation either. The good news, however, is that there is a generalization of the derivative that can be used in the non-differentiable case, namely the *subdifferential*.

The subdifferential of a function f at a point β is the set of all *subgradients* of f at that point. This means that the subdifferential is a set, rather than a single vector. A subgradient at a point β is a vector g such that

$$f(\beta') \geq f(\beta) + g^\intercal(\beta' - \beta).$$

In plain terms, this simply means that the linear approximation of the function at β always underestimates the function. If the function is differentiable, then the subdifferential is a singleton, containing only the gradient of the function. In Figure 14a, we show some of the subgradients of a one-dimensional lasso problem.

The existence of subgradients presents us with an intuitive solution to the problem of minimizing non-differentiable functions: use gradient descent, but replace the



(a) Subgradients of a one-dimensional lasso problem. At $\beta = 3$, there is just a single subgradient: the (ordinary) gradient, but at $\beta = 0$, there are multiple subgradients.

(b) The ℓ_1 norm (absolute value) of the coefficient β and its subgradient $\partial\|\beta\|_1$.

Figure 14: Subgradients of the lasso and the ℓ_1 norm

gradient with the subgradient, and take a step in its direction. We outline the algorithm in Algorithm 4.

Algorithm 4: The subgradient method. Note that g can be *any* subgradient of the subdifferential $\partial f(\beta)$.

Input: $\beta = \mathbf{0}$, $\tau > 0$
repeat
  $s \leftarrow s \in \partial f(\beta)$;
  $\beta \leftarrow \beta - \tau s$;
until *convergence*;

The attractiveness of the subgradient method is that it is general and works for a large class of problems, including the lasso, the elastic net, and SLOPE. The problem is that it converges slowly and is therefore never used in practice for these problems.

Proximal Gradient Descent

Fortunately, we have structure in this problem that we have yet to exploit, namely that although $f = g + h$ is not differentiable, g in fact *is*. Based on this, one idea is to simply

leave h alone and minimize the quadratic expansion of g plus h (untouched).

$$\begin{aligned}\arg\min_{\beta}\left(g(\beta')+\nabla g(\beta')^\intercal(\beta-\beta')+\frac{1}{2\tau}\|\beta-\beta'\|_2^2+h(\beta)\right)\\ =\arg\min_{\beta}\left(\frac{1}{2\tau}\|\beta-\underbrace{(\beta'-\tau\nabla g(\beta'))}_{\text{Gradient update}}\|_2^2+h(\beta)\right).\end{aligned}$$

Let us call the function that solves this problem the *proximal operator* of h, denoted $\text{prox}_{h,\tau}$, and define it as

$$\text{prox}_{h,\tau}(\boldsymbol{u})=\arg\min_{\boldsymbol{v}}\left(\frac{1}{2\tau}\|\boldsymbol{v}-\boldsymbol{u}\|_2^2+h(\boldsymbol{v})\right).$$

The proximal operator is a generalization of the projection operator that we introduced earlier, and we can obtain the latter by using the indicator function of the feasible region as the function h. We outline the basic version of the proximal gradient descent algorithm in Algorithm 5.

Algorithm 5: Proximal gradient descent. Note that the gradient is taken with respect to g and not f (for which it does not exist).

Input: $\beta = \mathbf{0}$, $\tau > 0$

repeat

  $\beta \leftarrow \text{prox}_{h,\tau}\left(\beta - \tau\nabla g(\beta)\right)$;

until *convergence*;

The reason for why all of this works is that the proximal operator has an explicit, and often efficient, form for all the problems we are interested in solving here. For the lasso, for instance, the proximal operator has a particularly simple form:

$$\text{prox}_{h,\tau}(\boldsymbol{u})=\arg\min_{\boldsymbol{v}}\left(\frac{1}{2\tau}\|\boldsymbol{v}-\boldsymbol{u}\|_2^2+\lambda\|\boldsymbol{v}\|_1\right)=\text{S}_{t\lambda}(\boldsymbol{u}),$$

where

$$\text{S}_\lambda(\boldsymbol{u})_i=\text{sign}(u_i)\max(|u_i|-\lambda,0),$$

is the *soft-thresholding operator* (Donoho and Johnstone 1995) and is illustrated in Figure 15. Using proximal gradient descent for the lasso is also known as the iterative shrinkage-thresholding algorithm (ISTA) (Beck and Teboulle 2009).

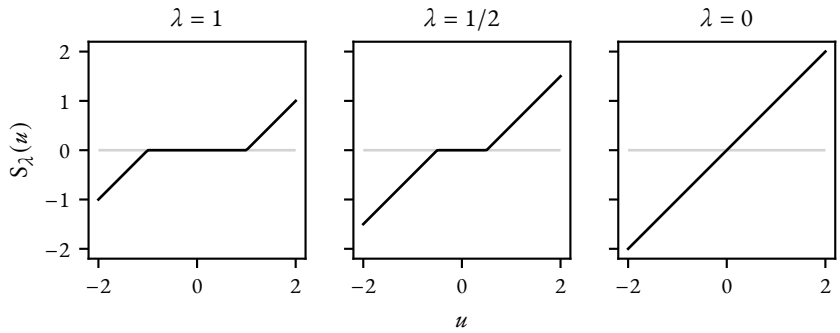


Figure 15: The soft-thresholding operator: the proximal operator in the case of the lasso. Here we show it for three different values of λ. For $\lambda = 0$, there is no thresholding (penalization), and the operator becomes the identity function. For all other values, the operator shrinks its input toward zero. For $|u| \leq \lambda$, the output is exactly zero.

Proximal gradient descent has a long history. Rockafellar (1970) was responsible for much of the early work. Nesterov (1983) introduced the first accelerated version of proximal gradient descent algorithms, which where later improved upon by, for instance, Beck and Teboulle (2009) in the form om the fast iterative shrinking-thresholding algorithm (FISTA). Other improvements include stochastic versions, line searches, improved step size settings, and other types of accleration, such as the Anderson variant (Mai and Johansson 2020).

At this point you may wonder what we have gained compared to the projected gradient method. For one thing, we have obtained an algorithm with more general applicability. Indeed, as Bogdan, Berg, Sabatti, et al. (2015) showed, there is an efficient algorithm for computing the proximal operator even in the case of the sorted ℓ_1 norm. Furthermore, we have also gained slightly in computational efficiency. In the lasso case, for instance, the soft-thresholding operator has $O(p)$ complexity whereas the projection onto the ℓ_1 ball has $O(p \log p)$ complexity. Please see Figure 16 for a small example on what this can amount to in practice.

In conclusion, proximal gradient descent is a simple algorithm with strong convergence guarantees that is easy to implement. As we shall see next, however, it is typically outperformed by a more naive method: coordinate descent.

Proximal Coordinate Descent

Coordinate descent is a simple algorithm: update one coefficient (coordinate) at a time by finding the minimizer with respect to that coefficient. In other words, we solve a

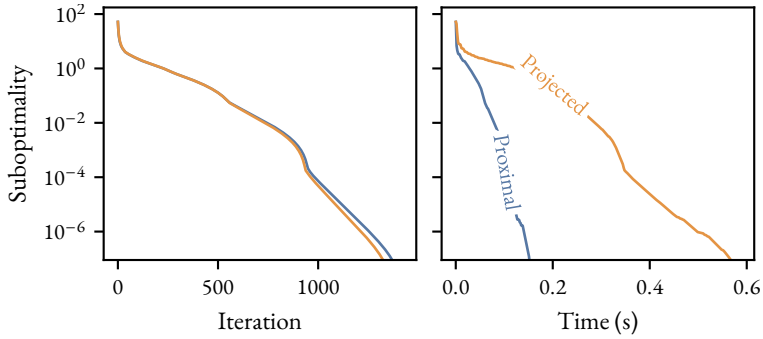


Figure 16: Comparison between projected and proximal gradient descent for an equivalent lasso problem. Per-iteration, the methods both perform almost on par with one another. In terms of wall-clock time, however, the proximal method is more efficient.

one-dimensional problem at each iteration of the algorithm. The algorithm extends naturally to the case when we have a composite objective of the form $f = h + g$, as in the case of proximal gradient descent, provided that h is differentiable and g is convex and *separable* in $\boldsymbol{\beta}$. Formally, this algorithm is called *proximal coordinate descent*. In this text, however, we will also allow ourselves a slight misuse of nomenclature and simply call it *coordinate descent*. The basic outline of the algorithm is given in Algorithm 6.

Algorithm 6: Proximal coordinate descent. Note that the implementation given here is designed for illustration; many improvements can be made that are critical to the practical performance of the algorithm.

Input: $\boldsymbol{\beta} = \mathbf{0}$, $\boldsymbol{\tau} \in \mathbb{R}^p_+$

1 **repeat**

2 Pick $j \in \{1, 2, \dots, p\}$;

3 $\beta_j \leftarrow \operatorname{prox}_{h,\tau}\left(\beta_j - \tau_j \nabla g(\boldsymbol{\beta})_j\right)$;

4 **until** *convergence*;

For the lasso, the coordinate descent update in Line 3 (Algorithm 6) is particularly cheap and amounts only to soft-thresholding of the partial residual. In Figure 17, we show a simple example of how coordinate descent works for a two-dimensional lasso problem.

There are varies strategies for picking the coefficient to optimize over in step three. The simplest and most common alternative is to cycle through the coefficients one by

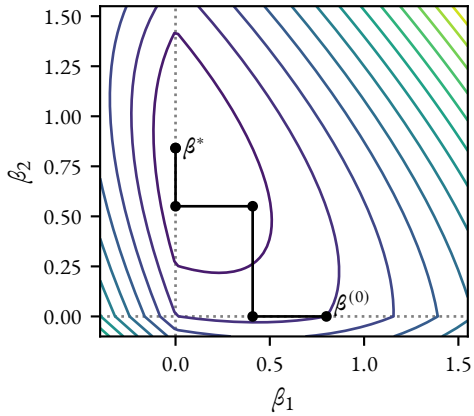


Figure 17: Cyclic coordinate descent for a lasso problem in two dimensions. At each step, we minimize only in the direction of one of he coefficients (along either the x or the y axis). In this case, g is the least-squares criterion, which means that the step always takes us to the minimum in that direction.

one, which is called *cyclic* coordinate descent. Another common alternative is to pick the coefficient at random, which makes for an algorithm that is easier to analyze than cyclic coordinate descent and somewhat more robust, but which also, in our experience, usually slightly slower.

At face-value it may seem somewhat surprising that this simple strategy should outperform proximal gradient descent, which considers all features simultaneously, particularly in light of the fact that proximal gradient descent sports better convergence rates (Wright 2015). Yet, barring a few exceptions, this is precisely the case (Figure 18).

The counterintuitive nature of this result may explain why coordinate descent algorithms were initially largely ignored in the literature on optimization. Daubechies, Defrise, and De Mol (2004), Fu (1998), and Shevade and S. S. Keerthi (2003) all wrote (relatively) early papers on coordinate-descent algorithms for the lasso, yet these initially received scant attention. It was only several years later that a paper by Friedman, Hastie, Höfling, et al. (2007) and corresponding software implementation in the `R` package `glmnet` (Friedman, Hastie, and Robert Tibshirani 2010) caused the method to grow in popularity.

The main reason for why coordinate descent works so well for the lasso, the elastic net, and many related problems is related to the size of the steps that the algorithms can take (while still guaranteeing convergence). In the least-squares case, proximal gradient descent is limited by a step size of $\tau = 1/\|\boldsymbol{X}\|_2$, whereas coordinate descent can take

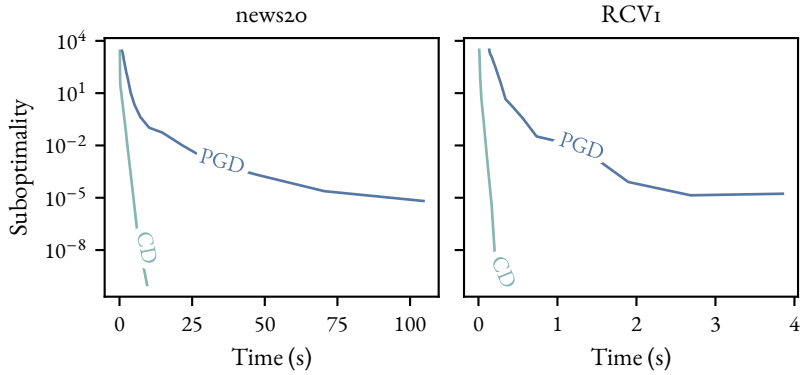


Figure 18: Proximal coordinate descent versus accelerated proximal gradient descent (FISTA) for two datasets: `news20` (S. Sathiya Keerthi and DeCoste 2005) and `RCV1` (Lewis et al. 2004). The dimensions of the data are $n = 199\,961$, $p = 355\,191$ for `news20` and $n = 20\,242$, $p = 47\,236$ for `RCV1`. The response in both cases is binary and we fit a standard lasso model In both cases we normalize with maximum–absolute value scaling. The comparisons were made using the `benchopt` package (Moreau et al. 2022).

steps of size $\tau_j = 1/\|\boldsymbol{x}_j\|_2^2$, which for much real data are decidedly larger for many of the features and hence promote faster convergence. For `news20`, for instance, we have $\|\boldsymbol{X}\|_2 = 161$ whereas the median of $\{\|\boldsymbol{x}_j\|_2^2 : j = 1, 2, \dots, p\}$ is 1.40 (interquartile range: 1.26–1.60, maximum: 34.4). The price we pay for this increase in step size is a limit to one direction at a time. This can pose problems in the case when there is large correlation between some features, since coordinate descent then may struggle to break this dependency apart.[12] Empirically, however, we have found coordinate descent to largely dominate proximal gradient descent.

As in the case proximal gradient descent, coordinate descent can also be accelerated via the Nesterov scheme (Nesterov 1983) or Anderson acceleration (Bertrand and Massias 2021). In practice, however, we have found that the benefit of acceleration is lower in the case of coordinate descent than it is for gradient descent.

One problem with coordinate descent, however, is that it requires the objective to be separable in the optimization variable $\boldsymbol{\beta}$; this is not the case for SLOPE, which makes the method inapplicable for this problem.

[12] Note that proximal gradient descent is also not well-equipped to deal with this situation. Instead, this is the setting where Hessian-based methods such as proximal Newton (Lee, Sun, and M. A. Saunders 2014) dominate.

We have introduced many optimization methods in this section, yet have only scratched the surface in the field. Several methods have been omitted, such as the alternating direction method of multipliers (ADMM) (Boyd et al. 2010), proximal Newton (Lee, Sun, and M. A. Saunders 2014), interior-point methods (Kim et al. 2007), and stochastic gradient descent (Bottou 2010; Robbins and Monro 1951) and its many derivatives. We have also omitted many details regarding the implementation of the algorithms, such as step size selection and convergence criteria. Yet, while these are important and interesting aspects of the algorithms, we omit them here for brevity and provide all the necessary details in the papers themselves.

Although optimization algorithms are fundamental in ensuring good performance in fitting sparse regression models such as the lasso, the elastic net, and SLOPE, they are not the only way to improve performance. In the next section, we will introduce *screening rules*, which drastically reduces the computational cost of fitting these models, particularly in the high-dimensional setting.

4 Screening Rules

Screening rules are a remarkably efficient method for speeding up the optimization of sparse regression methods. They are based on the following reasoning:

1. We know that the solution is going to be sparse, especially if $p \gg n$[13].
2. We also know something about the importance of the features, even before fitting the model, since we for instance (as in Figure 1) can compute the correlation between the features and the response. In addition to this, we are also typically interested in solving for a range of regularization parameters, which means that the problem we are currently trying to solve is likely related to a problem we have already solved.
3. Therefore, we might be better of by only considering a subset of the features when solving the problem. And if this subset is small and selecting it is cheap, then we should be able to save a lot of time.

This intuition turns out to be correct, and screening rules have consequently brought a pivotal discovery to the field of optimization for sparse methods, particularly in the high-dimensional domain.

The first screening rule for the lasso, SAFE (SAfe Feature Elimination), was introduced by El Ghaoui, Viallon, and Rabbani (2010). In essence, the authors showed that it was possible to confine the solution for a given λ to a region of the feature space,

[13] Recall, for instance, that the lasso can only select $\min(n, p)$ features.

and feature-by-feature ascertain whether it is possible for the feature be selected in the solution.

In practice, this means that if you have a dataset of, say, one million features (p) and one thousand observations (n) and want to fit the lasso for some value of λ, then there is a good chance that you may just have to fit a problem with a few hundred features, while at the same time guaranteeing that you will reach the same solution as the one-million-feature problem would. In addition, the cost of running the screening rule is negligible next to solving the problem (even the reduced problem). The net result is a remarkable gain in computational performance. It is not in our experience rare to see speedups of several orders of magnitude, for instance reducing the time taking to fit this size of model from hours to minutes (or even less).

The screening rule introduced by El Ghaoui, Viallon, and Rabbani (2010) is a *safe* screening rule, meaning that all of the features discarded by the rule are guaranteed to be absent at the optimum. As it turns out, however, this requirement on safety in general leads to rules that are overly conservative. This was made clear in a paper by Robert Tibshirani, Bien, et al. (2012), in which they introduced the *strong screening rule for the lasso*. This rule uses an initial screening test that is *heuristic* (as opposed to safe), which means that it may discard features that actually are part of the solution. This necessitates a check of the optimality conditions after solving the reduced problem and, in case any features were incorrectly discarded, a refitting of the model with these features icluded. As the authors showed, however, these *violations* of the screening rule are so rare in practice that there is often no need to refit. And if there is, the estimate from the reduced problem is often so close to the optimum of the real problem that only a few extra iterations of the optimization algorithm are needed.

It is important to highlight that the nomenclature for screening rules is quite misleading.[14] Even if the screening performed by the strong rule is *not* safe, the actual screening rule method *is*, which stems directly from the fact that all of these methods include checks of the optimality conditions in order to safeguard against discarding features that are part of the solution.[15] In other words, even though we call some rules *safe* and some *heuristic*, they are in fact *all* safe as they are implemented in practice.

Robert Tibshirani, Bien, et al. (2012) also introduced the notion of *sequential* screening rules, meaning that screening is performed along the regularization (lasso) path with the solution at the current step on the path as the starting point for the screening rule, which significantly impoves the effectiveness of the rule. One problem with both SAFE and the strong rule, however, is that their effectiveness decrease when features in the design are heavily correlated—the problem is particularly severe in SAFE.

[14] We stress this point since we have (repeatedly) found this fact to stump reviewers of our papers.

[15] There *are* screening rules that actually are unsafe, but they are used for an altogether different purpose in which one is interested not just in screening, but also changing the model (for instance the lasso) itself.

This lead Robert Tibshirani, Bien, et al. (2012) to conclude, which they also demonstrate empirically, that the most efficient screening method (at least in the sequential case) is to begin with the set of features that have so far *ever* been active along the path: $\mathcal{E}$, and solve the problem for this set first. After reaching a solution for this set, the optimality checks are then performed on the features present in the strong rule set $\mathcal{S}$, but not in $\mathcal{E}$ ($\mathcal{S} \setminus \mathcal{E}$). If there are any violations, which is often the case since no screening was performed, these features are entered into $\mathcal{E}$. The problem is then solved, again, but this time for the updated $\mathcal{E}$. This procedure is then repeated until no violations are found among $\mathcal{S} \setminus \mathcal{E}$. Finally, when this happens, the optimality conditions are checked for the entire set of features, and if no violations are found, the solution is deemed to be found. If there are violations, the procedure repeats with these features added to $\mathcal{E}$.

Since these two papers were published, there has been a slew of research around screening rules for the lasso as well as other sparse regression problems. Other notable examples of safe rules include the sphere tests (Zhen J. Xiang, Xu, and Peter J Ramadge 2011), R-region test (Zhen James Xiang and Peter J. Ramadge 2012), and Gap Safe (Fercoq, Gramfort, and Salmon 2015; Ndiaye et al. 2017).

We have so far discussed screening rules in relatively informal terms. In the next section, we will provide a more formal treatment of the topic, which we hope will provide a better understanding of how the methods work.

4.1 Screening Rules and the Correlation Vector

To better understand how screening rules work, we need to consider the optimality conditions of our problems. For the unconstrained versions of our problems, they are

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

where $\boldsymbol{s}$ is a subgradient of the penalty term h. Letting $\boldsymbol{c} = -\nabla g(\boldsymbol{\beta})$, we can rewrite this as

$$\boldsymbol{c} \in \boldsymbol{s} h(\boldsymbol{\beta}). \tag{7}$$

We will call $\boldsymbol{c}$ the *correlation vector*, which is simply the negative gradient of $g(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$.

For illustrative purposes, we will focus on the lasso here, for which the $\partial h(\boldsymbol{\beta})$ is the subdifferential of the ℓ_1 norm multiplied with λ, defined as

$$s_j \in \begin{cases} \{\lambda \operatorname{sign}(\hat{\beta}_j)\} & \text{if } \hat{\beta}_j \neq 0, \\ [-\lambda, \lambda] & \text{otherwise,} \end{cases}$$

which we previously visualized in Figure 14b. One consequence of this is that, if

$$|\nabla g(\boldsymbol{\beta})_j| < \lambda,$$

then it must also hold that $\beta_j^* = 0$. In other words, if the absolute value of the gradient with respect to feature j is smaller than our level of regularization, then the feature must be absent from the solution. Similarly, this also means that if $\beta_j^* \neq 0$, then its gradient will be $\pm\lambda$. We can observe this behavior in Figure 19, where we have plotted the lasso path for the diabetes dataset that we have encountered before. Note that we start at the left. At this point only a single feature is active. Then, as we lower λ (move from left-to-right in the plot), the features become active one-by-one. And at exactly the point where they do so, they also join either of the lines $c = \pm\lambda$; and as long as they stay active, that is where they remain.

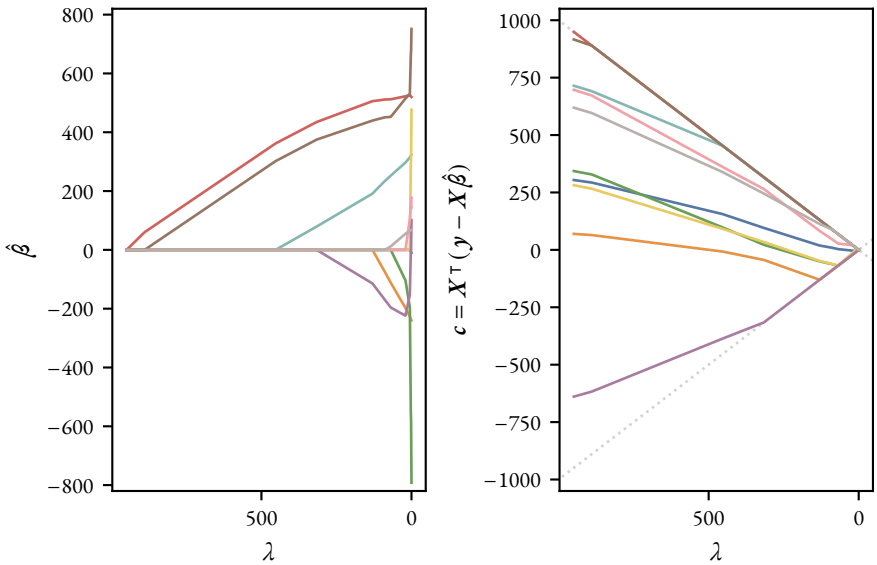


Figure 19: The lasso path for the diabetes data. This time, we have also plotted the correlation (negative gradients) along the path. In order for a feature to become active from one step on the path (one value of λ) to the next, its correlation c_j has to reach either of the $\lambda = \pm c$ (dotted) lines. Note that the x-axis is flipped to illustrate the fact that we typically proceed from a large λ towards a small one (left to right in the plot).

So, if we knew, before fitting, that $|c_j| < \lambda$, then we could safely discard feature j from the problem and still solve it as if it were always there. The problem, however, is that we do not have access to the correlation vector before fitting the model.

What we can do instead, however, is to estimate the correlation vector and use that in the place of the true value. Let us say that we are at step k on the path and have

computed the solution at this step; then we also know $\boldsymbol{c}$ at this step ($\boldsymbol{c}^{(k)}$). Next, we want to predict what $\boldsymbol{c}^{(k+1)}$ will be in order to screen features for this (upcoming) step. One rather conservative approach for this is to assume that gradient of the correlation vector is bounded by one, in other words, we introduce the approximation

$$\hat{c}_j^{(k+1)} = c_j^{(k)} + \operatorname{sign}\left(c_j^{(k)}\right)(\lambda_k - \lambda_{k+1})\,.$$

Using this approximation is in fact exactly the strong rule for the lasso. In Figure 20, we illustrate what this approximation amounts to in practice for the diabetes dataset.

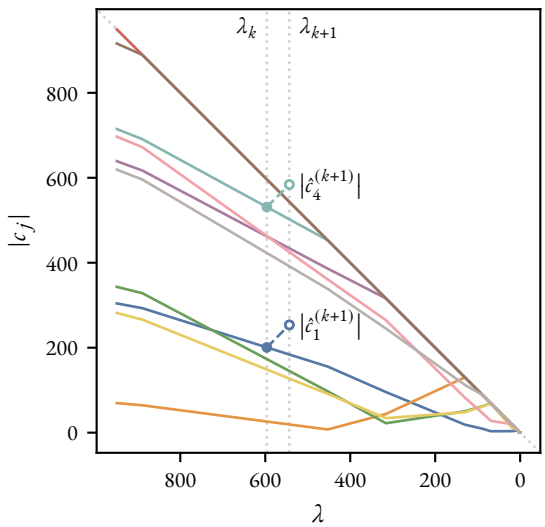


Figure 20: The sequential strong rule for the lasso. The data is the diabetes dataset that we have encountered several times already. We are at step k and want to screen features for step $k+1$. We show the screening rule for two features here. The filled circles show $|c_j|$ for each feature and the open circles show the strong rule estimate $|\hat{c}_j|$. For feature 1, the approximation $|\hat{c}_1|$ does not reach λ in time for the next step—so the rule discards it. For the fourth feature, however, $|\hat{c}_4|$ is larger than λ, and so feature 4 cannot be discarded.

As we mentioned previously, this is a somewhat conservative estimate of the gradient for the next step, which, in addition, uses no information at all about the shape of the path of the correlation vector. And even though the rule has lead to remarkable improvements in performance of optimization algorithms, this fact causes it to struggle

in less well-conditioned datasets where, for instance, there is large correlation between the features (Robert Tibshirani, Bien, et al. 2012).

Another consequence of Equation (7) is that it is possible to know at which λ the first feature enters the model. In the case when g is the least-squares objective, for instance, we have the following optimality condition:[16]

$$\left|\boldsymbol{X}^\intercal \left(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\right)\right| = \left|\boldsymbol{X}^\intercal \boldsymbol{y}\right| \leq \lambda$$

since we know that $\boldsymbol{\beta} = \boldsymbol{0}$ at this point. This means that the largest value for this equation to hold is

$$\lambda_{\max} = \max_j |\boldsymbol{x}_j^\intercal \boldsymbol{y}|.$$

We have used this setting repeatedly throughout this text as a starting point for the lasso path (in the unconstrained case).

This also, however, highlights a property of the lasso that is true of the elastic net and SLOPE as well: it is sensitive to the scales of the features. The larger the values of $\boldsymbol{x}_j$, the larger the inner product $\boldsymbol{x}_j^\intercal \boldsymbol{y}$ becomes, which means that if feature j for instance is the length of a person, then it will enter the path earlier if our measurements are in meters rather than centimeters. This is of course not desireable. To deal with this issue, we need to *normalize* the features, which is the topic of the next section.

5 Normalization

One important aspect of regularized models is that they are generally sensitive to the scale of the features in the problem. This is the case for the lasso, the elastic net, and SLOPE, since they penalize the magnitude of the coefficients. And it puts them apart from ordinary least squares and best-subset selection, which we introduced earlier.

For a simple example of this behavior, assume that we have a dataset that is generated from a normal distribution in the following manner:

$$\boldsymbol{X} \sim \text{Normal}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}4 & 0\\0 & 1\end{bmatrix}\right), \qquad \boldsymbol{\beta} = \begin{bmatrix}1/2\\1\end{bmatrix}, \qquad \boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

with $\varepsilon \sim \text{Normal}(0, 1)$, identically and independently for each $i = 1, 2, \dots, n$.

In other words, we have generated our features from two normally distributed variables, both with mean zero and standard deviation 2 and 4, respectively, and with no correlation between them. We have constructed our problem such that the effects on the response for the two features are equivalent, in the sense that a change of one standard deviation in either feature will result in a change of one in the response.

[16] Note that this equation is for the no-intercept case.

Since the errors are generated according to the assumptions of the ordinary least squares model, we will recover the true coefficients in expectance. And when we compute the standardized coefficients from our estimates, according to

$$\hat{\beta}_{\text{std}} = s_j \hat{\beta}_j,$$

where s_j is the standard deviation of the feature $\boldsymbol{x}_j$, the coefficients are both 1 (for ordinary least-squares regression).

If we fit the lasso to this data, however, we will find that the feature with the larger standard deviation will be penalized *less* than the other (Table 2), which we see from looking at the standardized coefficients. A similar effect occurs in ridge regression. This behavior is typically not desirable.

Table 2: The effect of regularization on the regression coefficients and standardized versions thereof.

Model	$\hat{\beta}$	$\hat{\beta}_{\text{std}}$
OLS	$\begin{bmatrix}0.50 & 1.00\end{bmatrix}^\intercal$	$\begin{bmatrix}1.00 & 1.00\end{bmatrix}^\intercal$
Lasso	$\begin{bmatrix}0.38 & 0.50\end{bmatrix}^\intercal$	$\begin{bmatrix}0.75 & 0.50\end{bmatrix}^\intercal$
Ridge	$\begin{bmatrix}0.47 & 0.78\end{bmatrix}^\intercal$	$\begin{bmatrix}0.93 & 0.78\end{bmatrix}^\intercal$

To tackle this, the standard approach is to *normalize* the design matrix $\boldsymbol{X}$ before fitting the model by centering (subtracting a value, for instance its mean) and scaling each feature (by dividing with a value, for instance its standard deviation).[17] The goal is to place the features on the same scale. For features that are normally distributed, this is relatively straightforward to do. You simply need to scale with the standard deviation. But when the distribution of the features is different, for instance when whey are binary $x_{ij} \in \{0, 1\}$, the situation is more complicated. Unfortunately, there is little literature to fall back on in this case; in fact, this holds in general for the case of normalization. Typically, researchers and practioners instead rely on "standard practice", but this differs from field to field. Two common types of normalization is *standardization*, which is shifting by the mean and scaling by the standard deviation[18] of each feature, and *max–abs* scaling, which does no shifting but scales with the maximum absolute value of each feature. The first type, standardization, is common in the statistical literature, whilst the second type, max–abs normalization, is common in machine learning, particularly when dealing with sparse data. As we can observe in Figure 21,

[17] Afterwards, the coefficients are typically returned to their original scale.

[18] The formula without Bessel's correction is typically used.

however, the two methods generate starkly different results for the estimated coefficients in many datasets.

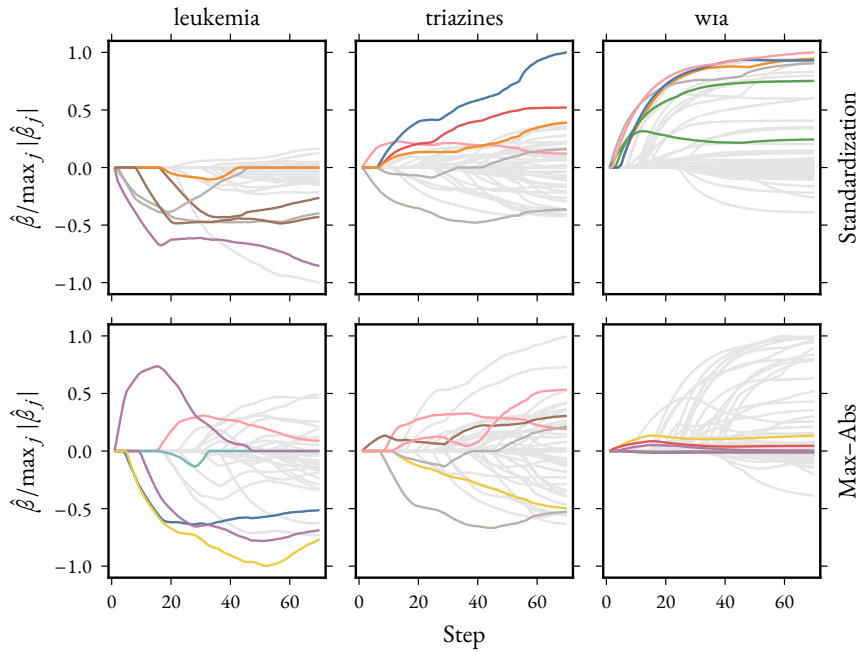


Figure 21: Lasso paths for real datasets using two types of normalization: standardization and maximum absolute value scaling (max–abs). We have fit the lasso path to three different datasets: `leukemia` (Golub et al. 1999), `triazines` (R. King 2024), and `w1a` (Platt 1998). For each dataset, we have colored the coefficients if they were among the first five features to become active in under either of the two types of normalization schemes. We see that the paths differ with regards to the size as well as the signs of the coefficients, and that, in addition, the coefficients to become active first differ between the normalization types.

This concludes our introduction to the topics of this thesis, including some of the issues and challenges faced by methods and practices in this field. In the next section, we will introduce the papers that this thesis is based on and briefly summarize their contributions in tackling these challenges.

6 Summary of the Papers

6.1 Paper I

In Section 4, we demonstrated the considerable effect that screening rules have had on sparse regression problems in the large-p setting. As a result, the papers by El Ghaoui, Viallon, and Rabbani (2010) and Robert Tibshirani, Bien, et al. (2012) led to a surge in interest for screening rules for the lasso and its derivatives, including the elastic net. For SLOPE, however, there existed no screening rule until our work in this paper, in which we introduced the *strong screening rule for SLOPE* (Larsson, Bogdan, and Wallin 2020).

A key contribution in our paper is the development of a practical characterization of the subdifferential for the sorted ℓ_1 norm, from which we derive an effective algorithm for implementing the screening rule in practice. Just as for the lasso, we demonstrate remarkable boosts in performance for fitting SLOPE (Table 3). The net result is a considerable extension in reach for SLOPE in the high-dimensional setting.

Table 3: Benchmarks measuring wall-clock time for four datasets fit with different models 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

6.2 Paper II

As we saw in Section 4, screening rules are particularly effective when they are sequential, that is, operate along the lasso, elastic net, or SLOPE paths. But another possibility that had previously not been explored is the idea of screening not only for the next step on the path, but for *all* of the remaining steps as well. This is the idea behind *look-ahead screening rules*, which we introduce in this short paper (Larsson 2021). We employ this idea for the gap-safe screening rule, which, as the name suggests, is a safe screening rule. This means that if a feature is screened out, it is guaranteed to be zero in the solution.

In Figure 22, we illustrate the effectiveness of look-ahead screening for the lasso path at the first step for the `leukemia` dataset. What the plot indicates is that many

features can be safely exempt from model fitting for a range of steps on the path, which means that we only need to re-screen them at the first grey square for each given feature.

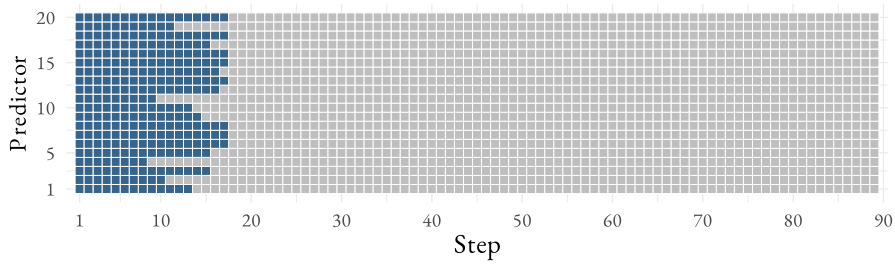


Figure 22: Look-ahead screening for the lasso at the first step on the path for the `leukemia` dataset. The plot shows a random sample of 20 features. Blue squares indicate that the corresponding feature can be discarded for those steps on the lasso path.

In the paper, we show that this effect has sizeable consequences for the computation time requires for fitting the full path.

6.3 Paper III

Even though the strong rule for the lasso is highly effective in general, there is one area in which it struggles, namely, when features are highly correlated. Robert Tibshirani, Bien, et al. (2012) in fact noted this themselves and forwarded it as the main motivation for using the working-set strategy (where the model is initially fit using the ever-active set, rather than the strong set).

The reason for this is that the strong rule, and every other screening rules we know of, ignores information about the gradient of the correlation vector $\boldsymbol{c}$, even though it contains useful information about the structure of the path. Looking at Figure 20, for instance, we see that the strong rule bound is indifferent to the slopes of the correlation vectors. This is the motivation for the *Hessian screening rule* that we introduce in the third paper of the thesis (Larsson and Wallin 2022). The name stems from the fact that we use second-order information about the optimization problem, which involves the Hessian matrix $\boldsymbol{X}^\intercal \boldsymbol{X}$. The rule offers a better estimate of the correlation vector (Figure 23), which in practice leads to better screening performance.

The screening method manages to be effective, particularly when g is the least-squares objective, because of efficient updates of the Hessian matrix and its inverse. And as a bonus, the availability of these quantities also allow for adaptively tailoring the λ sequence in order to better approximate the true lasso path.

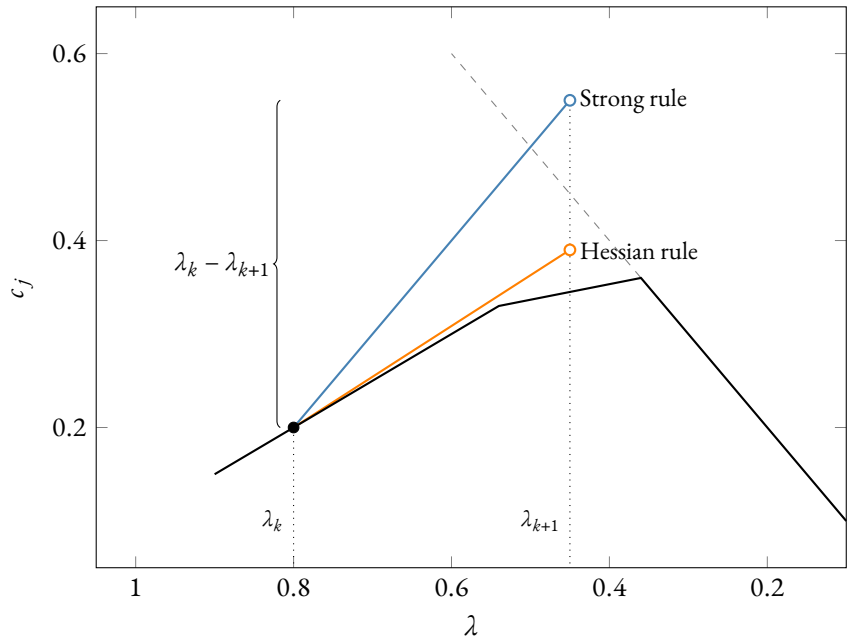


Figure 23: An illustration of the strong and Hessian screening rules for a lasso problem. We are at λ_k and are looking to screen features for λ_{k+1}. The black solid line shows the path of the correlation vector for the jth feature, which is inactive (its coefficient is zero) until the point where it joins the dashed line. In blue and orange colors, we show the predicted values for c_j between λ_k and λ_{k+1}. For the strong rule, the predicted value lies above the dashed line, so it is not discarded by the rule (even though it is inactive). For the Hessian rule, however, the prediction $\hat{c}_j$ *does* lie below the dashed line, and so the feature *is* discarded.

6.4 Paper IV

The few optimization methods that we have introduced so far amount to no more than a drop in the ocean of the vast volume of research that is the field of optimization. And although we have provided a few select benchmarks of our methods, we are still far removed from any kind of comprehensive study of their comparative effectiveness. This is not just an issue of our limited overview in this thesis, but in fact a general problem for research in optimization.

The problem is that there are now so many optimization methods to examine and so many different models and datasets on which to compare them on, that it has become difficult to keep track of which methods it is that actually do best on a given problem. You can easily find a paper A that studies optimization methods X and Y on datasets I and II and conclude that X is better than Y but then find another paper, B, which studies methods X, Y, and Z on datasets I and III and conclude that, actually, Y is better than X and by the way, Z happens to be best of them all. Then, later, you find paper C, which claims that Z actually is considerably worse than X and in fact also performs better for dataset IV. In addition to this confused state of affairs, it is commonly also the case that the authors of these papers 1) used different criteria for convergence, 2) programmed their methods in contrasting programming languages, and 3) benchmarked their experiments on different hardware.

In short, there is a dire need for a framework through which this process can be made simple, reproducible, and transparent. This is the motivation behind the `benchopt` package, which we present in the fifth of this thesis' papers (Moreau et al. 2022).

The goal of `benchopt` is to make life easier for both researchers in optimization and users of optimization software. For a researcher who has developed new optimization method for SLOPE, for instance, all you need to do is to write the optimization method for your algorithm and plug it into the existing `benchopt` benchmark for SLOPE and run it. The package will then automatically compare your method to all the other methods in the benchmark and output table and plots of the results (Figure 24). And if you instead are a user who is interested in using SLOPE for your applied work and want to know which algorithm to use, you can either browse the extensive database of results that other users have already uploaded or just download the benchmark and run it yourself on the data that you are interested in using it for.

6.5 Paper V

As we saw in Figure 18, proximal coordinate descent is an efficient optimization algorithm for fitting the lasso. But as we also noted, however, it cannot handle the case when the penalty term h is non-separable, which is the case in SLOPE. In practice, this has reduced the applicability of SLOPE to large data, which is unfortunate given the

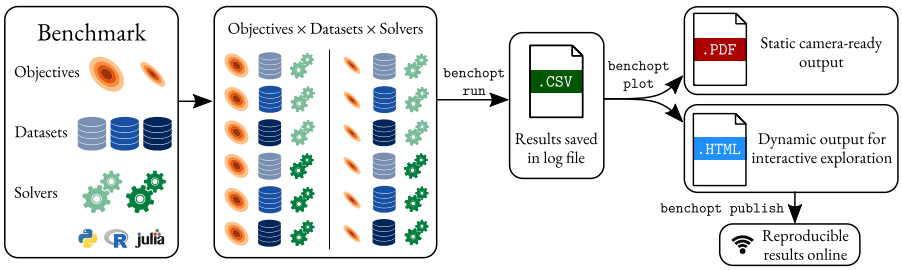


Figure 24: A schematic over how a benchmark is set up and run using `benchopt`. The benchmark consists of a set of files that define an objective, datasets, and solvers. When the user runs `benchopt run`, the package combines all of the possible combinations of objectives, datasets, and solvers and outputs a neatly formatted database of the results. Using `benchopt plot`, the user can then easily compare the different methods through interative visualizations or produce publication-ready plots to insert directly into a paper. Finally, to make the results available to the `benchopt` community, the user can run `benchopt publish`, which opens up a pull-request against the public repository of benchmark results.

many appealing properties of the model.

In paper v (Larsson, Klopfenstein, et al. 2023), however, we present a way to circumvent this issue by using a hybrid of proximal coordinate and proximal gradient descent (Figure 25). Our main discovery is that if we fix the clusters and optimize over each cluster in turn, rather than each feature, the problem becomes separable, which means that coordinate descent can be used. And if we combine this with proximal gradient descent steps, which allow us to discover the clusters, then we can guarantee convergence and at the same time benefit from the efficiency of coordinate descent.

6.6 Paper vi

In the final paper of this thesis, we tackle the issue of normalization of binary features, which we touched upon in Section 5. As we saw in that section, normalization is necessary in order to put the features on the "same scale". What this means, however, is not clear, yet has been met mostly with neglect in the literature. We think that this is both surprising and problematic given the almost universal use of normalization in regularized methods and the apparent and large effects it has on the solution paths (Figure 21).

In our paper, we begin to bridge this knowledge gap by studying normalization for the lasso and ridge regression when they are used on binary features (features that only

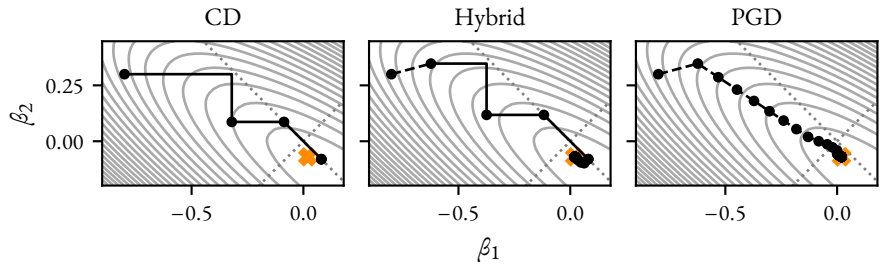


Figure 25: A illustration of the hybrid coordinate descent solver we developed for SLOPE.

contain values 0 or 1) or mixd of binary and normally distributed features. What we find is that there is a large effect of normalization with respect to the class balance of the features: the proportion of ones to zeros (or vice versa). Both the lasso and the ridge estimators turn out to be sensitive to this class balance and, depending on the type of normalization used, have trouble recovering effects that are associated with binary features as long as their class balance is severe enough (Figure 26).

References

Beck, A. and M. Teboulle (Jan. 1, 2009). "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems". In: *SIAM Journal on Imaging Sciences* 2.1, pp. 183–202. DOI: 10.1137/080716542.

Belloni, Alexandre, Victor Chernozhukov, and Lie Wang (Dec. 2011). "Square-Root Lasso: Pivotal Recovery of Sparse Signals via Conic Programming". In: *Biometrika* 98.4, pp. 791–806. ISSN: 0006-3444. DOI: 10.1093/biomet/asr043.

Bertrand, Quentin and Mathurin Massias (Mar. 18, 2021). "Anderson Acceleration of Coordinate Descent". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. AISTATS 2021. Vol. 130. San Diego, CA, USA: PMLR, pp. 1288–1296.

Bertsimas, Dimitris, Angela King, and Rahul Mazumder (Apr. 1, 2016). "Best Subset Selection via a Modern Optimization Lens". In: *The Annals of Statistics* 44.2, pp. 813–852. ISSN: 0090-5364. DOI: 10.1214/15-AOS1388.

Bogdan, Małgorzata, Ewout van den Berg, Chiara Sabatti, et al. (Sept. 2015). "SLOPE – Adaptive Variable Selection via Convex Optimization". In: *The annals of applied statistics* 9.3, pp. 1103–1140. ISSN: 1932-6157. DOI: 10.1214/15-AOAS842. pmid: 26709357.

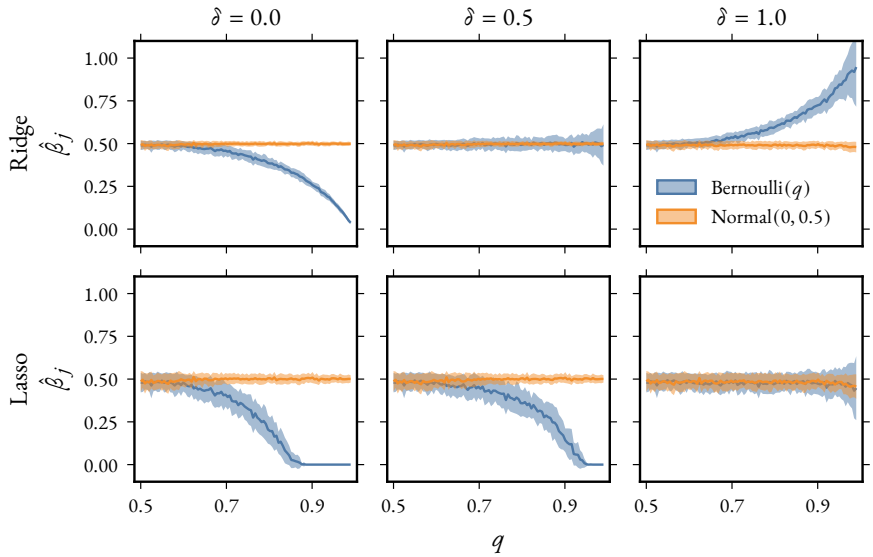


Figure 26: Estimated coefficients from the lasso and ridge for a two-feature problem where one of the features has a quasi-normal distribution (values deterministically set via the quantile function), with standard deviation 1/2, and the other is a binary (quasi-Bernoulli) feature with class-balance q. The normal feature is standardized in every case, whereas the binary feature is scaled with $(q - q^2)^\partial$—its variance to the power of ∂. In other words, we have no scaling for $\partial = 0$, standard deviation scaling when $\partial = 1/2$, and variance-scaling when $\partial = 1$.

Bogdan, Małgorzata, Ewout van den Berg, Weijie Su, et al. (Oct. 29, 2013). *Statistical Estimation and Testing via the Sorted L1 Norm*. DOI: 10.48550/arXiv.1310.1969. arXiv: 1310.1969 [math, stat]. URL: http://arxiv.org/abs/1310.1969 (visited on 04/16/2020). preprint.

Bogdan, Małgorzata, Xavier Dupuis, et al. (Mar. 22, 2022). *Pattern Recovery by SLOPE*. DOI: 10.48550/arXiv.2203.12086. arXiv: 2203.12086 [math, stat]. URL: http://arxiv.org/abs/2203.12086 (visited on 06/03/2022). preprint.

Bondell, Howard D. and Brian J. Reich (Mar. 2008). "Simultaneous Regression Shrinkage, Variable Selection, and Supervised Clustering of Predictors with OSCAR". In: *Biometrics* 64.1, pp. 115–123. ISSN: 0006-341X. DOI: 10.1111/j.1541-0420.2007.00843.x. JSTOR: 25502027.

Bottou, Léon (Sept. 30, 2010). "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of COMPSTAT'2010*. 19th International Conference on Computational Statistics. Ed. by Yves Lechevallier and Gilbert Saporta. Berlin, Germany: Physica-Verlag, pp. 177–186. ISBN: 978-3-7908-2604-3. DOI: 10.1007/978-3-7908-2604-3_16.

Boyd, Stephen et al. (2010). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends® in Machine Learning* 3.1, pp. 1–122. ISSN: 1935-8237. DOI: 10.1561/2200000016.

Daubechies, I., M. Defrise, and C. De Mol (Aug. 26, 2004). "An Iterative Thresholding Algorithm for Linear Inverse Problems with a Sparsity Constraint". In: *Communications on Pure and Applied Mathematics* 57.11, pp. 1413–1457. ISSN: 1097-0312. DOI: 10.1002/cpa.20042.

Davis, Damek (June 26, 2015). *An $O(N\log(n))$ Algorithm for Projecting onto the Ordered Weighted ℓ_1 Norm Ball*. DOI: 10.48550/arXiv.1505.00870. arXiv: 1505.00870 [cs, math]. URL: http://arxiv.org/abs/1505.00870 (visited on 04/23/2024). preprint.

Donoho, David L. and Iain M. Johnstone (Aug. 1994). "Ideal Spatial Adaptation by Wavelet Shrinkage". In: *Biometrika* 81.3, pp. 425–455. ISSN: 0006-3444. DOI: 10.2307/2337118. JSTOR: 2337118.

— (1995). "Adapting to Unknown Smoothness via Wavelet Shrinkage". In: *Journal of the American Statistical Association* 90.432, pp. 1200–1224. ISSN: 0162-1459. DOI: 10.2307/2291512. JSTOR: 2291512.

Duchi, John et al. (July 5–9, 2008). "Efficient Projections onto the L1-Ball for Learning in High Dimensions". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML 2008. Ed. by Andrew McCallum and Sam Roweis. Helsinki, Finland: Association for Computing Machinery, pp. 272–279. ISBN: 978-1-60558-205-4. DOI: 10.1145/1390156.1390191.

Dupuis, Xavier and Patrick J C Tardivel (Oct. 28, 2023). *The Solution Path of SLOPE*. HAL: hal-04100441v2. URL: https://hal.science/hal-04100441v2 (visited on 01/17/2024). preprint.

Efron, Bradley et al. (Apr. 2004). "Least Angle Regression". In: *Annals of Statistics* 32.2, pp. 407–499. ISSN: 0090-5364. DOI: 10.1214/009053604000000067.

El Ghaoui, Laurent, Vivian Viallon, and Tarek Rabbani (Sept. 21, 2010). *Safe Feature Elimination in Sparse Supervised Learning*. Technical report UCB/EECS-2010-126. Berkeley: EECS Department, University of California.

Fan, Jianqing and Runze Li (Dec. 1, 2001). "Variable Selection via Nonconcave Penalized Likelihood and Its Oracle Properties". In: *Journal of the American Statistical Association* 96.456, pp. 1348–1360. ISSN: 0162-1459. DOI: 10/fd7bfs.

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.

Figueiredo, Mário A. T. and Robert D. Nowak (Sept. 13, 2014). *Sparse Estimation with Strongly Correlated Variables Using Ordered Weighted L1 Regularization*. DOI: 10.48550/arXiv.1409.4005. arXiv: 1409.4005. URL: http://arxiv.org/abs/1409.4005 (visited on 06/03/2022). preprint.

Friedman, Jerome, Trevor Hastie, Holger Höfling, et al. (Dec. 2007). "Pathwise Coordinate Optimization". In: *The Annals of Applied Statistics* 1.2, pp. 302–332. ISSN: 1932-6157. DOI: 10/d88g8c.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (July 2008). "Sparse Inverse Covariance Estimation with the Graphical Lasso". In: *Biostatistics* 9.3, pp. 432–441. ISSN: 1465-4644. DOI: 10.1093/biostatistics/kxm045.

— (Jan. 2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1, pp. 1–22. DOI: 10.18637/jss.v033.i01.

Fu, Wenjiang J. (1998). "Penalized Regressions: The Bridge versus the Lasso". In: *Journal of Computational and Graphical Statistics* 7.3, pp. 397–416. ISSN: 1061-8600. DOI: 10.2307/1390712. JSTOR: 1390712.

Golub, T. R. et al. (Oct. 15, 1999). "Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring". In: *Science* 286.5439, pp. 531–537. ISSN: 0036-8075. DOI: 10.1126/science.286.5439.531. pmid: 10521349.

Guyon, Isabelle et al. (Dec. 13–18, 2004). "Result Analysis of the NIPS 2003 Feature Selection Challenge". In: *Advances in Neural Information Processing Systems 17*. Neural Information Processing Systems 2004. Ed. by Lawrence K. Saul, Yair Weiss,

and Léon Bottou. Vancouver, BC, Canada: MIT Press, pp. 545–552. ISBN: 978-0-262-19534-8.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. 2nd ed. Springer Series in Statistics. New York: Springer-Verlag. ISBN: 978-0-387-84857-0.

Hastie, Trevor, Robert Tibshirani, and Ryan Tibshirani (Nov. 2020). "Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons". In: *Statistical Science* 35.4, pp. 579–592. ISSN: 0883-4237. DOI: 10.1214/19-STS733.

Keerthi, S. Sathiya and Dennis DeCoste (Dec. 1, 2005). "A Modified Finite Newton Method for Fast Solution of Large Scale Linear SVMs". In: *The Journal of Machine Learning Research* 6.12, pp. 341–361. ISSN: 1532-4435.

Kim, Seung-Jean et al. (Dec. 2007). "An Interior-Point Method for Large-Scale ℓ_1-Regularized Least Squares". In: *IEEE Journal of Selected Topics in Signal Processing* 1.4, pp. 606–617. ISSN: 1932-4553, 1941-0484. DOI: 10.1109/JSTSP.2007.910971.

King, Ross (2024). *Qualitative Structure Activity Relationships*. UCI Machine Learning Repository. DOI: 10.24432/C5TP54.

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.

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.

Lawson, Charles L. and Richard J. Hanson (1995). *Solving Least Squares Problems*. 2nd ed. Classics in Applied Mathematics. Philadelphia, PA, USA: Society for In-

dustrial and Applied Mathematics. 351 pp. ISBN: 978-0-89871-356-5. DOI: 10.1137/1.9781611971217.

Lee, Jason D., Yuekai Sun, and Michael A. Saunders (Jan. 1, 2014). "Proximal Newton-type Methods for Minimizing Composite Functions". In: *SIAM Journal on Optimization* 24.3, pp. 1420–1443. ISSN: 1052-6234. DOI: 10.1137/130921428.

Lewis, David D. et al. (Dec. 1, 2004). "RCV1: A New Benchmark Collection for Text Categorization Research". In: *The Journal of Machine Learning Research* 5, pp. 361–397. ISSN: 1532-4435.

Li, Qinzhen and Xudong Li (July 29, 2021). "Fast Projection onto the Ordered Weighted L1 Norm Ball". In: *Science China Mathematics* 65.4, pp. 869–886. ISSN: 1869-1862. DOI: 10.1007/s11425-020-1743-9.

Mai, Vien V. and Mikael Johansson (July 13–18, 2020). "Anderson Acceleration of Proximal Gradient Methods". In: *Proceedings of the 37th International Conference on Machine Learning*. ICML 20. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Virtual: PMLR, pp. 6620–6629.

Mairal, Julien and Bin Yu (June 2012). "Complexity Analysis of the Lasso Regularization Path". In: *Proceedings of the 29th International Conference on Machine Learning*. International Conference on Machine Learning 2012. Edinburgh, United Kingdom, pp. 1835–1842.

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.

Ndiaye, Eugene et al. (2017). "Gap Safe Screening Rules for Sparsity Enforcing Penalties". In: *Journal of Machine Learning Research* 18.128, pp. 1–33. ISSN: 1533-7928.

Nesterov, Yuri (1983). "A Method of Solving a Convex Programming Problem with Convergence Rate O(1/K^2)". In: *Doklady Akademii Nauk SSSR* 269.3, pp. 543–547.

Nomura, Shunichi (Oct. 29, 2020). *An Exact Solution Path Algorithm for SLOPE and Quasi-Spherical OSCAR*. DOI: 10.48550/arXiv.2010.15511. arXiv: 2010.15511. URL: http://arxiv.org/abs/2010.15511 (visited on 05/27/2021). preprint.

Osborne, Michael R., Brett Presnell, and Berwin A. Turlach (July 1, 2000a). "A New Approach to Variable Selection in Least Squares Problems". In: *IMA Journal of Numerical Analysis* 20.3, pp. 389–403. ISSN: 1464-3642. DOI: 10.1093/imanum/20.3.389.

— (2000b). "On the LASSO and Its Dual". In: *Journal of Computational and Graphical Statistics* 9.2, pp. 319–337. ISSN: 1061-8600. DOI: 10.2307/1390657. JSTOR: 1390657.

Perez, Guillaume et al. (May 2022). "Efficient Projection Algorithms onto the Weighted Li Ball". In: *Artificial Intelligence* 306, pp. 1–14. ISSN: 00043702. DOI: 10.1016/j.artint.2022.103683.

Platt, John C. (Jan. 1998). "Fast Training of Support Vector Machines Using Sequential Minimal Optimization". In: *Advances in Kernel Methods: Support Vector Learning*. Ed. by Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola. 1st ed. Boston, MA, USA: MIT Press, pp. 185–208. ISBN: 978-0-262-28319-9. DOI: 10.7551/mitpress/1130.003.0016.

R Core Team (Feb. 2024). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Robbins, Herbert and Sutton Monro (Sept. 1951). "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3, pp. 400–407. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729586.

Rockafellar, R. Tyrrell (1970). *Convex Analysis*. Princeton Landmarks in Mathematics and Physics. Princeton University Press. 472 pp. ISBN: 978-0-691-01586-6. JSTOR: j.ctt14bs1ff.

Santosa, Fadil and William W. Symes (Oct. 1986). "Linear Inversion of Band-Limited Reflection Seismograms". In: *SIAM Journal on Scientific and Statistical Computing* 7.4, pp. 1307–1330. ISSN: 0196-5204. DOI: 10.1137/0907087.

Schneider, Ulrike and Patrick Tardivel (Oct. 1, 2022). "The Geometry of Uniqueness, Sparsity and Clustering in Penalized Estimation". In: *The Journal of Machine Learning Research* 23.331, pp. 1–36. ISSN: 1532-4435.

Shevade, S. K. and S. S. Keerthi (Nov. 22, 2003). "A Simple and Efficient Algorithm for Gene Selection Using Sparse Logistic Regression". In: *Bioinformatics* 19.17, pp. 2246–2253. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg308. pmid: 14630653.

Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society: Series B* 58.1, pp. 267–288. ISSN: 0035-9246. DOI: 10.1111/j.2517-6161.1996.tb02080.x. JSTOR: 2346178.

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

Tibshirani, Robert, Michael Saunders, et al. (Feb. 2005). "Sparsity and Smoothness via the Fused Lasso". In: *Journal of the Royal Statistical Society: Series B* 67.1, pp. 91–108. ISSN: 1467-9868. DOI: 10.1111/j.1467-9868.2005.00490.x.

Wright, Stephen (Mar. 25, 2015). "Coordinate Descent Algorithms". In: *Mathematical Programming: Series B* 151.1, pp. 3–34. ISSN: 00255610. DOI: 10.1007/s10107-015-0892-3.

Xiang, Zhen J., Hao Xu, and Peter J Ramadge (Dec. 12–17, 2011). "Learning Sparse Representations of High Dimensional Data on Large Scale Dictionaries". In: *Advances in Neural Information Processing Systems 24*. Neural Information Processing Systems 2011. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., pp. 900–908.

Xiang, Zhen James and Peter J. Ramadge (Mar. 2012). "Fast Lasso Screening Tests Based on Correlations". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2137–2140. DOI: 10.1109/ICASSP.2012.6288334.

Yuan, Ming and Yi Lin (Dec. 21, 2005). "Model Selection and Estimation in Regression with Grouped Variables". In: *Journal of the Royal Statistical Society: Series B* 68.1, pp. 49–67. ISSN: 1467-9868. DOI: 10.1111/j.1467-9868.2005.00532.x.

Zeng, Xiangrong and Mário A. T. Figueiredo (Oct. 2014). "Decreasing Weighted Sorted L1 Regularization". In: *IEEE Signal Processing Letters* 21.10, pp. 1240–1244. ISSN: 1070-9908, 1558-2361. DOI: 10.1109/LSP.2014.2331977.

— (Apr. 10, 2015). *The Ordered Weighted L1 Norm: Atomic Formulation, Projections, and Algorithms*. DOI: 10.48550/arXiv.1409.4271. arXiv: 1409.4271. URL: http://arxiv.org/abs/1409.4271 (visited on 11/22/2019). preprint.

Zhang, Cun-Hui (Apr. 2010). "Nearly Unbiased Variable Selection under Minimax Concave Penalty". In: *The Annals of Statistics* 38.2, pp. 894–942. ISSN: 0090-5364. DOI: 10/bp22zz.

Zou, Hui (Dec. 1, 2006). "The Adaptive Lasso and Its Oracle Properties". In: *Journal of the American Statistical Association* 101.476, pp. 1418–1429. ISSN: 0162-1459. DOI: 10.1198/016214506000000735.

Zou, Hui and Trevor Hastie (2005). "Regularization and Variable Selection via the Elastic Net". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67.2, pp. 301–320. ISSN: 1369-7412.

Papers

I

The Strong Screening Rule for SLOPE

Johan Larsson
Dept. of Statistics, Lund University
johan.larsson@stat.lu.se

Małgorzata Bogdan
Dept. of Mathematics, University of Wroclaw
Dept. of Statistics, Lund University
malgorzata.bogdan@uwr.edu.pl

Jonas Wallin
Dept. of Statistics, Lund University
jonas.wallin@stat.lu.se

Abstract

Extracting relevant features from data sets where the number of observations (n) is much smaller then the number of predictors (p) is a major challenge in modern statistics. Sorted L-One Penalized Estimation (SLOPE)—a generalization of the lasso—is a promising method within this setting. Current numerical procedures for SLOPE, however, lack the efficiency that respective tools for the lasso enjoy, particularly in the context of estimating a complete regularization path. A key component in the efficiency of the lasso is predictor screening rules: rules that allow predictors to be discarded before estimating the model. This is the first paper to establish such a rule for SLOPE. We develop a screening rule for SLOPE by examining its subdifferential and show that this rule is a generalization of the strong rule for the lasso. Our rule is heuristic, which means that it may discard predictors erroneously. In our paper, however, we show that such situations are rare and easily safeguarded against by a simple check of the optimality conditions. Our numerical experiments show that the rule performs well in practice, leading to improvements by orders of magnitude for data in the $p \gg n$ domain, as well as incurring no additional computational overhead when $n > p$.

1 Introduction

Extracting relevant features from data sets where the number of observations (n) is much smaller then the number of predictors (p) is one of the major challenges in modern statistics. The dominating method for this problem, in a regression setting, is the lasso [1]. Recently, however, an alternative known as Sorted L-One Penalized Estimation (SLOPE) has been proposed [2–4], which is a generalization of the Octagonal Shrinkage and Supervised Clustering Algorithm for Regression (OSCAR) [5].

SLOPE is a regularization method that uses the sorted ℓ_1 norm instead of the regular ℓ_1 norm, which is used in the lasso. SLOPE features several interesting properties, such as control of the false discovery rate [2, 6], asymptotic minimaxity [7], and clustering of regression coefficients in the presence of strong dependence between predictors [8].

In more detail, SLOPE solves the convex optimization problem

$$\text{minimize}_{\beta \in \mathbb{R}^p} \left\{ f(\beta) + J(\beta; \lambda) \right\}, \tag{1}$$

where $f(\beta)$ is smooth and convex and $J(\beta; \lambda) = \sum_{j=1}^{p} \lambda_j |\beta|_{(j)}$ is the convex but non-smooth sorted ℓ_1 norm [2, 8], where $|\beta|_{(1)} \geq |\beta|_{(2)} \geq \cdots \geq |\beta|_{(p)}$ and $\lambda_1 \geq \cdots \geq \lambda_p \geq 0$.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

It is easy to see that the lasso is a specific instance of SLOPE, obtained by setting all elements of λ to the same value. But in contrast to SLOPE based on a decreasing sequence λ, the lasso suffers from unpredictable behavior in the presence of highly correlated predictors, occasionally resulting in solutions wherein only a subset among a group of correlated predictors is selected. SLOPE, in contrast, turns out to be robust to correlated designs, which it accomplishes via clustering: setting coefficients of predictors to the same value [8]. Kremer et al. [9] showed that this clustering is related to similarity of the influence of respective variables on the likelihood function, which can occur due to strong correlation [8, 10] but also due to similarity of true regression coefficients [11]. This property in some cases allows SLOPE to select all p predictors if they are grouped into no more than n clusters [9, 11], while the lasso can at most select n predictors [12].

The choice of λ sequence in (1) typically needs to be based on cross-validation or similar schemes. Most algorithms for fitting sparse regression, such as as the one implemented for lasso in the `glmnet` package for R [13], accomplish this by constructing a path of decreasing λ. For SLOPE, we begin the path with $\lambda^{(1)}$ and finish at $\lambda^{(l)}$ with $\lambda_j^{(m)} \geq \lambda_j^{(m+1)}$ for $j = 1, 2, \dots, p$ and $m = 1, 2, \dots, l-1$. (See Section 3.1 for details regarding the construction of the path.) For any point along this path, we let $\hat{\beta}(\lambda^{(m)})$ be the respective SLOPE estimate, such that

$$\hat{\beta}(\lambda^{(m)}) = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left\{ f(\beta) + J(\beta; \lambda^{(m)}) \right\}.$$

Fitting the path repeatedly by cross-validation introduces a heavy computational burden. For the lasso, an important remedy for this issue arose with the advent of screening rules, which provide criteria for discarding predictors before fitting the model.

Screening rules can be broken down into two categories: *safe* and *heuristic* (unsafe) screening rules. The former of these guarantee that any predictors screened as inactive (determined to be zero by the rule) are in fact zero at the solution. Heuristic rules, on the other hand, may lead to *violations:* incorrectly discarding predictors, which means that heuristic rules must be supplemented with a check of the Karush–Kuhn–Tucker (KKT) conditions. For any predictors failing the test, the model must be refit with these predictors added back in order to ensure optimality.

Safe screening rules include the safe feature elimination rule (SAFE [14]), the dome test [15], Enhanced Dual-Polytope Projection (EDPP [16]), and the Gap Safe rule [17, 18]. Heuristic rules include Sure Independence Screening (SIS [19]), Blitz [20], and the strong rule [21]. There have also been attempts to design *dynamic* approaches to screening [22] as well as hybrid rules, utilizing both heuristic and safe rules in tandem [23].

The implications of screening rules have been remarkable, allowing lasso models in the $p \gg n$ domain to be solved in a fraction of the time required otherwise and with a much reduced memory footprint [21]. Implementing screening rules for SLOPE has, however, proven to be considerably more challenging. After the first version of this paper appeared on *arXiv* [24], a first safe rule for SLOPE has been published [25]. Yet, because of the non-separability of the penalty in SLOPE, this rule requires iterative screening during optimization, which means that predictors cannot be screened prior to fitting the model. This highlights the difficulty in developing screening rules for SLOPE.

Our main contribution in this paper is the presentation of a first heuristic screening rule for SLOPE based on the strong rule for the lasso. In doing so, we also introduce a novel formulation of the subdifferential for the sorted ℓ_1 norm. We then proceed to show that this rule is effective, rarely leads to violations, and offers performance gains comparable to the strong rule for the lasso.

1.1 Notation

We use uppercase letters for matrices and lowercase letters for vectors and scalars. $\mathbf{1}$ and $\mathbf{0}$ denote vectors with all elements equal to 1 and 0 respectively, with dimension inferred from context. We use $\prec$ and $\succ$ to denote element-wise relational operators. We also let $\operatorname{card} \mathcal{A}$ denote the cardinality of set $\mathcal{A}$ and define $\operatorname{sign} x$ to be the signum function with range $\{-1, 0, 1\}$. Furthermore, we define $x_\downarrow$ to refer to a version of x sorted in decreasing order and the cumulative sum function for a vector $x \in \mathbb{R}^n$ as $\operatorname{cumsum}(x) = [x_1, x_1 + x_2, \cdots, \sum_{i=1}^n x_i]^T$. We also let $|i|$ be the index operator of $y \in \mathbb{R}^p$ so that $|y_{|i|}| = |y|_{(i)}$ for all $i = 1, \dots, p$. Finally, we allow a vector to be indexed with an integer-valued set by eliminating those elements of this vector whose indices do not belong to the indexing set—for instance, if $\mathcal{A} = \{3, 1\}$ and $v = [v_1, v_2, v_3]^T$, then $v_\mathcal{A} = [v_1, v_3]^T$.

2 Theory

Proofs of the following theorem and propositions are provided in the supplementary material.

2.1 The Subdifferential for SLOPE

The basis of the strong rule for ℓ_1-regularized models is the subdifferential. By the same argument, we now turn to the subdifferential of SLOPE. The subdifferential for SLOPE has been derived previously as a characterization based on polytope mappings [26, 27]; here we present an alternative formulation that can be used as the basis of an efficient algorithm. First, however, let $\mathcal{A}_i(\beta) \subseteq \{1, \dots, p\}$ denote a set of indices for $\beta \in \mathbb{R}^p$ such that

$$\mathcal{A}_i(\beta) = \{j \in \{1, \dots, p\} \mid |\beta_i| = |\beta_j|\} \tag{2}$$

where $\mathcal{A}_i(\beta) \cap \mathcal{A}_l(\beta) = \varnothing$ if $l \notin \mathcal{A}_i(\beta)$. To keep notation concise, we let $\mathcal{A}_i$ serve as a shorthand for $\mathcal{A}_i(\beta)$. In addition, we define the operator $O : \mathbb{R}^p \to \mathbb{N}^p$, which returns a permutation that rearranges its argument in descending order by its absolute values and $R : \mathbb{R}^p \to \mathbb{N}^p$, which returns the ranks of the absolute values in its argument.

Example 1. *If $\beta = \{-3, 5, 3, 6\}$, then $\mathcal{A}_1 = \{1, 3\}$, $O(\beta) = \{4, 2, 1, 3\}$, and $R(\beta) = \{3, 2, 4, 1\}$.*

Theorem 1. *The subdifferential $\partial J(\beta; \lambda) \in \mathbb{R}^p$ is the set of all $g \in \mathbb{R}^p$ such that*

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

2.2 Screening Rule for SLOPE

2.2.1 Sparsity Pattern

Recall that we are attempting to solve the following problem: we know $\hat{\beta}(\lambda^{(m)})$ and want to predict the support of $\hat{\beta}(\lambda^{(m+1)})$, where $\lambda^{(m+1)} \preceq \lambda^{(m)}$. The KKT stationarity criterion for SLOPE is

$$\mathbf{0} \in \nabla f(\beta) + \partial J(\beta; \lambda), \tag{3}$$

where $\partial J(\beta; \lambda)$ is the subdifferential for SLOPE (Theorem 1). This means that if $\nabla f(\hat{\beta}(\lambda^{(m+1)}))$ was available to us, we could identify the support exactly. In Algorithm 1, we present an algorithm to accomplish this in practice.

Algorithm 1

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

$\mathcal{S}, \mathcal{B} \leftarrow \varnothing$
for $i \leftarrow 1, \dots, p$ **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 if
end for
return $\mathcal{S}$

Algorithm 2 Fast version of Algorithm 1.

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

$i \leftarrow 1, k \leftarrow 0, s \leftarrow 0$
while $i + k \leq p$ **do**
  $s \leftarrow s + c_{i+k} - \lambda_{i+k}$
  if $s \geq 0$ **then**
    $k \leftarrow k + i$
    $i \leftarrow 1$
    $s \leftarrow 0$
  else
    $i \leftarrow i + 1$
  end if
end while
return k

In Proposition 1, we show that the result of Algorithm 1 with $c \coloneqq |\nabla f(\hat{\beta}(\lambda^{(m+1)}))|_\downarrow$ and $\lambda \coloneqq \lambda^{(m+1)}$ as input is certified to contain the true support set of $\hat{\beta}(\lambda^{(m+1)})$.

Proposition 1. *Taking $c \coloneqq |\nabla f(\hat{\beta}(\lambda^{(m+1)}))|_\downarrow$ and $\lambda \coloneqq \lambda^{(m+1)}$ as input to Algorithm 1 returns a superset of the true support set of $\hat{\beta}(\lambda^{(m+1)})$.*

Remark 1. *In Algorithm 1, we implicitly make use of the fact that the results are invariant to permutation changes within each cluster $\mathcal{A}_i$ (as defined in (2))—a fact that follows directly from the definition of the subdifferential (Theorem 1). In particular, this means that the indices for the set of inactive predictors will be ordered last in both $|\hat{\beta}|_\downarrow$ and $|\nabla f(\hat{\beta})|_\downarrow$; that is, for all $i, j \in \{1, 2, \dots, p\}$ such that $\hat{\beta}_i = 0$, $\hat{\beta}_j \neq 0$,*

$$O(\nabla f(\hat{\beta}))_i > O(\nabla f(\hat{\beta}))_j \implies O(\hat{\beta})_i > O(\hat{\beta})_j,$$

which allows us to determine the sparsity in $\hat{\beta}$ via $\nabla f(\hat{\beta})$.

Proposition 1 implies that Algorithm 1 may lead to a conservative decision by potentially including some of the support of inactive predictors in the result, i.e. indices for which the corresponding coefficients are in fact zero. To see this, let $\mathcal{U} = \{l, l+1, \dots, p\}$ be a set of inactive predictors and take $c \coloneqq |\nabla f(\hat{\beta}(\lambda^{(m+1)}))|_\downarrow$. For every $k \in \mathcal{U}$, $k \geq l$ for which $\sum_{i=l}^{k}(c_i - \lambda_i) = 0$, $\{l, l+1, \dots, k\}$ will be in the result of Algorithm 1 in spite of being inactive. This situation, however, occurs only when c is the true gradient at the solution and for this reason is of little practical importance.

Since the check in Algorithm 1 hinges only on the last element of the cumulative sum at any given time, we need only to store and update a single scalar instead of the full cumulative sum vector. Using this fact, we can derive a fast version of the rule (Algorithm 2), which returns k: the predicted number of active predictors at the solution.[1]

Since we only have to take a single pass over the predictors, the cost of the algorithm is linear in p. To use the algorithm in practice, however, we first need to compute the gradient at the previous solution and sort it. Using least squares regression as an example, this results in a complexity of $\mathcal{O}(np + p \log p)$. To put this into perspective, this is (slightly) lower than the cost of a single gradient step if a first-order method is used to compute the SLOPE solution (since it also requires evaluation of the proximal operator).

2.2.2 Gradient Approximation

The validity of Algorithm 1 requires $\nabla f(\hat{\beta}(\lambda^{(m+1)}))$ to be available, which of course is not the case. Assume, however, that we are given a reasonably accurate surrogate of the gradient vector and suppose that we substitute this estimate for $\nabla f(\hat{\beta}(\lambda^{(m+1)}))$ in Algorithm 1. Intuitively, this should yield us an estimate of the active set—the better the approximation, the more closely this screened set should resemble the active set. For the sequel, let $\mathcal{S}$ and $\mathcal{T}$ be the screened and active set respectively.

An obvious consequence of using our approximation is that we run the risk of picking $\mathcal{S} \not\supseteq \mathcal{T}$, which we then naturally must safeguard against. Fortunately, doing so requires only a KKT stationarity check—whenever the check fails, we relax $\mathcal{S}$ and refit. If such failures are rare, it is not hard to imagine that the benefits of tackling the reduced problem might outweigh the costs of these occasional failures.

Based on this argument, we are now ready to state the strong rule for SLOPE, which is a natural extension of the strong rule for the lasso [21]. Let $\mathcal{S}$ be the output from running Algorithm 1 with

$$c \coloneqq \left(|\nabla f(\hat{\beta}(\lambda^{(m)}))| + \lambda^{(m)} - \lambda^{(m+1)}\right)_\downarrow, \qquad \lambda \coloneqq \lambda^{(m+1)}$$

as input. The strong rule for SLOPE then discards all predictors corresponding to $\mathcal{S}^\mathsf{c}$.

Proposition 2. *Let $c_j(\lambda) = (\nabla f(\hat{\beta}(\lambda)))_{|j|}$. If $|c_j'(\lambda)| \leq 1$ for all $j = 1, 2, \dots, p$ and $O(c(\lambda^{(m+1)})) = O(c(\lambda^{(m)}))$ (see Section 2.1 for the definition of O), the strong rule for SLOPE returns a superset of the true support set.*

Except for the assumption on fixed ordering permutation, the proof for Proposition 2 is comparable to the proof of the strong rule for the lasso [21]. The bound appearing in the proposition, $|c_j'(\lambda)| \leq 1$, is referred to as the *unit slope bound*, which results in the following rule for the lasso: discard the jth predictor if

$$\left|\nabla f(\beta(\lambda^{(m)}))_j\right| \leq 2\lambda_j^{(m+1)} - \lambda_j^{(m)}.$$

In Proposition 3, we formalize the connection between the strong rule for SLOPE and lasso.

[1] The active set is then retrieved by sub-setting the first k elements of the ordering permutation.

Proposition 3. *The strong rule for SLOPE is a generalization of the strong rule for the lasso; that is, when $\lambda_j = \lambda_i$ for all $i, j \in \{1, \dots, p\}$, the two rules always produce the same screened set.*

Finally, note that a non-sequential (basic) version of this rule is obtained by simply using the gradient for the null model as the basis for the approximation together with the penalty sequence corresponding to the point at which the first predictor enters the model (see Section 3.1).

2.2.3 Violations of the Rule

Violations of the strong rule for SLOPE occur only when the unit slope bound fails, which may happen for both inactive and active predictors—in the latter case, this can occur when the clustering or the ordering permutation changes for these predictors. This means that the conditions under which violations may arise for the strong rule for SLOPE differ from those corresponding to the strong rule for the lasso [21].

To safeguard against violations, we check the KKT conditions after each fit and add violating predictors to the screened set, refit, and repeat the checks until there are no violations. In Section 3.2.2, we will study the prevalence of violations in simulated experiments.

2.2.4 Algorithms

Tibshirani et al. [21] considered two algorithms using the strong rule for the lasso. In this paper, we consider two algorithms that are analogous except in one regard. First, however, let $\mathcal{S}(\lambda)$ be the strong set, i.e. the set obtained by application of the strong rule for SLOPE, and $\mathcal{T}(\lambda)$ the active set. Both algorithms begin with a set $\mathcal{E}$ of predictors, fit the model to this set, and then either expand this set, refit and repeat, or stop.

In the *strong set* algorithm (see supplementary material for details) we initialize $\mathcal{E}$ with the union of the strong set and the set of predictors active at the previous step on the regularization path. We then fit the model and check for KKT violations in the full set of predictors, expanding $\mathcal{E}$ to include any predictors for which violations occur and repeat until there are no violations.

In the *previous set* algorithm (see supplementary material for details) we initialize $\mathcal{E}$ with only the set of previously active predictors, fit, and check the KKT conditions against the strong rule set. If there are violations in the strong set, the corresponding predictors are added to $\mathcal{E}$ and the model is refit. Only when there are no violations in the strong set do we check the KKT conditions in the full set. This procedure is repeated until there are no violations in the full set.

These two algorithms differ from the strong and working set algorithms from Tibshirani et al. [21] in that we use only the set of previously active predictors rather than the set of predictors that have been active at any previous step on the path.

3 Experiments

In this section we present simulations that examine the effects of applying the screening rules. The problems here reflect our focus on problems in the $p \gg n$ domain, but we will also briefly consider the reverse in order to examine the potential overhead of the rules when $n > p$.

3.1 Setup

Unless stated otherwise, we will use the strong set algorithm with the strong set computed using Algorithm 2. Unless stated otherwise, we normalize the predictors such that $\bar{x}_j = 0$ and $\|x_j\|_2 = 1$ for $j = 1, \dots, p$. In addition, we center the response vector such that $\bar{y} = 0$ when $f(\beta)$ is the least squares objective.

We use the *Benjamini–Hochberg* (BH) method [3] for computing the sequence, which sets $\lambda_i^{\text{BH}} = \Phi^{-1}\big(1 - qi/(2p)\big)$ for $i = 1, 2, \dots, p$, where Φ^{-1} is the probit function.[2] To construct the regularization path, we parameterize the sorted ℓ_1 penalty as $J(\beta; \lambda, \sigma) = \sigma \sum_{j=1}^{p} |\beta|_{(j)} \lambda_j$, with

[2] Bogdan et al. [3] also presented a method called the *Gaussian* sequence that is a modification of the BH method, but it is not appropriate for our problems since it reduces to the lasso in the $p \gg n$ context.

$\sigma^{(1)} > \sigma^{(2)} > \cdots > \sigma^{(l)} > 0$. We pick $\sigma^{(1)}$ corresponding to the point at which the first predictor enters the model, which corresponds to maximizing $\sigma \in \mathbb{R}$ subject to $\operatorname{cumsum}(\nabla f(\mathbf{0})_\downarrow - \sigma\lambda) \preceq 0$, which is given explicitly as

$$\sigma^{(1)} = \max(\operatorname{cumsum}(\nabla f(\mathbf{0})_\downarrow) \oslash \operatorname{cumsum}(\lambda)),$$

where $\oslash$ is the Hadamard (element-wise) division operator. We choose $\sigma^{(l)}$ to be $t\sigma^{(1)}$ with $t = 10^{-2}$ if $n < p$ and 10^{-4} otherwise. Unless stated otherwise, we employ a regularization path of $l = 100$ λ sequences but stop this path prematurely if 1) the number of unique coefficient magnitudes exceed the number of observations, 2) the fractional change in deviance from one step to another is less than 10^{-5}, or 3) if the fraction of deviance explained exceeds 0.995.

Throughout the paper we use version 0.2.1 of the R package `SLOPE` [28], which uses the accelerated proximal gradient algorithm FISTA [29] to estimate all models; convergence is obtained when the duality gap as a fraction of the primal and the relative level of infeasibility [30] are lower than 10^{-5} and 10^{-3} respectively. All simulations were run on a dedicated high-performance computing cluster and the code for the simulations is available in the supplementary material and at `https://github.com/jolars/slope-screening-code/`.

3.2 Simulated Data

Let $X \in \mathbb{R}^{n\times p}$, $\beta \in \mathbb{R}^{p\times m}$, and $y \in \mathbb{R}^n$. We take

$$y_i = x_i^T\beta + \varepsilon_i, \qquad i = 1, 2, \ldots, n,$$

where ε_i are sampled from independently and identically distributed standard normal variables. X is generated such that each row is sampled independently and identically from a multivariate normal distribution $\mathcal{N}(\mathbf{0}, \Sigma)$. From here on out, we also let k denote the cardinality of the non-zero support set of the true coefficients, that is, $k = \operatorname{card}\{i \in \mathbb{N}^p \mid \beta_i \neq 0\}$.

3.2.1 Efficiency

We begin by studying the efficiency of the strong rule for SLOPE on problems with varying levels of correlation ρ. Here, we let $n = 200$, $p = 5000$, and $\Sigma_{ij} = 1$ if $i = j$ and ρ otherwise. We take $k = p/4$ and generate β_i for $i = 1, \ldots, k$ from $\mathcal{N}(0, 1)$. We then fit a least squares regression model regularized with the sorted ℓ_1 norm to this data and screen the predictors with the strong rule for SLOPE. Here we set $q = 0.005$ in the construction of the BH sequence.

The size of the screened set is clearly small next to the full set (Figure 1). Note, however, that the presence of strong correlation among the predictors both means that there is less to be gained by screening since many more predictors are active at the start of the path, as well as makes the rule more conserative. No violations of the rule were observed in these simulations.

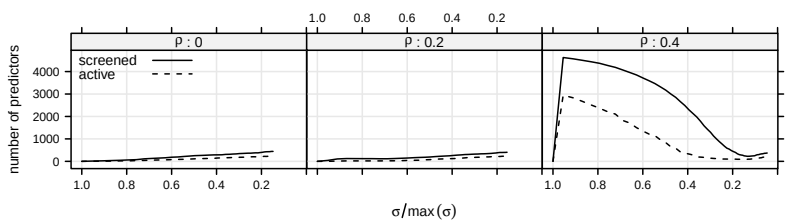


Figure 1: Number of screened and active predictors for sorted ℓ_1-regularized least squares regression using no screening or the strong rule for SLOPE.

3.2.2 Violations

To examine the number of violations of the rule, we generate a number of data sets with $n = 100$, $p \in \{20, 50, 100, 500, 1000\}$, and $\rho = 0.5$. We then fit a full path of 100 λ sequences across 100 iterations, averaging the results. (Here we disable the rules for prematurely aborting the path described

at the start of this section.) We sample the first fourth of the elements of β from $\{-2, 2\}$ and set the rest to zero.

Violations appear to be rare in this setting and occur only for the lower range of p values (Figure 2). For $p = 100$, for instance, we would at an average need to estimate roughly 100 paths for this type of design to encounter a single violation. Given that a complete path consists of 100 steps and that the warm start after the violation is likely a good initialization, this can be considered a marginal cost.

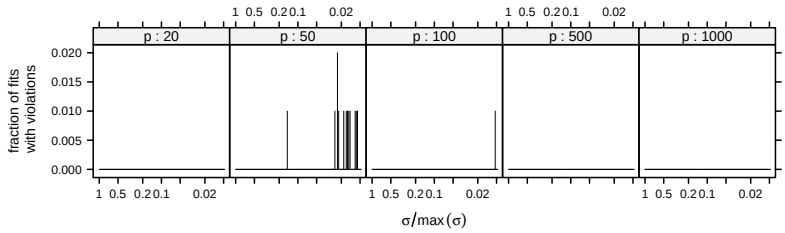


Figure 2: Fraction of model fits resulting in violations of the strong rule for sorted ℓ_1-regularized least squares regression.

3.2.3 Performance

In this section, we study the performance of the screening rule for sorted ℓ_1-penalized least squares, logistic, multinomial, and Poisson regression.

We now take $p = 20,000$, $n = 200$, and $k = 20$. To construct X, we let $X_1, X_2, \dots, X_p$ be random variables distributed according to

$$X_1 \sim \mathcal{N}(\mathbf{0}, I), \qquad X_j \sim \mathcal{N}(\rho X_{j-1}, I) \quad \text{for } j = 2, 3, \dots, p,$$

and sample the jth column in X from X_j for $j = 1, 2, \dots, p$.

For least squares and logistic regression data we sample the first $k = 20$ elements of β without replacement from $\{1, 2, \dots, 20\}$. Then we let $y = X\beta + \varepsilon$ for least squares regression and $y = \operatorname{sign}(X\beta + \varepsilon)$ for logistic regression, in both cases taking $\varepsilon \sim \mathcal{N}(\mathbf{0}, 20I)$. For Poisson regression, we generate β by taking random samples without replacement from $\{\frac{1}{40}, \frac{2}{40}, \dots, \frac{20}{40}\}$ for its first 20 elements. Then we sample y_i from $\operatorname{Poisson}\left(\exp((X\beta)_i)\right)$ for $i = 1, 2, \dots, n$. For multinomial regression, we start by taking $\beta \in \mathbb{R}^{p \times 3}$, initializing all elements to zero. Then, for each row in β we take a random sample from $\{1, 2, \dots, 20\}$ without replacement and insert it at random into one of the elements of that row. Then we sample y_i randomly from $\operatorname{Categorical}(3, p_i)$ for $i = 1, 2, \dots, n$, where

$$p_{i,l} = \frac{\exp\left((X\beta)_{i,l}\right)}{\sum_{l=1}^{3} \exp\left((X\beta)_{i,l}\right)}.$$

The benchmarks reveal a strong effect on account of the screening rule through the range of models used (Figure 3), leading to a substantial reduction in run time. As an example, the run time for fitting logistic regression when $\rho = 0.5$ decreases from roughly 70 to 5 seconds when the screening rule is used.

We finish this section with an examination of two types of algorithms outlined in Section 2.2.4: the strong set and previous set algorithm. In Figure 1 we observed that the strong rule is conservative when correlation is high among predictors, which indicates that the previous set algorithm might yield an improvement over the strong set algorithm.

In order to examine this, we conduct a simulation in which we vary the strength of correlation between predictors as well as the parameter q in the construction of the BH regularization sequence. Motivation for varying the latter comes from the relationship between coefficient clustering and the intervals in the regularization sequence—higher values of q cause larger gaps in the sequence, which in turn leads to more clustering among predictors. This clustering, in turn, is strongest at the start of the path when regularization is strong.

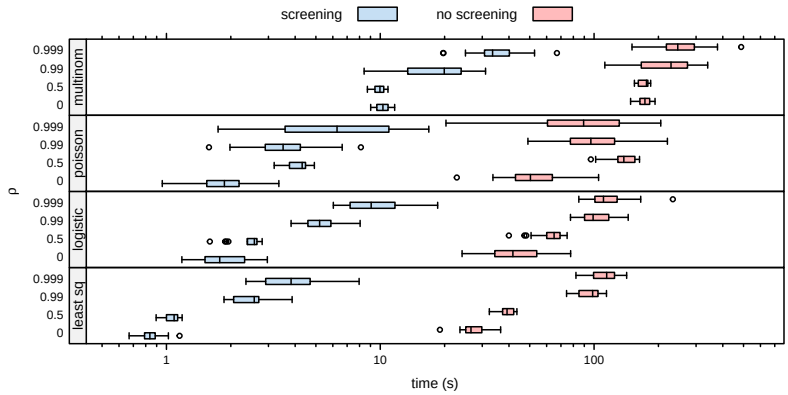


Figure 3: Time taken to fit SLOPE with or without the strong screening rule for randomly generated data.

For large enough q and ρ, this behavior in fact occasionally causes almost all predictors to enter the model at the second step on the path. As an example, using when $\rho = 0.6$ and fitting with $q = 10^{-2}$ and 10^{-4} leads to 2215 and 8 nonzero coefficients respectively at the second step in one simulation.

Here, we let $n = 200$, $p = 5000$, $k = 50$, and $\rho \in \{0, 0.1, 0.2, \dots, 0.8\}$. The data generation process corresponds to the setup at the start of this section for least squares regression data except for the covariance structure of X, which is equal to that in Section 3.2.1. We sample the non-zero entries in β independently from a random variable $U \sim \mathcal{N}(0, 1)$.

The two algorithms perform similarly for $\rho \leq 0.6$ (Figure 4). For larger ρ, the previous set strategy evidently outperforms the strong set strategy. This result is not surprising: consider Figure 1, for instance, which shows that the behavior of the regularization path under strong correlation makes the previous set strategy particularly effective in this context.

3.3 Real Data

3.3.1 Efficiency and Violations

We examine efficiency and violations for four real data sets: *arcene*, *dorothea*, *gisette*, and *golub*, which are the same data sets that were examined in Tibshirani et al. [21]. The first three originate from Guyon et al. [31] and were originally collected from the UCI (University of California Irvine) Machine Learning Repository [32], whereas the last data set, *golub*, was originally published in Golub et al. [33]. All of the data sets were collected from `http://statweb.stanford.edu/~tibs/strong/realdata/` and feature a response $y \in \{0, 1\}$. We fit both least squares and logistic regression models to the data sets and examine the effect of the level of coarseness in the path by varying the length of the path ($l = 20, 50, 100$).

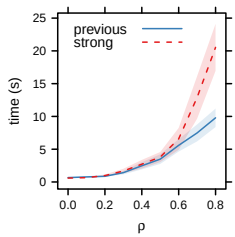


Figure 4: Time taken to fit a regularization path of SLOPE for least squares regression using either the strong or previous set algorithm.

There were no violations in any of the fits. The screening rule offers substantial reductions in problem size (Figure 5), particularly for the path length of 100, for which the size of the screened set of predictors ranges from roughly 1.5–4 times the size of the active set.

3.3.2 Performance

In this section, we introduce three new data sets: *e2006-tfidf* [34], *physician* [35], and *news20* [36]. *e2006-tfidf* was collected from Frandi [34], *news20* from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets` [37], and *physician* from `https://www.jstatsoft.org/`

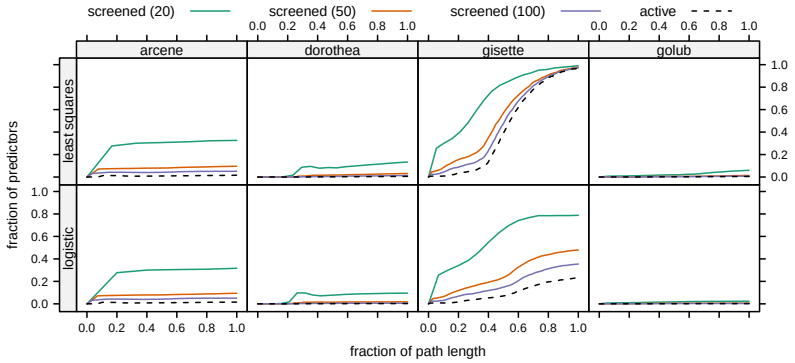


Figure 5: Proportion of predictors included in the model by the strong screening rule as a proportion of the total number of active predictors in the model for a path of λ sequences. Three types of paths have are examined, using path lengths of 20, 50, and 100.

Table 1: Benchmarks measuring wall-clock time for four data sets fit with different models using either the strong screening rule or no rule.

				time (s)	
dataset	model	n	p	no screening	screening
dorothea	logistic	800	88119	914	14
e2006-tfidf	least squares	3308	150358	43353	4944
news20	multinomial	1000	62061	5485	517
physician	poisson	4406	25	34	34

`article/view/v027i08` [38]. We use the test set for *e2006-tfidf* and a subset of 1000 observations from the training set for *news20*.

In Table 1, we summarize the results from fitting sorted ℓ_1-regularized least squares, logistic, Poisson, and multinomial regression to the four data sets. Once again, we see that the screening rule improves performance in the high-dimensional regime and presents no noticeable drawback even when $n > p$.

4 Conclusions

In this paper, we have developed a heuristic predictor screening rule for SLOPE and shown that it is a generalization of the strong rule for the lasso. We have demonstrated that it offers dramatic improvements in the $p \gg n$ regime, often reducing the time required to fit the full regularization path for SLOPE by orders of magnitude, as well as imposing little-to-no cost when $p < n$. At the time of this publication, an efficient implementation of the screening rule is available in the R package `SLOPE` [28].

The performance of the rule is demonstrably weaker when predictors in the design matrix are heavily correlated. This issue may be mitigated by the use of the previous set strategy that we have investigated here; part of the problem, however, is related to the clustering behavior that SLOPE exhibits: large portions of the total number of predictors often enter the model in a few clusters when regularization is strong. A possible avenue for future research might therefore be to investigate if screening rules for this clustering behavior might be developed and utilized to further enhance performance in estimating SLOPE.

Broader Impact

The predictor screening rules introduced in this article allow for a substantial improvement of the speed of SLOPE. This facilitates application of SLOPE to the identification of important predictors in huge data bases, such as collections of whole genome genotypes in Genome Wide Association Studies. It also paves the way for the implementation of cross-validation techniques and improved efficiency of the Adaptive Bayesian version SLOPE (ABSLOPE [39]), which requires multiple iterations of the SLOPE algorithm. Adaptive SLOPE bridges Bayesian and the frequentist methodology and enables good predictive models with FDR control in the presence of many hyper-parameters or missing data. Thus it addresses the problem of false discoveries and lack of replicability in a variety of important problems, including medical and genetic studies.

In general, the improved efficiency resulting from the predictor screening rules will make the SLOPE family of models (SLOPE [3], grpSLOPE [6], and ABSLOPE) accessible to a broader audience, enabling researchers and other parties to fit SLOPE models with improved efficiency. The time required to apply these models will be reduced and, in some cases, data sets that were otherwise too large to be analyzed without access to dedicated high-performance computing clusters can be tackled even with modest computational means.

We can think of no way by which these screening rules may put anyone at disadvantage. The methods we outline here do not in any way affect the model itself (other than boosting its performance) and can therefore only be of benefit. For the same reason, we do not believe that the strong rules for SLOPE introduces any ethical issues, biases, or negative societal consequences. In contrast, it is in fact possible that the reverse is true given that SLOPE serves as an alternative to, for instance, the lasso, and has superior model selection properties [10, 39] and lower bias [39].

Acknowledgments and Disclosure of Funding

We would like to thank Patrick Tardivel (Institut de Mathématiques de Bourgogne) and Yvette Baurne (Department of Statistics, Lund University) for their insightful comments on the paper. The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at Lunarc partially funded by the Swedish Research Council through grant agreement no. 2018-05973.

The research of Małgorzata Bogdan was supported by the grant of the Polish National Center of Science no. 2016/23/B/ST1/00454. The research of Jonas Wallin was supported by the Swedish Research Council, grant no. 2018-01726. These entities, however, have in no way influenced the work on this paper.

References

[1] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. ISSN 0035-9246.

[2] Małgorzata Bogdan, Ewout van den Berg, Weijie Su, and Emmanuel Candès. Statistical estimation and testing via the sorted L1 norm. *arXiv:1310.1969 [math, stat]*, October 2013.

[3] Małgorzata Bogdan, Ewout van den Berg, Chiara Sabatti, Weijie Su, and Emmanuel J. Candès. SLOPE – adaptive variable selection via convex optimization. *The annals of applied statistics*, 9(3):1103–1140, 2015. ISSN 1932-6157. doi: 10.1214/15-AOAS842.

[4] Xiangrong Zeng and Mário A. T. Figueiredo. Decreasing weighted sorted l1 regularization. *IEEE Signal Processing Letters*, 21(10):1240–1244, October 2014. ISSN 1070-9908, 1558-2361. doi: 10.1109/LSP.2014.2331977.

[5] Howard D. Bondell and Brian J. Reich. Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with OSCAR. *Biometrics*, 64(1):115–123, 2008. ISSN 0006-341X. doi: 10.1111/j.1541-0420.2007.00843.x.

[6] Damian Brzyski, Alexej Gossmann, Weijie Su, and Małgorzata Bogdan. Group SLOPE – adaptive selection of groups of predictors. *Journal of the American Statistical Association*, pages 1–15, January 2018. ISSN 0162-1459. doi: 10/gfrd93.

[7] Weijie Su and Emmanuel Candès. SLOPE is adaptive to unknown sparsity and asymptotically minimax. *The Annals of Statistics*, 44(3):1038–1068, June 2016. ISSN 0090-5364. doi: 10.1214/15-AOS1397.

[8] Xiangrong Zeng and Mário A. T. Figueiredo. The atomic norm formulation of OSCAR regularization with application to the Frank–Wolfe algorithm. In *EUSIPCO 2014*, pages 780–784, Lisbon, Portugal, September 2014. IEEE. ISBN 978-0-9928626-1-9.

[9] Philipp Kremer, Damian Brzyski, Malgorzata Bogdan, and Sandra Paterlini. Sparse index clones via the sorted L1-norm. SSRN Scholarly Paper ID 3412061, Social Science Research Network, Rochester, NY, June 2019.

[10] Mario Figueiredo and Robert Nowak. Ordered weighted L1 regularized regression with strongly correlated covariates: Theoretical aspects. In *Artificial Intelligence and Statistics*, pages 930–938, May 2016.

[11] Ulrike Schneider and Patrick Tardivel. The geometry of uniqueness and model selection of penalized estimators including SLOPE, LASSO, and basis pursuit. *arXiv:2004.09106 [math, stat]*, April 2020.

[12] Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, April 2004. ISSN 0090-5364. doi: 10.1214/009053604000000067.

[13] Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. *The Annals of Applied Statistics*, 1(2):302–332, December 2007. ISSN 1932-6157. doi: 10/d88g8c.

[14] Laurent El Ghaoui, Vivian Viallon, and Tarek Rabbani. Safe feature elimination in sparse supervised learning. Technical Report UCB/EECS-2010-126, EECS Department, University of California, Berkeley, September 2010.

[15] Zhen James Xiang and Peter J. Ramadge. Fast lasso screening tests based on correlations. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2137–2140, March 2012. doi: 10.1109/ICASSP.2012.6288334.

[16] Jie Wang, Peter Wonka, and Jieping Ye. Lasso screening rules via dual polytope projection. *Journal of Machine Learning Research*, 16(1):1063–1101, May 2015. ISSN 1532-4435.

[17] Eugene Ndiaye, Olivier Fercoq, Alexandre Gramfort, and Joseph Salmon. Gap safe screening rules for sparsity enforcing penalties. *Journal of Machine Learning Research*, 18(128):1–33, 2017.

[18] Olivier Fercoq, Alexandre Gramfort, and Joseph Salmon. Mind the duality gap: Safer rules for the lasso. In Francis Bach and David Blei, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 333–342, Lille, France, July 2015. PMLR.

[19] Jianqing Fan and Jinchi Lv. Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5):849–911, 2008. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2008.00674.x.

[20] Tyler B Johnson and Carlos Guestrin. Blitz: A principled meta-algorithm for scaling sparse optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, page 9, Lille, France, 2015. JMLR: W&CP.

[21] Robert Tibshirani, Jacob Bien, Jerome Friedman, Trevor Hastie, Noah Simon, Jonathan Taylor, and Ryan J. Tibshirani. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 74(2):245–266, March 2012. ISSN 1369-7412. doi: 10/c4bb85.

[22] Antoine Bonnefoy, Valentin Emiya, Liva Ralaivola, and Rémi Gribonval. A dynamic screening principle for the lasso. In *EUSIPCO 2014*, pages 6–10, Lisbon, Portugal, September 2014. IEEE. ISBN 978-0-9928626-1-9.

[23] Yaohui Zeng, Tianbao Yang, and Patrick Breheny. Hybrid safe–strong rules for efficient optimization in lasso-type problems. *Computational Statistics & Data Analysis*, 153:107063, January 2021. ISSN 0167-9473. doi: 10.1016/j.csda.2020.107063.

[24] Johan Larsson, Małgorzata Bogdan, and Jonas Wallin. The strong screening rule for SLOPE. *arXiv:2005.03730 [cs, stat]*, May 2020.

[25] Runxue Bao, Bin Gu, and Heng Huang. Fast OSCAR and OWL regression via safe screening rules. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, page 11, Vienna, Austria, July 2020. PMLR.

[26] Zhiqi Bu, Jason Klusowski, Cynthia Rush, and Weijie Su. Algorithmic analysis and statistical estimation of SLOPE via approximate message passing. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9361–9371, Vancouver, Canada, December 2019. Curran Associates, Inc.

[27] Patrick J. C. Tardivel, Rémi Servien, and Didier Concordet. Simple expressions of the LASSO and SLOPE estimators in low-dimension. *Statistics*, 54(2):340–352, February 2020. ISSN 0233-1888. doi: 10.1080/02331888.2020.1720019.

[28] Johan Larsson, Jonas Wallin, Malgorzata Bogdan, Ewout van den Berg, Chiara Sabatti, Emmanuel Candes, Evan Patterson, and Weijie Su. SLOPE: Sorted L1 penalized estimation, April 2020.

[29] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, January 2009. doi: 10.1137/080716542.

[30] Małgorzata Bogdan, Ewout van den Berg, Chiara Sabatti, Weijie Su, and Emmanuel J. Candès. Supplement to "SLOPE – adaptive variable selection via convex optimization". *Annals of Applied Statistics*, 9(3):1–7, 2015. ISSN 1932-6157. doi: 10.1214/15-AOAS842SUPP.

[31] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 545–552, Vancouver, Canada, December 2004. MIT Press. ISBN 978-0-262-19534-8.

[32] Dheeru Dua and Casey Graff. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2019.

[33] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, October 1999. ISSN 0036-8075. doi: 10.1126/science.286.5439.531.

[34] Emanuele Frandi. Replication data for: "Fast and scalable Lasso via stochastic Frank-Wolfe methods with a convergence guarantee", October 2015.

[35] Partha Deb and Pravin K. Trivedi. Demand for medical care by the elderly: A finite mixture approach. *Journal of Applied Econometrics*, 12(3):313–336, 1997. ISSN 0883-7252.

[36] Ken Lang. NewsWeeder: Learning to filter netnews. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the Twelfth International Conference on International Conference on Machine Learning*, pages 331–339, Tahoe City, CA, USA, July 1995. Morgan Kaufmann. ISBN 978-1-55860-377-6. doi: 10.1016/B978-1-55860-377-6.50048-7.

[37] Chih-chung Chang and Chih-jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, May 2011. doi: 10.1145/1961189.1961199.

[38] Achim Zeileis, Christian Kleiber, and Simon Jackman. Regression models for count data in R. *Journal of Statistical Software*, 27(1):1–25, July 2008. ISSN 1548-7660. doi: 10.18637/jss.v027.i08.

[39] Wei Jiang, Malgorzata Bogdan, Julie Josse, Blazej Miasojedow, Veronika Rockova, and TraumaBase Group. Adaptive Bayesian SLOPE – high-dimensional model selection with missing values. *arXiv:1909.06631 [stat]*, November 2019.

Erratum to *The Strong Screening Rule for SLOPE*

Johan Larsson
Dept. of Statistics, Lund University
johan.larsson@stat.lu.se

Małgorzata Bogdan
Dept. of Mathematics, University of Wroclaw
Dept. of Statistics, Lund University
malgorzata.bogdan@uwr.edu.pl

Jonas Wallin
Dept. of Statistics, Lund University
jonas.wallin@stat.lu.se

This file contains an erratum for Larsson et al. [1]. In Theorem 1 in the paper we left out a condition for the subdifferential of the SLOPE penalty, namely that $\operatorname{sign} \beta_{\mathcal{A}_i} = \operatorname{sign} s$. The updated and corrected theorem is given in Theorem 1. Note that the condition was in fact included in the proof of the theorem, which therefore requires no changes.

Theorem 1. *The subdifferential $\partial J(\beta; \lambda) \in \mathbb{R}^p$ is the set of all $g \in \mathbb{R}^p$ such that*

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

References

[1] Johan Larsson, Małgorzata Bogdan, and Jonas Wallin. The strong screening rule for SLOPE. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33*, volume 33, pages 14592–14603. Curran Associates, Inc. ISBN 978-1-71382-954-6. URL https://papers.nips.cc/paper%5Ffiles/paper/2020/hash/a7d8ae4569120b5bec12e7b6e9648b86-Abstract.html.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Supplement to "The Strong Screening Rule for SLOPE"

Johan Larsson[1,*], Małgorzata Bogdan[1,2], and Jonas Wallin[1]

[1] *Department of Statistics, Lund University*
[2] *Department of Mathematics, University of Wroclaw*
[*] *Corresponding author: johan.larsson@stat.lu.se*

October 21, 2020

1 Proofs

1.1 Proof of Theorem 1

By definition, the subdifferential $\partial J(\beta; \lambda)$ is the set of all $g \in \mathbb{R}^p$ such that

$$J(y; \lambda) \geq J(\beta; \lambda) + g^T(y - \beta) = \sum_{j=1}^{p} |\beta|_{(j)} \lambda_j + g^T(y - \beta), \tag{1}$$

for all $y \in \mathbb{R}^p$.

Assume that we have K clusters $\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_K$ (as defined per Equation 2 (main article)) and that $\beta = |\beta|_\downarrow$, which means we can rewrite (1) as

$$\begin{aligned} 0 &\geq J(\beta; \lambda) - J(y; \lambda) + g^T(y - \beta) \\ &= \sum_{i \in \mathcal{A}_1} (\lambda_i |\beta|_{(i)} - g_i \beta_i - \lambda_i |y|_{(i)} + g_i y_i) + \dots \\ &\quad + \sum_{i \in \mathcal{A}_K} (\lambda_i |\beta|_{(i)} - g_i \beta_i - \lambda_i |y|_{(i)} + g_i y_i). \end{aligned}$$

Notice that we must have $\sum_{i \in \mathcal{A}_j} (\lambda_i |\beta|_{(i)} - g_i \beta_i - \lambda_i |y|_{(i)} + g_i y_i) \leq 0$ for all $j \in \{1, 2, \dots, K\}$ since otherwise the inequality breaks by selecting $y_i = \beta_i$ for $i \in \mathcal{A}_j^c$. This means that it is sufficient to restrict attention to a single set as well as take this to be the set $\mathcal{A}_i = \{1, \dots, p\}$.

Case 1 ($\beta = \mathbf{0}$). In this case (1) reduces to $J(y; \lambda) \geq g^T y$. Now take a $c \in \mathcal{Z}$ where

$$\mathcal{Z} = \left\{ s \in \mathbb{R}^p \;\middle|\; \operatorname{cumsum}(|s|_\downarrow - \lambda) \preceq \mathbf{0} \right\} \tag{2}$$

and assume that $|c_1| \geq \dots \geq |c_p|$ without loss of generality.

Clearly, $J(y;\lambda) \geq c^T y$ holds if and only if $J(y^*;\lambda) - c^T y^* \geq 0$ where

$$y^* = \arg\min_y \left\{ J(y;\lambda) - c^T y \right\}.$$

Now, since $J(y;\lambda)$ is invariant to changes in signs and permutation of y, it follows from the rearrangement inequality [HLP52, Theorem 368] that $|y|_1^* \geq \cdots \geq |y|_p^*$. This permits us to formulate the following equivalent problem:

$$\begin{aligned} \text{minimize} \quad & y^T(\operatorname{sign}(y) \odot \lambda - c) \\ \text{subject to} \quad & \operatorname{sign}(y) = \operatorname{sign}(c), \\ & |y_1| \geq \cdots \geq |y_p|. \end{aligned}$$

To minimize the objective $y^T(\operatorname{sign}(y) \odot \lambda - |c|) = |y|^T(\lambda - |c|)$, recognize first that we must have $y_1^* = y_2^*$ since $c \in \mathcal{Z}$, which implies $\lambda_1 - |c_1| \geq 0$. Likewise, $y_2^*(\lambda_1 - |c_1|) + y_2^*(\lambda_2 - |c_2|) \geq 0$ since $\lambda_1 + \lambda_2 - (|c_1| + |c_2|) \geq 0$, which leads us to conclude that $y_2^* = y_3^*$. Then, proceeding inductively, it is easy to see that $y_p^* \sum_{i=1}^p (\lambda_i - |c_i|) \geq 0$, which implies $y_1^* = \cdots = y_p^* = 0$. At this point, we have shown that $c \in \mathcal{Z} \implies c \in \partial J(\beta;\lambda)$.

For the next part note that $g \in \mathcal{Z}$ is equivalent to requiring $|g|_{(1)} \leq \lambda_1$ and

$$|g|_{(i)} \leq \sum_{j=1}^{i} \lambda_j - \sum_{j=2}^{i} |g|_{(j)}, \qquad i = 1, \ldots, p. \tag{3}$$

Now assume that there is a c such that $c \in \partial J(\beta;\lambda)$ and $c \notin \mathcal{Z}$. Then there exists an $\varepsilon > 0$ and $i \in \{1, 2, \ldots, p\}$ such that

$$|c|_{(i)} \leq \sum_{j=1}^{i} \lambda_j - \sum_{j=2}^{i} |c|_{(j)} + \varepsilon, \qquad i = 1, \ldots, p.$$

Yet if $c = [\lambda_1, \ldots, \lambda_{i-1}, \lambda_i + \varepsilon, \lambda_{i+1}, \ldots, \lambda_p]^T$ then (1) breaks for $y = \mathbf{1}$, which implies that $c \notin \mathcal{Z} \implies c \notin \partial J(\beta;\lambda)$.

Case 2 ($\beta \neq \mathbf{0}$). Now let $|\beta_i| \coloneqq \alpha$ for all $i = 1, \ldots, p$, since by construction all β are equal in absolute value. Now (1) reduces to

$$\begin{aligned} J(y;\lambda) &\geq J(\beta;\lambda) - g^T\beta + g^T y \\ &= \sum_{i=1}^{p} \lambda_i \alpha - \sum_{i=1}^{p} g_i \operatorname{sign}(\beta_i)\alpha + g^T y \\ &= \alpha \sum_{i=1}^{p} \left(\lambda_i - g_i \operatorname{sign}(\beta_i)\right) + g^T y. \end{aligned} \tag{4}$$

The first term on the right-hand side of the last equality must be zero since otherwise the inequality breaks for $y = \mathbf{0}$. In addition, it must also hold that $\operatorname{sign}(\beta_i) = \operatorname{sign}(g_i)$ for all i such that $|\beta_i| > 0$. To show this, suppose the opposite

is true, that is, there exists at least one j such that $\text{sign}(g_j) \neq \text{sign}(\beta_j)$. But then if we take $y_j = \alpha \,\text{sign}(g_j)$ and $y_i = -\alpha \,\text{sign}(g_i)$, (4) is violated, which proves the statement by contradiction.

Taken together, this means that we have $g \in \mathcal{H}$ where

$$\mathcal{H} = \left\{ s \in \mathbb{R}^p \mid \sum_{j=1}^{p} (|s_j| - \lambda_j) = 0. \right\}$$

We are now left with $J(y; \lambda) \geq g^T y$, but this is exactly the setting from case one. Direct application of the reasoning from that part shows that we must have $g \in \mathcal{Z}$. Connecting the dots, we finally conclude that $c \in \mathcal{Z} \cap \mathcal{H} \implies c \in \partial J(\beta; \lambda)$.

1.2 Proof of Proposition 1

Suppose that we have $\mathcal{B} \neq \varnothing$ after running Algorithm 1 (main article). In this case we have

$$\text{cumsum}(c_\mathcal{B} - \lambda_\mathcal{B}) = \text{cumsum}\left(\left(\left| \nabla f(\hat{\beta}(\lambda^{(m+1)})) \right|_\downarrow \right)_\mathcal{B} - \lambda_\mathcal{B}^{(m+1)} \right) \prec \mathbf{0},$$

which implies via Theorem 1 (main article) and Equation 3 (main article) that all predictors in $\mathcal{B}$ must be inactive and that $\mathcal{S}$ contains the true support set.

1.3 Proof of Proposition 2

We need to show that the strong rule approximation does not violate the inequality on the fourth line in Algorithm 1 (main article). Since $\text{cumsum}(y) \succeq \text{cumsum}(x)$ for all $x, y \in \mathbb{R}^p$ if and only iff $y \succeq x$, it suffices to show that

$$|c_j(\lambda^{(m)})| + \lambda_j^{(m)} - \lambda_j^{(m+1)} \geq |c_j(\lambda^{(m+1)})|$$

for all $j = 1, 2, \dots, p$, which in turn means that Algorithm 1 (main article) with $|c_j(\lambda^{(m)})| + \lambda_j^{(m)} - \lambda_j^{(m+1)}$ as input cannot result in any violations.

From our assumptions we have

$$|c_j(\lambda^{(m+1)}) - c_j(\lambda^{(m)})| \leq |\lambda_j^{(m+1)} - \lambda_j^{(m)}|.$$

Using this fact, observe that

$$\begin{aligned} |c_j(\lambda^{(m+1)})| &\leq |c_j(\lambda^{(m+1)}) - c_j(\lambda^{(m)})| + |c_j(\lambda^{(m)})| \\ &\leq \lambda_j^{(m)} - \lambda_j^{(m+1)} + |c_j(\lambda^{(m)})|. \end{aligned}$$

1.4 Proof of Proposition 3

Let $c = (\nabla f(\hat{\beta}(\lambda)))$ and $\lambda_1 = \lambda_2$ and assume without loss of generality that $p = 2$ and $c_1 \geq c_2 \geq 0$. Recall that the strong rule for lasso discards the jth predictor whenever $c_j < \lambda_1$. There are three cases to consider.

Case 3 ($c_2 \leq c_1 < \lambda_1$). $\text{cumsum}(c - \lambda) \prec 0$, which means both predictors are discarded.

Case 4 ($c_1 \geq \lambda_1 > c_2$). The first predictor is retained since $\text{cumsum}(c - \lambda)_1 > 0$; the second is discarded because $c_2 \leq \lambda$.

Case 5 ($c_1 \geq c_2 \geq \lambda_1$). Both predictors are retained since $\text{cumsum}(c - \lambda) \succeq 0$.

The two results are equivalent for the lasso and thus the strong rule for SLOPE is a generalization of the strong rule for the lasso.

2 Algorithms

Algorithm 1 Strong set algorithm

$\mathcal{V} \leftarrow \varnothing$
$\mathcal{E} \leftarrow \mathcal{S}(\lambda^{(m+1)}) \cup \mathcal{T}(\lambda^{(m)})$
do
  compute $\hat{\beta}_{\mathcal{E}}(\lambda^{(m+1)})$
  $\mathcal{V} \leftarrow$ KKT violations in full set
  $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{V}$
while $\mathcal{V} \neq \varnothing$
return $\hat{\beta}_{\mathcal{E}}(\lambda^{(m+1)})$

Algorithm 2 Previous set algorithm

$\mathcal{V} \leftarrow \varnothing$
$\mathcal{E} \leftarrow \mathcal{T}(\lambda^{(m)})$
do
  compute $\hat{\beta}_{\mathcal{E}}(\lambda^{(m+1)})$
  $\mathcal{V} \leftarrow$ KKT violations in $\mathcal{S}(\lambda^{(m+1)})$
  if $\mathcal{V} = \varnothing$ **then**
    $\mathcal{V} \leftarrow$ KKT violations in full set
  end if
  $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{V}$
while $\mathcal{V} \neq \varnothing$
return $\hat{\beta}_{\mathcal{E}}(\lambda^{(m+1)})$

References

[1] G.H. Hardy, J.E. Littlewood, and G. Pólya. *Inequalities*. Second. Cambridge, United Kingdom: Cambridge University Press, 1952. ISBN: 0-521-05206-8.

II

Look-ahead screening rules for the Lasso

Johan Larsson[1]*

[1] *The Department of Statistics, Lund University*

Abstract: The lasso is a popular method to induce shrinkage and sparsity in the solution vector (coefficients) of regression problems, particularly when there are many predictors relative to the number of observations. Solving the lasso in this high-dimensional setting can, however, be computationally demanding. Fortunately, this demand can be alleviated via the use of *screening rules* that discard predictors prior to fitting the model, leading to a reduced problem to be solved. In this paper, we present a new screening strategy: *look-ahead screening*. Our method uses safe screening rules to find a range of penalty values for which a given predictor cannot enter the model, thereby screening predictors along the remainder of the path. In experiments we show that these look-ahead screening rules outperform the active warm-start version of the Gap Safe rules.

Keywords: lasso, sparse regression, screening rules, safe screening rules
AMS subject classification: 62J07

1 Introduction

The lasso [6] is a staple among regression models for high-dimensional data. It induces shrinkage and sparsity in the solution vector (regression coefficients) through penalization by the ℓ_1-norm. The optimal level of penalization is, however, usually unknown, which means we typically need to estimate it through model tuning across a grid of candidate values: the regularization path. This leads to a heavy computational load.

Thankfully, the advent of so-called *screening rules* have lead to remarkable advances in tackling this problem. Screening rules discard a subset of the predictors *before* fitting the model, leading to, often considerable, reductions in problem size. There are two types of screening rules: heuristic and safe rules. The latter kind provides a certificate that discarded predictors cannot be active at the optimum—that is, have a non-zero corresponding coefficients—whereas heuristic rules do not. In this paper, we will focus entirely on safe rules.

A prominent type of safe rules are the Gap Safe rules [5, 1], which use the duality gap in a problem to provide effective screening rules. There currently exists sequential versions of the Gap Safe rules, that discard predictors for the next step

*Corresponding author: johan.larsson@stat.lu.se

on the regularization path, as well as dynamic rules, which discard predictors during optimization at the current penalization value.

The objective of this paper is to introduce a new screening strategy based on Gap Safe screening: *look-ahead screening*, which screens predictors for a range of penalization parameters. We show that this method can be used to screen predictors for the entire stretch of the regularization path, leading to substantial improvements in the time to fit the entire lasso path.

2 Look-Ahead Screening

Let $X \in \mathbb{R}^{n \times p}$ be the design matrix with n observations and p predictors and $y \in \mathbb{R}^n$ the response vector. The lasso is represented by the following convex optimization problem:

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \left\{ P(\beta; \lambda) = \frac{1}{2} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\} \tag{1}$$

where $P(\beta; \lambda)$ is the *primal* objective. We let $\hat{\beta}_\lambda$ be the solution to (1) for a given λ. Moreover, the dual problem of (1) is

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

where $D(\theta; \lambda)$ is the *dual* objective. The relationship between the primal and dual problems is given by $y = X\hat{\beta}_\lambda + \lambda \hat{\theta}_\lambda$.

Next, we let G be the so-called *duality gap*, defined as

$$G(\beta, \theta; \lambda) = P(\beta; \lambda) - D(\theta; \lambda) = \frac{1}{2} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 - \lambda \theta^T y + \frac{\lambda^2}{2} \theta^T \theta. \tag{3}$$

In the case of the lasso, strong duality holds, which means that $G(\hat{\beta}_\lambda, \hat{\theta}_\lambda; \lambda) = 0$ for any choice of λ.

Suppose, now, that we have solved the lasso for λ; then for any given $\lambda^* \leq \lambda$, the Gap Safe rule [5] discards the jth predictor if

$$|X^T \theta_\lambda|_j + \|x_j\|_2 \sqrt{\frac{1}{\lambda_*^2} G(\beta_\lambda, \theta_\lambda; \lambda^*)} < 1 \tag{4}$$

where

$$\theta_\lambda = \frac{y - X\beta_\lambda}{\max\left(|X^T(y - X\beta_\lambda)|, \lambda\right)}$$

is a dual-feasible point [5] obtained through dual scaling.

Observe that (4) is a quadratic inequality with respect to λ_*, which means that it is trivial to discover the boundary points via the quadratic formula:

$$\lambda_* = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad \text{where} \quad \begin{aligned} a &= \left(1 - |x_j^T \theta_\lambda|\right)^2 - \frac{1}{2} \theta_\lambda^T \theta_\lambda \|x_j\|_2^2, \\ b &= \left(\theta_\lambda^T y - \|\beta_\lambda\|_1\right) \|x_j\|_2^2, \\ c &= -\frac{1}{2} \|y - X\beta_\lambda\|_2^2 \|x_j\|_2^2. \end{aligned}$$

By restricting ourselves to an index j corresponding to a predictor that is inactive at λ and recalling that we have $\lambda_* \leq \lambda$ by construction, we can inspect the signs of a, b, and c and find a range λ values for which predictor j must be inactive. Using this idea for the lasso path—a grid of λ values starting from the null (intercept-only) model, which corresponds to $\lambda_{\max} = \max_i |x_i^T y|$, and finishing at fraction of this (see section 3 for specifics)—we can screen predictor j for all upcoming λs, possibly discarding it for multiple steps on the path rather than just the next step. We call this idea *look-ahead screening*.

To illustrate the effectiveness of this screening method, we consider an instance of employing look-ahead screening for fitting a full lasso path to the *leukemia* data set [3]. At the first step of the path, the screening method discards 99.6% of the predictors for the steps up to and including step 5. The respective figures for steps 10 and 15 are 99.3% and 57%. At step 20, however, the rule does not discard a single predictor. In Figure 1, we have visualized the screening performance of look-ahead screening for a random sample of 25 predictors from this data set.

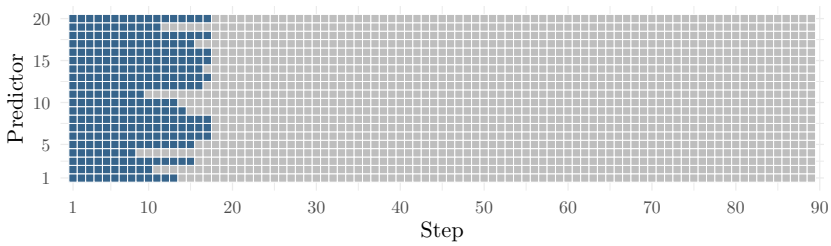


Figure 1: This figure shows the predictors screened at the first step of the lasso path via look-ahead screening for a random sample of 20 predictors from the *leukemia* data set. A blue square indicates that the corresponding predictor can be discarded at the respective step.

As is typical for all screening methods, the effectiveness of look-ahead screening is greatest at the start of the path and diminishes as the strength of penalization decreases later on in the path. Note, however, that all of the quantities involved in the rule are available as a by-product of solving the problem at the previous step, which means that the costs of look-ahead screening are diminutive.

3 Simulations

In this section, we study the effectiveness of the look-ahead screening rules by comparing them against the active warm start version of the Gap Safe rules [1, 5]. We follow the recommendations in [5] and run the screening procedure every tenth pass of the solver. Throughout the experiments, we center the response vector by its mean, as well as center and scale the predictors by their means and uncorrected sample standard deviations respectively.

To construct the regularization path, we employ the standard settings from glmnet, using a log-spaced path of 100 λ values from $\lambda_{\max}$ to $\varepsilon\lambda_{\max}$, where $\varepsilon = 10^{-2}$

if $p > n$ and 10^{-4} otherwise. We also use the default path stopping criteria from glmnet, that is, stop the path whenever the deviance ratio, $1-\text{dev}/\text{dev}_{\text{null}}$, is greater than or equal to 0.999, the fractional increase in deviance explained is lower than 10^{-5}, or, if $p \geq n$, when the number of active predictors exceeds or is equal to n.

To fit the lasso, we use cyclical coordinate descent [2]. We consider the solver to have converged whenever the duality gap as a fraction of the primal value for the null model is less than or equal to 10^{-6} and the amount of *infeasibility*, which we define as $\max_j \left(|x_j^T(y - X\beta_\lambda)| - \lambda\right)$, as a fraction of $\lambda_{\max}$ is lower than or equal to 10^{-5}.

Source code for the experiments, including a container to facilitate reproducibility, can be found at https://github.com/jolars/LookAheadScreening/. An HPC cluster node with two Intel Xeon E5-2650 v3 processors (Haswell, 20 compute cores per node) and 64 GB of RAM was used to run the experiments.

We run experiments on a design with $n = 100$ and $p = 50\,000$, drawing the rows of X i.i.d. from $\mathcal{N}(0, \Sigma)$ and y from $\mathcal{N}(X\beta, \sigma^2 I)$ with $\sigma^2 = \beta^T\Sigma\beta/\text{SNR}$, where SNR is the signal-to-noise ratio. We set 5 coefficients, equally spaced throughout the coefficient vector, to 1 and the rest to zero. Taking inspiration from **(author?)** [4], we consider SNR values of 0.1, 1, and 6.

Judging by the results (Figure 2), the addition of look-ahead screening results in sizable reductions in the solving time of the lasso path, particularly in the high signal-to-noise context.

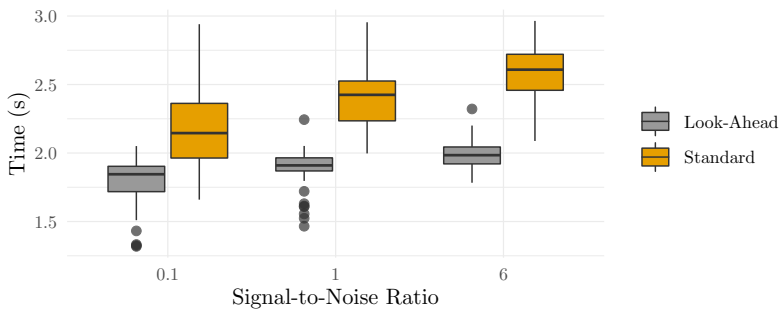


Figure 2: Standard box plots of timings to fit a full lasso path to a simulated data set with $n = 100$, $p = 50\,000$, and five true signals.

4 Discussion

In this paper, we have presented *look-ahead screening*, which is a novel method to screen predictors for a range of penalization values along the lasso regularization path using Gap Safe screening. Our results show that this type of screening can yield considerable improvements in performance for the standard lasso. For other loss functions, (4) may no longer reduce to a quadratic inequality and will hence

require more computation. Nevertheless, we believe that applying these rules in these cases is feasible and likely to result in comparable results.

Moreover, the idea is general and can therefore be extended to any type of safe screening rule and also used in tandem with heuristic screening rules in order to avoid expensive KKT computations. Finally, although we only cover one type of cyclical coordinate descent in our experiments, note that our screening method is agnostic to the solver used and that we expect the results hold for any solver that benefits from predictor screening.

Acknowledgements: I would like to thank my supervisor, Jonas Wallin, for valuable feedback on this work. The computations were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC) at LUNARC partially funded by the Swedish Research Council through grant agreement no. 2017-05973.

Bibliography

[1] O. Fercoq, A. Gramfort, and J. Salmon. Mind the duality gap: Safer rules for the lasso. In F. Bach and D. Blei, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 333–342, Lille, France, July 2015. PMLR.

[2] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. doi: 10.18637/jss.v033.i01.

[3] T.R. Golub, D.K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J.P. Mesirov, H. Coller, M.L. Loh, J.R. Downing, M.A. Caligiuri, C.D. Bloomfield, and E.S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, Oct. 1999. ISSN 0036-8075. doi: 10.1126/science.286.5439.531.

[4] T. Hastie, R. Tibshirani, and R. Tibshirani. Best subset, forward stepwise or lasso? Analysis and recommendations based on extensive comparisons. *Statistical Science*, 35(4):579–592, Nov. 2020. ISSN 0883-4237. doi: 10. 1214/19-STS733.

[5] E. Ndiaye, O. Fercoq, A. Gramfort, and J. Salmon. Gap safe screening rules for sparsity enforcing penalties. *Journal of Machine Learning Research*, 18(128):1–33, 2017.

[6] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. ISSN 0035-9246.

III

The Hessian Screening Rule

Johan Larsson
Department of Statistics
Lund University
johan.larsson@stat.lu.se

Jonas Wallin
Department of Statistics
Lund University
jonas.wallin@stat.lu.se

Abstract

Predictor screening rules, which discard predictors before fitting a model, have had considerable impact on the speed with which sparse regression problems, such as the lasso, can be solved. In this paper we present a new screening rule for solving the lasso path: the Hessian Screening Rule. The rule uses second-order information from the model to provide both effective screening, particularly in the case of high correlation, as well as accurate warm starts. The proposed rule outperforms all alternatives we study on simulated data sets with both low and high correlation for ℓ_1-regularized least-squares (the lasso) and logistic regression. It also performs best in general on the real data sets that we examine.

1 Introduction

High-dimensional data, where the number of features (p) exceeds the number of observations (n), poses a challenge for many classical statistical models. A common remedy for this issue is to regularize the model by penalizing the regression coefficients such that the solution becomes sparse. A popular choice of such a penalization is the ℓ_1-norm, which, when the objective is least-squares, leads to the well-known lasso [1]. More specifically, we will focus on the following convex optimization problem:

$$\underset{\beta \in \mathbb{R}^p}{\text{minimize}} \left\{ f(\beta; X) + \lambda \|\beta\|_1 \right\}, \tag{1}$$

where $f(\beta; X)$ is smooth and convex. We let $\hat{\beta}$ be the solution vector for this problem and, abusing notation, equivalently let $\hat{\beta} : \mathbb{R} \mapsto \mathbb{R}^p$ be a function that returns this vector for a given λ. Our focus lies in solving (1) along a regularization path $\lambda_1, \lambda_2 \ldots, \lambda_m$ with $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$. We start the path at λ_{max}, which corresponds to the null (all-sparse) model[1], and finish at some fraction of λ_{max} for which the model is either almost saturated (in the $p \geq n$ setting), or for which the solution approaches the ordinary least-squares estimate. The motivation for this focus is that the optimal λ is typically unknown and must be estimated through model tuning, such as cross-validation. This involves repeated refitting of the model to new batches of data, which is computationally demanding.

Fortunately, the introduction of so-called *screening rules* has improved this situation remarkably. Screening rules use tests that screen and possibly discard predictors from the model *before* it is fit, which effectively reduces the dimensions of the problem and leads to improvements in performance and memory usage. There are, generally speaking, two types of screening rules: *safe* and *heuristic* rules. Safe rules guarantee that discarded predictors are inactive at the optimum—heuristic rules do not and may therefore cause violations: discarding active predictors. The possibility of violations mean that heuristic methods need to validate the solution through checks of the Karush–Kuhn–Tucker (KKT) optimality conditions after optimization has concluded and, whenever there are violations, re-run optimization, which can be costly particularly because the KKT checks themselves are expensive. This means that the distinction between safe and heuristic rules only matters in regards to algorithmic

[1] λ_{max} is in fact available in closed form—for the lasso it is $\max_j |x_j^T y|$.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

details—all heuristic methods that we study here use KKT checks to catch these violations, which means that these methods are in fact also safe.

Screening rules can moreover also be classified as *basic*, *sequential*, or *dynamic*. Basic rules screen predictors based only on information available from the null model. Sequential rules use information from the previous step(s) on the regularization path to screen predictors for the next step. Finally, dynamic rules screen predictors during optimization, reducing the set of screened predictors repeatedly throughout optimization.

Notable examples of safe rules include the basic SAFE rule [2], the sphere tests [3], the R-region test [4], Slores [5], Gap Safe [6, 7], and Dynamic Sasvi [8]. There is also a group of dual polytope projection rules, most prominently Enhanced Dual Polytope Projection (EDPP) [9]. As noted by Fercoq, Gramfort, and Salmon [6], however, the sequential version of EDPP relies on exact knowledge of the optimal solution at the previous step along the path to be safe in practice, which is only available for $\lambda_{\max}$. Among the heuristic rules, we have the Strong Rule [10], SIS [11], and ExSIS [12]. But the latter two of these are not sequential rules and solve a potentially reduced form of the problem in (1)—we will not discuss them further here. In addition to these two types of rules, there has also recently been attempts to combine safe and heuristic rules into so-called hybrid rules [13].

There are various methods for employing these rules in practice. Of particular interest are so-called *working set* strategies, which use a subset of the screened set during optimization, iteratively updating the set based on some criterion. Tibshirani et al. [10] introduced the first working set strategy, which we in this paper will refer to simply as the *working set strategy*. It uses the set of predictors that have ever been active as an initial working set. After convergence on this set, it checks the KKT optimality conditions on the set of predictors selected by the strong rule, and then adds predictors that violate the conditions to the working set. This procedure is then repeated until there are no violations, at which point the optimality conditions are checked for the entire set, possibly triggering additional iterations of the procedure. Blitz [14] and Celer [15] are two other methods that use both Gap Safe screening and working sets. Instead of choosing previously active predictors as a working set, however, both Blitz and Celer assign priorities to each feature based on how close each feature is of violating the Gap Safe check and construct the working set based on this prioritization. In addition to this, Celer uses dual point acceleration to improve Gap Safe screening and speed up convergence. Both Blitz and Celer are heuristic methods.

One problem with current screening rules is that they often become conservative—including large numbers of predictors into the screened set—when dealing with predictors that are strongly correlated. Tibshirani et al. [10], for instance, demonstrated this to be the case with the strong rule, which was the motivation behind the working set strategy. (See Appendix F.4 for additional experiments verifying this). Yet because the computational complexity of the KKT checks in the working set strategy still depends on the strong rule, the effectiveness of the rule may nevertheless be hampered in this situation. A possible and—as we will soon show—powerful solution to this problem is to make use of the second-order information available from (1), and in this paper we present a novel screening rule based on this idea. Methods using second-order information (the Hessian) are often computationally infeasible for high-dimensional problems. We utilize two properties of the problem to remedy this issue: first, we need only to compute the Hessian for the active set, which is often much smaller than the full set of predictors. Second, we avoid constructing the Hessian (and it's inverse) from scratch for each λ along the path, instead updating it sequentially by means of the Schur complement. The availability of the Hessian also enables us to improve the warm starts (the initial coefficient estimate at the start of each optimization run) used when fitting the regularization path, which plays a key role in our method.

We present our main results in Section 3, beginning with a reformulation of the strong rule and working set strategy before we arrive at the screening rule that represents the main result of this paper. In Section 4, we present numerical experiments on simulated and real data to showcase the effectiveness of the screening rule, demonstrating that the rule is effective both when $p \gg n$ and $n \gg p$, out-performing the other alternatives that we study. Finally, in Section 5 we wrap up with a discussion on these results, indicating possible ways in which they may be extended.

2 Preliminaries

We use lower-case letters to denote scalars and vectors and upper-case letters for matrices. We use $\mathbf{0}$ and $\mathbf{1}$ to denote vectors with elements all equal to 0 or 1 respectively, with dimensions inferred from context. Furthermore, we let sign be the standard signum function with domain $\{-1, 0, 1\}$, allowing it to be overloaded for vectors.

Let $c(\lambda) \coloneqq -\nabla_\beta f\big(\hat{\beta}(\lambda); X\big)$ be the negative gradient, or so-called *correlation*, and denote $\mathcal{A}_\lambda = \{i : |c(\lambda)_i| > \lambda\}$ as the *active set* at λ: the support set of the non-zero regression coefficients corresponding to $\hat{\beta}(\lambda)$. In the interest of brevity, we will let $\mathcal{A} \coloneqq \mathcal{A}_\lambda$. We will consider β a solution to (1) if it satisfies the stationary criterion

$$\mathbf{0} \in \nabla_\beta f(\beta; X) + \lambda \partial. \tag{2}$$

Here ∂ is the subdifferential of $\|\beta\|_1$, defined as

$$\partial_j \in \begin{cases} \{\operatorname{sign}(\hat{\beta}_j)\} & \text{if } \hat{\beta}_j \neq 0, \\ [-1, 1] & \text{otherwise.} \end{cases}$$

This means that there must be a $\tilde{\partial} \in \partial$ for a given λ such that

$$\nabla_\beta f(\beta; X) + \lambda \tilde{\partial} = \mathbf{0}. \tag{3}$$

3 Main Results

In this section we derive the main result of this paper: the Hessian screening rule. First, however, we now introduce a non-standard perspective on screening rules. In this approach, we note that (2) suggests a simple and general formulation for a screening rule, namely: we substitute the gradient vector in the optimality condition of a ℓ_1-regularized problem with an estimate. More precisely, we discard the jth predictor for the problem at a given λ if the magnitude of the jth component of the gradient vector estimate is smaller than this λ, that is

$$|\tilde{c}(\lambda)_j| < \lambda. \tag{4}$$

In the following sections, we review the strong rule and working set method for this problem from this perspective, that is, by viewing both methods as gradient approximations. We start with the case of the standard lasso (ℓ_1-regularized least-squares), where we have $f(\beta; X) = \frac{1}{2}\|X\beta - y\|_2^2$.

3.1 The Strong Rule

The sequential strong rule for ℓ_1-penalized least-squares regression [10] discards the jth predictor at $\lambda = \lambda_{k+1}$ if

$$\big|x_j^T (X\hat{\beta}(\lambda_k) - y)\big| = |c(\lambda_k)_j| < 2\lambda_{k+1} - \lambda_k.$$

This is equivalent to checking that

$$\tilde{c}^S(\lambda_{k+1}) = c(\lambda_k) + (\lambda_k - \lambda_{k+1}) \operatorname{sign}(c(\lambda_k)) \tag{5}$$

satisfies (4). The strong rule gradient approximation (5) is also known as the *unit bound*, since it assumes the gradient of the correlation vector to be bounded by one.

3.2 The Working Set Method

A simple but remarkably effective alternative to direct use of the strong rule is the working set heuristic [10]. It begins by estimating β at the $(k+1)$th step using only the coefficients that have been previously active at any point along the path, i.e. $\mathcal{A}_{1:k} = \cup_{i=1}^{k} \mathcal{A}_i$. The working set method can be viewed as a gradient estimate in the sense that

$$\tilde{c}^W(\lambda_{k+1}) = X^T \Big(y - X_{\mathcal{A}_{1:k}} \tilde{\beta}(\lambda_{k+1}, \mathcal{A}_{1:k})\Big) = -\nabla f\big(\tilde{\beta}(\lambda_{k+1}, \mathcal{A}_{1:k}); X\big),$$

where $\tilde{\beta}(\lambda, \mathcal{A}) = \arg\min_\beta \frac{1}{2}||y - X_\mathcal{A}\beta||^2 + \lambda|\beta|$.

3.3 The Hessian Screening Rule

We have shown that both the strong screening rule and the working set strategy can be expressed as estimates of the correlation (negative gradient) for the next step of the regularization path. As we have discussed previously, however, basing this estimate on the strong rule can lead to conservative approximations. Fortunately, it turns out that we can produce a better estimate by utilizing second-order information.

We start by noting that (3), in the case of the standard lasso, can be formulated as

$$\begin{bmatrix} X_{\mathcal{A}}^T X_{\mathcal{A}} & X_{\mathcal{A}}^T X_{\mathcal{A}^c} \\ X_{\mathcal{A}^c}^T X_{\mathcal{A}} & X_{\mathcal{A}^c}^T X_{\mathcal{A}^c} \end{bmatrix} \begin{bmatrix} \hat{\beta}_{\mathcal{A}} \\ 0 \end{bmatrix} + \lambda \begin{bmatrix} \operatorname{sign}(\hat{\beta}(\lambda)_{\mathcal{A}}) \\ \partial_{\mathcal{A}^c} \end{bmatrix} = \begin{bmatrix} X_{\mathcal{A}}^T y \\ X_{\mathcal{A}^c}^T y \end{bmatrix},$$

and consequently that

$$\hat{\beta}(\lambda)_{\mathcal{A}} = (X_{\mathcal{A}}^T X_{\mathcal{A}})^{-1} \big(X_{\mathcal{A}}^T y - \lambda \operatorname{sign}(\hat{\beta}_{\mathcal{A}}) \big).$$

Note that, for an interval $[\lambda_l, \lambda_u]$ in which the active set is unchanged, that is, $\mathcal{A}_\lambda = \mathcal{A}$ for all $\lambda \in [\lambda_u, \lambda_k]$, then $\hat{\beta}(\lambda)$ is a continuous linear function in λ (Theorem 3.1)[2].

Theorem 3.1. *Let $\hat{\beta}(\lambda)$ be the solution of (1) where $f(\beta; X) = \frac{1}{2}\|X\beta - y\|_2^2$. Define*

$$\hat{\beta}^{\lambda^*}(\lambda)_{A_{\lambda^*}} = \hat{\beta}(\lambda^*)_{\mathcal{A}_{\lambda^*}} - (\lambda^* - \lambda) \left(X_{\mathcal{A}_{\lambda^*}}^T X_{\mathcal{A}_{\lambda^*}} \right)^{-1} \operatorname{sign}\left(\hat{\beta}(\lambda^*)_{\mathcal{A}_{\lambda^*}} \right)$$

and $\hat{\beta}^{\lambda^*}(\lambda)_{\mathcal{A}^c_{\lambda^*}} = 0$. If it for $\lambda \in [\lambda_0, \lambda^*]$ holds that (i) $\operatorname{sign}\left(\hat{\beta}^{\lambda^*}(\lambda)\right) = \operatorname{sign}\left(\hat{\beta}(\lambda^*)\right)$ and (ii) $\max |\nabla f(\hat{\beta}^{\lambda^*}(\lambda))_{\mathcal{A}_{\lambda^*}}| < \lambda$, then $\hat{\beta}(\lambda) = \hat{\beta}^{\lambda^*}(\lambda)$ for $\lambda \in [\lambda_0, \lambda^*]$.

See Appendix A for a full proof. Using Theorem 3.1, we have the following second-order approximation of $c(\lambda_{k+1})$:

$$\hat{c}^H(\lambda_{k+1}) = -\nabla f\left(\hat{\beta}^{\lambda_k}(\lambda_{k+1})_{A_{\lambda_k}}\right) = c(\lambda_k) + (\lambda_{k+1} - \lambda_k) X^T X_{\mathcal{A}_k} (X_{\mathcal{A}_k}^T X_{\mathcal{A}_k})^{-1} \operatorname{sign}\left(\hat{\beta}(\lambda_k)_{\mathcal{A}_k}\right). \tag{6}$$

Remark 3.2. If no changes in the active set occur in $[\lambda_{k+1}, \lambda_k]$, (6) is in fact an exact expression for the correlation at the next step, that is, $\hat{c}^H(\lambda_{k+1}) = c(\lambda_{k+1})$.

One problem with using the gradient estimate in (6) is that it is expensive to compute due to the inner products involving the full design matrix. To deal with this, we use the following modification, in which we restrict the computation of these inner products to the set indexed by the strong rule, assuming that predictors outside this set remain inactive:

$$\tilde{c}^H(\lambda_{k+1})_j \coloneqq \begin{cases} \lambda_{k+1} \operatorname{sign} \hat{\beta}(\lambda_k)_j & \text{if } j \in \mathcal{A}_{\lambda_k}, \\ 0 & \text{if } |\tilde{c}^S(\lambda_{k+1})_j| < \lambda_{k+1} \text{ and } j \notin \mathcal{A}_{\lambda_k}, \\ \hat{c}^H(\lambda_{k+1})_j & \text{else.} \end{cases}$$

For high-dimensional problems, this modification leads to large computational gains and seldom proves inaccurate, given that the strong rule only rarely causes violations [10]. Lastly, we make one more adjustment to the rule, which is to add a proportion of the unit bound (used in the strong rule) to the gradient estimate:

$$\check{c}^H(\lambda_{k+1})_j \coloneqq \tilde{c}^H(\lambda_{k+1})_j + \gamma(\lambda_{k+1} - \lambda_k) \operatorname{sign}(c(\lambda_k)_j),$$

where $\gamma \in \mathbb{R}_+$. Without this adjustment there would be no upwards bias on the estimate, which would cause more violations than would be desirable. In our experiments, we have used $\gamma = 0.01$, which has worked well for most problems we have encountered. This finally leads us to the *Hessian screening rule*: discard the jth predictor at λ_{k+1} if $|\check{c}^H(\lambda_{k+1})_j| < \lambda_{k+1}$.

We make one more modification in our implementation of the Hessian Screening Rule, which is to use the union of the ever-active predictors and those screened by the screening rule as our final set of screened predictors. We note that this is a marginal improvement to the rule, since violations of the rule are already quite infrequent. But it is included nonetheless, given that it comes at no cost and occasionally prevents violations.

[2] This result is not a new discovery [16], but is included here for convenience because the following results depend on it.

As an example of how the Hessian Screening Rule performs, we examine the screening performance of several different strategies. We fit a full regularization path to a design with $n = 200$, $p = 20\,000$, and pairwise correlation between predictors of ρ. (See Section 4 and Appendix F.4 for more information on the setup.) We compute the average number of screened predictors across iterations of the coordinate descent solver. The results are displayed in Figure 1 and demonstrate that our method gracefully handles high correlation among predictors, offering a screened set that is many times smaller than those produced by the other screening strategies. In Appendix F.4 we extend these results to ℓ_1-regularized logistic regression as well and report the frequency of violations.

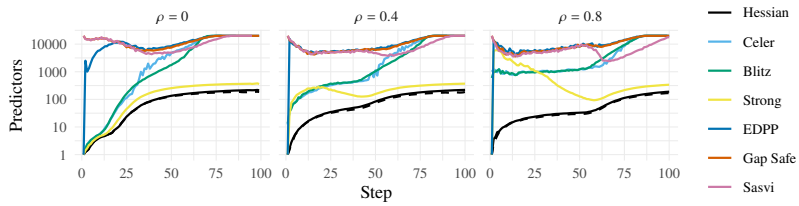


Figure 1: The number of predictors screened (included) for when fitting a regularization path of ℓ_1-regularized least-squares to a design with varying correlation (ρ), $n = 200$, and $p = 20000$. The values are averaged over 20 repetitions. The minimum number of active predictors at each step across iterations is given as a dashed line. Note that the y-axis is on a $\log_{10}$ scale.

Recall that the Strong rule bounds its gradient of the correlation vector estimate at one. For the Hessian rule, there is no such bound. This means that it is theoretically possible for the Hessian rule to include more predictors than the Strong rule[3]. In fact, it is even possible to design special cases where the Hessian rule could be more conservative than the Strong rule. In practice, however, we have not encountered any situation in which this is the case.

3.3.1 Updating the Hessian

A potential drawback to using the Hessian screening rule is the computational costs of computing the Hessian and its inverse. Let $\mathcal{A}_k$ be the active set at step k on the lasso path. In order to use the Hessian screening rule we need $H_k^{-1} = (X_{\mathcal{A}_k}^T X_{\mathcal{A}_k})^{-1}$. Computing $(X_{\mathcal{A}_k}^T X_{\mathcal{A}_k})^{-1}$ directly, however, has numerical complexity $O(|\mathcal{A}_k|^3 + |\mathcal{A}_k|^2 n)$. But if we have stored (H_{k-1}^{-1}, H_{k-1}) previously, we can utilize it to compute (H_k^{-1}, H_k) more efficiently via the so-called sweep operator [17]. We outline this technique in Algorithm 1 (Appendix B). The algorithm has a reduction step and an augmentation step; in the reduction step, we reduce the Hessian and its inverse to remove the presence of any predictors that are no longer active. In the augmentation step, we update the Hessian and its inverse to account for predictors that have just become active.

The complexity of the steps depends on the size of the sets $\mathcal{C} = \mathcal{A}_{k-1} \setminus \mathcal{A}_k$,$\mathcal{D} = \mathcal{A}_k \setminus \mathcal{A}_{k-1}$, and $\mathcal{E} = \mathcal{A}_k \cap \mathcal{A}_{k-1}$ The complexity of the reduction step is $O(|\mathcal{C}|^3 + |\mathcal{C}|^2|\mathcal{E}| + |\mathcal{C}||\mathcal{E}|^2)$ and the complexity of the augmentation step is $O(|\mathcal{D}|^2 n + n|\mathcal{D}||\mathcal{E}| + |\mathcal{D}|^2|\mathcal{E}| + |\mathcal{D}|^3)$ since $n \geq \max(|\mathcal{E}|, |\mathcal{D}|)$. An iteration of Algorithm 1 therefore has complexity $O(|\mathcal{D}|^2 n + n|\mathcal{D}||\mathcal{E}| + |\mathcal{C}|^3 + |\mathcal{C}||\mathcal{E}|^2)$.

In most applications, the computationally dominant term will be $n|\mathcal{D}||\mathcal{E}|$ (since, typically, $n > |\mathcal{E}| > \mathcal{D} > \mathcal{C}$) which could be compared to evaluating the gradient for $\beta_{\mathcal{A}_k}$, which is $n\,(|\mathcal{D}| + |\mathcal{E}|)$ when $\beta_{\mathcal{A}_k^c} = 0$. Note that we have so far assumed that the inverse of the Hessian exists, but this need not be the case. To deal with this issue we precondition the Hessian. See Appendix C for details.

3.3.2 Warm Starts

The availability of the Hessian and its inverse offers a coefficient warm start that is more accurate than the standard, naive, approach of using the estimate from the previous step. With the Hessian screening rule, we use the following warm start.

$$\hat{\beta}(\lambda_{k+1})_{\mathcal{A}_k} \coloneqq \hat{\beta}(\lambda_k)_{\mathcal{A}_k} + (\lambda_k - \lambda_{k+1}) H_{\mathcal{A}_k}^{-1} \operatorname{sign}\left(\hat{\beta}(\lambda_k)_{\mathcal{A}_k}\right), \tag{7}$$

[3]The chance of this happening is tied to the setting of γ.

where $H_{\mathcal{A}_k}^{-1}$ is the Hessian matrix for the differentiable part of the objective. Our warm start is equivalent to the one used in Park and Hastie [18], but is here made much more efficient due due to the efficient updates of the Hessian and its inverse that we use.

Remark 3.3. The warm start given by (7) is the exact solution at λ_k if the active set remains constant in $[\lambda_{k+1}, \lambda_k]$.

As a first demonstration of the value of this warm start, we look at two data sets: *YearPredicitionMSD* and *colon-cancer*. We fit a full regularization path using the setup as outlined in Section 4, with or without Hessian warm starts. For YearPredictionMSD we use the standard lasso, and for colon-cancer ℓ_1-regularized logistic regression.

The Hessian warm starts offer sizable reductions in the number of passes of the solver (Figure 2), for many steps requiring only a single pass to reach convergence. On inspection, this is not a surprising find. There are no changes in the active set for many of these steps, which means that the warm start is almost exact—"almost" due to the use of a preconditioner for the Hessian (see Appendix C).

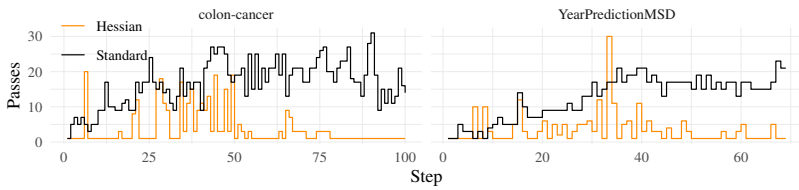


Figure 2: Number of passes of coordinate descent along a full regularization path for the *colon-cancer* ($n = 62$, $p = 2\,000$) and *YearPredictionMSD* ($n = 463\,715$, $p = 90$) data sets, using either Hessian warm starts (7) or standard warm starts (the solution from the previous step).

3.3.3 General Loss Functions

We now venture beyond the standard lasso and consider loss functions of the form

$$f(\beta; X) = \sum_{i=1}^{n} f_i(x_i^T \beta) \tag{8}$$

where f_i is convex and twice differentiable. This, for instance, includes logistic, multinomial, and Poisson loss functions. For the strong rule and working set strategy, this extension does not make much of a difference. With the Hessian screening rule, however, the situation is different.

To see this, we start by noting that our method involving the Hessian is really a quadratic Taylor approximation of (1) around a specific point β_0. For loss functions of the type (8), this approximation is equal to

$$\begin{aligned} Q(\beta, \beta_0) &= f(\beta_0; X) + \sum_{i=1}^{n} \left(x_i^T f_i'(x_i^T \beta_0)(\beta - \beta_0) + \frac{1}{2}(\beta - \beta_0)^T x_i^T f_i''(x_i^T \beta_0) x_i (\beta - \beta_0) \right) \\ &= \frac{1}{2} \left(\tilde{y}(x_i^T \beta_0) - X\beta \right)^T D\left(w(\beta_0)\right) \left(\tilde{y}(x_i^T \beta_0) - X\beta \right) + C(\beta_0), \end{aligned}$$

where $D(w(\beta_0))$ is a diagonal matrix with diagonal entries $w(\beta_0)$ where $w(\beta_0)_i = f''(x_i^T \beta_0)$ and $\tilde{y}(z)_i = f_i'(z) / f_i''(z) - x_i^T \beta_0$, whilst $C(\beta_0)$ is a constant with respect to β.

Suppose that we are on the lasso path at λ_k and want to approximate $c(\lambda_{k+1})$. In this case, we simply replace $f(\beta; X)$ in (1) with $Q(\beta, \hat{\beta}(\lambda_k))$, which leads to the following gradient approximation:

$$c^H(\lambda_{k+1}) = c(\lambda_k) + (\lambda_{k+1} - \lambda_k) X^T D(w) X_{\mathcal{A}_k} (X_{\mathcal{A}_k}^T D(w) X_{\mathcal{A}_k})^{-1} \operatorname{sign}\left(\hat{\beta}(\lambda_k)_{\mathcal{A}_k}\right),$$

where $w = w\big(\hat{\beta}(\lambda_k)\big)$. Unfortunately, we cannot use Algorithm 1 to update $X_{\mathcal{A}_k}^T D(w) X_{\mathcal{A}_k}$. This means that we are forced to either update the Hessian directly at each step, which can be computationally demanding when $|\mathcal{A}_k|$ is large and inefficient when X is very sparse, or to approximate

$D(w)$ with an upper bound. In logistic regression, for instance, we can use ¼ as such a bound, which also means that we once again can use Algorithm 1.

In our experiments, we have employed the following heuristic to decide whether to use an upper bound or compute the full Hessian in these cases: we use full updates at each step if $\text{sparsity}(X)n/\max\{n,p\} < 10^{-3}$ and the upper bound otherwise.

3.3.4 Reducing the Impact of KKT Checks

The Hessian Screening Rule is heuristic, which means there may be violations. This necessitates that we verify the KKT conditions after having reached convergence for the screened set of predictors, and add predictors back into the working set for which these checks fail. When the screened set is small relative to p, the cost of optimization is often in large part consumed by these checks. Running these checks for the full set of predictors always needs to be done once, but if there are violations during this step, then we need repeat this check, which is best avoided. Here we describe two methods to tackle this issue.

We employ a procedure equivalent to the one used in Tibshirani et al. [10] for the working set strategy: we first check the KKT conditions for the set of predictors singled out by the strong rule and then, if there are no violations in that set, check the full set of predictors for violations. This works well because the strong rule is conservative—violations are rare—which means that we seldom need to run the KKT checks for the entire set more than once.

If we, in spite of the augmentation of the rule, run into violations when checking the full set of predictors, that is, when the strong rule fails to capture the active set, then we can still avoid repeating the full KKT check by relying on Gap Safe screening: after having run the KKT checks and have failed to converge, we screen the set of predictors using the Gap Safe rule. Because this is a safe rule, we can be sure that the predictors we discard will be inactive, which means that we will not need to include them in our upcoming KKT checks. Because Gap Safe screening and the KKT checks rely on exactly the same quantity—the correlation vector–we can do so at marginal extra cost. To see how this works, we now briefly introduce Gap Safe screening. For details, please see Fercoq, Gramfort, and Salmon [6].

For the ordinary lasso (ℓ_1-regularized least squares), the primal (1) is $P(\beta) = \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$ and the corresponding dual is

$$D(\theta) = \frac{1}{2}\|y\|_2^2 - \frac{\lambda^2}{2}\left\|\theta - \frac{y}{\lambda}\right\|_2^2 \tag{9}$$

subject to $\|X^T\theta\|_\infty \leq 1$. The duality gap is then $G(\beta,\theta) = P(\beta) - D(\theta)$ and the relation between the primal and dual problems is given by $y = \lambda\hat{\theta} + X\hat{\beta}$, where $\hat{\theta}$ is the maximizer to the dual problem (9). In order to use Gap Safe screening, we need a feasible dual point, which can be obtained via dual point scaling, taking $\theta = (y - X\beta)/\max\left(\lambda, \|X^T(y - X\beta)\|_\infty\right)$. The Gap Safe screening rule then discards the jth feature if $|x_j^T\theta| < 1 - \|x_j\|_2\sqrt{2G(\beta,\theta)/\lambda^2}$. Since we have computed $X^T(y - X\beta)$ as part of the KKT checks, we can perform Gap Safe screening at an additional (and marginal) cost amounting to $O(n) + O(p)$.

Since this augmentation benefits the working set strategy too, we adopt it in our implementation of this method as well. To avoid ambiguity, we call this version working+. Note that this makes the working set strategy quite similar to Blitz. In Appendix F.8 we show the benefit of adding this type of screening.

3.3.5 Final Algorithm

The Hessian screening method is presented in full in Algorithm 2 (Appendix B).

Lemma 3.4. *Let $\beta \in \mathbb{R}^{p\times m}$ be the output of Algorithm 2 for a path of length m and convergence threshold $\varepsilon > 0$. For each step k along the path and corresponding solution $\beta^{(k)} \in \mathbb{R}^p$, there is a dual-feasible point $\theta^{(k)}$ such that $G(\beta^{(k)}, \theta^{(k)}) < \zeta\varepsilon$.*

Proof. First note that Gap safe screening [7, Theorem 6] ensures that $\mathcal{G} \supseteq \mathcal{A}_k$. Next, note that the algorithm guarantees that the working set, $\mathcal{W}$, grows with each iteration until $|x_j^T r| < \lambda_k$ for all

$j \in \mathcal{G} \setminus \mathcal{W}$, at which point

$$\max\left(\lambda_k, \|X_{\mathcal{W}}^T(y - X_{\mathcal{W}}\beta_{\mathcal{W}}^{(k)})\|_\infty\right) = \max\left(\lambda_k, \|X_{\mathcal{G}}^T(y - X_{\mathcal{G}}\beta_{\mathcal{G}}^{(k)})\|_\infty\right).$$

At this iteration, convergence at line 2, for the subproblem $(X_{\mathcal{W}}, y)$, guarantees convergence for the full problem, (X, y), since

$$\theta^{(k)} = \frac{y - X_{\mathcal{W}}\beta_{\mathcal{W}}^{(k)}}{\max\left(\lambda_k, \|X_{\mathcal{W}}^T(y - X_{\mathcal{W}}\beta_{\mathcal{W}}^{(k)})\|_\infty\right)}$$

is dual-feasible for the full problem. □

3.3.6 Extensions

Approximate Homotopy In addition to improved screening and warm starts, the Hessian also allows us to construct the regularization path adaptively via approximate homotopy [19]. In brief, the Hessian screening rule allows us to choose the next λ along the path adaptively, in effect distributing the grid of λs to better approach the exact (homotopy) solution for the lasso, avoiding the otherwise heuristic choice, which can be inappropriate for some data sets.

Elastic Net Our method can be extended to the elastic net [20], which corresponds to adding a quadratic penalty $\phi\|\beta\|_2^2/2$ to (1). The Hessian now takes the form $X_{\mathcal{A}}^T X_{\mathcal{A}} + \phi I$. Loosely speaking, the addition of this term makes the problem "more" quadratic, which in turn improves both the accuracy and stability of the screening and warm starts we use in our method. As far as we know, however, there is unfortunately no way to update the inverse of the Hessian efficiently in the case of the elastic net. More research in this area would be welcome.

4 Experiments

Throughout the following experiments, we scale and center predictors with the mean and uncorrected sample standard deviation respectively. For the lasso, we also center the response vector, y, with the mean.

To construct the regularization path, we adopt the default settings from `glmnet`: we use a log-spaced path of 100 λ values from λ_{max} to $\xi\lambda_{\text{max}}$, where $\xi = 10^{-2}$ if $p > n$ and 10^{-4} otherwise. We stop the path whenever the deviance ratio, $1 - \text{dev}/\text{dev}_{\text{null}}$, reaches 0.999 or the fractional decrease in deviance is less than 10^{-5}. Finally, we also stop the path whenever the number of coefficients ever to be active predictors exceeds p.

We compare our method against working+ (the modified version of the working set strategy from Tibshirani et al. [10]), Celer [15], and Blitz [14]. We initially also ran our comparisons against EDPP [9], the Gap Safe rule [6], and Dynamic Sasvi [8] too, yet these methods performed so poorly that we omit the results in the main part of this work. The interested reader may nevertheless consult Appendix F.6 where results from simulated data has been included for these methods too.

We use cyclical coordinate descent with shuffling and consider the model to converge when the duality gap $G(\beta, \theta) \leq \varepsilon\zeta$, where we take ζ to be $\|y\|_2^2$ when fitting the ordinary lasso, and $n \log 2$ when fitting ℓ_1-regularized logistic regression. Unless specified, we let $\varepsilon = 10^{-4}$. These settings are standard settings and, for instance, resemble the defaults used in Celer. For all of the experiments, we employ the line search algorithm used in Blitz[4].

The code used in these experiments was, for every method, programmed in C++ using the Armadillo library [21, 22] and organized as an R package via Rcpp [23]. We used the renv package [24] to maintain dependencies. The source code, including a Singularity [25] container and its recipe for reproducing the results, are available at `https://github.com/jolars/HessianScreening`. Additional details of the computational setup are provided in Appendix D.

[4] Without the line search, all of the tested methods ran into convergence issues, particularly for the high-correlation setting and logistic regression.

4.1 Simulated Data

Let $X \in \mathbb{R}^{n \times p}$, $\beta \in \mathbb{R}^p$, and $y \in \mathbb{R}^n$ be the predictor matrix, coefficient vector, and response vector respectively. We draw the rows of the predictor matrix independently and identically distributed from $\mathcal{N}(0, \Sigma)$ and generate the response from $\mathcal{N}(X\beta, \sigma^2 I)$ with $\sigma^2 = \beta^T \Sigma \beta / \text{SNR}$, where SNR is the signal-to-noise ratio. We set s coefficients, equally spaced throughout the coefficient vector, to 1 and the rest to zero.

In our simulations, we consider two scenarios: a low-dimensional scenario and a high-dimensional scenario. In the former, we set $n = 10\,000$, $p = 100$, $s = 5$, and the SNR to 1. In the high-dimensional scenario, we take $n = 400$, $p = 40\,000$, $s = 20$, and set the SNR to 2. These SNR values are inspired by the discussion in Hastie, Tibshirani, and Tibshirani [26] and intend to cover the middle-ground in terms of signal strength. We run our simulations for 20 iterations.

From Figure 3, it is clear that the Hessian screening rule performs best, taking the least time in every setting examined. The difference is largest for the high-correlation context in the low-dimensional setting and otherwise roughly the same across levels of correlation.

The differences between the other methods are on average small, with the working+ strategy performing slightly better in the $p > n$ scenario. Celer and Blitz perform largely on par with one another, although Celer sees an improvement in a few of the experiments, for instance in logistic regression when $p > n$.

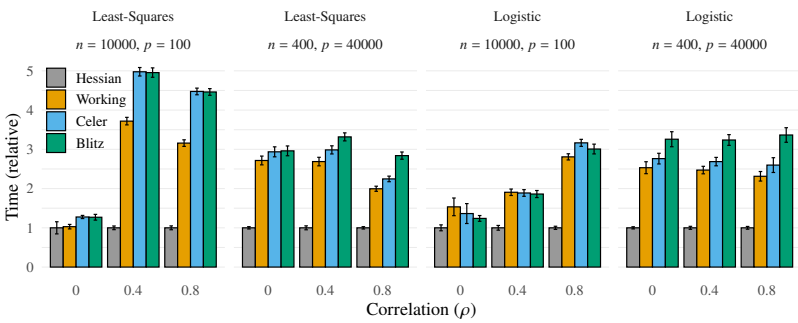


Figure 3: Time to fit a full regularization path for ℓ_1-regularized least-squares and logistic regression to a design with n observations, p predictors, and pairwise correlation between predictors of ρ. Time is relative to the minimal mean time in each group. The error bars represent ordinary 95% confidence intervals around the mean.

4.2 Real Data

In this section, we conduct experiments on real data sets. We run 20 iterations for the smaller data sets studied and three for the larger ones. For information on the sources of these data sets, please see Appendix E. For more detailed results of these experiments, please see Appendix F.5.

Starting with the case of ℓ_1-regularized least-squares regression, we observe that the Hessian screening rule performs best for all five data sets tested here (Table 1), in all but one instance taking less than half the time compared to the runner-up, which in each case is the working+ strategy. The difference is particularly large for the YearPredictionMSD and e2006-tfidf data sets.

In the case of ℓ_1-regularized logistic regression, the Hessian method again performs best for most of the examined data sets, for instance completing the regularization path for the madelon data set around five times faster than the working+ strategy. The exception is the arcene data set, for which the working+ strategy performs best out of the four methods.

We have provided additional results related to the effectiveness of our method in Appendix F.

Table 1: Average time to fit a full regularization path of ℓ_1-regularized least-squares and logistic regression to real data sets. Density represents the fraction of non-zero entries in X. Density and time values are rounded to two and three significant figures respectively.

					Time (s)			
Data Set	n	p	Density	Loss	Hessian	Working	Blitz	Celer
bcTCGA	536	17 322	1	Least-Squares	3.00	7.67	11.7	10.6
e2006-log1p	16 087	4 272 227	1.4×10^{-3}	Least-Squares	205	438	756	835
e2006-tfidf	16 087	150 360	8.3×10^{-3}	Least-Squares	14.3	143	277	335
scheetz	120	18 975	1	Least-Squares	0.369	0.643	0.706	0.801
YearPredictionMSD	463 715	90	1	Least-Squares	78.8	541	706	712
arcene	100	10 000	5.4×10^{-1}	Logistic	4.35	3.27	4.42	3.99
colon-cancer	62	2000	1	Logistic	0.0542	0.134	0.177	0.169
duke-breast-cancer	44	7129	1	Logistic	0.111	0.210	0.251	0.262
ijcnn1	35 000	22	1	Logistic	0.939	5.53	4.68	3.50
madelon	2000	500	1	Logistic	48.2	232	240	247
news20	19 996	1 355 191	3.4×10^{-4}	Logistic	1290	1620	2230	2170
rcv1	20 242	47 236	1.6×10^{-3}	Logistic	132	266	384	378

5 Discussion

We have presented the Hessian Screening Rule: a new heuristic predictor screening rule for ℓ_1-regularized generalized linear models. We have shown that our screening rule offers large performance improvements over competing methods, both in simulated experiments but also in the majority of the real data sets that we study here. The improved performance of the rule appears to come not only from improved effectiveness in screening, particularly in the high-correlation setting, but also from the much-improved warm starts, which enables our method to dominate in the $n \gg p$ setting. Note that although we have focused on ℓ_1-regularized least-squares and logistic regression here, our rule is applicable to any composite objective for which the differentiable part is twice-differentiable.

One limitation of our method is that it consumes more memory than its competitors owing to the storage of the Hessian and its inverse. This cost may become prohibitive for cases when $\min\{n, p\}$ is large. In these situations the next-best choice may instead be the working set strategy. Note also that we, in this paper, focus entirely on the lasso *path*. The Hessian Screening Rule is a sequential rule and may therefore not prove optimal when solving for a single λ, in which case a dynamic strategy such as Celer and Blitz likely performs better.

With respect to the relative performance of the working set strategy, Celer, and Blitz, we note that our results deviate somewhat from previous comparisons [15, 14]. We speculate that these differences might arise from the fact that we have used equivalent implementations for all of the methods and from the modification that we have used for the working set strategy.

Many avenues remain to be explored in the context of Hessian-based screening rules and algorithms, such as developing more efficient methods for updating of the Hessian matrix for non-least-squares objectives, such as logistic regression and using second-order information to further improve the optimization method used. Other interesting directions also include adapting the rules to more complicated regularization problems, such as the fused lasso [27], SLOPE [28], SCAD [29], and MCP [30]. Although the latter two of these are non-convex problems, they are locally convex for intervals of the regularization path [31], which enables the use of our method. Adapting the method for use in batch stochastic gradient descent would also be an interesting topic for further study, for instance by using methods such as the ones outlined in Asar et al. [32] to ensure that the Hessian remains positive definite.

Finally, we do not expect there to be any negative societal consequences of our work given that it is aimed solely at improving the performance of an optimization method.

Acknowledgments and Disclosure of Funding

We would like to thank Małgorzata Bogdan for valuable comments. This work was funded by the Swedish Research Council through grant agreement no. 2020-05081 and no. 2018-01726. The computations were enabled by resources provided by LUNARC. The results shown here are in part based upon data generated by the TCGA Research Network: https://www.cancer.gov/tcga.

References

[1] Robert Tibshirani. “Regression Shrinkage and Selection via the Lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. ISSN: 0035-9246. JSTOR: 2346178.

[2] Laurent El Ghaoui, Vivian Viallon, and Tarek Rabbani. *Safe Feature Elimination in Sparse Supervised Learning*. UCB/EECS-2010-126. Berkeley: EECS Department, University of California, Sept. 21, 2010.

[3] Zhen J. Xiang, Hao Xu, and Peter J Ramadge. “Learning Sparse Representations of High Dimensional Data on Large Scale Dictionaries”. In: *Advances in Neural Information Processing Systems 24*. Neural Information Processing Systems 2011. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., Dec. 12–17, 2011, pp. 900–908.

[4] Zhen James Xiang and Peter J. Ramadge. “Fast Lasso Screening Tests Based on Correlations”. In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Mar. 2012, pp. 2137–2140. DOI: 10.1109/ICASSP.2012.6288334.

[5] Jie Wang et al. “A Safe Screening Rule for Sparse Logistic Regression”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’14. Cambridge, MA, USA: MIT Press, Dec. 8, 2014, pp. 1053–1061.

[6] Olivier Fercoq, Alexandre Gramfort, and Joseph Salmon. “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, July 6–11, 2015, pp. 333–342.

[7] Eugene Ndiaye et al. “Gap Safe Screening Rules for Sparsity Enforcing Penalties”. In: *Journal of Machine Learning Research* 18.128 (2017), pp. 1–33.

[8] Hiroaki Yamada and Makoto Yamada. “Dynamic Sasvi: Strong Safe Screening for Norm-Regularized Least Squares”. In: *Advances in Neural Information Processing Systems*. NeurIPS 2021. Ed. by M. Ranzato et al. Vol. 34. New Orleans, USA: Curran Associates, Inc., 2021, pp. 14645–14655.

[9] Jie Wang, Peter Wonka, and Jieping Ye. “Lasso Screening Rules via Dual Polytope Projection”. In: *Journal of Machine Learning Research* 16.1 (May 15, 2015), pp. 1063–1101. ISSN: 1532-4435.

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

[11] Jianqing Fan and Jinchi Lv. “Sure Independence Screening for Ultrahigh Dimensional Feature Space”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 70.5 (2008), pp. 849–911. ISSN: 1467-9868. DOI: 10.1111/j.1467-9868.2008.00674.x.

[12] Talal Ahmed and Waheed U. Bajwa. “ExSIS: Extended Sure Independence Screening for Ultrahigh-Dimensional Linear Models”. In: *Signal Processing* 159 (June 1, 2019), pp. 33–48. ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2019.01.018.

[13] Yaohui Zeng, Tianbao Yang, and Patrick Breheny. “Hybrid Safe–Strong Rules for Efficient Optimization in Lasso-Type Problems”. In: *Computational Statistics & Data Analysis* 153 (Jan. 1, 2021), p. 107063. ISSN: 0167-9473. DOI: 10.1016/j.csda.2020.107063.

[14] Tyler B Johnson and Carlos Guestrin. “Blitz: A Principled Meta-Algorithm for Scaling Sparse Optimization”. In: *Proceedings of the 32nd International Conference on Machine Learning*. International Conference on Machine Learning. Vol. 37. Lille, France: JMLR: W&CP, 2015, p. 9.

[15] Mathurin Massias, Alexandre Gramfort, and Joseph Salmon. “Celer: A Fast Solver for the Lasso with Dual Extrapolation”. In: *Proceedings of the 35th International Conference on Machine Learning*. ICML 2018. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholm, Sweden: PMLR, July 10–15, 2018, pp. 3315–3324.

[16] Bradley Efron et al. “Least Angle Regression”. In: *Annals of Statistics* 32.2 (Apr. 2004), pp. 407–499. ISSN: 0090-5364. DOI: 10.1214/009053604000000067.

[17] James H. Goodnight. “A Tutorial on the SWEEP Operator”. In: *The American Statistician* 33.3 (1979), pp. 149–158. ISSN: 0003-1305. DOI: 10.2307/2683825. JSTOR: 2683825.

[18] Mee Young Park and Trevor Hastie. "L1-Regularization Path Algorithm for Generalized Linear Models". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 69.4 (2007), pp. 659–677. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2007.00607.x.

[19] Julien Mairal and Bin Yu. "Complexity Analysis of the Lasso Regularization Path". In: *Proceedings of the 29th International Conference on Machine Learning*. International Conference on Machine Learning 2012. Edinburgh, United Kingdom, June 2012, pp. 1835–1842.

[20] Hui Zou and Trevor Hastie. "Regularization and Variable Selection via the Elastic Net". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320. ISSN: 1369-7412.

[21] Dirk Eddelbuettel and Conrad Sanderson. "RcppArmadillo: Accelerating R with High-Performance C++ Linear Algebra". In: *Computational Statistics and Data Analysis* 71 (Mar. 2014), pp. 1054–1063.

[22] Conrad Sanderson and Ryan Curtin. "Armadillo: A Template-Based C++ Library for Linear Algebra". In: *The Journal of Open Source Software* 1.2 (2016), p. 26. DOI: 10.21105/joss.00026.

[23] Dirk Eddelbuettel and Romain François. "Rcpp: Seamless R and C++ Integration". In: *Journal of Statistical Software* 40.8 (2011), pp. 1–18. DOI: 10/gc3hqm.

[24] Kevin Ushey. *Renv: Project Environments*. Version 0.13.2. R Studio, 2021.

[25] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. "Singularity: Scientific Containers for Mobility of Compute". In: *PLOS ONE* 12.5 (May 11, 2017), e0177459. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0177459.

[26] Trevor Hastie, Robert Tibshirani, and Ryan Tibshirani. "Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons". In: *Statistical Science* 35.4 (Nov. 2020), pp. 579–592. ISSN: 0883-4237. DOI: 10.1214/19-STS733.

[27] Robert Tibshirani et al. "Sparsity and Smoothness via the Fused Lasso". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.1 (2005), pp. 91–108. ISSN: 1467-9868. DOI: 10.1111/j.1467-9868.2005.00490.x.

[28] Małgorzata Bogdan et al. "SLOPE – Adaptive Variable Selection via Convex Optimization". In: *The annals of applied statistics* 9.3 (Sept. 2015), pp. 1103–1140. ISSN: 1932-6157. DOI: 10.1214/15-AOAS842. pmid: 26709357.

[29] Jianqing Fan and Runze Li. "Variable Selection via Nonconcave Penalized Likelihood and Its Oracle Properties". In: *Journal of the American Statistical Association* 96.456 (Dec. 1, 2001), pp. 1348–1360. ISSN: 0162-1459. DOI: 10/fd7bfs.

[30] Cun-Hui Zhang. "Nearly Unbiased Variable Selection under Minimax Concave Penalty". In: *The Annals of Statistics* 38.2 (Apr. 2010), pp. 894–942. ISSN: 0090-5364, 2168-8966. DOI: 10/bp22zz.

[31] Patrick Breheny and Jian Huang. "Coordinate Descent Algorithms for Nonconvex Penalized Regression, with Applications to Biological Feature Selection". In: *The Annals of Applied Statistics* 5.1 (Mar. 2011), pp. 232–253. ISSN: 1932-6157, 1941-7330. DOI: 10.1214/10-AOAS388.

[32] Özgür Asar et al. "Linear Mixed Effects Models for Non-Gaussian Continuous Repeated Measurement Data". In: *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 69.5 (Sept. 9, 2020), pp. 1015–1065. ISSN: 1467-9876. DOI: 10.1111/rssc.12405.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes]
(b) Did you include complete proofs of all theoretical results? [Yes] See the supplementary material.

3. If you ran experiments...
(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Section 4.
(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 as well as the supplementary details and the references to existing data sets for discussions on test and training splits.
(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
(a) If your work uses existing assets, did you cite the creators? [Yes] See the supplement and Section 4.
(b) Did you mention the license of the assets? [Yes] A `LICENSE.md` file has been included along with the code.
(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See Section 4.
(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All data we used has been made available in the public domain.
(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] To the best of our knowledge, it does not.

5. If you used crowdsourcing or conducted research with human subjects...
(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] Not applicable to our work.
(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Not applicable to our work.
(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] Not applicable to our work.

Supplement to *The Hessian Screening Rule*

Johan Larsson
Department of Statistics
Lund University
johan.larsson@stat.lu.se

Jonas Wallin
Department of Statistics
Lund University
jonas.wallin@stat.lu.se

A Proofs

A.1 Proof of Theorem 1

It suffices to verify that the KKT conditions hold for $\hat{\beta}^{\lambda^*}(\lambda)$, i.e. that $\mathbf{0}$ is in the subdifferential. By (ii) it follows that the indices $\mathcal{A}^c_{\lambda^*}$ in the subdifferential contain zero. That leaves us only to show that $\nabla f\big(\hat{\beta}^{\lambda^*}(\lambda); X\big)_{\mathcal{A}_{\lambda^*}} = \lambda \operatorname{sign}\big(\hat{\beta}^{\lambda^*}(\lambda)\big)_{\mathcal{A}_{\lambda^*}}$.

$$
\begin{aligned}
&\nabla f\big(\hat{\beta}^{\lambda^*}(\lambda); X\big)_{\mathcal{A}_{\lambda^*}} \\
&= X^T_{\mathcal{A}_{\lambda^*}}\big(y - X_{\mathcal{A}_{\lambda^*}}\hat{\beta}^{\lambda^*}(\lambda)_{\mathcal{A}_{\lambda^*}}\big) \\
&= X^T_{\mathcal{A}_{\lambda^*}}\left(y - X_{\mathcal{A}_{\lambda^*}}\beta(\lambda^*)_{\mathcal{A}_{\lambda^*}} - \big(\lambda^* - \lambda\big)X_{\mathcal{A}_{\lambda^*}}\big(X^T_{\mathcal{A}_{\lambda^*}}X_{\mathcal{A}_{\lambda^*}}\big)^{-1}\operatorname{sign}\beta(\lambda^*)_{\mathcal{A}_{\lambda^*}}\right) \\
&= \nabla f\big(\hat{\beta}^{\lambda^*}(\lambda^*)\big)_{\mathcal{A}_{\lambda^*}} - (\lambda^* - \lambda)\operatorname{sign}\hat{\beta}(\lambda^*)_{\mathcal{A}_{\lambda^*}} \\
&= \lambda \operatorname{sign}\hat{\beta}(\lambda^*)_{\mathcal{A}_{\lambda^*}},
\end{aligned}
$$

which by (i) equals $\lambda \operatorname{sign}(\hat{\beta}^{\lambda^*}(\lambda))_{\mathcal{A}_{\lambda^*}}$.

B Algorithms

In this section we present the algorithms for efficiently updating the Hessian and its inverse (Algorithm 1) and the full algorithm for the Hessian screening method (Algorithm 2).

C Singular or Ill-Conditioned Hessians

In this section, we discuss situations in which the Hessian is singular or ill-conditioned and propose remedies for these situations.

Inversion of the Hessian demands that the null space corresponding to the active predictors $\mathcal{A}_\lambda$ contains only the zero vector, which typically holds when the columns of X are in general position, such as in the case of data simulated from continuous distributions. It is not, however, generally the case with discrete-valued data, particularly not in when $p \gg n$. In Lemma C.1, we formalize this point.

Lemma C.1. *Suppose that we have $e \in \mathbb{R}^p$ such that $Xe = 0$. Let $\hat{\beta}(\lambda)$ be the solution to the primal problem (1) and $\mathcal{E} = \{i : e_i \neq 0\}$; then $|\hat{\beta}(\lambda)_\mathcal{E}| > 0$ only if there exists a $z \in \mathbb{R}^p$ where $z_\mathcal{E} \in \{-1, 1\}^{|\mathcal{E}|}$ such that $z^T e = 0$.*

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Algorithm 1 This algorithm provides computationally efficient updates for the inverse of the Hessian. Note the slight abuse of notation here in that $\mathcal{E}$ is used both for X and Q. It is implicitly understood that $Q_{\mathcal{E}\mathcal{E}}$ is the sub-matrix of Q that corresponds to the columns $\mathcal{E}$ of X.

Input: $X, H = X_{\mathcal{A}}^T X_{\mathcal{A}}, Q \coloneqq H^{-1}, \mathcal{A}, \mathcal{B}$
  $\mathcal{C} \coloneqq \mathcal{A} \setminus \mathcal{B}$
  $\mathcal{D} \coloneqq \mathcal{B} \setminus \mathcal{A}$
  if $\mathcal{C} \neq \varnothing$ **then**
    $\mathcal{E} \coloneqq \mathcal{A} \cap \mathcal{B}$
    $Q \coloneqq Q_{\mathcal{E}\mathcal{E}} - Q_{\mathcal{E}\mathcal{E}^c} Q_{\mathcal{E}^c\mathcal{E}^c}^{-1} Q_{\mathcal{E}\mathcal{E}^c}^T$
    $\mathcal{A} \coloneqq \mathcal{E}$
  end if
  if $\mathcal{D} \neq \varnothing$ **then**
    $S \coloneqq X_{\mathcal{D}}^T X_{\mathcal{D}} - X_{\mathcal{D}}^T X_{\mathcal{A}} Q X_{\mathcal{A}}^T X_{\mathcal{D}}$
    $Q \coloneqq \begin{bmatrix} Q + Q X_{\mathcal{A}}^T X_{\mathcal{D}} S^{-1} X_{\mathcal{D}}^T X_{\mathcal{A}} Q & -Q X_{\mathcal{A}}^T X_{\mathcal{D}} S^{-1} \\ -S^{-1} X_{\mathcal{D}}^T X_{\mathcal{A}} Q & S^{-1} \end{bmatrix}$
  end if
  Return H^*

Algorithm 2 The Hessian screening method for the ordinary least-squares lasso

Input: $X \in \mathbb{R}^{n \times p}$, $y \in \mathbb{R}^n$, $\lambda \in \{\mathbb{R}_+^m : \lambda_1 = \lambda_{\max}, \lambda_1 > \lambda_2 > \cdots > \lambda_m\}$, $\varepsilon > 0$
Initalize: $k \leftarrow 1$, $\beta^{(0)} \leftarrow 0$, $\zeta \leftarrow \|y\|_2^2$, $\mathcal{W} \leftarrow \varnothing$, $\mathcal{A} \leftarrow \varnothing$, $\mathcal{S} \leftarrow \varnothing$, $\mathcal{G} \leftarrow \{1, 2, \dots, p\}$

while $k \leq m$ **do**
  $\beta_{\mathcal{W}}^{(k)} \leftarrow \big\{\beta \in \mathbb{R}^{|\mathcal{W}|} : G\big(\beta, (y - X_{\mathcal{W}}\beta)/\max(\lambda_k, \|X_{\mathcal{W}}^T(y - X_{\mathcal{W}}\beta)\|_\infty)\big) < \zeta\varepsilon\big\}$
  $\beta_{\mathcal{W}^c}^{(k)} \leftarrow 0$
  $\mathcal{A} \leftarrow \{j : \beta_j \neq 0\}$
  $r \leftarrow y - X_{\mathcal{W}}\beta_{\mathcal{W}}^{(k)}$
  $\mathcal{V} \leftarrow \{j \in \mathcal{S} \setminus \mathcal{W} : |x_j^T r| \geq \lambda_k\}$ ▷ *Check for violations in Strong set*
  if $\mathcal{V} = \varnothing$ **then**
    $\theta \leftarrow r/\max\big(\lambda_k, \|X_{\mathcal{G}}^T r\|_\infty\big)$ ▷ *Compute dual-feasible point*
    if $G(\beta^{(k)}, \theta) < \varepsilon\zeta$ **then**
      Update H and H^{-1} via Algorithm 1
      $\mathcal{W} \leftarrow \{j : |\tilde{c}^H(\lambda_{k+1})| < \lambda_{k+1}\} \cup \mathcal{A}$ ▷ *Hessian rule screening*
      $\mathcal{S} \leftarrow \{j : |\tilde{c}^S(\lambda_{k+1})| < \lambda_{k+1}\}$ ▷ *Strong rule screening*
      Initialize $\beta_{\mathcal{A}}^{(k+1)}$ using (7) ▷ *Hessian warm start*
      $\mathcal{G} \leftarrow \{1, 2, \dots, p\}$ ▷ *Reset Gap-Safe set*
      $k \leftarrow k + 1$ ▷ *Move to next step on path*
    else
      $\mathcal{G} \leftarrow \Big\{j \in \mathcal{G} : |x_j^T\theta| \geq 1 - \|x_j\|_2\sqrt{2G(\beta^{(k)}, \theta)/\lambda_k^2}\Big\}$ ▷ *Gap-Safe screening*
      $\mathcal{V} \leftarrow \{j \in \mathcal{G} \setminus (\mathcal{S} \cup \mathcal{W}) : |x_j^T r| \geq \lambda_k\}$ ▷ *Check for violations in Gap-Safe set*
      $\mathcal{W} \leftarrow \mathcal{W} \cap \mathcal{G}$
      $\mathcal{S} \leftarrow \mathcal{S} \cap \mathcal{G}$
    end if
  end if
  $\mathcal{W} \leftarrow \mathcal{W} \cup \mathcal{V}$ ▷ *Augment working set with violating predictors*
end while
return β

Proof. $\sum_{j\in\mathcal{E}} x_j e_j = 0$ by assumption. Then, since $\hat{\beta}(\lambda)$ is the solution to the primal problem, it follows that $x_j^T \nabla f(X\beta) = \operatorname{sign}(\beta_j)\lambda$ for all $j \in \mathcal{E}$. Hence

$$\sum_{j\in\mathcal{E}} x_j^T \nabla f(X\beta) e_j = \sum_{j\in\mathcal{E}} \operatorname{sign}(\beta_j)\lambda e_j = \lambda \sum_{j\in\mathcal{E}} \operatorname{sign}(\beta_j) e_j = 0$$

and $z_{\mathcal{E}^C} = 0$, $z_{\mathcal{E}} = \operatorname{sign}(\beta_{\mathcal{E}})$. □

In our opinion, the most salient feature of this result is that if all predictors in $\mathcal{E}$ except i are known to be active, then predictor i is active iff $e_i = \sum_{j\in\mathcal{E}\setminus i} \pm e_j$. If the columns of X are independent and normally distributed, this cannot occur and hence one will never see a null space in $X_{\mathcal{A}}$. Yet if $X_{ij} \in \{0, 1\}$, one should expect the null space to be non-empty frequently. A simple instance of this occurs when the columns of X are duplicates, in which case $|e| = 2$.

Duplicated predictors are fortunately easy to handle since they enter the model simultaneously. And we have, in our program, implemented measures that deal efficiently with this issue by dropping them from the solution after fitting and adjust $\hat{\beta}$ accordingly.

Dealing with the presence of rank-deficiencies due to the existence of linear combinations among the predictors is more challenging. In the work for this paper, we developed a strategy to deal with this issue directly by identifying such linear combinations through spectral decompositions. During our experiments, however, we discovered that this method often runs into numerical issues that require other modifications that invalidate its potential. We have therefore opted for a different strategy.

To deal with singularities and ill-conditioned Hessian matrices, we instead use preconditioning. At step k, we form the spectral decomposition

$$H_{\mathcal{A}_k} = Q\Lambda Q^T.$$

Then, if $\min_i \big(\operatorname{diag}(\Lambda)\big) < \alpha$, we add a factor α to the diagonal of $H_{\mathcal{A}_k}$. Then we substitute

$$\hat{H}_{\mathcal{A}_k}^{-1} = Q^T(I\alpha + \Lambda)^{-1}Q$$

for the true Hessian inverse. An analogous approach is taken when updating the Hessian incrementally as in Algorithm 1. In our experiments, we have set $\alpha := n10^{-4}$.

D Computational Setup Details

The computer used to run the experiments had the following specifications:

CPU Intel i7-10510U @ 1.80Ghz (4 cores)

Memory 64 GB (3.2 GB/core)

OS Fedora 36

Compiler GNU GCC compiler v9.3.0, C++17

BLAS/LAPACK OpenBLAS v0.3.8

R version 4.1.3

E Real Data Sets

All of the data sets except *arcene*, *scheetz*, and *bc_tcga* werew retrieved from `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/` [1, 2]. arcene was retrieved from `https://archive.ics.uci.edu/ml/datasets/Arcene` [3, 4] and scheetz and bc_tcga from `https://myweb.uiowa.edu/pbreheny` [5]. Their original sources have been listed in Table 1. In each case where it is available we use the training partition of the data set and otherwise the full data set.

F Additional Results

In this section, we present additional results related to the performance of the Hessian Screening Rule.

Table 1: Source for the real data sets used in our experiments.

Dataset	Sources
arcene	Guyon et al. [3] and Dua and Graff [4]
bcTCGA	National Cancer Institute [6]
colon-cancer	Alon et al. [7]
duke-breast-cancer	West et al. [8]
e2006-log1p	Kogan et al. [9]
e2006-tfidf	Kogan et al. [9]
ijcnn1	Prokhorov [10]
madelon	Guyon et al. [3]
news20	Keerthi and DeCoste [11]
rcv1	Lewis et al. [12]
scheetz	Scheetz et al. [13]
YearPredictionMSD	Bertin-Mahieux et al. [14] and Dua and Graff [4]

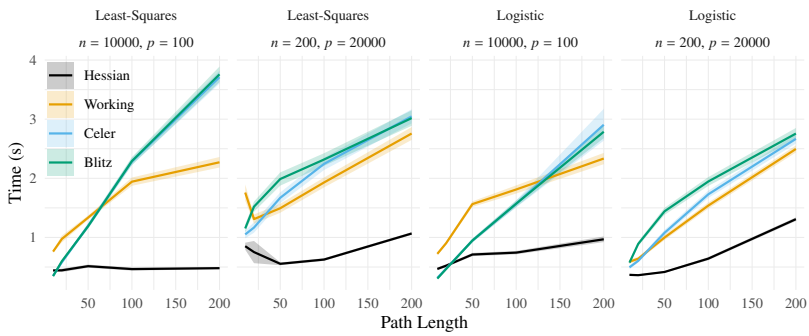


Figure 1: The time in seconds required to fit a full regularization path with length given on the x axis.

F.1 Path Length

Using the same setup as in Section 4 but with $n = 200$, $p = 20\,000$ for the high-dimensional setting, we again benchmark the time required to fit a full regularization path using the different methods studied in this paper. The results (Figure 1) show that the Hessian Screening Method out-performs the studied alternatives except for the low-dimensional situation and a path length of 10 λs. The results demonstrate that our method pays a much smaller price for increased path resolution compared to the other methods but that the increased marginal costs of updating the Hessian may make the method less appealing in this case.

F.2 Convergence Tolerance

To better understand if and how the stopping threshold used in the solver affects the performance of the various methods we test, we conduct simulations where we vary the tolerance, keeping the remaining parameters constant. We use the same situation as in the high-dimensional scenario (see Section 4) but use $n = 200$, $p = 20\,000$. We run the experiment for tolerances 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6}. The results (Figure 2) indicate that the choice of stopping threshold has some importance for convergence time but that the gap between our method and the alternatives tested never disappears.

F.3 The Benefit of Augmenting Heuristic Methods with Gap Safe Screening

To study the effectiveness of augmenting the Hessian Screening and working methods with a gap-safe check, we conduct experiments using the high-dimensional setup in Section 4 but with $n = 200$ and

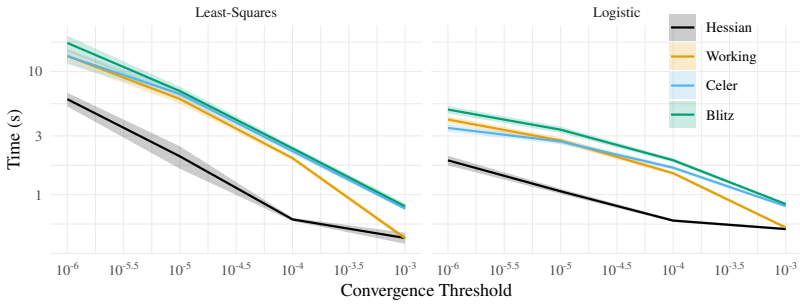


Figure 2: Time required to fit a full regularization path for the high-dimensional scenario setup in Section 4 for both ℓ_1-regularized least-squares and logistic regression, with $n = 200$ and $p = 20\,000$. Both the x and y axis are on a $\log_{10}$ scale.

$p = 20\,000$, either enabling this augmentation or disabling it. We also vary the level of correlation, ρ. Each combination is benchmarked across 20 iterations.

The results indicate that the addition of gap safe screening makes a definite, albeit modest, contribution to the performance of the methods, particularly in the case of the working strategy, which is to be expected given that the working strategy typically runs more KKT checks that the Hessian method does since it causes many more violations.

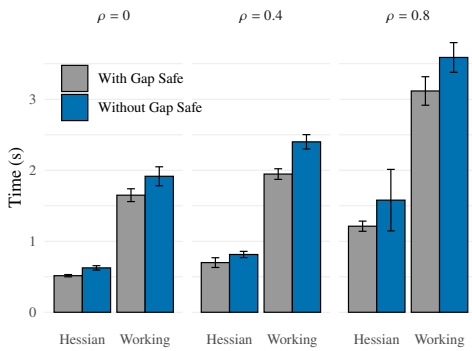


Figure 3: Average time in seconds required to fit a full regularization path for the high-dimensional scenario setup in Section 4 for ℓ_1-regularized least-squares regression, with $n = 200$ and $p = 20\,000$, using the Hessian and working set methods with or without the addition of Gap Safe screening. The bars represent ordinary 95% confidence intervals.

F.4 Effectiveness and Violations

To study the effectiveness of the screening rule, we conduct as experiment using the setup in Section 4 but with $n = 200$ and $p = 20\,000$. We run 20 iterations and average the number of screened predictors as well as violations across the entire path.

Looking at the effectiveness of the screening rules, we see that the Hessian screening rule performs as desired for both ℓ_1-regularized least-squares and logistic regression (Figure 4), leading to a screened set that lies very close to the true size. In particular, the rule works much better than all alternatives in the case of high correlation,

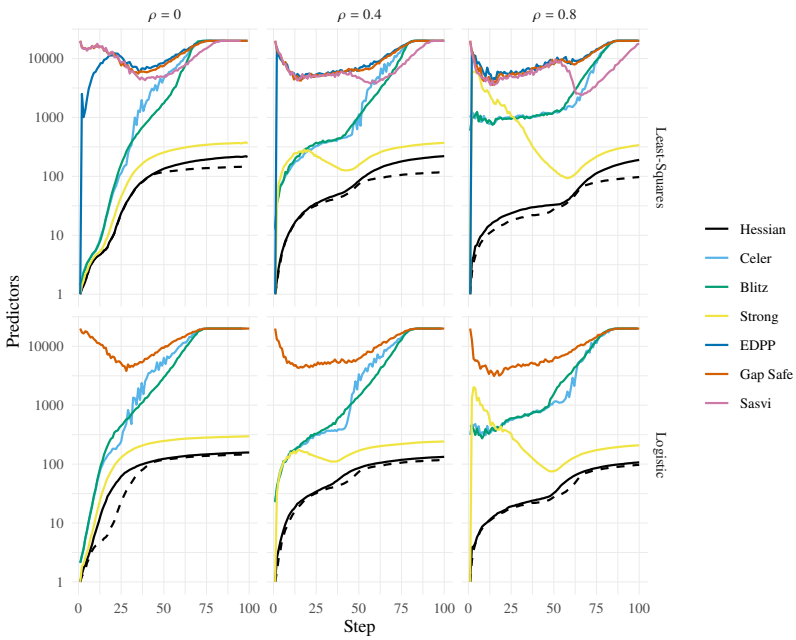


Figure 4: The number of predictors screened (included) for each given screening rule, as well as the minimum number of active predictors at each step as a dashed line. The values are averaged over 20 repetitions in each condition. Note that the y-axis is on a $\log_{10}$ scale.

In Table 2, we show the average numbers of screened (included) predictors and violations for the heuristic screening rules across the path. We note, first, that EDPP never lead to any violations and that the Strong rule only did so once throughout the experiments. The Hessian rule, on the other hand, leads to more violations, particularly when there is high correlation. On the other hand, the Hessian screening rule successfully discards many more predictors than the other two rules do. And because the Hessian method always checks for violations in the strong rule set first, which is demonstrably conservative, these violations are of little importance in practice.

F.5 Detailed Results on Real Data

In Table 3 we show Table 1 with additional detail, including confidence intervals and higher figure resolutions. Please see Section 4 for commentary on these results, where they have been covered in full.

F.6 Additional Results on Simulated Data

In Figure 5, we show results for the ordinary least-squares lasso for the Sasvi, Gap Safe, and EDPP methods, which were not included in the main paper.

F.7 Gamma

In this section we present the results of experiments targeting γ, the parameter for the Hessian rule that controls how much of the unit bound (used in the Strong Rule) that is included in the correlation vector estimate from the Hessian rule.

Table 2: Numbers of screened predictors and violations averaged over the entire path and 20 iterations for simulated data with $n = 20\,000$ $p = 200$ and correlation level equal to ρ.

Model	ρ	Method	Screened	Violations
Least-Squares	0	Hessian	112	0.081
Least-Squares	0	Strong	203	0
Least-Squares	0	EDPP	11 928	0
Least-Squares	0.4	Hessian	103	0.099
Least-Squares	0.4	Strong	238	0
Least-Squares	0.4	EDPP	10 561	0
Least-Squares	0.8	Hessian	66	0.37
Least-Squares	0.8	Strong	897	0.0010
Least-Squares	0.8	EDPP	10 652	0
Logistic	0	Hessian	102	0.020
Logistic	0	Strong	201	0
Logistic	0.4	Hessian	77	0.033
Logistic	0.4	Strong	173	0
Logistic	0.8	Hessian	49	0.051
Logistic	0.8	Strong	297	0

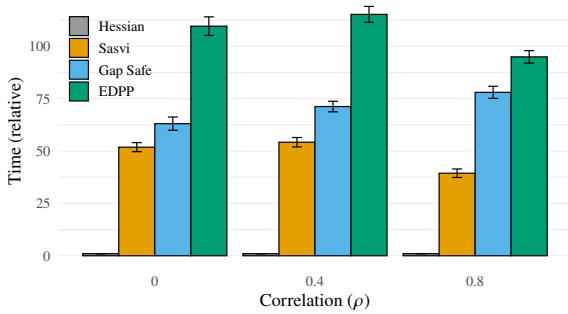


Figure 5: Additional results on simulated data for methods not included in the main article. The results correspond to the ordinary (least-squares) lasso with $n = 400$, $p = 40\,000$ and varying levels of pairwise correlation between predictors, ρ.

We run 50 iterations of the high-dimensional setup from Section 4 and measure the number of predictors screened (included) by the Hessian screening rule, the number of violations, and the time taken to fit the full path. We vary γ from 0.001 to 0.3.

The results are presented in Figure 6. From the figure it is clear that the number of violations in fact has a slightly negative impact on the speed at which the path is fit. We also see that the number of violations is small considering the dimension of the data set ($p = 40\,000$) and approach zero at γ values around 0.1 for the lowest level of correlation, but have yet to reach exactly zero at 0.3 for the highest level of correlation. The size of the screened set increase only marginally as γ increases fro 0.001 to 0.01, but eventually increase rapidly at γ approaches 0.3. Note, however, that the screened set is still very small relative to the full set of predictors.

F.8 Ablation Analysis

In this section we report an experiment wherein we study the effects of the various features of the Hessian screening method by incrementally adding them and timing the result.

We add features incrementally in the following order, such that each step includes all of the previous features.

1. Hessian screening

Table 3: Time to fit a full regularization path of ℓ_1-regularized least-squares and logistic regression to real data sets. Density and time values are rounded to two and four significant figures respectively. The estimates are based on 20 repetitions for arcene, colon-cancer, duke-breast-cancer, and ijcnn1 and three repetitions otherwise. Standard 95% confidence levels are included.

							95% CI	
Dataset	n	p	Density	Loss	Method	Time (s)	Lower	Upper
arcene	100	10 000	5.4×10^{-1}	Logistic	Blitz	4.42	4.39	4.45
arcene	100	10 000	5.4×10^{-1}	Logistic	Celer	3.99	3.98	3.99
arcene	100	10 000	5.4×10^{-1}	Logistic	Hessian	4.35	4.32	4.38
arcene	100	10 000	5.4×10^{-1}	Logistic	Working	3.27	3.25	3.28
bcTCGA	536	17 322	1	Least-Squares	Blitz	11.7	11.5	11.8
bcTCGA	536	17 322	1	Least-Squares	Celer	10.6	10.5	10.7
bcTCGA	536	17 322	1	Least-Squares	Hessian	3.00	2.85	3.14
bcTCGA	536	17 322	1	Least-Squares	Working	7.67	7.57	7.77
colon-cancer	62	2000	1	Logistic	Blitz	0.177	0.176	0.178
colon-cancer	62	2000	1	Logistic	Celer	0.169	0.168	0.170
colon-cancer	62	2000	1	Logistic	Hessian	0.0542	0.0534	0.0550
colon-cancer	62	2000	1	Logistic	Working	0.134	0.132	0.136
duke-breast-cancer	44	7129	1	Logistic	Blitz	0.251	0.248	0.253
duke-breast-cancer	44	7129	1	Logistic	Celer	0.262	0.260	0.264
duke-breast-cancer	44	7129	1	Logistic	Hessian	0.111	0.110	0.112
duke-breast-cancer	44	7129	1	Logistic	Working	0.210	0.209	0.212
e2006-log1p	16 087	4 272 227	1.4×10^{-3}	Least-Squares	Blitz	756	749	764
e2006-log1p	16 087	4 272 227	1.4×10^{-3}	Least-Squares	Celer	835	831	839
e2006-log1p	16 087	4 272 227	1.4×10^{-3}	Least-Squares	Hessian	205	203	207
e2006-log1p	16 087	4 272 227	1.4×10^{-3}	Least-Squares	Working	438	434	441
e2006-tfidf	16 087	150 360	8.3×10^{-3}	Least-Squares	Blitz	277	275	280
e2006-tfidf	16 087	150 360	8.3×10^{-3}	Least-Squares	Celer	335	334	337
e2006-tfidf	16 087	150 360	8.3×10^{-3}	Least-Squares	Hessian	14.3	14.3	14.4
e2006-tfidf	16 087	150 360	8.3×10^{-3}	Least-Squares	Working	143	139	146
ijcnn1	35 000	22	1	Logistic	Blitz	4.68	3.82	5.53
ijcnn1	35 000	22	1	Logistic	Celer	3.50	3.42	3.58
ijcnn1	35 000	22	1	Logistic	Hessian	0.939	0.869	1.01
ijcnn1	35 000	22	1	Logistic	Working	5.53	4.57	6.48
madelon	2000	500	1	Logistic	Blitz	240	223	258
madelon	2000	500	1	Logistic	Celer	247	243	251
madelon	2000	500	1	Logistic	Hessian	48.2	43.2	53.1
madelon	2000	500	1	Logistic	Working	232	227	238
news20	19 996	1 355 191	3.4×10^{-4}	Logistic	Blitz	2230	2230	2240
news20	19 996	1 355 191	3.4×10^{-4}	Logistic	Celer	2170	2160	2180
news20	19 996	1 355 191	3.4×10^{-4}	Logistic	Hessian	1290	1290	1290
news20	19 996	1 355 191	3.4×10^{-4}	Logistic	Working	1620	1610	1630
rcv1	20 242	47 236	1.6×10^{-3}	Logistic	Blitz	384	380	387
rcv1	20 242	47 236	1.6×10^{-3}	Logistic	Celer	378	373	384
rcv1	20 242	47 236	1.6×10^{-3}	Logistic	Hessian	132	127	137
rcv1	20 242	47 236	1.6×10^{-3}	Logistic	Working	266	258	275
scheetz	120	18 975	1	Least-Squares	Blitz	0.706	0.689	0.722
scheetz	120	18 975	1	Least-Squares	Celer	0.801	0.777	0.826
scheetz	120	18 975	1	Least-Squares	Hessian	0.369	0.354	0.383
scheetz	120	18 975	1	Least-Squares	Working	0.643	0.639	0.647
YearPredictionMSD	463 715	90	1	Least-Squares	Blitz	706	704	707
YearPredictionMSD	463 715	90	1	Least-Squares	Celer	712	711	714
YearPredictionMSD	463 715	90	1	Least-Squares	Hessian	78.8	78.1	79.5
YearPredictionMSD	463 715	90	1	Least-Squares	Working	541	516	565

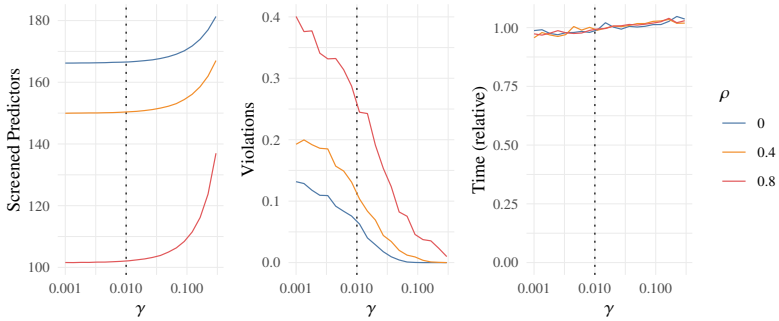


Figure 6: The number of predictors screened (included), the number of violations, and the time taken to fit the full path. All measures in the plots represent means across combinations of ρ and γ over 50 iterations. The time recorded here is the time relative to the mean time for each level of ρ. The choice of γ in this work, 0.01, is indicated by a dotted line in the plots. Note that x is on a $\log_{10}$ scale.

2. Hessian warm starts
3. Effective updates of the Hessian matrix and its inverse using the sweep operator
4. Gap safe screening

We then run an experiment on a design with $n = 200$ and $p = 20\,000$ and two levels of pairwise correlation between the predictors. The results (Figure 7) show that both screening and warm starts make considerable contributions in this example.

Note that these results are conditional on the order with which they are added and also on the specific design. The Hessian updates, for instance, make a larger contribution when $\min\{n, p\}$ is larger and n and p are more similar. And when $n \gg p$, the contribution of the warm starts dominate whereas screening no longer plays as much of a role.

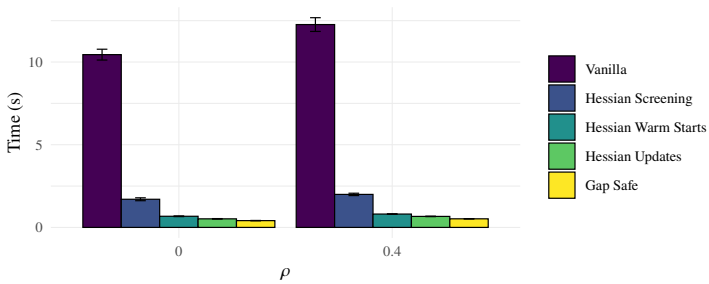


Figure 7: Incremental contribution to the decrease in running time from Hessian screening, Hessian warm starts, our effective updates of the Hessian and its inverse, and gap safe screening. In other words, *Gap Safe*, for instance, includes *all* of the other features, whilst *Hessian Warm Starts* includes only *Hessian Screening*. *Vanilla* does not include any screening and only uses standard warm starts (from the solution at the previous step along the path). The example shows an example of ordinary (least-squares) lasso fit to a design with $n = 200$ and $p = 20\,000$ with pairwise correlation between predictors given by ρ. (See Section 4 for more details on the setup). The error bars indicate standard 95% confidence intervals. The results are based on 10 iterations for each condition.

F.9 ℓ_1-Regularized Poisson Regression

In this experiment, we provide preliminary results for ℓ_1-regularized Poisson regression. The setup is the same as Section 4 except for the following remarks:

- The response, y, is randomly sampled such that $y_i \sim \text{Poisson}\left(\exp(x_i^T \beta)\right)$.
- We set ζ in the convergence criterion to $n + \sum_{i=1}^{n} \log(y_i!)$.
- We do not use the line search procedure from Blitz.
- Due to convergence issues for higher values of ρ, we use values 0.0, 0.15, and 0.3 here. Tackling higher values of ρ would likely need considerable modifications to the coordinate descent solver we use.
- The gradient of the negative Poisson log-likelihood is not Lipschitz continuous, which means that Gap safe screening [15] no longer works. As a result, we have excluded the Blitz and Celer algorithms, which rely on Gap safe screening, from these benchmarks, and deactivated the additional Gap safe screening from our algorithm.

The results from the comparison are shown in Figure 8, showing that our algorithm is noticeably faster than the working algorithm also in this case.

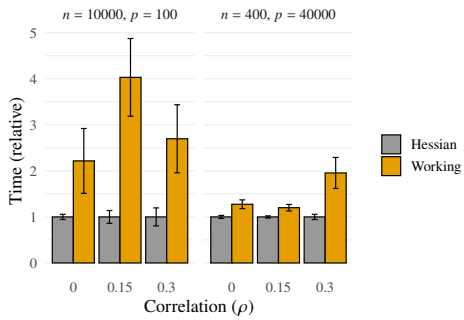


Figure 8: Time to fit a full regularization path for ℓ_1-regularized Poisson regression to a design with observations, p predictors, and pairwise correlation between predictors of ρ. Time is relative to the minimal mean time in each group. The error bars represent ordinary 95% confidence intervals around the mean.

F.10 Runtime Breakdown Along Path

In this section we take a closer look at the running time of fitting the full regularization path and study the impact the Hessian screening rule and its warm starts have on the time spent on optimization of the problem using coordinate descent (CD).

To illustrate these cases we take a look at three data sets here: *e2006-tfidf*, *madelon*, and *rcv1*. The first of these, e2006-tfidf, is a sparse data set of dimensions $16\,087 \times 150\,360$ with a numeric response, to which we fit the ordinary lasso. The second two are both data sets with a binary response, for which we use ℓ_1-regularized logistic regression. The dimensions of madelon are 2000×500 and the dimensions of rcv1 are $20\,242 \times 47\,236$.

We study the contribution to the total running time per step, comparing the Hessian screening rule with the working+ strategy. For the working+ strategy, all time is spent inside the CD optimizer and in checks of the KKT conditions. For the Hessian screening rule, time is also spent updating the Hessian and computing the correlation estimate $\tilde{c}^H$.

Beginning with Figure 9 we see that the Hessian strategy dominates the Working+ strategy, which spends most of its running time on coordinate descent iterations, which the Hessian strategy ensures are completed in much less time.

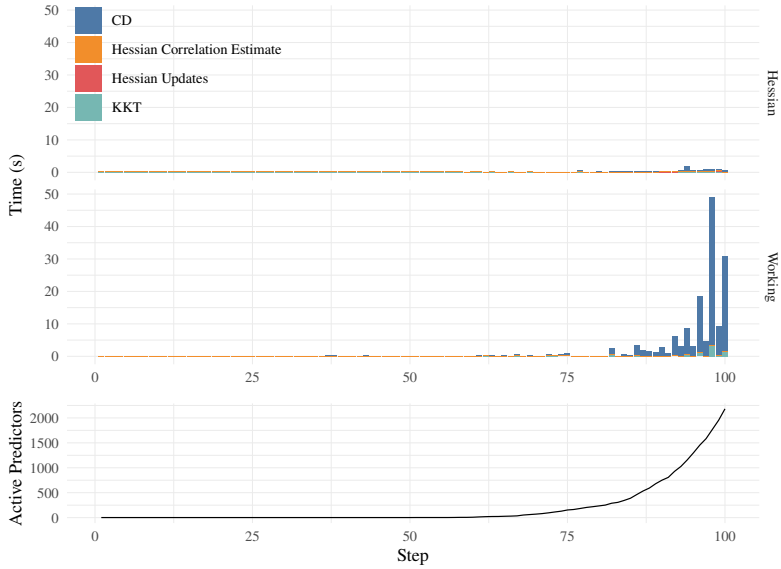


Figure 9: Relative contribution to the full running time when fitting a complete regularization path to the *e2006-tfidf* data set.

In Figure 10, we see an example of ℓ_1-regularized logistic regression. In this case updating the Hessian exactly (and directly) dominates the other approaches. The size of the problem makes the cost of updating the Hessian negligible and offers improved screening and warm starts, which in turn greatly reduces the time spent on coordinate descent iterations and consequently the full time spent fitting the path.

Finally, in Figure 11 we consider the *rcv1* data set. In contrast to the case for *madelon*, the cost of directly forming the Hessian (and inverse) proves more time-consuming here (although the benefits still show in the time spent on coordinate descent iterations).

As a final remark, note that the pattern by which predictors enter the model (bottom panels) differ considerably between these three cases (Figures 9 to 11). Consider, for instance, *madelon* viz-a-viz *e2006-tfidf*. In *Approximate Homotopy* (Section 3.3.6), we discuss a remedy for this solution that is readily available through our method.

References

[1] Chih-chung Chang and Chih-jen Lin. "LIBSVM: A Library for Support Vector Machines". In: *ACM Transactions on Intelligent Systems and Technology* 2.3 (May 2011), 27:1–27:27. DOI: 10.1145/1961189.1961199.

[2] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM Data: Classification, Regression, and Multi-Label*. LIBSVM - A library for Support Vector Machines. Dec. 22, 2016. URL: https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/ (visited on 03/12/2018).

[3] Isabelle Guyon et al. "Result Analysis of the NIPS 2003 Feature Selection Challenge". In: *Advances in Neural Information Processing Systems 17*. Neural Information Processing Systems 2004. Ed. by L. K. Saul, Y. Weiss, and L. Bottou. Vancouver, BC, Canada: MIT Press, Dec. 13–18, 2004, pp. 545–552. ISBN: 978-0-262-19534-8.

[4] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. UCI machine learning repository. 2019. URL: http://archive.ics.uci.edu/ml (visited on 04/14/2021).

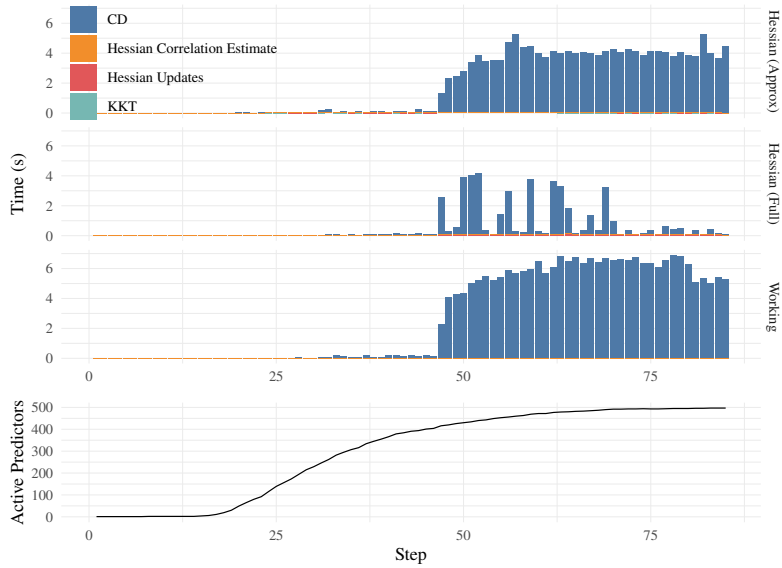


Figure 10: Relative contribution to the full running time when fitting a complete regularization path to the *madelon* data set.

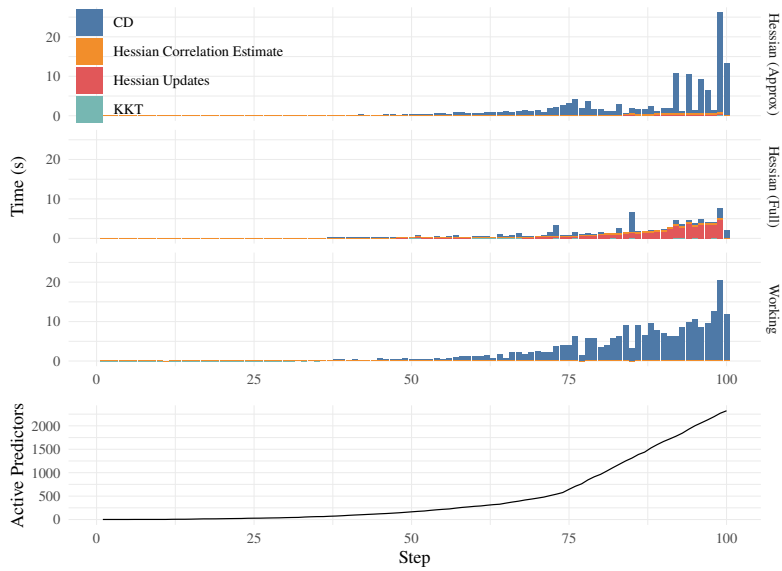


Figure 11: Relative contribution to the full running time when fitting a complete regularization path to the *rcv1* data set.

[5] Patrick Breheny. *Patrick Breheny*. Patrick Breheny. May 16, 2022. URL: https://myweb.uiowa.edu/pbreheny/ (visited on 05/17/2022).

[6] National Cancer Institute. *The Cancer Genome Atlas Program*. National Cancer Institute. May 16, 2022. URL: https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga (visited on 05/17/2022).

[7] U. Alon et al. "Broad Patterns of Gene Expression Revealed by Clustering Analysis of Tumor and Normal Colon Tissues Probed by Oligonucleotide Arrays". In: *Proceedings of the National Academy of Sciences* 96.12 (June 8, 1999), pp. 6745–6750. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.96.12.6745. pmid: 10359783.

[8] M. West et al. "Predicting the Clinical Status of Human Breast Cancer by Using Gene Expression Profiles". In: *Proceedings of the National Academy of Sciences of the United States of America* 98.20 (Sept. 25, 2001), pp. 11462–11467. ISSN: 0027-8424. DOI: 10.1073/pnas.201162998. pmid: 11562467.

[9] Shimon Kogan et al. "Predicting Risk from Financial Reports with Regression". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. NAACL-HLT 2009. Boulder, Colorado: Association for Computational Linguistics, June 2009, pp. 272–280.

[10] Danil Prokhorov. "IJCNN 2001 Neural Network Competition". Oral Presentation. IJCNN01 (Washington, DC, USA). July 15–19, 2001.

[11] S. Sathiya Keerthi and Dennis DeCoste. "A Modified Finite Newton Method for Fast Solution of Large Scale Linear SVMs". In: *The Journal of Machine Learning Research* 6.12 (Dec. 1, 2005), pp. 341–361. ISSN: 1532-4435.

[12] David D. Lewis et al. "RCV1: A New Benchmark Collection for Text Categorization Research". In: *The Journal of Machine Learning Research* 5 (Dec. 1, 2004), pp. 361–397. ISSN: 1532-4435.

[13] Todd E. Scheetz et al. "Regulation of Gene Expression in the Mammalian Eye and Its Relevance to Eye Disease". In: *Proceedings of the National Academy of Sciences* 103.39 (Sept. 26, 2006), pp. 14429–14434. DOI: 10.1073/pnas.0602562103.

[14] Thierry Bertin-Mahieux et al. "The Million Song Dataset". In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*. ISMIR 2011. Miami, FL, USA: University of Miami, Oct. 24–28, 2011. ISBN: 978-0-615-54865-4. DOI: 10.7916/D8NZ8J07.

[15] Eugene Ndiaye et al. "Gap Safe Screening Rules for Sparsity Enforcing Penalties". In: *Journal of Machine Learning Research* 18.128 (2017), pp. 1–33.

IV

Benchopt: Reproducible, efficient and collaborative optimization benchmarks

Thomas Moreau[1,*], **Mathurin Massias**[2,*], **Alexandre Gramfort**[1,*], **Pierre Ablin**[3], **Pierre-Antoine Bannier, Benjamin Charlier**[4], **Mathieu Dagréou**[1], **Tom Dupré la Tour**[6] **Ghislain Durif**[4], **Cassio F. Dantas**[7], **Quentin Klopfenstein**[8], **Johan Larsson**[9], **En Lai**[1], **Tanguy Lefort**[4], **Benoit Malézieux**[1], **Badr Moufad**[2], **Binh T. Nguyen**[10], **Alain Rakotomamo Zaccharie Ramzi**[12], **Joseph Salmon**[4,5], **Samuel Vaiter**[13]

[1] Université Paris-Saclay, Inria, CEA, 91120 Palaiseau, France
[2] Univ Lyon, Inria, CNRS, ENS de Lyon, UCB Lyon 1, LIP UMR 5668, F-69342, Lyon, Fran
[3] Université Paris-Dauphine, PSL University, CNRS, 75016, Paris, France
[4] IMAG, Univ Montpellier, CNRS, Montpellier, France [5] Institut Universitaire de France (IU
/ersity of California, Berkeley, CA 94720, USA [7] TETIS, Univ Montpellier, INRAE, Montpellier,
[8] University of Luxembourg, LCSB, Esch-sur-Alzette, Luxembourg
[9] The Department of Statistics, Lund University [10] LTCI, Télécom Paris, 91120 Palaiseau, Fra
[11] Criteo AI Lab, Paris, France [12] ENS Ulm, CNRS, UMR 8553, Paris, France
[13] CNRS & Université Côte d'Azur, Laboratoire J.A. Dieudonné, CNRS, Nice, France

Abstract

Numerical validation is at the core of machine learning research as it allows to assess the actual impact of new methods, and to confirm the agreement between theory and practice. Yet, the rapid development of the field poses several challenges: researchers are confronted with a profusion of methods to compare, limited transparency and consensus on best practices, as well as tedious re-implementation work. As a result, validation is often very partial, which can lead to wrong conclusions that slow down the progress of research. We propose `Benchopt`, a collaborative framework to automate, reproduce and publish optimization benchmarks in machine learning across programming languages and hardware architectures. `Benchopt` simplifies benchmarking for the community by providing an off-the-shelf tool for running, sharing and extending experiments. To demonstrate its broad usability, we showcase benchmarks on three standard learning tasks: ℓ_2-regularized logistic regression, Lasso, and ResNet18 training for image classification. These benchmarks highlight key practical findings that give a more nuanced view of the state-of-the-art for these problems, showing that for practical evaluation, the devil is in the details. We hope that `Benchopt` will foster collaborative work in the community hence improving the reproducibility of research findings.

1 Introduction

Numerical experiments have become an essential part of statistics and machine learning (ML). It is now commonly accepted that every new method needs to be validated through comparisons with existing approaches on standard problems. Such validation provides insight into the method's benefits and limitations and thus adds depth to the results. While research aims at advancing knowledge and not just improving the state of the art, experiments ensure that results are reliable and support theoretical claims (Sculley et al., 2018). Practical validation also helps the ever-increasing number of ML users in applied sciences to choose the right method for their task. Performing rigorous and extensive experiments is, however, time-consuming (Raff, 2019), particularly because comparisons

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

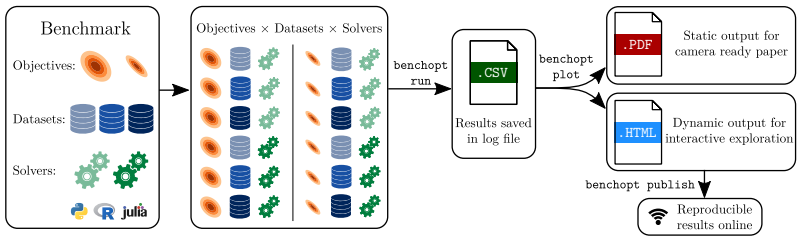


Figure 1: A visual summary of `Benchopt`. Each `Solver` is run (in parallel) on each `Dataset` and each variant of the `Objective`. Results are exported as a `CSV` file that is easily shared and can be automatically plotted as interactive `HTML` visualizations or `PDF` figures.

against existing methods in new settings often requires reimplementing baseline methods from the literature. In addition, ingredients necessary for a proper reimplementation may be missing, such as important algorithmic details, hyperparameter choices, and preprocessing steps (Pineau et al., 2019).

In the past years, the ML community has actively sought to overcome this "reproducibility crisis" (Hutson, 2018) through collaborative initiatives such as open datasets (`OpenML`, Vanschoren et al. 2013), standardized code sharing (Forde et al., 2018), benchmarks (`MLPerf`, Mattson et al. 2020), the NeurIPS and ICLR reproducibility challenges (Pineau et al., 2019; Pineau et al., 2021) and new journals (*e.g.,* Rougier and Hinsen 2018). As useful as these endeavors may be, they do not, however, fully address the problems in optimization for ML since, in this area, there are no clear community guidelines on how to perform, share, and publish benchmarks.

Optimization algorithms pervade almost every area of ML, from empirical risk minimization, variational inference to reinforcement learning (Sra et al., 2012). It is thus crucial to know which methods to use depending on the task and setting (Bartz-Beielstein et al., 2020). While some papers in optimization for ML provide extensive validations (Lueckmann et al., 2021), many others fall short in this regard due to lack of time and resources, and in turn feature results that are hard to reproduce by other researchers. In addition, both performance and hardware evolve over time, which eventually makes static benchmarks obsolete. An illustration of this is the recent work by Schmidt et al. (2021), which extensively evaluates the performances of 15 optimizers across 8 deep-learning tasks. While their benchmark gives an overall assessment of the considered solvers, this assessment is bound to become out-of-date if it is not updated with new solvers and new architectures. Moreover, the benchmark does not reproduce state-of-the-art results on the different datasets, potentially indicating that the considered architectures and optimizers could be improved.

> We firmly believe that this critical task of **maintaining an up-to-date benchmark in a field cannot be solved without a collective effort**. We want to empower the community to take up this challenge and **build a living, reproducible and standardized state of the art that can serve as a foundation for future research.**

`Benchopt` provides the tools to structure the optimization for machine learning (Opt-ML) community around standardized benchmarks, and to aggregate individual efforts for reproducibility and results sharing. `Benchopt` can handle algorithms written in `Python`, `R`, `Julia` or `C/C++` via binaries. It offers built-in functionalities to ease the execution of benchmarks: parallel runs, caching, and automatic results archiving. Benchmarks are meant to evolve over time, which is why `Benchopt` offers a modular structure through which a benchmark can be easily extended with new objective functions, datasets, and solvers by the addition of a single file of code.

The paper is organized as follows. We first detail the design and usage of `Benchopt`, before presenting results on three canonical problems:

- ℓ_2-regularized logistic regression: a convex and smooth problem which is central to the evaluation of many algorithms in the Opt-ML community, and remains of high relevance for practitioners;
- the Lasso: the prototypical example of non-smooth convex problem in ML;
- training of ResNet18 architecture for image classification: a large scale non-convex deep learning problem central in the field of computer vision.

The reported benchmarks, involving dozens of implementations and datasets, shed light on the current state-of-the-art solvers for each problem, across various settings, highlighting that the best algorithm largely depends on the dataset properties (*e.g.,* size, sparsity), the hyperparameters, as well as hardware. A variety of other benchmarks (*e.g.,* MCP, TV1D, etc.) are also presented in Appendix, with the goal to facilitate contributions from the community.

By the open source and collaborative design of `Benchopt` (BSD 3-clause license), we aim to open the way towards community-endorsed and peer-reviewed benchmarks that will improve the tracking of progress in optimization for ML.

2 The Benchopt library

The `Benchopt` library aims to provide a standard toolset and structure to implement benchmarks for optimization in ML, where the problems depend on some input dataset $\mathcal{D}$. The considered problems are of the form

$$\theta^* \in \arg\min_{\theta \in \Theta} f(\theta; \mathcal{D}, \Lambda) \ , \tag{1}$$

where f is the objective function, Λ are its hyperparameters, and Θ is the feasible set for θ. The **scope** of the library is to evaluate optimization methods in their wide sense by considering the sequence $\{\theta^t\}_t$ produced to approximate θ^*. We emphasize than `Benchopt` does not provide a fixed set of benchmarks, but a framework to create, extend and share benchmarks on any problem of the form (1). To provide a flexible and extendable coding standard, benchmarks are defined as the association of three types of object classes:

Objective: It defines the function f to be minimized as well as the hyperparameters Λ or the set Θ, and the metrics to track along the iterations (*e.g.,* objective value, gradient norm for smooth problems, or validation loss). Multiple metrics can be registered for each θ^t.

Datasets: The `Dataset` objects provide the data $\mathcal{D}$ to be passed to the `Objective` class. They control how data is loaded and preprocessed. `Datasets` are separated from the `Objective`, making it easy to add new ones, provided they are coherent with the `Objective`.

Solvers: The `Solver` objects define how to run the algorithm. They are provided with the `Objective` and `Dataset` objects and output a sequence $\{\theta^t\}_t$. This sequence can be obtained using a single run of the method, or with multiple runs in case the method only returns its final iterate.

```
benchmark/
├── objective.py
├── datasets/
│   ├── dataset1.py
│   └── dataset2.py
└── solvers/
    ├── solver1.py
    └── solver2.py
```

Figure 2: Standard benchmark structure

Each of these objects can have parameters that change their behavior, *e.g.,* the regularization parameters for the `Objective`, the choice of preprocessing for the `Datasets`, or the step size for the `Solvers`. By exposing these parameters in the different objects, `Benchopt` can evaluate their influence on the benchmark results. The `Benchopt` library defines an application programming interface (API) for each of these concepts and provides a command line interface (CLI) to make them work together. A benchmark is defined as a folder that contains an `Objective` as well as subfolders containing the `Solvers` and `Datasets`. Appendix B presents a concrete example on Ridge regression of how to construct a `Benchopt` benchmark while additional design design choices of `Benchopt` are discussed in Appendix C.

For each `Dataset` and `Solver`, and for each set of parameters, `Benchopt` retrieves a sequence $\{\theta^t\}_t$ and evaluates the metrics defined in the `Objective` for each θ^t. To ensure fair evaluation, the computation of these metrics is done off-line. The metrics are gathered in a `CSV` file that can be used to display the benchmark results, either locally or as `HTML` files published on a website that reference the benchmarks run with `Benchopt`. This workflow is described in Figure 1.

This modular and standardized organization for benchmarks empowers the optimization community by making numerical experiments easily reproducible, shareable, flexible and extendable. The benchmark can be shared as a git repository or a folder containing the different definitions for the `Objective`, `Datasets` and `Solvers` and it can be run with the `Benchopt` CLI, hence becoming a convenient reference for future comparisons. This ensures fair evaluation of baselines in follow-up experiments, as implementations validated by the community are available. Moreover, benchmarks can be extended easily as one can add a `Dataset` or a `Solver` to the comparison by adding a single

file. Finally, by supporting multiple metrics – *e.g.,* training and testing losses, error on parameter estimates, sparsity of the estimate – the `Objective` class offers the flexibility to define the concurrent evaluation, which can be extended to track extra metrics on a per benchmark basis, depending on the problem at hand.

As one of the goal of `Benchopt` is to make benchmarks as simple as possible, it also provides a set of features to make them easy to develop and run. `Benchopt` is written in `Python`, but `Solvers` run with implementations in different languages (*e.g.,* `R` and `Julia`, as in Section 4) and frameworks (*e.g.,* `PyTorch` and `TensorFlow`, as in Section 5). Moreover, benchmarks can be run in parallel with checkpointing of the results, enabling large scale evaluations on many CPU or GPU nodes. `Benchopt` also makes it possible to run solvers with many different hyperparameters' values , allowing to assess their sensitivity on the method performance. Benchmark results are also automatically exported as interactive visualizations, helping with the exploration of the many different settings.

Benchmarks All presented benchmarks are run on 10 cores of an Intel Xeon Gold 6248 CPUs @ 2.50GHz and NVIDIA V100 GPUs (16GB). The results' interactive plots and data are available at `https://benchopt.github.io/results/preprint_results.html`.

3 First example: ℓ_2-regularized logistic regression

Logistic regression is a very popular method for binary classification. From a design matrix $X \in \mathbb{R}^{n \times p}$ with rows X_i and a vector of labels $y \in \{-1, 1\}^n$ with corresponding element y_i, ℓ_2-regularized logistic regression provides a generalized linear model indexed by $\theta^* \in \mathbb{R}^p$ to discriminate the classes by solving

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^p} \sum_{i=1}^{n} \log\left(1 + \exp(-y_i X_i^\top \theta)\right) + \frac{\lambda}{2}\|\theta\|_2^2 \ , \tag{2}$$

where $\lambda > 0$ is the regularization hyperparameter. Thanks to the regularization part, Problem (2) is strongly convex with a Lipschitz gradient, and thus its solution can be estimated efficiently using many iterative optimization schemes.

The most classical methods to solve this problem take inspiration from Newton's method (Wright and Nocedal, 1999). On the one hand, quasi-Newton methods aim at approximating the Hessian of the cost function with cheap to compute operators. Among these methods, L-BFGS (Liu and Nocedal, 1989) stands out for its small memory footprint, its robustness and fast convergence in a variety of settings. On the other hand, truncated Newton methods (Dembo et al., 1982) try to directly approximate Newton's direction by using *e.g.,* the conjugate gradient method (Fletcher and Reeves, 1964) and Hessian-vector products to solve the associated linear system. Yet, these methods suffer when n is large: each iteration requires a pass on the whole dataset.

In this context, methods based on stochastic estimates of the gradient have become standard (Bottou, 2010), with Stochastic Gradient Descent (SGD) as a main instance. The core idea is to use cheap and noisy estimates of the gradient (Robbins and Monro, 1951; Kiefer and Wolfowitz, 1952). While SGD generally converges either slowly due to decreasing step sizes, or to a neighborhood of the solution for constant step sizes, variance-reduced adaptations such as SAG (Schmidt et al., 2017), SAGA (Defazio et al., 2014) and SVRG (Johnson and Zhang, 2013) make it possible to solve the problem more efficiently and are often considered to be state-of-the-art for large scale problems.

Finally, methods based on coordinate descent (Bertsekas, 1999) have also been proposed to solve Problem (2). While these methods are usually less popular, they can be efficient in the context of sparse datasets, where only few samples have non-zero values for a given feature, or when accelerated on distributed systems or GPU (Dünner et al., 2018).

The code for the benchmark is available at `https://github.com/benchopt/benchmark_logreg_l2/`. To reflect the diversity of solvers available, we showcase a `Benchopt` benchmark with 3 datasets, 10 optimization strategies implemented in 5 packages, leveraging GPU hardware when possible. We also consider different scenarios for the objective function: (i) **scaling** (or not) the features, a recommended data preprocessing step, crucial in practice to have comparable regularization strength on all variables; (ii) fitting (or not) an unregularized **intercept** term, important in practice and making optimization harder when omitted from the regularization term (Koh et al., 2007); (iii) working (or

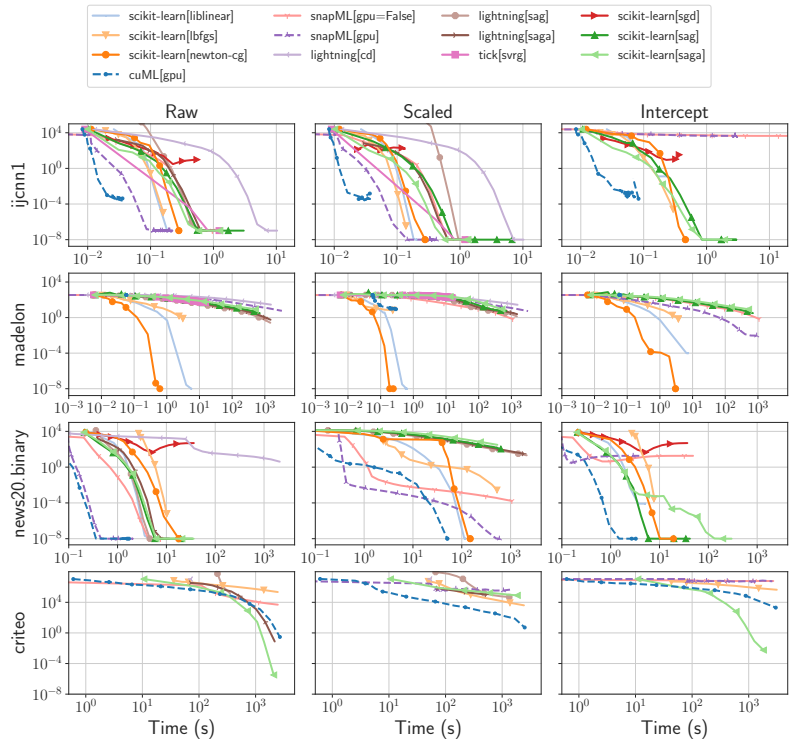


Figure 3: Benchmark for the ℓ_2-regularized logistic regression, on 13 solvers, 4 datasets (*rows*), and 3 variants of the `Objective` (*columns*) with $\lambda = 1$. The curves display the suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, as a function of time. The first column corresponds to the objective function detailed in Problem (2). In the second column, datasets were preprocessed by normalizing each feature to unit standard deviation. The third column is for an objective function which includes an unregularized intercept.

not) with **sparse** features, which prevent explicit centering during preprocessing to keep memory usage limited. Details on packages, datasets and additional scenarios are available in Appendix D.

Results Figure D.1 presents the results of the benchmarks, in terms of suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, for three datasets and three scenarios. Here, because the problem is convex, θ^* is approximated by the best iterate across all runs (see Section C.1). Overall, the benchmark shows the benefit of using GPU solvers (`cuML` and `snapML`), even for small scale tasks such as *ijcnn1*. Note that these two accelerated solvers converge to a higher suboptimality level compared to other solvers, due to operating with 32-bit float precision. Another observation is that data scaling can drastically change the picture. In the case of *madelon*, most solvers have a hard time converging for the scaled data. For the solvers that converge, we note that the convergence time is one order of magnitude smaller with the scaled dataset compared to the raw one. This stems from the fact that in this case, the scaling improves the conditioning of the dataset.[1] For *news20.binary*, the stochastic solvers such as SAG and SAGA have degraded performances on scaled data. Here, the scaling makes the problem harder.[2]

[1]The condition number of the dataset is divided by 5.9 after scaling.
[2]The condition number is multiplied by 407 after scaling.

On CPU, quasi-Newton solvers are often the most efficient ones, and provide a reasonable choice in most situations. For large scale *news20.binary*, stochastic solvers such as SAG, SAGA or SVRG –that are often considered as state of the art for such problem– have worst performances for the presented datasets. While this dataset is often used as a test bed for benchmarking new stochastic solvers, we fail to see an improvement over non-stochastic ones for this experimental setup. In contrast, the last row in Figure D.1 displays an experiment with the larger scale *criteo* dataset, which demonstrates a regime where variance-reduced stochastic gradient methods outperform quasi-Newton methods. For future benchmarking of stochastic solvers, we therefore recommend using such a large dataset.

Finally, the third column in Figure D.1 illustrates a classical problem when benchmarking different solvers: their specific (and incompatible) definition and resolution of the corresponding optimization problem. Here, the objective function is modified to account for an intercept (bias) in the linear model. In most situations, this intercept is not regularized when it is fitted. However, `snapML` and `liblinear` solvers do regularize it, leading to incomparable losses.

4 Second example: The Lasso

The Lasso, (Tibshirani, 1996; Chen et al., 1998), is an archetype of non-smooth ML problems, whose impact on ML, statistics and signal processing in the last three decades has been considerable (Bühlmann and van de Geer, 2011; Hastie et al., 2015). It consists of solving

$$\theta^* \in \arg\min_{\theta \in \mathbb{R}^p} \frac{1}{2} \|y - X\theta\|^2 + \lambda \|\theta\|_1 \quad , \tag{3}$$

where $X \in \mathbb{R}^{n \times p}$ is a design matrix containing p features as columns, $y \in \mathbb{R}^n$ is the target vector, and $\lambda > 0$ is a regularization hyperparameter. The Lasso estimator was popularized for variable selection: when λ is high enough, many entries in θ^* are exactly equal to 0. This leads to more interpretable models and reduces overfitting compared to the least-squares estimator.

Solvers for Problem (3) have evolved since its introduction by Tibshirani (1996). After generic quadratic program solvers, new dedicated solvers were proposed based on iterative reweighted least-squares (IRLS) (Grandvalet, 1998), followed by LARS (Efron et al., 2004), a homotopy method computing the full Lasso path[3]. The LARS solver helped popularize the Lasso, yet the algorithm suffers from stability issues and can be very slow for worst case situations (Mairal and Yu, 2012). General purpose solvers became popular for Lasso-type problems with the introduction of the iterative soft thresholding algorithm (ISTA, Daubechies et al. 2004), an instance of forward-backward splitting (Combettes and Wajs, 2005). These algorithms became standard in signal and image processing, especially when accelerated (FISTA, Beck and Teboulle 2009).

In parallel, proximal coordinate descent has proven particularly relevant for the Lasso in statistics. Early theoretical results were proved by Tseng (1993) and Sardy et al. (2000), before it became the standard solver of the widely distributed packages `glmnet` in `R` and `scikit-learn` in Python. For further improvements, some solvers exploit the sparsity of θ^*, trying to identify its support to reduce the problem size. Best performing variants of this scheme are screening rules (*e.g.,* El Ghaoui et al., 2012; Bonnefoy et al., 2015; Ndiaye et al., 2017) and working/active sets (*e.g.,* Johnson and Guestrin 2015; Massias et al. 2018), including strong rules (Tibshirani et al., 2012).

While reviews of Lasso solvers have already been performed (Bach et al., 2012, Sec. 8.1), they are limited to certain implementation and design choices, but also naturally lack comparisons with more recent solvers and modern hardware, hence drawing biased conclusions.

The code for the benchmark is available at `https://github.com/benchopt/benchmark_lasso/`. Results obtained on 4 datasets, with 9 standard packages and some custom reimplementations, possibly leveraging GPU hardware, and 17 different solvers written in `Python`/`numba`/`Cython`, `R`, `Julia` or `C++` (Table E.1) are presented in Figure 4. All solvers use efficient numerical implementations, possibly leveraging calls to `BLAS`, precompiled code in `Cython` or just-in-time compilation with `numba`.

The different parameters influencing the setup are

- the regularization strength λ, controlling the sparsity of the solution, parameterized as a fraction of $\lambda_{\max} = \left\|X^\top y\right\|_\infty$ (the minimal hyperparameter such that $\theta^* = 0$),

[3] The Lasso path is the set of solutions of Problem (3) as λ varies in $(0, \infty)$.

- the dataset dimensions: *MEG* has small n and medium p; *rcv1.binary* has medium n and p; *news20.binary* has medium n and very large p while *MillionSong* has very large n and small p (Table E.2).

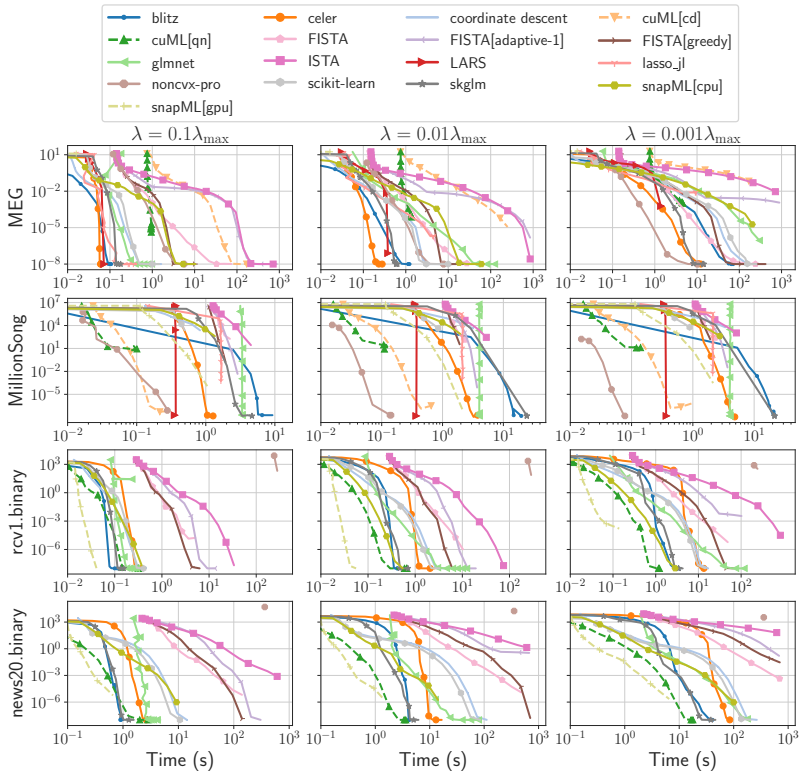


Figure 4: Benchmark for the Lasso, on 17 solvers, 4 datasets (*rows*), and 3 variants of the `Objective` (*columns*) with decreasing regularization λ. The curves display the suboptimality of the objective function, $f(\theta^t) - f(\theta^*)$, as a function of time.

Results Figure 4 presents the result of the benchmark on the Lasso, in terms of objective suboptimality $f(\theta^t) - f(\theta^*)$ as a function of time.

Similarly to Section 3, the GPU solvers obtain good performances in most settings, but their advantage is less clear. A consistent finding across all settings is that coordinate descent-based methods outperform full gradient ones (ISTA and FISTA, even restarted), and are improved by the use of working set strategies (`blitz`, `celer`, `skglm`, `glmnet`). This observation is even more pronounced when the regularization parameter is large, as the solution is sparser.

When observing the influence of the dataset dimensions, we observe 3 regimes. When n is small (*MEG*), the support of the solution is small and coordinate descent, LARS and `noncvx-pro` perform the best. When n is much larger than p (*MillionSong*), `noncvx-pro` clearly outperforms other solvers, and working set methods prove useless. Finally, when n and p are large (*rcv1.binary*, *news20.binary*), CD and working sets vastly outperforms the rest while `noncvx-pro` fails, as it requires solving a linear system of size $\min(n, p)$. We note that this setting was not tested in the original experiment of Poon and Peyré (2021), which highlights the need for extensive, standard experimental setups.

When the support of the solution is small (either small λ, either small n since the Lasso solution has at most n nonzero coefficients), LARS is a competitive algorithm. We expect this to degrade when n increases, but as the LARS solver in `scikit-learn` does not support sparse design matrices we could not include it for *news20.binary* and *rcv1.binary*.

This benchmark is the first to evaluate solvers across languages, showing the competitive behavior of `lasso.jl` and `glmnet` compared to `Python` solvers. Both solvers have a large initialization time, and then converge very fast. To ensure that the benchmark is fair, even though the `Benchopt` library is implemented in `Python`, we made sure to ignore conversion overhead, as well as just-in-time compilation cost. We also checked the timing's consistency with native calls to the libraries.

Since the Lasso is massively used for it feature selection properties, the speed at which the solvers identify the support of the solution is also an important performance measure. Monitoring this with `Benchopt` is straightforward, and a figure reporting this benchmark is in Appendix E.

5 Third example: How standard is a benchmark on ResNet18?

As early successes of deep learning have been focused on computer vision tasks (Krizhevsky et al., 2012), image classification has become a *de facto* standard to validate novel methods in the field. Among the different network architectures, ResNets (He et al., 2016) are extensively used in the community as they provide strong and versatile baselines (Xie et al., 2017; Tan and Le, 2019; Dosovitskiy et al., 2021; Brock et al., 2021; Liu et al., 2022). While many papers present results with such model on classical datasets, with sometimes extensive ablation studies (He et al., 2019; Wightman et al., 2021; Bello et al., 2021; Schmidt et al., 2021), the lack of standardized codebase and missing implementation details makes it hard to replicate their results.

The code for the benchmark is available at `https://github.com/benchopt/benchmark_resnet_classif/`. We provide a cross-dataset –*SVHN*, Netzer et al. (2011); *MNIST*, LeCun et al. (2010) and *CIFAR-10*, Krizhevsky (2009)– and cross-framework –`TensorFlow/Keras`, Abadi et al. (2015) and Chollet et al. (2015); `PyTorch`, Paszke et al. (2019)– evaluation of the training strategies for image classification with ResNet18 (see Appendix F for details on architecture and datasets). We train the network by minimizing the cross entropy loss relatively to the weights θ of the model. Contrary to logistic regression and the Lasso, this problem is non-convex due to the non-linearity of the model f_θ. Another notable difference is that we report the evolution of the test error rather than the training loss.

Because we chose to monitor the test loss, the `Solvers` are defined as the combination of an optimization algorithm, its hyperparameters, the learning rate (LR) and weight decay schedules, and the data augmentation strategy. This is in contrast to a case where we would monitor the train loss, and therefore make the LR and weight decay schedules, as well as the data augmentation policy, part of the objective. We focus on 2 standard methods: stochastic gradient descent (SGD) with momentum and Adam (Kingma and Ba, 2015), as well as a more recently published one: Lookahead (Zhang et al., 2019). The LR schedules are chosen among fixed LR, step LR[4], and cosine annealing (Loshchilov and Hutter, 2017). We also consider decoupled weight decay for Adam (Loshchilov and Hutter, 2019), and coupled weight decay (*i.e.,* ℓ_2-regularization) for SGD. Regarding data augmentation, we use random cropping for all datasets and add horizontal flipping only for *CIFAR-10*, as the digits datasets do not exhibit a mirror symmetry. We detail the remaining hyperparameters in Table F.2, and discuss their selection as well as their sensitivity in Appendix F.

Aligning cross-framework implementations Due to some design choices, components with the same name in the different frameworks do not have the same behavior. For instance, when it comes to applying weight decay, `PyTorch`'s SGD uses coupled weight decay, while in `TensorFlow/Keras` weight decay always refers to decoupled weight decay. These two methods lead to significantly different performance and it is not straightforward to apply coupled weight decay in a post-hoc manner in `TensorFlow/Keras` (see further details in Section F.3). We conducted an extensive effort to align the networks implementation in different frameworks using unit testing to make the conclusions of our benchmarks independent of the chosen framework. We found additional significant differences (reported in Table F.3) in the initialization, the batch normalization, the convolutional layers and the weight decay scaling.

[4] decreasing the learning rate by a factor 10 at mid-training, and again at 3/4 of the training

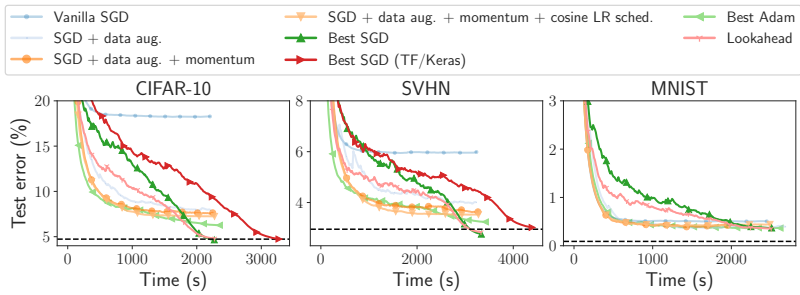


Figure 5: **ResNet18 image classification benchmark with `PyTorch Solvers`.** The best SGD configuration features data augmentation, momentum, cosine learning rate schedule and weight decay. In dashed black is the state of the art for the corresponding datasets with a ResNet18 measured by Zhang et al. (2019) for *CIFAR-10*, by Zheng et al. (2021) for *SVHN* with a PreAct ResNet18, by PapersWithCode for *MNIST* with all networks considered. Off-the-shelf ResNet implementations in `TensorFlow/Keras` do not support images smaller than 32×32 and is hence not shown for *MNIST*. Curves are exponentially smoothed.

Results The results of the benchmark are reported in Figure 5. Each graph reports the test error relative to time, with an ablation study on the solvers' parameters. Note that we only report selected settings for clarity but that we run every possible combination.[5]

Firstly, reaching the state of the art for a vanilla ResNet18 is not straightforward. On the popular website Papers with code it has been so far underestimated. It can achieve 4.45% and 2.65% test error rates on *CIFAR-10* and *SVHN* respectively (compared to 4.73% and 2.95% – for a PreAct ResNet18 – before that). Our ablation study shows that a variety of techniques is required to reach it. The most significant one is an appropriate data augmentation strategy, which lowers the error rate on *CIFAR-10* from about 18% to about 8%. The second most important one is weight decay, but it has to be used in combination with a proper LR schedule, as well as momentum. While these techniques are not novel, they are regularly overlooked in baselines, resulting in underestimation of their performance level.

This reproducible benchmark not only allows a researcher to get a clear understanding of how to achieve the best performances for this model and datasets, but also provides a way to reproduce and extend these performances. In particular, we also include in this benchmark the original implementation of Lookahead (Zhang et al., 2019). We confirm that it slightly accelerates the convergence of the Best SGD, even with a cosine LR schedule – a setting that had not been studied in the original paper.

Our benchmark also evaluates the relative computational performances of the different frameworks. We observe that `PyTorch-Lightning` is significantly slower than the other frameworks we tested, in large part due to their callbacks API. We also notice that our `TensorFlow/Keras` implementation is significantly slower ($\approx 28\%$) than the `PyTorch` ones, despite following the best practices and our profiling efforts. Note that we do not imply that `TensorFlow` is intrinsically slower than `PyTorch`, but a community effort is needed to ensure that the benchmark performances are framework-agnostic.

A recurrent criticism of such benchmarks is that only the best test error is reported. In Figure 6, we measure the effect of using a train-validation-test split, by keeping a fraction of the training set as a validation set. The splits we use are detailed in Table F.1. Our finding is that the results of the ablation study do not change significantly when using such procedure, even though their validity is reinforced by the use of multiple trainings. Yet, a possible limitation of our findings is that some of the hyperparameters we used for our study, coming from the PyTorch-CIFAR GitHub repository, may have been tuned while looking at the test set.

[5]The results are available online as a user-friendly interactive `HTML` file.

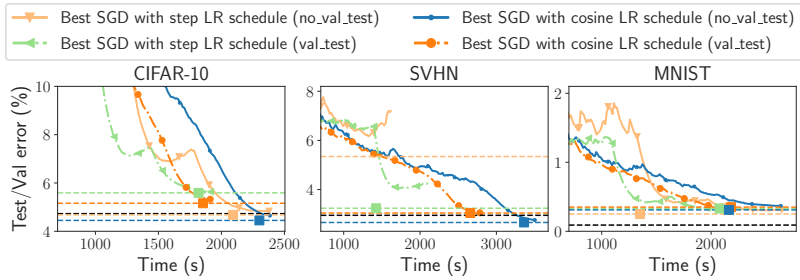


Figure 6: **ResNet18 image classification benchmark with a validation split.** In dashed black is the state of the art (see caption of Figure 5 for more details). In addition, we show in colored horizontal dashed lines, the test results for early stopping on the validation and on the test set for the different solvers, the square mark indicating the moment this stopping would happen. The curves for the train-val splits show the exponentially smoothed median results for five different random seeds.

6 Conclusion and future work

We have introduced `Benchopt`, a library that makes it easy to collaboratively develop fair and extensive benchmarks of optimization algorithms, which can then be seamlessly published, reproduced, and extended. In the future, we plan on supporting the creation of new benchmarks, that could become the standards the community builds on. This work is part of a wider effort to improve reproducibility of machine learning results. It aims to contribute to raising the standard of numerical validation for optimization, which is pervasive in the statistics and ML community as well as for the experimental sciences that rely more and more on these tools for research.

7 Acknowledgements

It can not be stressed enough how much the `Benchopt` library relies on contributions from the community and in particular the Python open source ecosystem. In particular, it could not exist without the libraries mentioned in Appendix A.

This work was granted access to the HPC resources of IDRIS under the allocation 2022-AD011011172R2 and 2022-AD011013570 made by GENCI, which was used to run all the benchmarks. MM also gratefully acknowledges the support of the Centre Blaise Pascal's IT test platform at ENS de Lyon (Lyon, France) for Machine Learning facilities. The platform operates the SIDUS solution (Quemener and Corvellec, 2013).

TL, CFD and JS contributions were supported by the Chaire IA CaMeLOt (ANR-20-CHIA-0001-01). AG, EL and TM contributions were supported by the Chaire IA ANR BrAIN (ANR-20-CHIA-0016). BMa contributions were supported by a grant from Digiteo France. MD contributions were supported by a public grant overseen by the French National Research Agency (ANR) through the program UDOPIA, project funded by the ANR-20-THIA-0013-01 and DATAIA convergence institute (ANR-17-CONV-0003). BN work was supported by the Télécom Paris's Chaire DSAIDIS (Data Science & Artificial Intelligence for Digitalized Industry Services). BMo contributions were supported by a grant from the Labex MILYON.

References

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org.

Akiba, T., S. Sano, T. Yanase, T. Ohta, and M. Koyama (2019). *Optuna: A Next-generation Hyperparameter Optimization Framework*.

Bach, F., R. Jenatton, J. Mairal, and G. Obozinski (2012). "Optimization with sparsity-inducing penalties". In: *Foundations and Trends in Machine Learning* 4.1, pp. 1–106.

Bacry, E., M. Bompaire, S. Gaïffas, and S. Poulsen (2017). "Tick: A Python Library for Statistical Learning, with a Particular Emphasis on Time-Dependent Modeling". In: *ArXiv e-prints*.

Barbero, A. and S. Sra (2018). "Modular Proximal Optimization for Multidimensional Total-Variation Regularization". In: *Journal of Machine Learning Research* 19.56, pp. 1–82.

Barlow, R. E. and H. D. Brunk (1972). "The Isotonic Regression Problem and Its Dual". In: *Journal of the American Statistical Association* 67.337, pp. 140–147.

Bartz-Beielstein, T., C. Doerr, D. van den Berg, J. Bossek, S. Chandrasekaran, T. Eftimov, A. Fischbach, P. Kerschke, W. La Cava, M. Lopez-Ibanez, et al. (2020). "Benchmarking in optimization: Best practice and open issues". In: *arXiv preprint arXiv:2007.03488*.

Beck, A. and M. Teboulle (2009). "A fast iterative shrinkage-thresholding algorithm for linear inverse problems". In: *SIAM J. Imaging Sci.* 2.1, pp. 183–202.

Bello, I., W. Fedus, X. Du, E. D. Cubuk, A. Srinivas, T.-Y. Lin, J. Shlens, and B. Zoph (2021). "Revisiting ResNets: Improved Training and Scaling Strategies". In: *Advances in Neural Information Processing Systems*.

Bergstra, J. and Y. Bengio (2012). "Random search for hyper-parameter optimization". In: *J. Mach. Learn. Res.* 13.2.

Bergstra, J., D. Yamins, and D. Cox (2013). "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures". In: *ICML*. Vol. 28. 1, pp. 115–123.

Bertin-Mahieux, T., D. P. Ellis, B. Whitman, and P. Lamere (2011). "The Million Song Dataset". In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

Bertrand, Q., Q. Klopfenstein, P.-A. Bannier, G. Gidel, and M. Massias (2022). "Beyond L1: Faster and Better Sparse Models with skglm". In: *arXiv preprint arXiv:2204.07826*.

Bertsekas, D. P. (1999). *Nonlinear programming*. Athena Scientific.

Bezanson, J., A. Edelman, S. Karpinski, and V. B. Shah (2017). "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1, pp. 65–98.

Bleakley, K. and J.-P. Vert (2011). "The group fused Lasso for multiple change-point detection". In.

Blondel, M. and F. Pedregosa (2016). *Lightning: Large-Scale Linear Classification, Regression and Ranking in Python*.

Boisbunon, A., R. Flamary, and A. Rakotomamonjy (2014). "Active set strategy for high-dimensional non-convex sparse optimization problems". In: *ICASSP*. IEEE, pp. 1517–1521.

Bolte, J., S. Sabach, and M. Teboulle (2014). "Proximal alternating linearized minimization for nonconvex and nonsmooth problems". In: *Mathematical Programming* 146.1, pp. 459–494.

Bonnefoy, A., V. Emiya, L. Ralaivola, and R. Gribonval (2015). "Dynamic screening: accelerating first-order algorithms for the Lasso and Group-Lasso". In: *IEEE Trans. Signal Process.* 63.19, p. 20.

Bottou, L. (2010). "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *COMPSTAT*. Physica-Verlag, pp. 177–186.

Boyd, S., N. Parikh, E. Chu, B. Peleato, and J. Eckstein (2011). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3.1.

Boyd, S. P. and L. Vandenberghe (2004). *Convex Optimization*. Cambridge, UK ; New York: Cambridge University Press.

Boykov, Y., O. Veksler, and R. Zabih (2001). "Fast approximate energy minimization via graph cuts". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.11, pp. 1222–1239.

Brandl, G. (2010). "Sphinx documentation". In: *URL http://sphinx-doc. org/sphinx. pdf*.

Bredies, K., K. Kunisch, and T. Pock (2010). "Total generalized variation". In: *SIAM J. Imaging Sci.* 3.3, pp. 492–526.

Breheny, P. and J. Huang (2011). "Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection". In: *Ann. Appl. Stat.* 5.1, p. 232.

Brock, A., S. De, S. L. Smith, and K. Simonyan (2021). "High-performance large-scale image recognition without normalization". In: *ICML*. PMLR, pp. 1059–1071.

Bühlmann, P. and S. van de Geer (2011). *Statistics for high-dimensional data*. Springer Series in Statistics. Methods, theory and applications. Heidelberg: Springer.

Candès, E. J., M. B. Wakin, and S. P. Boyd (2008). "Enhancing Sparsity by Reweighted l_1 Minimization". In: *J. Fourier Anal. Applicat.* 14.5-6, pp. 877–905.

Chambolle, A. and P.-L. Lions (1997). "Image recovery via total variation minimization and related problems". In: *Numerische Mathematik* 76.2, pp. 167–188.

Chambolle, A. and T. Pock (2011). "A first-order primal-dual algorithm for convex problems with applications to imaging". In: *Journal of Mathematical Imaging and Vision* 40.1.

Chen, S. S., D. L. Donoho, and M. A. Saunders (1998). "Atomic decomposition by basis pursuit". In: *SIAM J. Sci. Comput.* 20.1, pp. 33–61.

Cherkaoui, H., T. Moreau, A. Halimi, and P. Ciuciu (2019). "Sparsity-Based Semi-Blind Deconvolution of Neural Activation Signal in fMRI". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Brighton, UK.

Chollet, F. et al. (2015). *Keras*.

Clark, A. (2015). *Pillow (PIL Fork) Documentation*.

Combettes, P. L. and V. Wajs (2005). "Signal recovery by proximal forward-backward splitting". In: *Multiscale modeling & simulation* 4.4, pp. 1168–1200.

Combettes, P. L. and L. E. Glaudin (2021). "Solving Composite Fixed Point Problems with Block Updates". In: *Advances in Nonlinear Analysis*.

Condat, L. (2013a). "A Direct Algorithm for 1-D Total Variation Denoising". In: *IEEE SIGNAL PROCESSING LETTERS* 20.12.

Condat, L. (2013b). "A primal-dual splitting method for convex optimization involving Lipschitzian, proximable and linear composite terms". In: *Journal of Optimization Theory and Applications, Springer Verlag*.

Criteo-Labs (2015). "Criteo releases industry's largest-ever dataset for machine learning to academic community". In.

Daubechies, I., M. Defrise, and C. De Mol (2004). "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint". In: *Commun. Pure Appl. Math.* 57.11, pp. 1413–1457.

Defazio, A., F. Bach, and S. Lacoste-Julien (2014). "SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives". In: *Advances in Neural Information Processing Systems*. Vol. 28, pp. 1646–1654.

Dembo, R. S., S. C. Eisenstat, and T. Steihaug (1982). "Inexact Newton Methods". In: *SIAM J. Numer. Anal.* 19.2, pp. 400–408.

Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *ICLR*.

Dünner, C., T. Parnell, D. Sarigiannis, N. Ioannou, A. Anghel, G. Ravi, M. Kandasamy, and H. Pozidis (2018). "Snap ML: A hierarchical framework for machine learning". In: *Advances in Neural Information Processing Systems*. Vol. 31.

Efron, B., T. J. Hastie, I. M. Johnstone, and R. Tibshirani (2004). "Least angle regression". In: *Ann. Statist.* 32.2. With discussion, and a rejoinder by the authors, pp. 407–499.

El Ghaoui, L., V. Viallon, and T. Rabbani (2012). "Safe feature elimination in sparse supervised learning". In: *J. Pacific Optim.* 8.4, pp. 667–698.

Elad, M., P. Milanfar, and R. Rubinstein (2006). "Analysis versus synthesis in signal priors". In: *2006 14th European Signal Processing Conference*.

Falcon, W., J. Borovec, A. Wälchli, N. Eggert, J. Schock, J. Jordan, N. Skafte, Ir1dXD, V. Bereznyuk, E. Harris, T. Murrell, P. Yu, S. Præsius, T. Addair, J. Zhong, D. Lipin, S. Uchida, S. Bapat, H. Schröter, B. Dayma, A. Karnachev, A. Kulkarni, S. Komatsu, Martin.B, J.-B. SCHIRATTI, H. Mary, D. Byrne, C. Eyzaguirre, cinjon, and A. Bakhtin (May 2020). *PyTorchLightning/pytorch-lightning: 0.7.6 release*. Version 0.7.6.

Fan, J. and R. Li (2001). "Variable selection via nonconcave penalized likelihood and its oracle properties". In: *J. Amer. Statist. Assoc.* 96.456, pp. 1348–1360.

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin (2008). "LIBLINEAR: A Library for Large Linear Classification". In: *J. Mach. Learn. Res.* 9.

Farrens, S., A. Grigis, L. El Gueddari, Z. Ramzi, G. Chaithya, S. Starck, B. Sarthou, H. Cherkaoui, P. Ciuciu, and J.-L. Starck (2020). "PySAP: Python Sparse Data Analysis Package for multidisciplinary image processing". In: *Astronomy and Computing* 32, p. 100402.

Fletcher, R. and C. M. Reeves (Jan. 1964). "Function Minimization by Conjugate Gradients". In: *The Computer Journal* 7.2, pp. 149–154.

Forde, J., T. Head, C. Holdgraf, Y. Panda, G. Nalvarete, B. Ragan-Kelley, and E. Sundell (2018). *Reproducible research environments with repo2docker*. Tech. rep.

Friedman, J., T. Hastie, and R. Tibshirani (2010). "Regularization paths for generalized linear models via coordinate descent". In: *J. Stat. Softw.* 33.1, pp. 1–22.

Gao, F. and L. Han (2012). "Implementing the Nelder-Mead simplex algorithm with adaptive parameters". In: *Computational Optimization and Applications* 51.1, pp. 259–277.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *AISTATS*. Vol. 9, pp. 249–256.

Golub, T. R., D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, et al. (1999). "Molecular classification of cancer: class discovery and class prediction by gene expression monitoring". In: *science* 286.5439, pp. 531–537.

Gong, P., C. Zhang, Z. Lu, J. Huang, and J. Ye (2013). "A general iterative shrinkage and thresholding algorithm for non-convex regularized optimization problems". In: *ICML*, pp. 37–45.

Gramfort, A., M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, L. Parkkonen, and M. S. Hämäläinen (2014). "MNE software for processing MEG and EEG data". In: *NeuroImage* 86, pp. 446–460.

Grandvalet, Y. (1998). "Least absolute shrinkage is equivalent to quadratic penalization". In: *International Conference on Artificial Neural Networks*. Springer, pp. 201–206.

Guyon, I., S. Gunn, A. Ben-Hur, and G. Dror (2004). "Result analysis of the nips 2003 feature selection challenge". In: *Advances in neural information processing systems* 17.

Hansen, N., A. Auger, R. Ros, O. Mersmann, T. Tušar, and D. Brockhoff (2021). "COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting". In: *Optimization Methods and Software* 36.1. ArXiv e-prints, arXiv:1603.08785, pp. 114–144.

Hansen, N. and A. Ostermeier (2001). "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary computation* 9.2, pp. 159–195.

Harris, C. R., K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, et al. (2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362.

Hastie, T. J., R. Tibshirani, and M. Wainwright (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. CRC Press.

He, K., X. Zhang, S. Ren, and J. Sun (2015). "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *CVPR*, pp. 1026–1034.

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *CVPR*.

He, T., Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li (2019). "Bag of tricks for image classification with convolutional neural networks". In: *CVPR*. Vol. 2019-June, pp. 558–567.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3, pp. 90–95.

Hutson, M. (2018). "Artificial intelligence faces reproducibility crisis". In: *Science* 359.6377, pp. 725–726.

Inc., P. T. (2015). *Collaborative data science*. URL: https://plot.ly.

Ioffe, S. and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *ICML*, pp. 448–456.

Johnson, R. and T. Zhang (2013). "Accelerating Stochastic Gradient Descent Using Predictive Variance Reduction". In: *Advances in Neural Information Processing Systems*. Vol. 26.

Johnson, T. B. and C. Guestrin (2015). "Blitz: A Principled Meta-Algorithm for Scaling Sparse Optimization". In: *ICML*. Vol. 37, pp. 1171–1179.

Karahanoğlu, F. I., C. Caballero-Gaudes, F. Lazeyras, and D. Van De Ville (June 2013). "Total Activation: fMRI Deconvolution through Spatio-Temporal Regularization". In: *NeuroImage* 73, pp. 121–134.

Keerthi, S. S., D. DeCoste, and T. Joachims (2005). "A modified finite Newton method for fast solution of large scale linear SVMs." In: *Journal of Machine Learning Research* 6.3.

Kiefer, J. and J. Wolfowitz (1952). "Stochastic estimation of the maximum of a regression function". In: *Ann. Math. Stat.*, pp. 462–466.

Kingma, D. P. and J. Ba (2015). "Adam: A Method for Stochastic Optimization". In: *ICLR*, pp. 1–10.

Koh, K., J. Kim, and S. Boyd (2007). "An interior-point method for large-scale l1-regularized logistic regression." In: *J. Mach. Learn. Res.* 8.8, pp. 1519–1555.

Kolmogorov, V. and R. Zabin (2004). "What energy functions can be minimized via graph cuts?" In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.2, pp. 147–159.

Komodakis, N. and J.-C. Pesquet (2015). "Playing with Duality: An overview of recent primal-dual approaches for solving large-scale optimization problems". In: *IEEE Signal Processing Magazine, Institute of Electrical and Electronics Engineers*.

Kornblith, S. (Oct. 28, 2021). *Lasso.jl*. Version 0.6.3. JuliaStats.

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images*. Tech. rep.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. South Lake Tahoe, NV, USA, pp. 1097–1105.

Lalanne, C., M. Rateaux, L. Oudre, M. P. Robert, and T. Moreau (July 2020). "Extraction of Nystagmus Patterns from Eye-Tracker Data with Convolutional Sparse Coding". In: *Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. Montreal, QC, Canada: IEEE, pp. 928–931.

LeCun, Y., C. Cortes, and C. Burges (2010). "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2.

Lewis, D. D., Y. Yang, T. Russell-Rose, and F. Li (2004). "Rcv1: A new benchmark collection for text categorization research". In: *Journal of machine learning research* 5.Apr, pp. 361–397.

Liang, J., T. Luo, and C.-B. Schöenlieb (2022). "Improving "Fast Iterative Shrinkage-Thresholding Algorithm": Faster, Smarter, and Greedier". In: *SIAM J. Sci. Comput.* 44.3, A1069–A1091.

Liu, D. C. and J. Nocedal (Aug. 1989). "On the Limited Memory BFGS Method for Large Scale Optimization". In: *Math. Program.* 45.1-3, pp. 503–528.

Liu, Z., H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie (2022). "A ConvNet for the 2020s". In: *CVPR*.

Loshchilov, I. and F. Hutter (2017). "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *ICLR*.

Loshchilov, I. and F. Hutter (2019). "Decoupled Weight Decay Regularization". In: *ICLR*.

Lueckmann, J.-M., J. Boelts, D. S. Greenberg, P. J. Gonçalves, and J. H. Macke (2021). "Benchmarking Simulation-Based Inference". In: *AISTATS*. Vol. 130. PMLR, pp. 343–351.

Mairal, J. and B. Yu (2012). "Complexity analysis of the Lasso regularization path". In: *ICML*, pp. 353–360.

Mairal, J. (2019). "Cyanure: An open-source toolbox for empirical risk minimization for python, c++, and soon more". In: *arXiv preprint arXiv:1912.08165*.

Massias, M., A. Gramfort, and J. Salmon (2018). "Celer: a fast solver for the lasso with dual extrapolation". In: *ICML*, pp. 3315–3324.

Mattson, P., V. Reddi, C. Cheng, C. Coleman, G. Diamos, D. Kanter, P. Micikevicius, D. Patterson, G. Schmuelling, H. Tang, et al. (2020). "MLPerf: An industry standard benchmark suite for machine learning performance". In: *IEEE Micro* 40.2, pp. 8–16.

Mazumder, R., J. H. Friedman, and T. Hastie (2011). "Sparsenet: Coordinate descent with nonconvex penalties". In: *J. Amer. Statist. Assoc.* 106.495, pp. 1125–1138.

McKinney, W. et al. (2010). "Data structures for statistical computing in python". In: *Proceedings of the 9th Python in Science Conference*. Vol. 445. Austin, TX, pp. 51–56.

Moreau, T., P. Glaser, R. Yurchak, and O. Grisel (June 2017). *Loky*. Version 3.0. URL: https://github.com/joblib/loky.

Nájera, Ó., E. Larson, L. Estève, G. Varoquaux, L. Liu, J. Grobler, E. S. de Andrade, C. Holdgraf, A. Gramfort, M. Jas, J. Nothman, O. Grisel, N. Varoquaux, E. Gouillart, M. Luessi, A. Lee, J. Vanderplas, T. Hoffmann, T. A. Caswell, B. Sullivan, A. Batula, jaeilepp, T. Robitaille, S. Appelhoff, P. Kunzmann, M. Geier, Lars, K. Sunden, D. Stańczak, and A. Y. Shih (May 2020). *sphinx-gallery/sphinx-gallery: Release v0.7.0*. Version v0.7.0.

Ndiaye, E., O. Fercoq, A. Gramfort, and J. Salmon (2017). "Gap Safe screening rules for sparsity enforcing penalties". In: *J. Mach. Learn. Res.* 18.128, pp. 1–33.

Nesterov, Y. E. (1983). "A method for solving the convex programming problem with convergence rate $O(1/k^2)$". In: *Dokl. akad. nauk Sssr*. Vol. 269, pp. 543–547.

Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng (2011). "Reading Digits in Natural Images with Unsupervised Feature Learning". In: *Advances in Neural Information Processing Systems*.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*, pp. 8024–8035.

Pedregosa, F., G. Negiar, and G. Dresdner (2020). "copt: composite optimization in Python". In.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. (2011). "Scikit-learn: Machine learning in Python". In: *J. Mach. Learn. Res.* 12, pp. 2825–2830.

Pineau, J., P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and H. Larochelle (2021). "Improving reproducibility in machine learning research: a report from the NeurIPS 2019 reproducibility program". In: *J. Mach. Learn. Res.* 22.

Pineau, J., K. Sinha, G. Fried, R. N. Ke, and H. Larochelle (2019). "ICLR reproducibility challenge 2019". In: *ReScience C* 5.2, p. 5.

Poon, C. and G. Peyré (2021). "Smooth Bilevel Programming for Sparse Regularization". In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 1543–1555.

Powell, M. J. (1964). "An efficient method for finding the minimum of a function of several variables without calculating derivatives". In: *The computer journal* 7.2, pp. 155–162.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Prokhorov, D. (2001). "IJCNN 2001 neural network competition". In: *Slide presentation in IJCNN* 1.97, p. 38.

Quemener, E. and M. Corvellec (2013). "SIDUS—the solution for extreme deduplication of an operating system". In: *Linux Journal* 2013.235, p. 3.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria.

Raff, E. (2019). "A step toward quantifying independently reproducible machine learning research". In: *Advances in Neural Information Processing Systems*. Vol. 32, pp. 5486–5496.

Rapin, J. and O. Teytaud (2018). *Nevergrad - A gradient-free optimization platform*. https://GitHub.com/FacebookResearch/Nevergrad.

Raschka, S., J. Patterson, and C. Nolet (2020). "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence". In: *Information-an International Interdisciplinary Journal* 11.4, p. 193.

Robbins, H. and S. Monro (1951). "A stochastic approximation method". In: *Ann. Math. Stat.*, pp. 400–407.

Rodola, G. (2016). "Psutil package: a cross-platform library for retrieving information on running processes and system utilization". In: *Google Scholar*.

Rougier, N. P. and K. Hinsen (2018). "ReScience C: a journal for reproducible replications in computational science". In: *International Workshop on Reproducible Research in Pattern Recognition*. Springer, pp. 150–156.

Rudin, L. I., S. Osher, and E. Fatemi (1992). "Nonlinear total variation based noise removal algorithms". In: *Physica D: Nonlinear Phenomena* 60.1-4, pp. 259–268.

Sardy, S., A. G. Bruce, and P. Tseng (2000). "Block coordinate relaxation methods for nonparametric wavelet denoising". In: *J. Comput. Graph. Stat.* 9.2, pp. 361–379.

Schmidt, M., N. Le Roux, and F. Bach (2017). "Minimizing Finite Sums with the Stochastic Average Gradient". In: *Math. Program.* 162.arXiv:1309.2388, pp. 83–112.

Schmidt, R. M., F. Schneider, and P. Hennig (2021). "Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers". In: *ICML*. Vol. 139. PMLR, pp. 9367–9376.

Sculley, D., J. Snoek, A. Wiltschko, and A. Rahimi (2018). "Winner's curse? On pace, progress, and empirical rigor". In.

Silva, T. S. (2019). "How to Add Regularization to Keras Pre-trained Models the Right Way". In: *https://sthalles.github.io*.

Simonyan, K. and A. Zisserman (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ICLR*.

Sra, S., S. Nowozin, and S. J. Wright (2012). *Optimization for machine learning*. MIT Press.

Tan, M. and Q. Le (2019). "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *ICML*, pp. 6105–6114.

Tibshirani, R., J. Bien, J. Friedman, T. Hastie, N. Simon, J. Taylor, and R. J. Tibshirani (2012). "Strong rules for discarding predictors in lasso-type problems". In: *J. R. Stat. Soc. Ser. B Stat. Methodol.* 74.2, pp. 245–266.

Tibshirani, R. (1996). "Regression shrinkage and selection via the lasso". In: *J. R. Stat. Soc. Ser. B Stat. Methodol.* 58.1, pp. 267–288.

Tibshirani, R. J. (Feb. 2014). "Adaptive Piecewise Polynomial Estimation via Trend Filtering". In: *The Annals of Statistics* 42.1.

Tibshirani, R. J. and J. Taylor (2011). "The solution path of the generalized lasso". In: *Ann. Statist.* 39.3, pp. 1335–1371.

Tseng, P. (1993). "Dual coordinate ascent methods for non-strictly convex minimization". In: *Math. Program.* 59.1, pp. 231–247.

Vanschoren, J., J. van Rijn, B. Bischl, and L. Torgo (2013). "OpenML: networked science in machine learning". In: *SIGKDD Explorations* 15.2, pp. 49–60.

Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors (2020). "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17, pp. 261–272.

Wales, D. J. and J. P. Doye (1997). "Global optimization by basin-hopping and the lowest energy structures of Lennard-Jones clusters containing up to 110 atoms". In: *The Journal of Physical Chemistry A* 101.28, pp. 5111–5116.

Wightman, R., H. Touvron, and H. Jégou (2021). *ResNet strikes back: An improved training procedure in timm*. Tech. rep., pp. 1–22.

Wright, S. and J. Nocedal (1999). *Numerical Optimization*. Science Springer.

Xie, S., R. Girshick, P. Dollár, Z. Tu, and K. He (2017). "Aggregated residual transformations for deep neural networks". In: *CVPR*, pp. 1492–1500.

Zhang, . (2010a). "Nearly unbiased variable selection under minimax concave penalty". In: *Ann. Statist.* 38.2, pp. 894–942.

Zhang, M., J. Lucas, J. Ba, and G. E. Hinton (2019). "Lookahead optimizer: k steps forward, 1 step back". In: *Advances in Neural Information Processing Systems*. Vol. 32.

Zhang, T. (2010b). "Analysis of multi-stage convex relaxation for sparse regularization". In: *J. Mach. Learn. Res.* 11.Mar, pp. 1081–1107.

Zheng, Y., R. Zhang, and Y. Mao (2021). "Regularizing neural networks via adversarial model perturbation". In: *CVPR*, pp. 8156–8165.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] These are specified in Appendix, on a per-benchmark basis.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Error bars are not reported for clarity, but `Benchopt` allows this in particular in html versions of the plots that can be found in `https://benchopt.github.io/results/preprint_results.html`.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] In the introduction.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Software ecosystem acknowledgement

The command line interface and API use the `click`, `pyyaml` and `psutil` (Rodola, 2016) libraries.

Numerical computations involve `numpy` (Harris et al., 2020) and `scipy` (Virtanen et al., 2020). For cross-language processing, we used `rpy2` for calling `R` (R Core Team, 2017) libraries and `PyJulia` for interfacing with `Julia` (Bezanson et al., 2017). The benchmark runs extensively use `joblib`, `loky` (Moreau et al., 2017) and `submitit` for parallelization.

The results are stored and processed for visualizations using `pandas` (McKinney et al., 2010), `matplotlib` (Hunter, 2007) for static rendering, `mako` and `plotly` (Inc., 2015) for interactive webpages. The participative results website relies partially on `pygithub`.

Our documentation is generated by multiple `sphinx`-based (Brandl, 2010) libraries (`sphinx-bootstrap-theme`, `sphinx-click`, `sphinx-gallery` (Nájera et al., 2020) and `sphinx-prompt`), and also the `numpydoc` and `pillow` (Clark, 2015) libraries.

B A complete Benchmark example: `Objective`, `Dataset` and `Solver` classes for Ridge regression

Here, we provide code examples for a simple benchmark on Ridge regression. The Ridge regression – also called ℓ_2-regularized least-squares or Tikhonov regression – is a popular method to solve least-square problems in the presence of noisy observations or correlated features. The problem reads:

$$\min_{\theta} \frac{1}{2}\|y - X\theta\|_2^2 + \frac{\lambda}{2}\|\theta\|_2^2 \ , \tag{4}$$

where $X \in \mathbb{R}^{n \times p}$ is a design matrix, $y \in \mathbb{R}^n$ is the target vector and λ is the regularization parameter. This problem is strongly convex and many methods can be used to solve it. Direct computation of the close form solution $\theta^* = (X^\top X + \lambda Id)^{-1} X^\top y$ can be obtained using matrix factorization methods such as Cholesky decomposition or the SVD (Press et al., 2007) or iterative linear solver such as Conjugate-Gradient (Liu and Nocedal, 1989). One can also resort on first order methods such as gradient descent, coordinate descent (known as the Gauss-Seidel method in this context), or their stochastic variant.

The code for the benchmark is available at `https://github.com/benchopt/benchmark_ridge/`. The following code snippets are provided in the documentation as a template for new benchmarks.

B.1 `Objective` class

The `Objective` class is the central part of the benchmark, defining the objective function. This class allows us to monitor the quantities of interest along the iterations of the solvers, amongst which the objective function value. An `Objective` class should define 3 methods:

- `set_data(**data)`: it allows specifying the nature of the data used in the benchmark. The data is passed as a dictionary of Python variables, so no constraint is enforced to what can be passed here.
- `compute(theta)`: it allows evaluating the objective function for a given value of the iterate, here called θ. This method should take only one parameter, the output returned by the `Solver`. All other parameters should be stored in the class with the `set_data` method. The compute function should return a float (understood as the objective value) or a dictionary. If a dictionary is returned it should contain a key called `value` (the objective value) and all other keys should correspond to float values allowing tracking more than one quantity of interest (e.g. train and test errors).
- `get_objective()`: a method that returns a dictionary to be passed to the `set_objective()` method of a `Solver`.

An `Objective` class needs to inherit from a base class, `benchopt.BaseObjective`. Below is the implementation of the Ridge regression `Objective` class.

```python
from benchopt import BaseObjective


class Objective(BaseObjective):
    name = "Ridge regression"
    parameters = {"reg": [0.1, 1, 10]}

    def __init__(self, reg=1):
        self.reg = reg

    def set_data(self, X, y):
        self.X, self.y = X, y

    def compute(self, theta):
        res = self.y - self.X @ theta
        return .5 * res @ res + 0.5 * self.reg * theta @ theta

    def get_objective(self):
        return dict(X=self.X, y=self.y, reg=self.reg)
```

B.2 `Dataset` class

A `Dataset` class defines data to be passed to the `Objective`. More specifically, a `Dataset` class should implement one method:

- `get_data()`: A method outputting a dictionary that can be passed as keyword arguments `**data` to the `set_data` method of the `Objective`.

A `Dataset` class also needs to inherit from a base class, `benchopt.BaseDataset`.

If a `Dataset` requires some packages to function, `Benchopt` allows listing some requirements. The necessary packages should be available via `conda` or `pip`.

Below is an example of a `Dataset` definition using the `libsvmdata` library, which exposes datasets from `libsvm`, such as *leukemia*, *bodyfat* and *gisette* – described in Table B.1.

```python
from benchopt import BaseDataset
from benchopt import safe_import_context

# This context allow to manipulate the Dataset object even if
# libsvmdata is not installed. It is used in `benchopt install`.
with safe_import_context() as import_ctx:
    from libsvmdata import fetch_libsvm

class Dataset(BaseDataset):
    name = "libsvm"
    install_cmd = "conda"
    requirements = ["libsvmdata"]
    parameters = {"dataset": ["bodyfat", "leukemia", "gisette"]}

    def __init__(self, dataset="bodyfat"):
        self.dataset = dataset

    def get_data(self):
        X, y = fetch_libsvm(self.dataset)
        return dict(X=self.X, y=self.y)
```

B.3 `Solver` class

A `Solver` class must define three methods:

- `set_objective(**objective_dict)`: This method will be called with the dictionary `objective_dict` returned by the method `get_objective` from the `Objective`. The goal of this method is to provide all necessary information to the `Solver` so it can optimize the objective function.
- `run(stop_value)`: This method takes only one parameter that controls the stopping condition of the `Solver`. Typically this is either a number of iterations `n_iter` or a tolerance parameter `tol`. Alternatively, a callback function that will be called at each iteration can be passed. The callback should return `False` once the computation should stop. The parameter `stop_value` is controlled by the `stopping_strategy`, see below for details.
- `get_result()`: This method returns a variable that can be passed to the compute method from the `Objective`. This is the output of the `Solver`.

If a Python `Solver` requires some packages such as `scikit-learn`, `Benchopt` allows listing some requirements. The necessary packages must be available via `conda` or `pip`.

Below is a simple `Solver` example using `scikit-learn` implementation of Ridge regression with different optimization algorithms.

```python
from benchopt import BaseSolver
from benchopt import safe_import_context

# This context allow to manipulate the Solver object even if
# scikit-learn is not installed. It is used in `benchopt install`.
with safe_import_context() as import_ctx:
    from sklearn.linear_model import Ridge


class Solver(BaseSolver):
    name = "scikit-learn"
    install_cmd = "conda"
    requirements = ["scikit-learn"]
    parameters = {
        "alg": ["svd", "cholesky", "lsqr", "sparse_cg", "saga"],
    }

    def __init__(self, alg="svd"):
        self.alg = alg

    def set_objective(self, X, y, reg=1):
        self.X, self.y  = X, y
        self.clf = Ridge(
            fit_intercept=False, alpha=reg, solver=self.alg,
            tol=1e-10
        )

    def run(self, n_iter):
        self.clf.max_iter = n_iter + 1
        self.clf.fit(self.X, self.y)

    def get_result(self):
        return self.clf.coef_
```

B.4 Results from the benchmark

Descriptions of datasets Table B.1 describes the datasets used in this benchmarks.

Table B.1: List of the datasets used in Ridge regression in Appendix B

Datasets	References	Samples (n)	Features (p)
leukemia	Golub et al. (1999)	38	7129
bodyfat	Guyon et al. (2004)	252	8
gisette	Guyon et al. (2004)	6000	5000

We also run the solvers on the simulated data described bellow.

Generation process for `simulated` dataset We generate a linear regression scenario with decaying correlation for the design matrix, *i.e.,* the ground-truth covariance matrix is a Toeplitz matrix, with each element $\Sigma_{ij} = \rho^{|i-j|}$. As a consequence, the generated features have 0 mean, a variance of 1, and the correlation structure as:

$$\mathbb{E}[X_i] = 0\,, \quad \mathbb{E}[X_i^2] = 1 \quad \text{and} \quad \mathbb{E}[X_i X_j] = \rho^{|i-j|}\,. \tag{5}$$

Our simulation scheme also includes the parameter `density` $= 0.2$ that controls the proportion of non-zero elements in θ^*. The target vector is generated according to linear relationship with Gaussian noise:

$$y = X\theta^* + \varepsilon\,,$$

such that the signal-to-noise ratio is `snr` $= \frac{\|X\theta^*\|_2}{\|\varepsilon\|_2}$.

We use a signal-to-noise ratio `snr` $= 3$, a correlation ρ of 0 or 0.6 with $n = 500$ samples and $p = 1000$ features.

Description of the solvers Table B.2 describes the different solvers compared in this benchmark.

Table B.2: List of solvers used in the Ridge benchmark in Appendix B

Solver	References	Description	Language
`GD`	Boyd and Vandenberghe (2004)	Gradient Descent	`Python`
`Accelerated GD`	Nesterov (1983)	Gradient Descent + acceleration	`Python`
`scikit-learn[svd]`	Pedregosa et al. (2011)	SVD (Singular Value Decomposition)	`Python (Cython)`
`scikit-learn[cholesky]`	Pedregosa et al. (2011)	Cholesky decomposition	`Python (Cython)`
`scikit-learn[lsqr]`	Pedregosa et al. (2011)	regularized least-squares	`Python (Cython)`
`scikit-learn[saga]`	Pedregosa et al. (2011)	SAGA (Varianced reduced stochastic method)	`Python (Cython)`
`scikit-learn[cg]`	Pedregosa et al. (2011)	Conjugate Gradient	`Python (Cython)`
`CD`	Bertsekas (1999)	Cyclic Coordinate Descent	`Python (Numba)`
`lightning[cd]`	Blondel and Pedregosa (2016)	Cyclic Coordinate Descent	`Python (Cython)`
`snapML[cpu]`	Dünner et al. (2018)	CD	`Python, C++`
`snapML[gpu]`	Dünner et al. (2018)	CD + GPU	`Python, C++`

Results Figure B.1 presents the performance of the different methods for different values of the regularization parameter in the benchmark. The algorithms based on the direct computation of the closed-form solution outperform iterative ones in a majority of presented datasets. Among closed-form algorithms, the Cholesky solver converges faster.

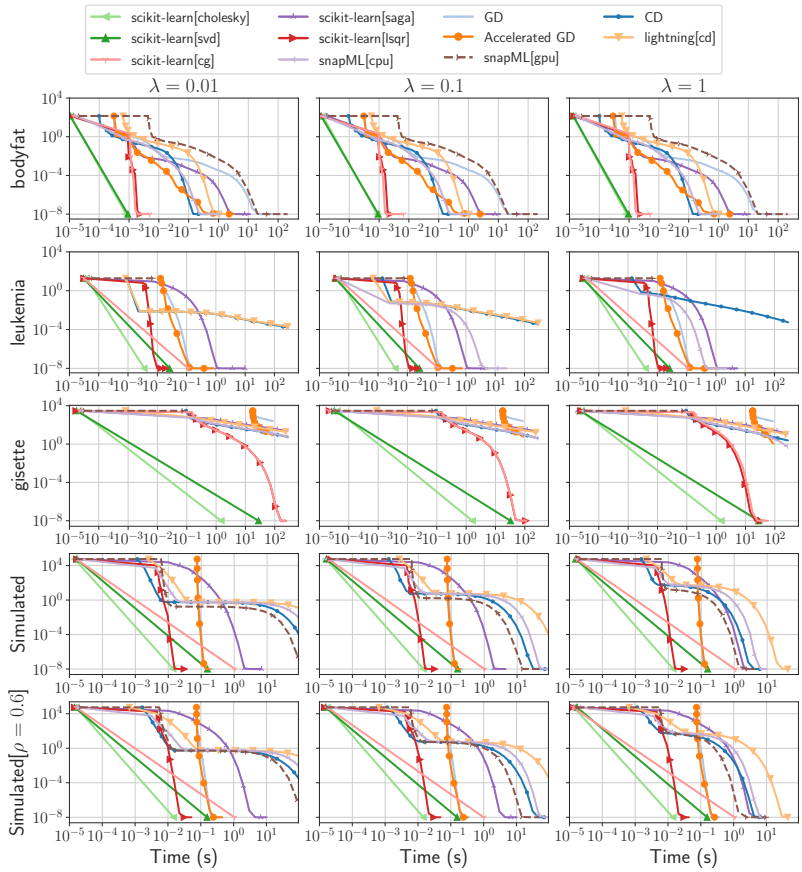


Figure B.1: Benchmark for the Ridge regression, on 10 solvers, 5 datasets (*rows*), and 3 variants of the `Objective` (*columns*) each with a different regularization value $\lambda \in \{0.01, 0.1, 1\}$. The curves display the suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, as a function of time.

C Design choices

Benchopt has made some design choices, while trying as much as possible to leave users free of customizing the behavior on each benchmark. We detail the most important ones in this section.

C.1 Estimating θ^* for convex problems

When the problem is convex, many solvers are guaranteed to converge to a global minimizer θ^* of the objective function f. To estimate θ^* and $f(\theta^*)$, Benchopt approximates θ^* by the iterate achieving the lowest `objective_value` among all solvers for a given `Dataset` and `Objective`. This means that the sub-optimality plot proposed by Benchopt are only valid if at least one solver has converged to the optimal solution. Else, the curves are a lower bound estimate of the sub-optimality. In practice, for most considered convex problems, running the `Solver` for long enough ensures that $f(\theta^*)$ is correctly estimated.

C.2 Stopping solvers

Benchopt offers many ways to stop running a solver. The most common is to stop the solver when the objective value does not decrease significantly between iterations. For some convex problems, we also propose to track the duality gap (which upper bounds the suboptimality), as is done for the Lasso. For non convex problems, criteria such as gradient norm or violation of first order conditions can be used, as users do in practice. These criteria can easily be customized.

C.3 Wall-clock time versus number of iterations

Measuring time or iteration are two alternatives that make sense in their respective contexts. Practitioners mostly care about the time it takes to solve their problem, while researchers in mathematical optimization may want to abstract away the implementation and hardware details and only consider iteration. The benchmarks we have presented showcase efficient implementations and are also interested in hardware and implementation differences (e.g. CPU vs GPU solvers for Section 3, Section 4, torch versus tensorflow for Section F.4), hence our focus on time. However, Benchopt does not impose a choice between the two measures: it is perfectly possible to create plots as a function of the number of iterations as evidenced for example in Section E.3.

D ℓ_2-regularized logistic regression

D.1 List of solvers and datasets used in the benchmark in Section 3

Table D.1 and Table D.2 respectively present the `Solvers` and `Datasets` used in this benchmark.

Table D.1: List of solvers used in the ℓ_2-regularized logistic regression benchmark in Section 3

Solver	References	Description	Language
`lightning[sag]`	Blondel and Pedregosa (2016)	SAG	`Python (Cython)`
`lightning[saga]`	Blondel and Pedregosa (2016)	SAGA	`Python (Cython)`
`lightning[cd]`	Blondel and Pedregosa (2016)	Cyclic Coordinate Descent	`Python (Cython)`
`Tick[svrg]`	Bacry et al. (2017)	Stochastic Variance Reduced Gradient	`Python, C++`
`scikit-learn[sgd]`	Pedregosa et al. (2011)	Stochastic Gradient Descent	`Python (Cython)`
`scikit-learn[sag]`	Pedregosa et al. (2011)	SAG	`Python (Cython)`
`scikit-learn[saga]`	Pedregosa et al. (2011)	SAGA	`Python (Cython)`
`scikit-learn[liblinear]`	Pedregosa et al. (2011), Fan et al. (2008)	Truncated Newton Conjugate-Gradient	`Python (Cython)`
`scikit-learn[lbfgs]`	Pedregosa et al. (2011), Virtanen et al. (2020)	L-BFGS (Quasi-Newton Method)	`Python (Cython)`
`scikit-learn[newton-cg]`	Pedregosa et al. (2011), Virtanen et al. (2020)	Truncated Newton Conjugate-Gradient	`Python (Cython)`
`snapml[cpu]`	Dünner et al. (2018)	CD	`Python, C++`
`snapml[gpu]`	Dünner et al. (2018)	CD + GPU	`Python, C++`
`cuML[gpu]`	Raschka et al. (2020)	L-BFGS + GPU	`Python, C++`

Table D.2: List of the datasets used in ℓ_2-regularized logistic regression in Section 3

Datasets	References	Samples (n)	Features (p)	Density
ijcnn1	Prokhorov (2001)	49 990	22	4.5×10^{-2}
madelon	Guyon et al. (2004)	2000	500	2.0×10^{-3}
news20.binary	Keerthi et al. (2005)	19 996	1 355 191	3.4×10^{-4}
criteo	Criteo-Labs (2015)	45 840 617	1 000 000	3.9×10^{-5}

D.2 Results

Figure D.1 presents the performance results for the different solvers on the different datasets using various regularization parameter values, on unscaled raw data. We observe that when the regularization parameter λ increases, the problem tends to become easier and faster to solve for most methods. Also, the relative order of the method does not change significantly for the considered range of regularization.

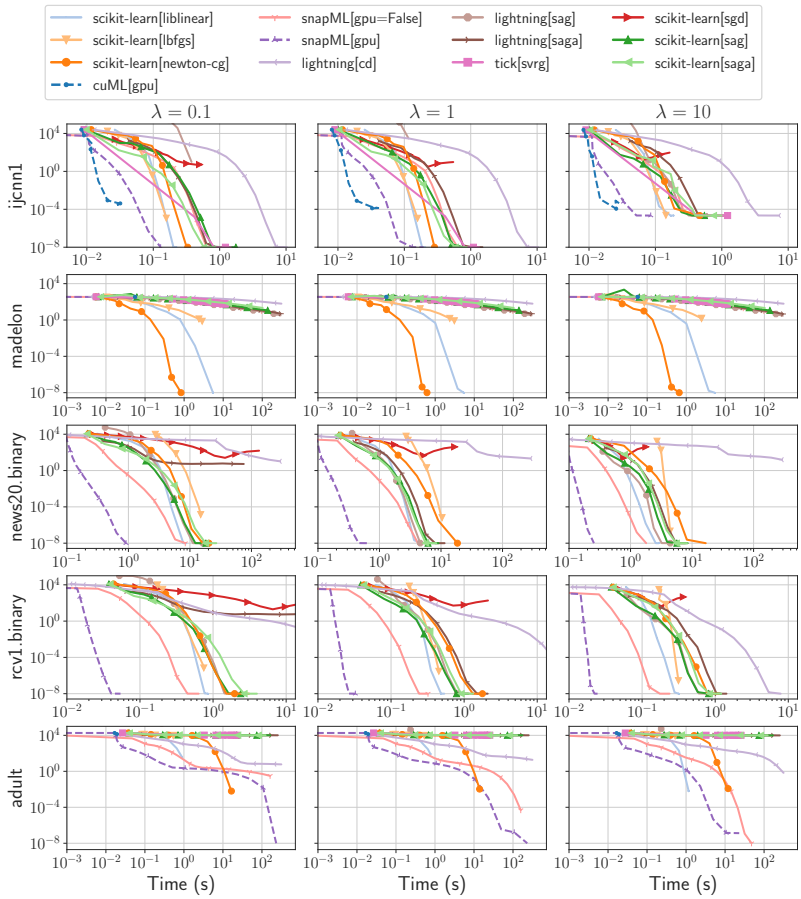


Figure D.1: Additional benchmark for the ℓ_2-regularized logistic regression on variants of the `Objective` (*columns*) with `fit_intercept=False`. The curves display the suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, as a function of time. The columns correspond to the objective detailed in Problem (2) with different value of λ: (*first*) $\lambda = 0.1$, (*second*) $\lambda = 1$ and (*third*) $\lambda = 10$.

E Lasso

E.1 List of solvers and datasets used in the Lasso benchmark in Section 4

Table E.1 and Table E.2 respectively present the `Solvers` and `Datasets` used in this benchmark.

Table E.1: List of solvers used in the Lasso benchmark in Section 4

Solver	References	Description	Language
`blitz`	Johnson and Guestrin (2015)	CD + working set	`Python`, `C++`
`coordinate descent`	Friedman et al. (2010)	(Cyclic) Minimization along coordinates	`Python (Numba)`
`celer`	Massias et al. (2018)	CD + working set + dual extrapolation	`Python (Cython)`
`cuML[cd]`	Raschka et al. (2020)	(Cyclic) Minimization along coordinates	`Python`, `C++`
`cuML[qn]`	Raschka et al. (2020)	Orthant-Wise Limited Memory Quasi-Newton (OWL-QN)	`Python`, `C++`
`FISTA`	Beck and Teboulle (2009)	ISTA + acceleration	`Python`
`glmnet`	Friedman et al. (2010)	CD + working set + strong rules	`R`, `C++`
`ISTA`	Daubechies et al. (2004)	ISTA (Proximal GD)	`Python`
`LARS`	Efron et al. (2004)	Least-Angle Regression algorithm (LARS)	`Python (Cython)`
`FISTA[adaptive-1]`	Liang et al. (2022, Algo 4), Farrens et al. (2020)	FISTA + adaptive restart	`Python`
`FISTA[greedy]`	Liang et al. (2022, Algo 5), Farrens et al. (2020)	FISTA + greedy restart	`Python`
`noncvx-pro`	Poon and Peyré (2021)	Bilevel optim + L-BFGS	`Python (Cython)`
`skglm`	Bertrand et al. (2022)	CD + working set + primal extrapolation	`Python (Numba)`
`scikit-learn`	Pedregosa et al. (2011)	CD	`Python (Cython)`
`snapML[gpu]`	Dünner et al. (2018)	CD + GPU	`Python`, `C++`
`snapML[cpu]`	Dünner et al. (2018)	CD	`Python`, `C++`
`lasso.jl`	Kornblith (2021)	CD	`Julia`

Table E.2: List of datasets used in the Lasso benchmark in Section 4

Dataset	References	Samples (n)	Features (p)	Density
MEG	Gramfort et al. (2014)	305	7498	1.0
news20	Keerthi et al. (2005)	19 996	1 355 191	3.4×10^{-4}
rcv1	Lewis et al. (2004)	20 242	47 236	3.6×10^{-3}
MillionSong	Bertin-Mahieux et al. (2011)	463 715	90	1

E.2 Support identification speed benchmark

Since the Lasso is massively used for its feature selection properties, the speed at which the solvers identify the support of the solution is also an important performance measure. To evaluate the behavior of solvers in this task, it is sufficient to add a single new variable in the `Objective`, namely the ℓ_0 pseudonorm of the iterate, allowing to produce Figure E.1 in addition to Figure 4.

E.3 Convergence in terms of iteration

While practitioners are mainly concerned with the time it takes to solve their optimization problem, one may also be interested in the convergence as a function of the number of iterations. This is

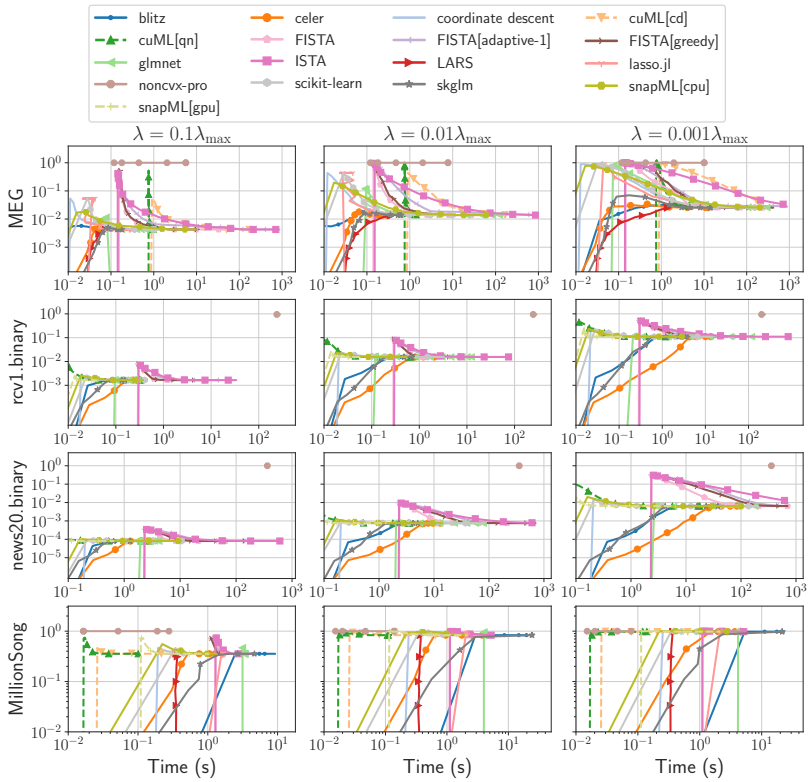


Figure E.1: Additional benchmark for the Lasso on variants of the `Objective` (*columns*). The curves display the fraction of non-zero coefficients in iterates θ_t ($\|\theta_t\|_0/p$), as a function of time.

particularly relevant to compare theoretical convergence rates with experiments. Benchopt natively supports such functionality. Yet, this makes sense only if one iteration of each algorithm costs the same. Figure E.2 presents such a case on the *leukemia* dataset, using algorithms for which one iteration costs $n \times p$. One can observe that cyclic coordinate descent as implemented in `Cython` in `scikit-learn` or in `Numba` lead to identical results, while they outperform proximal gradient methods.

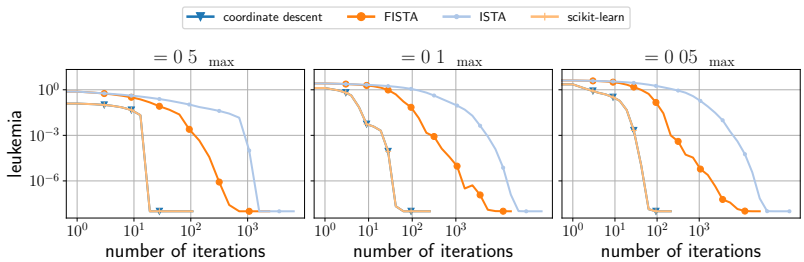


Figure E.2: Convergence speed with respect to the number of iterations for some solvers of the Lasso benchmark on the *leukemia* dataset.

F ResNet18

F.1 Description of the benchmark

Setting up the benchmark The three currently supported frameworks are `TensorFlow/Keras` (Abadi et al., 2015; Chollet et al., 2015), `PyTorch` (Paszke et al., 2019) and `PyTorch Lightning` (Falcon et al., 2020). We report here results for `TensorFlow/Keras` and `PyTorch`. To guarantee that the model behaves consistently across the different considered frameworks, we implemented several consistency unit tests. We followed the best practice of each framework to make sure to achieve the optimal computational efficiency. In particular, we tried as much as possible to use official code from the frameworks, and not third-party code. We also optimized and profiled the data pipelines to make sure that our training was not IO-bound. Our benchmarks were run using `TensorFlow` version 2.8 and `PyTorch` version 1.10.

Descriptions of the datasets

Table F.1: Description of the datasets used in the ResNet18 image classification benchmark

Dataset	Content	References	Classes	Train Size	Val. Size	Test Size	Image Size	RGB
CIFAR-10	natural images	Krizhevsky (2009)	10	40k	10k	10k	32	✓
SVHN	digits in natural images	Netzer et al. (2011)	10	58.6k	14.6k	26k	32	✓
MNIST	handwritten digits	LeCun et al. (2010)	10	50k	10k	10k	28	✗

In Table F.1, we present some characteristics of the different datasets used for the ResNet18 benchmark. In particular, we specify the size of each splits when using the train-validation-test split strategy. The test split is always fixed, and is the official one.

While the datasets are downloaded and preprocessed using the official implementations of the frameworks, we made sure to test that they matched using a unit test.

ResNet The ResNet18 is the smallest variant of the architecture introduced by He et al. (2016). It consists in 3 stages:

1. A feature extension convolution that goes from 3 channels (RGB, or a repeated grayscale channel in the *MNIST* case) to 64, followed by a batch normalization and a ReLU.
2. A series of residual blocks. Residual blocks are grouped by scale, and each individual group starts with a strided convolution to reduce the image scale (except the first one). As the scale increases, so does the number of features (64, 128, 256, 512). In the ResNet18 case, each scale group has two individual residual blocks and there are four scales. A residual block is comprised of three convolution operations, all followed by a batch normalization layer, and the first two also followed by a ReLU. The input is then added to the output of the third batch normalization layer before being fed to a ReLU.
3. A classification head that performs global average pooling, before applying a fully connected (i.e. dense) layer to obtain logits.

Training's hyperparameters

Table F.2: Hyperparameters used for each solver. If a hyperparameter's value is not specified in the table, it was set as the default of the implementation (checked to be consistent across frameworks).

Hyperparameter	SGD	Adam
Learning Rate	0.1	0.001
Momentum	0.9	N/A
Weight Decay	5×10^{-4}	0.02
Batch Size	128	128

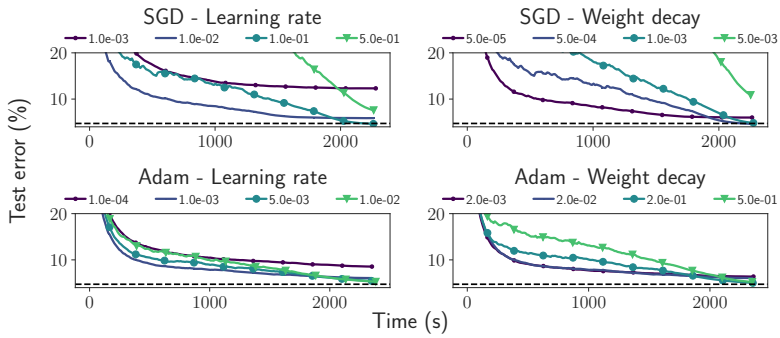


Figure F.1: **ResNet18 image classification benchmark on *CIFAR-10* for different values of learning rate and weight decay for SGD and Adam.** The default values are that reported in Table F.2. In dashed black is the state of the art for *CIFAR-10* with a ResNet18 measured by Zhang et al. (2019). Curves are exponentially smoothed.

In Table F.2, we specify the hyperparameters we used for the benchmark. For the SGD, the values were taken from the pytorch-cifar GitHub repository, while for Adam we took the most relevant ones from the work of Wightman et al. (2021).

F.2 Hyperparameter sensitivity

In the benchmark presented in Section 5, we consider fixed hyperparameters chosen from common practices to train ResNet18 models for an image classification task. However, in practice, these hyperparameters must be carefully set, either via a grid search, or via more adapted algorithms such as random search (**bergstra2012random**) or bayesian optimization (Paszke et al., 2019). It is therefore important to evaluate how sensitive an optimizer is to choosing the right parameters, as more sensitive methods will require more exhaustive hyperparameters search. In Figure F.1, we study this issue using `Benchopt` for image classification on *CIFAR-10*. Despite achieving the best results in terms of accuracy, SGD is way more sensitive to the choice of hyperparameters than Adam.[6]

Another way to look at hyperparameter sensitivity is to evaluate how a given selection of hyperparameters performs for different tasks. Figure 5 shows that while SGD is sensitive to the choice of learning rate and weight decay, the selected values work very well across 3 different datasets.

F.3 Aligning TensorFlow and PyTorch ResNet18 training

We summarized in Table F.3 the different elements that have to be considered to align the training of a ResNet18 in PyTorch and TensorFlow. Let us detail here some lines of this table:

- **Bias in convolutions:** It can be seen in TensorFlow/Keras official implementation, that convolutions operations use a bias. This is in contrast to PyTorch's official implementation in `torchvision` which does not. Since the convolutions are followed by batch normalization layers, with a mean removal, the convolutions' bias is a spurious parameter, as was noted by Ioffe and Szegedy (2015). We therefore chose to use unbiased convolutions.
- **Decoupled weight decay scaling:** this led us to scale manually the weight decay used in TensorFlow by the learning rate when setting it. Moreover, because the weight decay is completely decoupled from the learning rate, it is important to update it accordingly when using a learning rate schedule, as noted in the TensorFlow documentation.
- **Batch normalization momentum:** an important note here is that the convention used to implement the batch normalization momentum is not the same in the 2 frameworks. Indeed we have the relationship $\texttt{momentum}_{\text{TF}} = 1 - \texttt{momentum}_{\text{PT}}$.

[6] We ran the same experiment on two other datasets obtaining similar figures.

Table F.3: Differences in off-the-shelf implementations of various components when training ResNet18 for image classification in `PyTorch` and `TensorFlow`. The selected versions are put in bold font for components that we were able to reconcile. This highlights the numerous details to consider when comparing experimental results.

Component	`PyTorch`	`TensorFlow/Keras`
Bias in convolutions	✗	✓
Decoupled weight decay scaling	**Multiplied by learning rate**	Completely decoupled
Batch normalization momentum	**0.9**	0.99
Conv2D weights init.	**Fan out, normal**	Fan average, uniform
Classification head init. (weights)	**Fan in, uniform**	Fan average, uniform
Classification head init. (bias)	**Fan in, uniform**	Zeros
Striding in convolutions	**Starts one off**	Ends one off
Variance estimation in batch norm	unbiased (eval)/biased (training)	biased

- **Conv2D weights intialization:** TensorFlow/Keras uses the default intialization which is a Glorot uniform intialization (Glorot and Bengio, 2010). PyTorch uses a He normal initialization (He et al., 2015). We used TensorFlow's Variance Scaling framework to differentiate the 2.
- **Striding in convolutions:** when using a stride of 2 in convolutions on an even-size image, one needs to specify where to start the convolution in order to know which lines (one in every two) in the image will be removed. The decision is different between TensorFlow and PyTorch. This is not expected to have an effect on the final performance, but it makes it more difficult to compare the architectures when unit testing. We therefore decided to align the models on this aspect as well.
- **Variance estimation in batch normalization:** in order to estimate the batch variance during training for batch normalization layers, it is possible to chose between the unbiased and the biased variance estimator. The unbiased variance estimator applies a Bessel correction to the biased variance estimator, namely a multiplication by a factor $\frac{m}{m-1}$, where m is the number of samples used to estimate. It is to be noted that PyTorch does uses the biased estimator in training, but stores the unbiased estimator for use during inference. TensorFlow does not allow for such a behaviour, and the 2 are therefore not reconcilable[7]. Arguably this inconsistency should not play a big role with large batch sizes, but can be significant for smaller batches, especially in deeper layers where the feature map size (and therefore the number of samples used to compute the estimates) is reduced.

Adapting official ResNet implementations to small images In addition to these elements, it is important to adapt the reference implementations of both frameworks to the small image case. Indeed, for the case of ImageNet, the ResNet applies two downsampling operations (a stride-2 convolution and a max pooling) at the very beginning of the network to make the feature maps size more manageable. In the case of smaller images, it is necessary to do without these downsampling operations (i.e. perform the convolution with stride 1 and get rid of the max pooling).

Coupled weight decay in TensorFlow In TensorFlow, the SGD implementation does not allow the setting of coupled weight decay. Rather, one has to rely on the equivalence (up to a scale factor of 2) between coupled weight decay and L2 regularization. However, in TensorFlow/Keras, adding L2 regularization on an already built model (which is the case for the official ResNet implementation), is not straightforward and we relied on the workaround of Silva (2019).

F.4 VGG benchmark on CIFAR-10

In order to show how flexible `Benchopt` is, we also ran a smaller version of the ResNet benchmark using a VGG16 (Simonyan and Zisserman, 2015) network instead of a ResNet18. In the `Benchopt` framework, this amounts to specifying a different model in the objective, while all the other pieces of code in the benchmark remain unchanged. Note that the VGG official implementations also need to

[7] It is possible to use the unbiased estimator in TensorFlow for the batch normalization, even if not documented, but its application is consistent between training and inference unlike PyTorch.

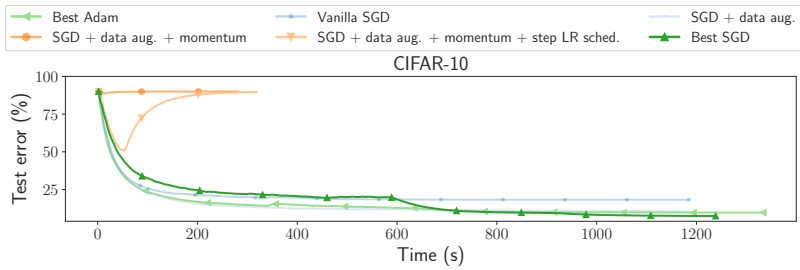


Figure F.2: **VGG16 image classification benchmark with `PyTorch` solvers.** The best SGD configuration features data augmentation, momentum, step learning rate schedule and weight decay.

be adapted to the CIFAR-10 case by changing the classification head. This was not specified in the original paper, where no experiment was conducted on small-scale datasets, and we relied on available open source implementations (cifar10-vgg16 and cifar-vgg) to make this decision. Importantly, these implementations use batch normalization to make the training of VGG more robust to initialization, which is not the case in the official framework implementations.

In Figure F.2, we see that for the case of VGG, the application of weight decay is so important that without it, in cases with momentum, the model does not converge.

G ℓ_1-regularized logistic regression

This additional benchmark is dedicated to ℓ_1-regularized logistic regression, in the same setting as Problem (2) but this time with an ℓ_1-regularization for the parameters of the model:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^p} \sum_{i=1}^{n} \log\left(1 + \exp(-y_i X_i^\top \theta)\right) + \lambda \|\theta\|_1 \ . \tag{6}$$

G.1 List of solvers and datasets used in the ℓ_1-regularized logistic regression benchmark

The code for the benchmark is available at https://github.com/benchopt/benchmark_logreg_l1/. Table G.1 and Table G.2 present the solvers and datasets used in this benchmark.

Table G.1: List of solvers used in the ℓ_1-regularized logistic regression benchmark

Solver	References	Description	Language
`blitz`	Johnson and Guestrin (2015)	CD + working set	`Python, C++`
`coordinate descent`	Friedman et al. (2010)	(Cyclic) Minimization along coordinates	`Python (Numba)`
`coordinate descent (Newton)`	Friedman et al. (2010)	CD + Newton	`Python (Numba)`
`celer`	Massias et al. (2018)	CD + working set + dual extrapolation	`Python (Cython)`
`copt[FISTA line search]`	Pedregosa et al. (2020), Beck and Teboulle (2009)	FISTA (ISTA + acceleration) + line search	`Python (Cython)`
`copt[PGD]`	Pedregosa et al. (2020), Combettes and Wajs (2005)	Proximal Gradient Descent	`Python (Cython)`
`copt[PGD linesearch]`	Pedregosa et al. (2020), Combettes and Wajs (2005)	Proximal Gradient Descent + linesearch	`Python (Cython)`
`copt[saga]`	Pedregosa et al. (2020)	SAGA (Variance reduced stochastic method)	`Python (Cython)`
`copt[svrg]`	Pedregosa et al. (2020)	SVRG (Variance reduced stochastic method)	`Python (Cython)`
`cuML[gpu]`	Raschka et al. (2020)	L-BFGS + GPU	`Python, C++`
`cuML[qn]`	Raschka et al. (2020)	Orthant-Wise Limited Memory Quasi-Newton (OWL-QN)	`Python, C++`
`cyanure`	Mairal (2019)	Proximal Minimization by Incremental Surrogate Optimization (MISO)	`Python, C++`
`lightning`	Blondel and Pedregosa (2016)	(Cyclic) Coordinate Descent	`Python (Cython)`
`scikit-learn[liblinear]`	Pedregosa et al. (2011), Fan et al. (2008)	Truncated Newton Conjugate-Gradient	`Python (Cython)`
`scikit-learn[lbfgs]`	Pedregosa et al. (2011), Virtanen et al. (2020)	L-BFGS (Quasi-Newton Method)	`Python (Cython)`
`scikit-learn[newton-cg]`	Pedregosa et al. (2011), Virtanen et al. (2020)	Truncated Newton Conjugate-Gradient	`Python (Cython)`
`snapml[gpu=True]`	Dünner et al. (2018)	CD + GPU	`Python, C++`
`snapml[gpu=False]`	Dünner et al. (2018)	CD	`Python, C++`

Table G.2: List of the datasets used in the ℓ_1-regularized logistic regression benchmark

Datasets	**References**	**Samples (n)**	**Features (p)**	**Density**
gisette	Guyon et al. (2004)	6000	5000	9.9×10^{-1}
colon-cancer	Guyon et al., 2004	62	2000	1.0
news20.binary	Keerthi et al. (2005)	19 996	1 355 191	3.4×10^{-4}
rcv1.binary	Guyon et al., 2004	20 242	19 959	3.6×10^{-3}

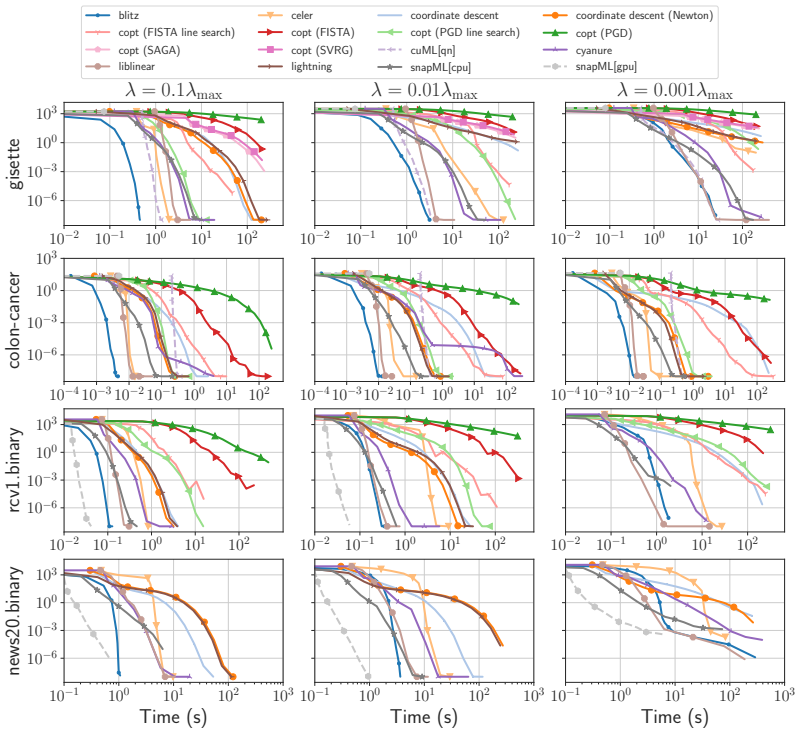


Figure G.1: Benchmark for the ℓ_1-regularized logistic regression on variants of the `Objective` (*columns*). The curves display the suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, as a function of time. The first column corresponds to the objective detailed in Problem (6) with $\lambda = 0.1\|X^\top y\|_\infty/2$, the second one with $\lambda = 0.01\|X^\top y\|_\infty/2$ and the third column with $\lambda = 0.001\|X^\top y\|_\infty/2$.

G.2 Results

The results of the ℓ_1-regularized logistic regression benchmark are in Figure G.1.

H Unidimensional total variation

The use of 1D Total Variation regularization takes its root in the taut-string algorithm (Barlow and Brunk, 1972) and can be seen as a special case of either the Rudin-Osher-Fatemi model (Rudin et al., 1992) or the Generalized Lasso (Tibshirani and Taylor, 2011) for a quadratic data fit term. It reads

$$\theta^* \in \arg\min_{\theta \in \mathbb{R}^p} F(y, X\theta) + \lambda \left\| D\theta \right\|_1 \quad , \tag{7}$$

where F is a data fidelity term, $X \in \mathbb{R}^{n \times p}$ is a design matrix with n samples and p features, $y \in \mathbb{R}^n$ is the target vector, $\lambda > 0$ is a regularization hyperparameter, and $D \in \mathbb{R}^{(p-1) \times p}$ is a finite difference operator defined by $(D\theta)_k = \theta_{k+1} - \theta_k$ for all $1 \leq k \leq p-1$ (it is also possible to use cyclic differences).

Most often, the data fidelity term is the ℓ^2-loss $F(y, z) = \frac{1}{2} \left\| y - z \right\|_2^2$, following an additive Gaussian noise hypothesis. But the data fit term can also account for other types of noises, such as noises with heavy tails using the Huber loss $F(y, z) = |y - z|_\mu$ where $|\cdot|_\mu$ is defined coordinate-wise by

$$|t|_\mu = \begin{cases} \frac{1}{2} t^2 & \text{if } |t| \leq \mu \\ \mu |t| - \frac{\mu^2}{2} & \text{otherwise.} \end{cases}$$

Problem (7) promotes piecewise-constant solutions – alternatively said, solutions such that their gradients is sparse – and was proved to be useful in several applications, in particular for change point detection (Bleakley and Vert, 2011; Tibshirani, 2014), for BOLD signal deconvolution in functional MRI (Karahanoğlu et al., 2013; Cherkaoui et al., 2019) or for detrending in oculomotor recordings (Lalanne et al., 2020).

The penalty $\theta \mapsto \left\| D\theta \right\|_1$ is convex but non-smooth, and its proximity operator has no closed form. Yet as demonstrated by Condat (2013a), the taut-string algorithm allows to compute this proximity operator in $O(p^2)$ operations in the worst case, but it enjoys a $O(p)$ complexity in most cases. Other methods do not rely on this proximity operator and directly solve Problem (7), using either primal-dual approaches (Chambolle and Pock, 2011; Condat, 2013b), or solving the dual problem (Komodakis and Pesquet, 2015). Finally, for 1-dimensional TV regularization, one can also use the synthesis formulation (Elad et al., 2006) to solve the problem. By setting $z = D\theta$ and $\theta = Lz + \rho$ where $L \in \mathbb{R}^{p \times p-1}$ is a lower trianglar matrix representing an integral operator (cumulative sum), the problem is equivalent to a Lasso problem, and ρ^* has a closed-form expression (see *e.g.,* Bleakley and Vert 2011 for a proof). As a consequence, any lasso solver can be used to obtain the solution of the Lasso problem z^* and the solution of the original Problem (7) u^* is retrieved as $u^* = Lz^* + \rho^*$.

The code for the benchmark is available at `https://github.com/benchopt/benchmark_tv_1d/` and Table H.1 details the different algorithms used in this benchmark.

Table H.1: List of solvers used in the 1D Total Variation benchmarks

Solver	References	Formulation	Description
`ADMM`	Boyd et al. (2011)	Analysis	Primal-Dual Augmented Lagrangian
`ChambollePock`	Chambolle and Pock (2011)	Analysis	Primal-Dual Hybrid Gradient
`CondatVu`	Condat (2013b)	Analysis	Primal-Dual Hybrid Gradient
`DPGD`	Komodakis and Pesquet (2015)	Analysis	Dual proximal GD (+ acceleration)
`PGD`	Condat (2013a)	Analysis	Proximal GD + taut–string
	Barbero and Sra (2018)		ProxTV (+ acceleration)
`celer`	Massias et al. (2018)	Synthesis	CD + working set (lasso) *only for ℓ_2 data-fit*
`FP`	Combettes and Glaudin (2021)	Synthesis	Fixed point with block updates
`ISTA`	Daubechies et al. (2004)	Synthesis	Proximal GD (+ acceleration)
`skglm`	Bertrand et al. (2022)	Synthesis	CD + working set

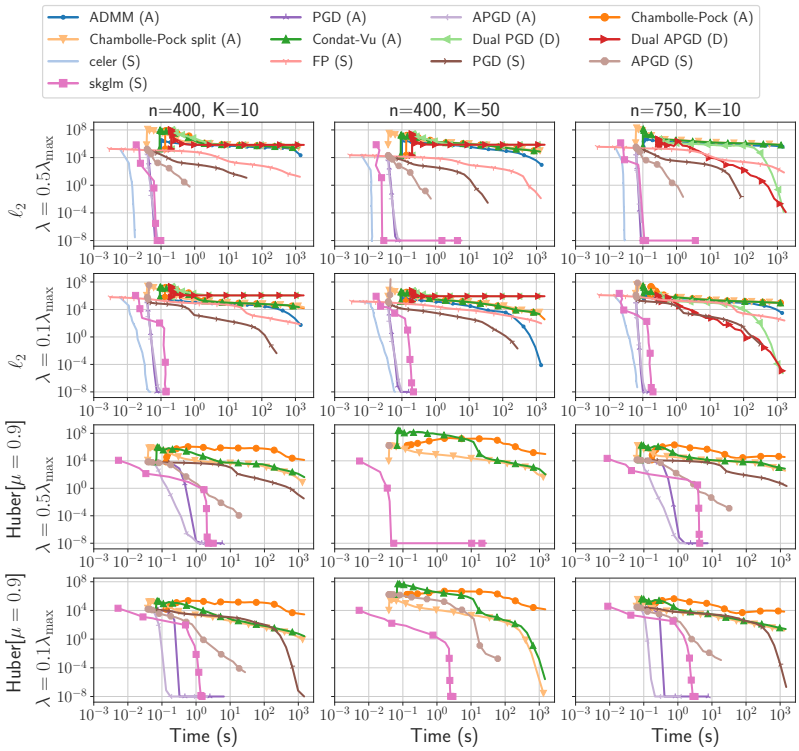


Figure H.1: Benchmark for the TV-regularized regression regression, on 13 solvers, 4 variants of the `Objective` (*rows*), and 3 configurations for a simulated dataset (*columns*). The curves display the suboptimality of the iterates, $f(\theta^t) - f(\theta^*)$, as a function of time. The solvers in this benchmark showcase the three resolution approaches with the Analysis (A), Dual (D) and Synthesis (S) formulations.

Simulated dataset We use here simulated data, as applications based on fMRI and EOG signals require access to open and preprocessed data that we will make available on `OpenML` Vanschoren et al., 2013 in the future. The data are generated as follows: a block signal $\bar{\theta} \in \mathbb{R}^p$ is generated by sampling first a sparse random vector $z \in \mathbb{R}^p$ with K non-zero coefficients positioned randomly, and taking random values following a $\mathcal{N}(0, 1)$ distribution. Finally, $\bar{\theta}$ is obtained by discrete integration as $\bar{\theta}_i = \sum_{k=1}^{i} z_k$ for $1 \leq i \leq p$. The design matrix $X \in \mathbb{R}^{n \times p}$ is a Gaussian random design with $X_{ij} \sim \mathcal{N}(0, 1)$. The observations y are obtained as $y = X\bar{\theta} + \epsilon$, with $\epsilon \sim \mathcal{N}(0, 0.01)$ a Gaussian white noise. For all experiments, we used $p = 500$ and vary the number of non-zero coefficient K, and the number of rows n of the matrix X.

Results Figure H.1 shows that the solvers using the synthesis formulation and coordinate descent-based solvers for the Lasso (ℓ_2 data fit term) work best on this type of problem. For the Huber data fit term, the solver using the analysis formulation and the taut-string algorithm for the proximal operator are faster. An interesting observation from this benchmark is the behavior of the solvers based on primal-dual splitting or dual formulation. We observe that for all these solvers, the objective starts by increasing. This is probably due to a sub-optimal initialization of the dual variable compared to the primal one. While this initialization is seldom described in the literature, it seems to have a large

impact on the speed of these algorithms. This shows how `Benchopt` allows to reveal such behavior, and could lead to practical guidelines on how to select this dual initialization.

Extensions We plan to extend this benchmark in the future to consider higher dimensional problems – *e.g.,* 2D TV problems for images – or higher order TV regularization, such as Total Generalized Variation Bredies et al., 2010 or inf-convolution of TV functionals Chambolle and Lions, 1997 – used of instance for change point detection (Tibshirani, 2014). Yet, for 2D or higher dimensional problems, we can no longer use the synthesis formulation. It is however possible to apply the taut-string method of Condat (2013a) and graph-cut methods of Boykov et al. (2001) and Kolmogorov and Zabin (2004) for anisotropic TV, and dual or primal-dual methods for isotropic, such as Primal-Dual Hybrid Gradient algorithm (Chambolle and Pock, 2011).

I Linear regression with minimax concave penalty (MCP)

The Lasso problem (Tibshirani, 1996) is a least-squares regression problem with a convex non-smooth penalty that induces sparsity in its solution. However, despite its success and large adoption by the machine learning and signal processing communities, it is plagued with some statistical drawbacks, such as bias for large coefficients. To overcome these issues, the standard approach is to consider non-convex sparsity-inducing penalties. Several penalties have been proposed: *Smoothly Clipped Absolute Deviation* (SCAD, Fan and Li 2001), the *Log Sum penalty* (Candès et al., 2008), the *capped-ℓ_1 penalty* (Zhang, 2010b) or the *Minimax Concave Penalty* (MCP, Zhang 2010a).

This benchmark is devoted to least-squares regression with the latter, namely the problem:

$$\theta^* \in \arg\min_{\theta \in \mathbb{R}^p} \frac{1}{2n} \|y - X\theta\|^2 + \sum_{j=1}^{p} \rho_{\lambda,\gamma}(\theta_j) \ , \tag{8}$$

where $X \in \mathbb{R}^{n \times p}$ is a design matrix containing p features as columns, $y \in \mathbb{R}^n$ is the target vector, and $\rho_{\lambda,\gamma}$ the penalty function that reads as:

$$\rho_{\lambda,\gamma}(t) = \begin{cases} \lambda|t| - \frac{t^2}{2\gamma} \ , & \text{if } |t| \leq \gamma\lambda \ , \\ \frac{\lambda^2\gamma}{2} \ , & \text{if } |t| > \gamma\lambda \ . \end{cases}$$

Similarly to the Lasso, Problem (8) promotes sparse solutions but the optimization problem raises some difficulties due to the non-convexity and non-smoothness of the penalty. Nonetheless, several efficient algorithms have been derived for solving it. The ones we use in the benchmark are listed in Table I.1.

Table I.1: List of solvers used in the MCP benchmark

Solver	References	Short Description
`CD`	Breheny and Huang (2011), Mazumder et al. (2011)	Proximal coordinate descent
`PGD`	Bolte et al. (2014)	Proximal gradient descent
`GIST`	Gong et al. (2013)	Proximal gradient + Barzilai-Borwein rule
`WorkSet CD`	Boisbunon et al. (2014)	Coordinate descent + working set
`skglm`	Bertrand et al. (2022)	Accelerated coordinate descent + Working set

The code for the benchmark is available at `https://github.com/benchopt/benchmark_mcp/`. For this benchmark, we run the solvers on the *colon-cancer* dataset and on the simulated dataset described in Table B.4. We use a signal-to-noise ratio $\texttt{snr} = 3$, a correlation $\rho = 0.6$ with $n = 500$ observations and $p = 2000$ features.

I.1 Results

The result of the benchmark is presented in Figure I.1 The problem is non-convex and solvers are only guaranteed to converge to local minima; hence in Figure I.1 we monitor the distance of the negative gradient to the Fréchet subdifferential of the MCP, representing the violation of the first order optimality condition. Other metrics, such as objective of iterates sparsity, are monitored in the full benchmark, allowing to compare the different limit points obtained by the solvers.

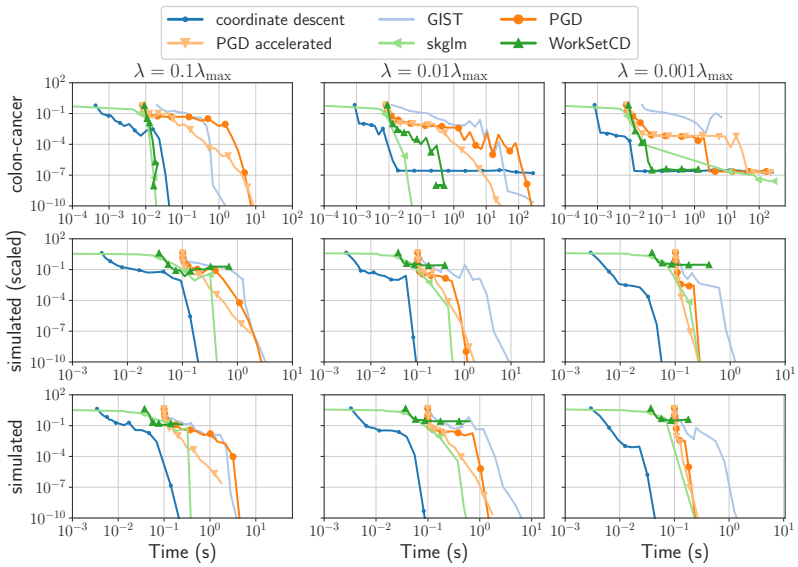


Figure I.1: Benchmark for the MCP regression on variants of the `Objective` (*columns*). The curves display the violation of optimality conditions, $\operatorname{dist}(-X^\top(X\theta_t - y)/n, \partial\rho_{\lambda,\gamma}(\theta_t))$, as a function of time. γ is set to 3, and λ is parameterized as a fraction of the Lasso's $\lambda_{\max}$, $\left\|X^\top y\right\|_\infty / n$.

J Zero-order optimization on standard functions

Zero-order optimization refers to scenarios where only calls to the function to minimize are possible. This is in contrast with first-order optimization where gradient information is available. Grid search, random search, evolution strategies (ES) or Bayesian optimization (BO) are popular methods to tackle such a problem and are most commonly employed for hyperparameter optimization. This setting is also known as black-box optimization.

This benchmark demonstrates the usability of `Benchopt` for zero-order optimization considering functions classically used in the literature (Hansen et al., 2021). The functions are available in the PyBenchFCN package https://github.com/Y1fanHE/PyBenchFCN/. In particular, among the 61 functions of interest we present here (see Figure J.1) a benchmark for three functions, defined for any $\mathbf{x} = (x_1, \dots, x_N) \in \mathbb{R}^N$:

$$f(\mathbf{x}) = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \qquad \text{(Rosenbrock)}$$

$$f(\mathbf{x}) = 10 \cdot N + \sum_{i=1}^{N} [(x_i^2 - 10 \cdot \cos(2\pi x_i)] \qquad \text{(Rastrigin)}$$

$$f(\mathbf{x}) = -20 \cdot \exp\left[-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{N} x_i^2}\right] - \exp\left[\frac{1}{d}\sum_{i=1}^{N} \cos(2\pi x_i)\right] + e + 20 \qquad \text{(Ackley)} \ .$$

For each function, the domain is restricted to a box: $\|\mathbf{x}\|_\infty \leq 32$ for Ackley, $\|\mathbf{x}\|_\infty \leq 30$ for Rosenbrock and $\|\mathbf{x}\|_\infty \leq 5.12$ for Rastrigin. The algorithms considered in the benchmark are listed in Table J.1. As `BFGS` requires first-order information, gradients are approximated with finite-differences.

Table J.1: List of solvers used in the zero-order benchmark

Solver	References	Description
`Basin-hopping`	Wales and Doye (1997) and Virtanen et al. (2020)	Two-phase method: global step + local min.
`Nevergrad-RandomSearch`	Rapin and Teytaud (2018) and Bergstra and Bengio (2012)	Sampler by random search
`Nevergrad-CMA`	Rapin and Teytaud (2018) and Hansen and Ostermeier (2001)	CMA evolutionary strategy
`Nevergrad-TwoPointsDE`	Rapin and Teytaud (2018)	Evolutionary strategy
`Nevergrad-NGOpt`	Rapin and Teytaud (2018)	Adaptive evolutionary algorithm
`Nelder-Mead`	Gao and Han (2012) and Virtanen et al. (2020)	Direct search (downhill simplex)
`BFGS`	Virtanen et al. (2020)	BFGS with finite differences
`Powell`	Powell (1964) and Virtanen et al. (2020)	Conjugate direction method
`optuna-TPE`	Akiba et al. (2019) and Bergstra et al. (2013)	Sampler by Tree Parzen Estimation (TPE)
`optuna-CMA`	Akiba et al. (2019) and Hansen and Ostermeier (2001)	CMA evolutionary strategy

The code for the benchmark is available at https://github.com/benchopt/benchmark_zero_order/.

J.1 Results

The results of the benchmark are presented in Figure J.1. The functions are non-convex and solvers are only guaranteed to converge to local minima; hence in Figure J.1 we monitor the value of the function. The functions are designed such that the global minimum of the function is always 0. One

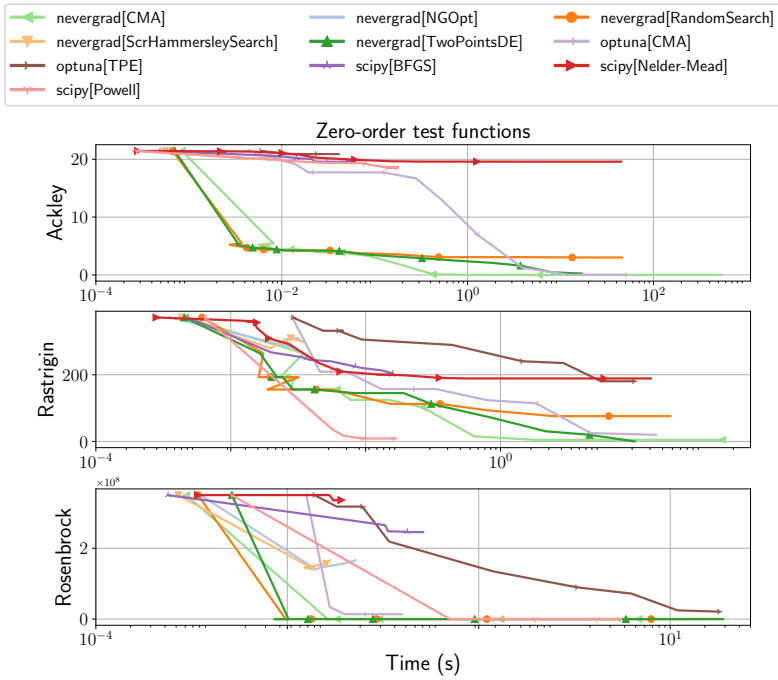


Figure J.1: Benchmark for the zero-order optimization on the Ackley, Rosenbrock and Rastrigin functions in dimension $N = 20$.

can observe that the CMA and TwoPointsDE implementations from `nevergrad` consistently reaches the global minimum. In addition, the CMA implementation from `optuna` is a bit slower than the one from `nevergrad`. Also one can notice that random search offers reasonable results. The TPE method seems to suffer from the curse of dimensionality, as most kernel methods in non-parametric estimation. Finally regarding the `scipy` solvers, `Powell` can be competitive, while `Nelder-Mead` and `BFGS` suffer a lot from local minima.

V

Coordinate Descent for SLOPE

Johan Larsson
Department of Statistics
Lund University, Sweden
johan.larsson@stat.lu.se

Quentin Klopfenstein
Luxembourg Centre for Systems Biomedicine
University of Luxembourg, Luxembourg
quentin.klopfenstein@uni.lu

Mathurin Massias
Univ. Lyon, Inria, CNRS, ENS de Lyon,
UCB Lyon 1, LIP UMR 5668, F-69342
Lyon, France
mathurin.massias@inria.fr

Jonas Wallin
Department of Statistics
Lund University, Sweden
jonas.wallin@stat.lu.se

Abstract

The lasso is the most famous sparse regression and feature selection method. One reason for its popularity is the speed at which the underlying optimization problem can be solved. Sorted L-One Penalized Estimation (SLOPE) is a generalization of the lasso with appealing statistical properties. In spite of this, the method has not yet reached widespread interest. A major reason for this is that current software packages that fit SLOPE rely on algorithms that perform poorly in high dimensions. To tackle this issue, we propose a new fast algorithm to solve the SLOPE optimization problem, which combines proximal gradient descent and proximal coordinate descent steps. We provide new results on the directional derivative of the SLOPE penalty and its related SLOPE thresholding operator, as well as provide convergence guarantees for our proposed solver. In extensive benchmarks on simulated and real data, we demonstrate our method's performance against a long list of competing algorithms.

1 INTRODUCTION

In this paper we present a novel numerical algorithm for Sorted L-One Penalized Estimation (SLOPE, Bogdan et al., 2013; Bogdan et al., 2015; Zeng and Figueiredo, 2014), which, for a design matrix $X \in \mathbb{R}^{n\times p}$ and response vector $y \in \mathbb{R}^n$, is defined as

$$\beta^* \in \operatorname*{arg\,min}_{\beta\in\mathbb{R}^p} P(\beta) = \frac{1}{2}\|y - X\beta\|^2 + J(\beta) \tag{1}$$

where

$$J(\beta) = \sum_{j=1}^{p} \lambda_j |\beta_{(j)}| \tag{2}$$

is the *sorted* ℓ_1 *norm*, defined through

$$|\beta_{(1)}| \geq |\beta_{(2)}| \geq \cdots \geq |\beta_{(p)}| \ , \tag{3}$$

with λ being a fixed non-increasing and non-negative sequence.

The sorted ℓ_1 norm is a sparsity-enforcing penalty that has become increasingly popular due to several appealing properties, such as its ability to control false discovery rate (Bogdan et al., 2015; Kos and Bogdan, 2020), cluster coefficients (Figueiredo and Nowak, 2016; Schneider and Tardivel, 2020), and recover sparsity and ordering patterns in the solution (Bogdan et al., 2022). Contrary to other coefficient clustering approaches such as the fused Lasso (Tibshirani et al., 2005), it is independent of feature order, and in addition does not require prior knowledge of the number of clusters. Finally, unlike other competing sparse regularization methods such as MCP (Zhang, 2010) and SCAD (Fan and Li, 2001), SLOPE has the advantage of being a convex problem (Bogdan et al., 2015).

In spite of the availability of predictor screening rules (Larsson, Bogdan, and Wallin, 2020; Elvira and Herzet, 2021), which help speed up SLOPE in the high-dimensional regime, current state-of-the-art algorithms

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

for SLOPE perform poorly in comparison to those of more established penalization methods such as the lasso (ℓ_1 norm regularization) and ridge regression (ℓ_2 norm regularization). As a small illustration of this issue, we compared the speed at which the SLOPE (Larsson et al., 2022) and glmnet (Friedman et al., 2022) packages solve a SLOPE and lasso problem, respectively, for the `bcTCGA` data set. SLOPE takes 43 seconds to reach convergence, whilst glmnet requires only 0.14 seconds[1]. This lackluster performance has hampered the applicability of SLOPE to many real-world applications. In this paper, we present a remedy for this issue: an algorithm that reaches convergence in only 2.9 seconds on the same problem[2].

A major reason for why algorithms for solving ℓ_1-, MCP-, or SCAD-regularized problems enjoy better performance is that they use coordinate descent (Tseng, 2001; Friedman, Hastie, and Tibshirani, 2010; Breheny and Huang, 2011). Current SLOPE solvers, on the other hand, rely on proximal gradient descent algorithms such as FISTA (Beck and Teboulle, 2009) and the alternating direction method of multipliers method (ADMM, Boyd et al., 2010), which have proven to be less efficient than coordinate descent in empirical benchmarks on related problems, such as the lasso (Moreau et al., 2022). In addition to FISTA and ADMM, there has also been research into Newton-based augmented Lagrangian methods to solve SLOPE (Luo et al., 2019). But this method is adapted only to the $p \gg n$ regime and, as we show in our paper, is outperformed by our method even in this scenario. Applying coordinate descent to SLOPE is not, however, straightforward since convergence guarantees for coordinate descent require the non-smooth part of the objective to be coordinate-wise separable, which is not the case for SLOPE. As a result, naive coordinate descent schemes can get stuck (Figure 1).

In this article, we address this problem by introducing a new, highly effective algorithm for SLOPE based on a hybrid proximal gradient and coordinate descent scheme. Our method features convergence guarantees and reduces the time required to fit SLOPE by orders of magnitude in our empirical experiments.

Notation Let $(i)^-$ be the inverse of (i) such that $\big((i)^-\big)^- = (i)$; see Table 1 for an example of this operator for a particular β. This means that

$$J(\beta) = \sum_{j=1}^{p} \lambda_j |\beta_{(j)}| = \sum_{j=1}^{p} \lambda_{(j)^-} |\beta_j| \,.$$

[1]See Appendix B.1 for details on this experiment.

[2]Note that we do not use any screening rule in the current implementation of our algorithm, unlike the SLOPE package, which uses the strong screening rule for SLOPE (Larsson, Bogdan, and Wallin, 2020).

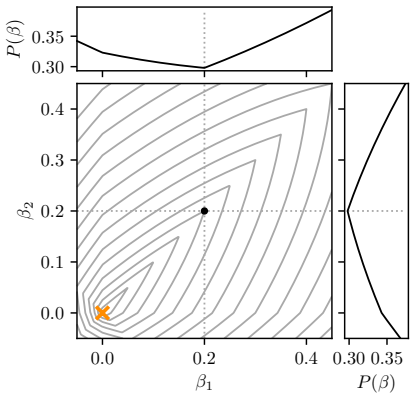


Figure 1: An example of standard coordinate descent getting stuck on a two-dimensional SLOPE problem. The main plot shows level curves for the primal objective (1), with the minimizer $\beta^* = [0, 0]^T$ indicated by the orange cross. The marginal plots display objective values at $\beta_1 = 0.2$ when optimizing over β_2 and vice versa. At $\beta = [0.2, 0.2]^T$, standard coordinate descent can only move in the directions indicated by the dashed lines—neither of which are descent directions for the objective. As a result, the algorithm is stuck at a suboptimal point.

Sorted ℓ_1 norm penalization leads to solution vectors with clustered coefficients in which the absolute values of several coefficients are set to exactly the same value. To this end, for a fixed β such that $|\beta_j|$ takes m distinct values, we introduce $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ and $c_1, c_2, \ldots, c_m$ for the indices and coefficients respectively of the m clusters of β, such that $\mathcal{C}_i = \{j : |\beta_j| = c_i\}$ and $c_1 > c_2 > \cdots > c_m \geq 0$. For a set $\mathcal{C}$, let $\bar{\mathcal{C}}$ denote its complement. Furthermore, let $(e_i)_{i \in [d]}$ denote the canonical basis of $\mathbb{R}^d$, with $[d] = \{1, 2, \ldots, d\}$. Let $X_{i:}$ and $X_{:i}$ denote the i-th row and column, respectively, of the matrix X. Finally, let $\operatorname{sign}(x) = x/|x|$ (with the convention $0/0 = 1$) be the scalar sign, that acts entrywise on vectors.

Table 1: Example of the permutation operator (i) and its inverse $(i)^-$ for $\beta = [0.5, -5, 4]^T$

i	β_i	(i)	$(i)^-$
1	0.5	2	3
2	−5	3	1
3	4	1	2

2 COORDINATE DESCENT FOR SLOPE

Proximal coordinate descent cannot be applied to Problem (1) because the non-smooth term is not separable. If the clusters $\mathcal{C}_1^*, \dots, \mathcal{C}_{m^*}^*$ and signs of the solution β^* were known, however, then the values $c_1^*, \dots, c_{m^*}^*$ taken by the clusters of β^* could be computed by solving

$$\min_{z \in \mathbb{R}^{m^*}} \left(\frac{1}{2} \Big\| y - X \sum_{i=1}^{m^*} \sum_{j \in \mathcal{C}_i^*} z_i \operatorname{sign}(\beta_j^*) e_j \Big\|^2 + \sum_{i=1}^{m^*} |z_i| \sum_{j \in \mathcal{C}_i^*} \lambda_j \right). \tag{4}$$

Conditionally on the knowledge of the clusters and the signs of the coefficients, the penalty becomes separable (Dupuis and Tardivel, 2022), which means that coordinate descent could be used.

Based on this idea, we derive a coordinate descent update for minimizing the SLOPE problem (1) with respect to the coefficients of a single cluster at a time (Section 2.1). Because this update is limited to updating and, possibly, merging clusters, we intertwine it with proximal gradient descent in order to correctly identify the clusters (Section 2.2). In Section 2.3, we present this hybrid strategy and show that is guaranteed to converge. In Section 3, we show empirically that our algorithm outperforms competing alternatives for a wide range of problems.

2.1 Coordinate Descent Update

In the sequel, let β be fixed with m clusters $\mathcal{C}_1, \dots, \mathcal{C}_m$ corresponding to values $c_1, \dots, c_m$. In addition, let $k \in [m]$ be fixed and $s_k = \operatorname{sign} \beta_{\mathcal{C}_k}$. We are interested in updating β by changing only the value taken on the k-th cluster. To this end, we define $\beta(z) \in \mathbb{R}^p$ by:

$$\beta_i(z) = \begin{cases} \operatorname{sign}(\beta_i) z \,, & \text{if } i \in \mathcal{C}_k \,, \\ \beta_i \,, & \text{otherwise} \,. \end{cases} \tag{5}$$

Minimizing the objective in this direction amounts to solving the following one-dimensional problem:

$$\min_{z \in \mathbb{R}} \Big(G(z) = P(\beta(z)) = \frac{1}{2} \|y - X\beta(z)\|^2 + H(z) \Big) \,, \tag{6}$$

where

$$H(z) = |z| \sum_{j \in \mathcal{C}_k} \lambda_{(j)_z^-} + \sum_{j \notin \mathcal{C}_k} |\beta_j| \lambda_{(j)_z^-} \tag{7}$$

is the *partial sorted* ℓ_1 *norm* with respect to the k-th cluster and where we write $\lambda_{(j)_z^-}$ to indicate that the inverse sorting permutation $(j)_z^-$ is defined with respect to $\beta(z)$. The optimality condition for Problem (6) is

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

where $G'(z; \delta)$ is the directional derivative of G in the direction δ. Since the first part of the objective is differentiable, we have

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

where $H'(z; \delta)$ is the directional derivative of H.

Throughout the rest of this section, we derive the solution to (6). To do so, we will introduce the directional derivative for the sorted ℓ_1 norm with respect to the coefficient of the k-th cluster. First, as illustrated in Figure 2, note that H is piecewise affine, with breakpoints at 0 and all $\pm c_i$'s for which $i \neq k$. Hence, the partial derivative is piecewise constant, with jumps at these points; in addition, $H'(\cdot; 1) = H'(\cdot, -1)$ except at these points.

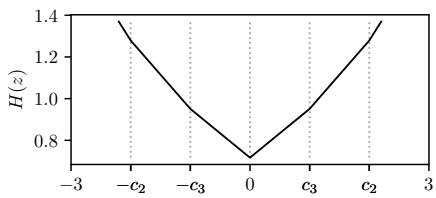


Figure 2: Graph of the partial sorted ℓ_1 norm with $\beta = [-3, 1, 3, 2]^T$, $k = 1$, and so $c_1, c_2, c_3 = (3, 2, 1)$.

Let $C(z)$ be the function that returns the cluster of $\beta(z)$ corresponding to $|z|$, that is

$$C(z) = \{ j : |\beta(z)_j| = |z| \} \,. \tag{8}$$

Remark 2.1. Note that if z is equal to some c_i, then $C(z) = \mathcal{C}_i \cup \mathcal{C}_k$, and otherwise $C(z) = \mathcal{C}_k$. Related to the piecewise affineness of H is the fact that the permutation[3] corresponding to $\beta(z)$ is

$$\begin{cases} \mathcal{C}_k, \mathcal{C}_m, \dots, C_1 & \text{if } z \in (0, c_m) \,, \\ \mathcal{C}_m, \dots, \mathcal{C}_i, \mathcal{C}_k, \mathcal{C}_{i-1}, \dots, C_1 & \text{if } z \in (c_i, c_{i-1}) \\ & \quad \text{and } i \in [\![2, m]\!] \,, \\ \mathcal{C}_m, \dots C_1, \mathcal{C}_k & \text{if } z \in (c_1, +\infty) \,, \end{cases}$$

and that this permutation also reorders $\beta(z \pm h)$ for $z \neq c_i$ $(i \neq k)$ and h small enough. The only change in permutation happens when $z = 0$ or $z = c_i$ $(i \neq k)$. Finally, the permutations differ between $\beta(z + h)$ and $\beta(z - h)$ for arbitrarily small h if and only if $z = c_i \neq 0$.

[3] the permutation is in fact not unique, without impact on our results. This is discussed when needed in the proofs.

We can now state the directional derivative of H.

Theorem 2.2. *Let $c^{\setminus k}$ be the set containing all elements of c except the k-th one: $c^{\setminus k} = \{c_1, \dots c_{k-1}, c_{k+1}, \dots, c_m\}$. Let $\varepsilon_c > 0$ such that*

$$\varepsilon_c < \left|c_i - c_j\right|, \quad \forall\, i \neq j \text{ and } \varepsilon_c < c_m \text{ if } c_m \neq 0\,. \quad (9)$$

The directional derivative of the partial sorted ℓ_1 norm with respect to the k-th cluster, H, in the direction δ is

$$H'(z;\delta) = \begin{cases} \sum_{j \in C(\varepsilon_c)} \lambda_{(j)^-_{\varepsilon_c}} & \text{if } z = 0\,, \\ \operatorname{sign}(z)\delta \sum_{j \in C(z+\varepsilon_c\delta)} \lambda_{(j)^-_{z+\varepsilon_c\delta}} & \text{if } |z| \in c^{\setminus k} \setminus \{0\}, \\ \operatorname{sign}(z)\delta \sum_{j \in C(z)} \lambda_{(j)^-_z} & \text{otherwise}\,. \end{cases}$$

The proof is in Appendix A.1; in Figure 3, we show an example of the directional derivative and the objective function.

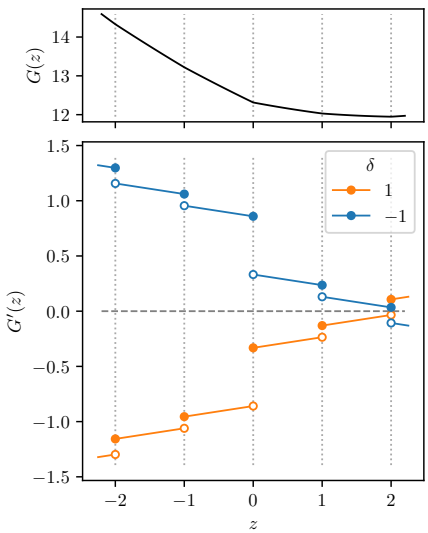


Figure 3: The function G and its directional derivative $G'(\cdot;\delta)$ for an example with $\beta = [-3, 1, 3, 2]^T$, $k = 1$, and consequently $c^{\setminus k} = \{1, 2\}$. The solution of Problem (6) is the value of z for which $G'(z;\delta) \geq 0$ for $\delta \in \{-1, 1\}$, which holds only at $z = 2$, which must therefore be the solution.

Using the directional derivative, we can now introduce the SLOPE thresholding operator.

Theorem 2.3 (The SLOPE Thresholding Operator)**.** *Define $S(x) = \sum_{j \in C(x)} \lambda_{(j)^-_x}$ and let*

$$T(\gamma;\omega,c,\lambda) = \begin{cases} 0 & \text{if } |\gamma| \leq S(\varepsilon_c), \\ \operatorname{sign}(\gamma)c_i & \text{if } \omega c_i + S(c_i - \varepsilon_c) \leq |\gamma| \leq \omega c_i + S(c_i + \varepsilon_c), \\ \frac{\operatorname{sign}(\gamma)}{\omega}\left(|\gamma| - S(c_i + \varepsilon_c)\right) & \text{if } \omega c_i + S(c_i + \varepsilon_c) < |\gamma| < \omega c_{i-1} + S(c_{i-1} - \varepsilon_c), \\ \frac{\operatorname{sign}(\gamma)}{\omega}\left(|\gamma| - S(c_1 + \varepsilon_c)\right) & \text{if } |\gamma| \geq \omega c_1 + S(c_1 + \varepsilon_c). \end{cases}$$

with ε_c defined as in (9). Let $\tilde{x} = X_{\mathcal{C}_k} \operatorname{sign}(\beta_{\mathcal{C}_k})$ and $r = y - X\beta$. Then

$$T\left(c_k\|\tilde{x}\|^2 + \tilde{x}^T r; \|x\|^2, c^{\setminus k}, \lambda\right) = \operatorname*{arg\,min}_{z \in \mathbb{R}} G(z)\,. \quad (10)$$

An illustration of this operator is given in Figure 4.

Remark 2.4. The minimizer is unique because G is the sum of a quadratic function in one variable and a norm.

Remark 2.5. In the lasso case where the λ_i's are all equal, the SLOPE thresholding operator reduces to the soft thresholding operator.

In practice, it is rarely necessary to compute all sums in Theorem 2.3. Instead, we first check in which direction we need to search relative to the current order for the cluster and then search in that direction until we find the solution. The complexity of this operation depends on how far we need to search and the size of the current cluster and other clusters we need to consider. In practice, the cost is typically larger at the start of optimization and becomes marginal as the algorithm approaches convergence and the cluster permutation stabilizes.

2.2 Proximal Gradient Descent Update

The coordinate descent update outlined in the previous section updates the coefficients of each cluster in unison, which allows clusters to merge—but not to split. This means that the coordinate descent updates are not guaranteed to identify the clusters of the solution on their own, and thus are not guaranteed to converge to a solution of (1). To circumvent this issue, we combine these coordinate descent steps with a full proximal gradient step (Bogdan et al., 2015). This enables the algorithm to identify the cluster structure (Liang, Fadili, and Peyré, 2014) due to the partial smoothness property of the sorted ℓ_1 norm that we prove in Appendix A.4. A similar idea has previously been used in

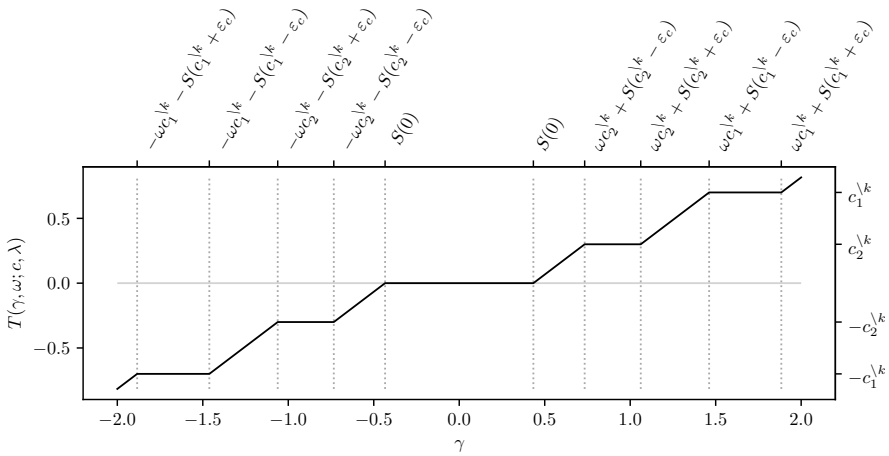


Figure 4: An example of the SLOPE thresholding operator for $\beta = [0.5, -0.5, 0.3, 0.7]^T$, $c = (0.7, 0.5, 0.3)$ with an update for the second cluster ($k = 2$), such that $c^{\setminus k} = (0.5, 0.3)$. Across regions where the function is constant, the operator sets the result to be either exactly 0 or to the value of one of the elements of $\pm c^{\setminus k}$.

Bareilles, Iutzeler, and Malick (2022), wherein Newton steps are taken on the problem structure identified after a proximal gradient descent step.

2.3 Hybrid Strategy

We now present the proposed solver in Algorithm 1. For the first and every v-th iteration[4], we perform a proximal gradient descent update. For the remaining iterations, we take coordinate descent steps.

The combination of the proximal gradient steps and proximal coordinate descent allows us to overcome the problem of vanilla proximal coordinate descent getting stuck because of non-separability and allows us to enjoy the speed-up provided by making local updates on each cluster, as we illustrate in Figure 5.

We now state that our proposed hybrid algorithm converges to a solution of Problem (1).

Lemma 2.6. *Let $\beta^{(t)}$ be an iterate generated by Algorithm 1. Then*

$$\lim_{t\to\infty} \big(P(\beta^{(t)}) - P(\beta^*)\big) = 0.$$

Computional Complexity We examine the complexity of the proximal gradient step and the coordinate descent separately. For the proximal gradient step, the complexity is $\mathcal{O}\big(np + p\log(p)\big)$, where np comes from the matrix-vector multiplication and $p\log(p)$ from the computation of the proximal operator of the sorted ℓ_1 norm (Zeng and Figueiredo, 2014). For the coordinate descent step, the worst case complexity is $\mathcal{O}\big(np + m(m+n)\big)$. As we will see in Section 3, however, the cost of the coordinate descent step turns out to be much lower in practice. The reason for this is

[4]Our experiments suggest that v has little impact on performance as long as it is at least 3 (Appendix B.2). We have therefore set it to 5 in our experiments.

Algorithm 1 Hybrid coordinate descent and proximal gradient descent algorithm for SLOPE

input: $X \in \mathbb{R}^{n\times p}$, $y \in \mathbb{R}^n$, $\lambda \in \{\mathbb{R}^p : \lambda_1 \geq \lambda_2 \geq \cdots > 0\}$, $v \in \mathbb{N}$, $\beta \in \mathbb{R}^p$

for $t \leftarrow 0, 1, \ldots$ **do**
  if $t \bmod v = 0$ **then**
    $\beta \leftarrow \operatorname{prox}_{J/\|X\|_2^2}\Big(\beta - \frac{1}{\|X\|_2^2} X^T(X\beta - y)\Big)$
    Update c, $\mathcal{C}$
  else
    $k \leftarrow 1$
    while $k \leq |\mathcal{C}|$ **do**
      $\tilde{x}_k \leftarrow X_{\mathcal{C}_k} \operatorname{sign}(\beta_{\mathcal{C}_k})$
      $z \leftarrow T(c_k\|\tilde{x}\|^2 - \tilde{x}^T(X\beta - y); \|x\|^2, c^{\setminus k}, \lambda)$
      $\beta_{\mathcal{C}_k} \leftarrow z \operatorname{sign}(\beta_{\mathcal{C}_k})$
      Update c, $\mathcal{C}$
      $k \leftarrow k + 1$
return β

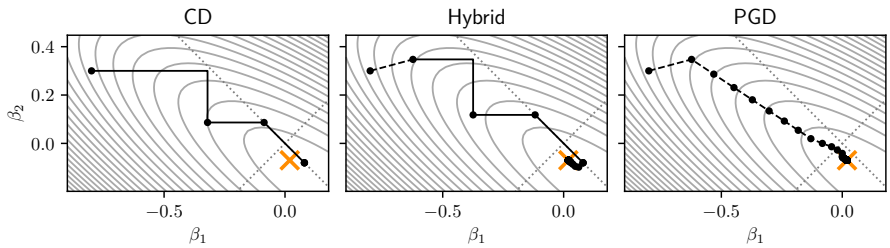


Figure 5: Illustration of the proposed solver. The figures show progress until convergence for the coordinate descent (CD) solver that we use as part of the hybrid method, our hybrid method, and proximal gradient descent (PGD). The orange cross marks the optimum. Dotted lines indicate where the coefficients are equal in absolute value. The dashed lines indicate PGD steps and solid lines CD steps. Each dot marks a complete epoch, which may correspond to only a single coefficient update for the CD and hybrid solvers if the coefficients flip order. Each solver was run until the duality gap was smaller than 10^{-10}. Note that the CD algorithm cannot split clusters and is therefore stuck after the third epoch. The hybrid and PGD algorithms, meanwhile, reach convergence after 67 and 156 epochs respectively.

that the order of the clusters becomes increasingly stable during optimization. If, for instance, the order of the clusters is unchanged with respect to the previous step, then the complexity, in our implementation of Algorithm 1, reduces to $\mathcal{O}(np + mn)$.

Alternative Datafits So far we have only considered sorted ℓ_1-penalized least squares regression. In Appendix C, we consider possible extensions to alternative datafits.

3 EXPERIMENTS

To investigate the performance of our algorithm, we performed an extensive benchmark against the following competitors:

- Alternating direction method of multipliers (`ADMM`, Boyd et al., 2010). We considered several alternative for the choice of the augmented Lagragian parameter: an adaptive method to update the parameter throughout the algorithm (Boyd et al., 2010, Sec. 3.4.1) and fixed values. In the following sections, we only kept the `ADMM` solver with a fixed value of 100 for the augmented Lagrangian parameter. We present in Appendix B.3 a more detailed benchmarks for `ADMM` solvers with different values of this parameter and the adaptive setting. Choosing this parameter is not straightforward and the best value changes across datasets and regularization strengths.
- Anderson acceleration for proximal gradient descent (`Anderson PGD`, Zhang, O'Donoghue, and Boyd, 2020)
- Proximal gradient descent (`PGD`, Combettes and Wajs, 2005)
- Fast Iterative Shrinkage-Thresholding Algorithm (`FISTA`, Beck and Teboulle, 2009)
- Semismooth Newton-Based Augmented Lagrangian (`Newt-ALM`, Luo et al., 2019)
- The hybrid (our) solver (see Algorithm 1) combines proximal gradient descent and coordinate descent to overcome the non-separability of the SLOPE problem. We perform a coordinate descent step every fifth iteration ($v = 5$) in the algorithm. (See Section 2.3.)
- The oracle solver (`oracle CD`) solves Problem (4) with coordinate descent, using the clusters obtained via another solver. Note that it cannot be used in practice as it requires knowledge of the solution's clusters.

We used Benchopt (Moreau et al., 2022) to obtain the convergence curves for the different solvers. Benchopt is a collaborative framework that allows reproducible and automatic benchmarks. The repository to reproduce the benchmark is available at `github.com/klopfe/benchmark_slope`.

Unless we note otherwise, we used the Benjamini–Hochberg method to compute the λ sequence (Bogdan et al., 2015), which sets $\lambda_j = \eta^{-1}(1 - q \times j/(2p))$ for $j = 1, 2, \dots, p$ where η^{-1} is the probit function. For the rest of the experiments section, the parameter q of this sequence has been set to 0.1 if not stated otherwise.[5] We let $\lambda_{\max}$ be the λ sequence such that $\beta^* = 0$, but for

[5] We initially experimented with various settings for q but found that they made little difference to the relative performance of the algorithms.

which any scaling with a strictly positive scalar smaller than one produces a solution with at least one non-zero coefficient. We then parameterize the experiments by scaling $\lambda_{\max}$, using the fixed factors 1/2, 1/10, and 1/50, which together cover the range of very sparse solutions to the almost-saturated case.

We pre-process datasets by first removing features with less than three non-zero values. Then, we center and scale each feature by its mean and standard deviation respectively. For sparse data, we scale each feature by its maximum absolute value. This approach is in line with recommendations in, for instance, Pedregosa et al. (2022).

Each solver was coded in python, using numpy (Harris et al., 2020) and numba (Lam, Pitrou, and Seibert, 2015) for performance-critical code. The code is available at `github.com/jolars/slopecd`. In Appendix D, we provide additional details on the implementations of some of the solvers used in our benchmarks.

The computations were carried out on a computing cluster with dual Intel Xeon CPUs (28 cores) and 128 GB of RAM.

3.1 Simulated Data

The design matrix X was generated such that features had mean one and unit variance, with correlation between features j and j' equal to $0.6^{|j-j'|}$. We generated $\beta \in \mathbb{R}^p$ such that k entries, chosen uniformly at random throughout the vector, were sampled from a standard Gaussian distribution. The response vector, meanwhile, was set to $y = X\beta + \varepsilon$, where ε was sampled from a multivariate Gaussian distribution with variance such that $\|X\beta\|/\|\varepsilon\| = 3$. The different scenarios for the simulated data are described in Table 2.

Table 2: Scenarios for the simulated data in our benchmarks, including the number of rows (n), columns (p), signals (k), and the fraction of non-zero entries (density) of X.

Scenario	n	p	k	Density
1	200	20 000	20	1
2	20 000	200	40	1
3	200	200 000	20	0.001

In Figure 6, we present the results of the benchmarks on simulated data. We see that for smaller fractions of $\lambda_{\max}$ our hybrid algorithm allows significant speedup in comparison to its competitors mainly when the number of features is larger than the number of samples. On very large scale data such as in simulated data setting 3, we see that the hybrid solver is faster than its competitors by one or two orders of magnitude.

For the second scenario, notice that all solvers take considerably longer than the `oracle CD` method to reach convergence. This gap is a consequence of Cholesky factorization in the case of `ADMM` and computation of $\|X\|_2$ in the remaining cases. For the hybrid method, we can avoid this cost, with little impact on performance, since $\|X\|_2$ is used only in the PGD step.

3.2 Real data

The datasets used for the experiments have been described in Table 3 and were obtained from Chang and Lin (2011), Chang and Lin (2022), and Breheny (2022).

Table 3: List of real datasets used in our experiments, including the number of rows (n), columns (p), signals (k), and the fraction of non-zero entries (density) of the corresponding X matrices. See Table 6 in Appendix E for references on these datasets.

Dataset	n	p	Density
`bcTCGA`	536	17 322	1
`news20`	19 996	1 355 191	0.000 34
`rcv1`	20 242	44 504	0.0017
`Rhee2006`	842	360	0.025

Figure 7 shows the suboptimality for the objective function P as a function of the time for the four different datasets. We see that when the regularization parameter is set at $\lambda_{\max}/2$ and $\lambda_{\max}/10$, our proposed solver is faster than all its competitors—especially when the datasets become larger. This is even more visible for the `news20` dataset where we see that our proposed method is faster by at least one order of magnitude.

When the parametrization value is set to $\lambda_{\max}/50$, our algorithm remains competitive on the different datasets. It can be seen that the different competitors do not behave consistently across the datasets. For example, the `Newt-ALM` method is very fast on the `bcTCGA` dataset but is very slow on the `news20` dataset whereas the `hybrid` method remains very efficient in both settings.

4 DISCUSSION

In this paper we have presented a new, fast algorithm for solving Sorted L-One Penalized Estimation (SLOPE). Our method relies on a combination of proximal gradient descent to identify the cluster structure of the solution and coordinate descent to allow the algorithm to take large steps. In our results, we have shown that our method often outperforms all competitors by orders of magnitude for high-to-medium levels of regularization and typically performs among the best algorithms for low levels of regularization.

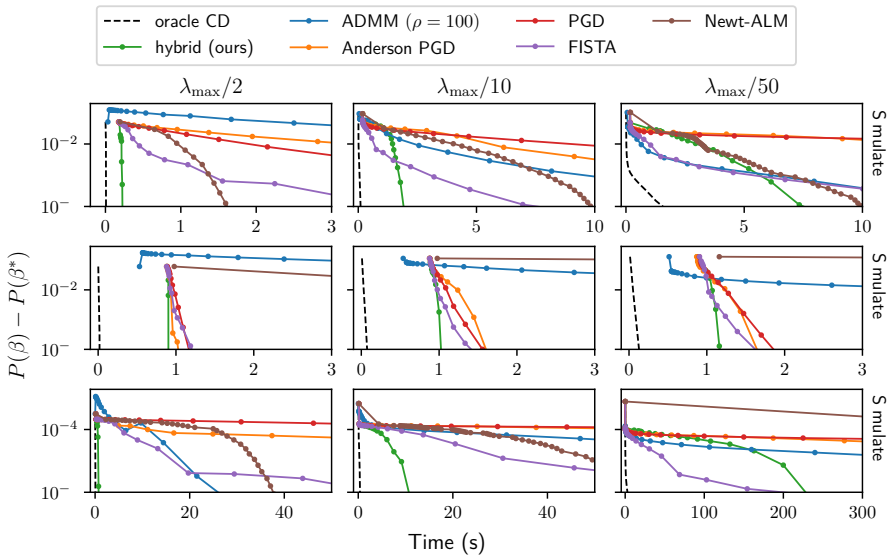


Figure 6: Benchmark on simulated datasets. The plots show suboptimality (difference between the objective at the current iterate, $P(\beta)$, and the optimum, $P(\beta^*)$) as a function of time for SLOPE on multiple simulated datasets and λ sequences of varying strength.

We have not, in this paper, considered using screening rules for SLOPE (Larsson, Bogdan, and Wallin, 2020; Elvira and Herzet, 2021). Although screening rules work for any algorithm considered in this article, they are particularly effective when used in tandem with coordinate descent (Fercoq, Gramfort, and Salmon, 2015) and, in addition, easy to implement due to the nature of coordinate descent steps. Coordinate descent is moreover especially well-adapted to fitting a path of λ sequences (Friedman et al., 2007; Friedman, Hastie, and Tibshirani, 2010), which is standard practice during cross-validating to obtain an optimal λ sequence.

Future research directions may include investigating alternative strategies to split clusters, for instance by considering the directional derivatives with respect to the coefficients of an entire cluster at once. Another potential approach could be to see if the full proximal gradient steps might be replaced with batch stochastic gradient descent in order to reduce the costs of these steps. It would also be interesting to consider whether gap safe screening rules might be used not only to screen predictors, but also to deduce whether clusters are able to change further during optimization. Finally, combining cluster identification of proximal gradient descent with solvers such as second order ones as in Bareilles, Iutzeler, and Malick (2022) is a direction of interest.

Acknowledgements

The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg (Varrette et al., 2022) (see `hpc.uni.lu`).

The results shown here are in whole or part based upon data generated by the TCGA Research Network: `https://www.cancer.gov/tcga`.

References

Bareilles, Gilles, Franck Iutzeler, and Jérôme Malick (2022). "Newton acceleration on manifolds identified by proximal gradient methods". In: *Mathematical Programming*, pp. 1–34.

Beck, Amir and Marc Teboulle (2009). "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems". In: *SIAM Journal on Imaging Sciences* 2.1, pp. 183–202.

Bogdan, Małgorzata, Ewout van den Berg, Weijie Su, and Emmanuel Candès (2013). "Statistical Estima-

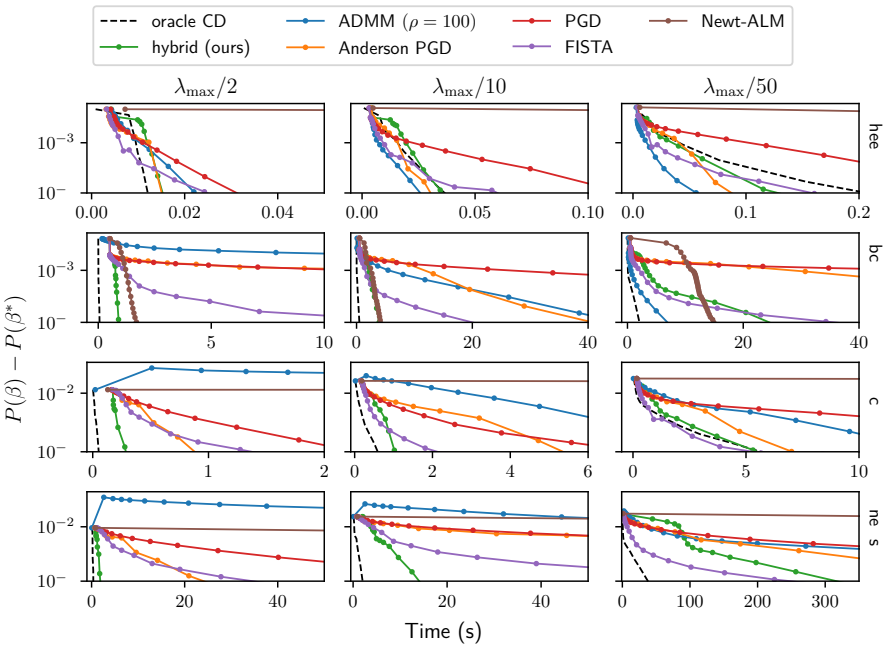


Figure 7: Benchmark on real datasets. The plots show suboptimality (difference between the objective at the current iterate, $P(\beta)$, and the optimum, $P(\beta^*)$) as a function of time for SLOPE on multiple simulated datasets and λ sequences of varying strength.

tion and Testing via the Sorted L1 Norm". arXiv: 1310.1969 [math, stat].

Bogdan, Małgorzata, Xavier Dupuis, Piotr Graczyk, Bartosz Kołodziejek, Tomasz Skalski, Patrick Tardivel, and Maciej Wilczyński (May 17, 2022). "Pattern Recovery by SLOPE". arXiv: 2203.12086 [math, stat]. URL: http://arxiv.org/abs/2203.12086 (visited on 06/03/2022).

Bogdan, Małgorzata, Ewout van den Berg, Chiara Sabatti, Weijie Su, and Emmanuel Candès (Sept. 2015). "SLOPE - Adaptive Variable Selection via Convex Optimization". In: *The annals of applied statistics* 9.3, pp. 1103–1140.

Boyd, Stephen, N. Parikh, E. Chu, B. Peleato, and J. Eckstein (2011). *MATLAB Scripts for Alternating Direction Method of Multipliers*. Stanford University. URL: https://web.stanford.edu/~boyd/papers/admm/ (visited on 10/11/2022).

Boyd, Stephen, Neil Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein (2010). "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends® in Machine Learning* 3.1, pp. 1–122. ISSN: 1935-8237, 1935-8245.

Breheny, Patrick (2022). *Patrick Breheny*. University of Iowa. URL: https://myweb.uiowa.edu/pbreheny/ (visited on 05/17/2022).

Breheny, Patrick and Jian Huang (2011). "Coordinate Descent Algorithms for Nonconvex Penalized Regression, with Applications to Biological Feature Selection". In: *The Annals of Applied Statistics* 5.1, pp. 232–253.

Chang, Chih-Chung and Chih-Jen Lin (2011). "LIBSVM: A Library for Support Vector Machines". In: *ACM Transactions on Intelligent Systems and Technology* 2.3, 27:1–27:27.

– (2022). *LIBSVM Data: Classification, Regression, and Multi-Label*. LIBSVM - A library for Support Vector Machines. URL: https://www.csie.ntu.

edu.tw/~cjlin/libsvmtools/datasets/ (visited on 10/05/2022).

Combettes, Patrick and Valérie Wajs (2005). "Signal recovery by proximal forward-backward splitting". In: *Multiscale modeling & simulation* 4.4, pp. 1168–1200.

Dupuis, Xavier and Patrick Tardivel (2022). "Proximal operator for the sorted l1 norm: Application to testing procedures based on SLOPE". In: *Journal of Statistical Planning and Inference* 221, pp. 1–8.

Elvira, Clément and Cédric Herzet (2021). *Safe Rules for the Identification of Zeros in the Solutions of the SLOPE Problem*. arXiv: 2110.11784.

Fan, Jianqing and Runze Li (2001). "Variable Selection via Nonconcave Penalized Likelihood and Its Oracle Properties". In: *Journal of the American Statistical Association* 96.456, pp. 1348–1360.

Fercoq, Olivier, Alexandre Gramfort, and Joseph Salmon (2015). "Mind the Duality Gap: Safer Rules for the Lasso". In: *ICML*. Vol. 37. Proceedings of Machine Learning Research, pp. 333–342.

Figueiredo, Mario and Robert Nowak (2016). "Ordered Weighted L1 Regularized Regression with Strongly Correlated Covariates: Theoretical Aspects". In: *AISTATS*, pp. 930–938.

Friedman, Jerome, Trevor Hastie, Holger Höfling, and Robert Tibshirani (2007). "Pathwise Coordinate Optimization". In: *The Annals of Applied Statistics* 1.2, pp. 302–332.

Friedman, Jerome, Trevor Hastie, Rob Tibshirani, Balasubramanian Narasimhan, Kenneth Tay, Noah Simon, Junyang Qian, and James Yang (Apr. 15, 2022). *Glmnet: Lasso and Elastic-Net Regularized Generalized Linear Models*. Version 4.1-4. URL: https://CRAN.R-project.org/package=glmnet (visited on 09/20/2022).

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1, pp. 1–22.

Harris, Charles R, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. (2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362.

Keerthi, S. Sathiya and Dennis DeCoste (2005). "A Modified Finite Newton Method for Fast Solution of Large Scale Linear SVMs". In: *JMLR* 6.12, pp. 341–361.

Kos, Michał and Małgorzata Bogdan (2020). "On the Asymptotic Properties of SLOPE". In: *Sankhya A* 82.2, pp. 499–532.

Lam, Siu Kwan, Antoine Pitrou, and Stanley Seibert (2015). "Numba: A llvm-based python jit compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pp. 1–6.

Larsson, Johan, Małgorzata Bogdan, and Jonas Wallin (2020). "The Strong Screening Rule for SLOPE". In: *NeurIPS*. Vol. 33, pp. 14592–14603.

Larsson, Johan, Jonas Wallin, Malgorzata Bogdan, Ewout van den Berg, Chiara Sabatti, et al. (Mar. 14, 2022). *SLOPE: Sorted L1 Penalized Estimation*. Version 0.4.1. URL: https://CRAN.R-project.org/package=SLOPE (visited on 09/20/2022).

Lewis, A. S. (Jan. 2002). "Active Sets, Nonsmoothness, and Sensitivity". In: *SIAM Journal on Optimization* 13.3, pp. 702–725. ISSN: 1052-6234, 1095-7189.

Lewis, David D., Yiming Yang, Tony G. Rose, and Fan Li (2004). "RCV1: A New Benchmark Collection for Text Categorization Research". In: *JMLR* 5, pp. 361–397.

Liang, Jingwei, Jalal Fadili, and Gabriel Peyré (2014). "Local linear convergence of Forward–Backward under partial smoothness". In: *Advances in neural information processing systems* 27.

Luo, Ziyan, Defeng Sun, Kim-Chuan Toh, and Naihua Xiu (2019). "Solving the OSCAR and SLOPE Models Using a Semismooth Newton-Based Augmented Lagrangian Method". In: *Journal of Machine Learning Research* 20.106, pp. 1–25.

Moreau, Thomas, Mathurin Massias, Alexandre Gramfort, Pierre Ablin, Pierre-Antoine Bannier, et al. (2022). "Benchopt: Reproducible, efficient and collaborative optimization benchmarks". In: *NeurIPS*.

National Cancer Institute (2022). *The Cancer Genome Atlas Program*. National Cancer Institute. URL: https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga (visited on 05/17/2022).

Paige, Christopher C. and Michael A. Saunders (1982). "LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares". In: *ACM Transactions on Mathematical Software* 8.1, pp. 43–71. ISSN: 0098-3500.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al. (Sept. 16, 2022). *Preprocessing Data*. scikit-learn. URL: https://scikit-learn/stable/modules/preprocessing.html (visited on 01/29/2023).

Rhee, Soo-Yon, Jonathan Taylor, Gauhar Wadhera, Asa Ben-Hur, Douglas L. Brutlag, and Robert W. Shafer (2006). "Genotypic Predictors of Human Immunodeficiency Virus Type 1 Drug Resistance". In: *Proceedings of the National Academy of Sciences* 103.46, pp. 17355–17360.

Schneider, Ulrike and Patrick Tardivel (2020). *The Geometry of Uniqueness, Sparsity and Clustering in Penalized Estimation*. arXiv: 2004.09106. URL: http://arxiv.org/abs/2004.09106.

Tibshirani, Robert, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight (2005). "Sparsity and smoothness via the fused lasso". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.1, pp. 91–108.

Tseng, Paul (2001). "Convergence of a block coordinate descent method for nondifferentiable minimization". In: *Journal of optimization theory and applications* 109.3, pp. 475–494.

Vaiter, Samuel, Charles Deledalle, Jalal Fadili, Gabriel Peyré, and Charles Dossal (Aug. 2017). "The Degrees of Freedom of Partly Smooth Regularizers". In: *Annals of the Institute of Statistical Mathematics* 69.4, pp. 791–832. ISSN: 00203157.

Varrette, S., H. Cartiaux, S. Peter, E. Kieffer, T. Valette, and A. Olloh (July 2022). "Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0". In: *Proc. of the 6th ACM High Performance Computing and Cluster Technologies Conf. (HPCCT 2022)*. Fuzhou, China: Association for Computing Machinery (ACM). ISBN: 978-1-4503-9664-6.

Zangwill, Willard I. (1969). *Nonlinear Programming: A Unified Approach*. 1st ed. New Orleans, USA: Prentice-Hall. 384 pp. ISBN: 978-0-13-623579-8.

Zeng, Xiangrong and Mario Figueiredo (2014). *The Ordered Weighted ℓ_1 Norm: Atomic Formulation, Projections, and Algorithms*. arXiv: 1409.4271.

Zhang, Cun-Hui (Apr. 2010). "Nearly Unbiased Variable Selection under Minimax Concave Penalty". In: *The Annals of Statistics* 38.2, pp. 894–942.

Zhang, Junzi, Brendan O'Donoghue, and Stephen Boyd (2020). "Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations". In: *SIAM Journal on Optimization* 30.4, pp. 3170–3197.

Supplement to *Coordinate Descent for SLOPE*

A PROOFS

A.1 Proof of Theorem 2.2

Let $c^{\setminus k}$ be the set containing all elements of c except the k-th one: $c^{\setminus k} = \{c_1, \dots c_{k-1}, c_{k+1}, \dots, c_m\}$.

From the observations in Remark 2.1, we have the following cases to consider: $|z| \in c^{\setminus k}$, $|z| = 0$, and $|z| \notin \{0\} \cup c^{\setminus k}$.

Since $C(z + \delta h) = C(z) = \mathcal{C}_k$ and $\operatorname{sign}(z + \delta h) = \operatorname{sign}(z)$ for h small enough,

$$\begin{aligned}
H(z+\delta h) - H(z) &= \sum_{j=1}^{p} |\beta(z+\delta h)_j| \lambda_{(j)^-_{z+\delta h}} - \sum_{j=1}^{p} |\beta(z)_j| \lambda_{(j)^-_z} \\
&= \sum_{j=1}^{p} (|\beta(z+\delta h)_j| - |\beta(z)_j|) \lambda_{(j)^-_z} \\
&= \sum_{j=1}^{p} (|\beta(z+\delta h)_j| - |\beta(z)_j|) \lambda_{(j)^-_z} \\
&= \sum_{j \in C(z)}^{p} (|\beta(z+\delta h)_j| - |\beta(z)_j|) \lambda_{(j)^-_z} \\
&= \sum_{j \in C(z)} \operatorname{sign}(\beta(z)_j)(z + \delta h - z) \lambda_{(j)^-_z} \\
&= \sum_{j \in C(z)} \operatorname{sign}(z) \delta h \lambda_{(j)^-_z} \\
&= \sum_{j \in \mathcal{C}_k} \operatorname{sign}(z) \delta h \lambda_{(j)^-_z} \,. \qquad (11)
\end{aligned}$$

Case 2 Then if $z \neq 0$ and $|z|$ is equal to one of the c_i's, $i \neq k$, one has $C(z) = \mathcal{C}_k \cup \mathcal{C}_i$, $C(z + \delta h) = \mathcal{C}_k$, and $\operatorname{sign}(z + \delta h) = \operatorname{sign}(z)$ for h small enough. Thus

$$\begin{aligned}
H(z+\delta h) - H(z) &= \sum_{j=1}^{p} |\beta(z+\delta h)_j| \lambda_{(j)^-_{z+\delta h}} - \sum_{i=1}^{p} |\beta(z)_j| \lambda_{(j)^-_z} \\
&= \sum_{j \in \mathcal{C}_k \cup \mathcal{C}_i} \left(|\beta(z+\delta h)_j| \lambda_{(j)^-_{z+\delta h}} - |\beta(z)_j| \lambda_{(j)^-_z} \right) \\
&= \sum_{j \in \mathcal{C}_k} (c_i + \delta h) \, \lambda_{(i)^-_{z+\delta h}} - c_i \lambda_{(i)^-_z} + \sum_{j \in \mathcal{C}_i} \left(c_i \lambda_{(j)^-_{z+\delta h}} - c_i \lambda_{(i)^-_z} \right) . \qquad (12)
\end{aligned}$$

Note that there is an ambiguity in terms of permutation, since, due to the clustering, there can be more than one permutation reordering $\beta(z)$. However, choosing any such permutation result in the same values for the computed sums.

Case 3 Finally let us treat the case $z = 0$. If $c_m = 0$ then the proof proceeds as in case 2, with the exception that $|\beta(z + \delta h)| = h$ and so the result is just:

$$H(z + \delta h) - H(z) = h \sum_{j \in \mathcal{C}_k} \lambda_{(i)^-_{z+\delta h}}. \tag{13}$$

If $c_m \neq 0$, then the computation proceeds exactly as in case 1.

A.2 Proof of Theorem 2.3

Recall that $G(z) : \mathbb{R} \to \mathbb{R}$ is a convex, continuous piecewise-differentiable function with breakpoints whenever $|z| = c_i^{\setminus k}$ or $z = 0$. Let $\gamma = c_k \|\tilde{x}\|^2 + \tilde{x}^T r$ and $\omega = \|\tilde{x}\|^2$ and note that the optimality criterion for (6) is

$$\delta(\omega z - \gamma) + H'(z; \delta) \geq 0, \quad \forall \delta \in \{-1, 1\},$$

which is equivalent to

$$\omega z - H'(z; -1) \leq \gamma \leq \omega z + H'(z; 1). \tag{14}$$

We now proceed to show that there is a solution $z^* \in \arg\min_{z \in \mathbb{R}} H(z)$ for every interval over $\gamma \in \mathbb{R}$.

First, assume that the first case in the definition of T holds and note that this is equivalent to (14) with $z = 0$ since $C(\varepsilon_c) = C(-\varepsilon_c)$ and $\lambda_{(j)^-_{-\varepsilon_c}} = \lambda_{(j)^-_{\varepsilon_c}}$. This is sufficient for $z^* = 0$.

Next, assume that the second case holds and observe that this is equivalent to (14) with $z = c_i^{\setminus k}$, since $C(c_i + \varepsilon_c) = C(-c_i - \varepsilon_c)$ and $C(-c_i + \varepsilon_c) = C(c_i - \varepsilon_c)$. Thus $z^* = \operatorname{sign}(\gamma) c_i^{\setminus k}$.

For the third case, we have

$$\sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j)^-_{c_i + \varepsilon_c}} = \sum_{j \in C(c_{i-1} - \varepsilon_c)} \lambda_{(j)^-_{c_{i-1} - \varepsilon_c}}$$

and therefore (14) is equivalent to

$$c_i < \frac{1}{\omega} \left(|\gamma| - \sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j)^-_{c_i + \varepsilon_c}} \right) < c_{i-1}.$$

Now let

$$z^* = \frac{\operatorname{sign}(\gamma)}{\omega} \left(|\gamma| - \sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j)^-_{c_i + \varepsilon_c}} \right) \tag{15}$$

and note that $|z^*| \in \left(c_i^{\setminus k}, c_{i-1}^{\setminus k} \right)$ and hence

$$\frac{1}{\omega} \left(|\gamma| - \sum_{j \in C(c_i + \varepsilon_c)} \lambda_{(j)^-_{c_i + \varepsilon_c}} \right) = \frac{1}{\omega} \left(|\gamma| - \sum_{j \in C(z^*)} \lambda_{(j)^-_{z^*}} \right).$$

Furthermore, since G is differentiable in $\left(c_i^{\setminus k}, c_{i-1}^{\setminus k} \right)$, we have

$$\frac{\partial}{\partial z} G(z) \Big|_{z = z^*} = \omega z^* - \gamma + \operatorname{sign}(z^*) \sum_{j \in C(z^*)} \lambda_{(j)^-_{z^*}} = 0,$$

and therefore (15) must be the solution.

The solution for the last case follows using reasoning analogous to that of the third case.

A.3 Proof of Lemma 2.6

To prove the lemma, we will show that $\lim_{t \to \infty} \beta^{(t)} \in \Omega = \{\beta : 0 \in \partial P(\beta)\}$ using Convergence Theorem A in Zangwill (1969, p. 91). For simplicity, we assume that the point to set map A is generated by v iterations of Algorithm 1, that is $A(\beta^{(0)}) = \{\beta^{(vi)}\}_{i=0}^{\infty}$. To be able to use the theorem, we need the following assumptions to hold.

1. The set of iterates, $A(\beta^{(0)})$ is in a compact set.
2. P is continuous and if $\beta \notin \Omega = \{\beta : 0 \in \partial P(\beta)\}$, then for any $\hat{\beta} \in A(\beta)$ it holds that $P(\hat{\beta}) < P(\beta)$.
3. If $\beta \in \Omega = \{\beta : 0 \in \partial P(\beta)\}$, then for any $\hat{\beta} \in A(\beta)$ it holds that $P(\hat{\beta}) \leq P(\beta)$.

Before tackling these three assumptions, we decompose the map into two parts: $v - 1$ coordinate descent steps, T_{CD}, and one proximal gradient decent step, T_{PGD}. This clearly means that

$$P(T_{\text{CD}}(\beta)) \leq P(\beta)$$

for all $\beta \in \mathbb{R}^p$. For T_{PGD}, we have two useful properties: first, if $||T_{\text{PGD}}(\beta) - \beta|| = 0$, then by Lemma 2.2 in Beck and Teboulle (2009) it follows that $\beta \in \Omega$. Second, by Lemma 2.3 in Beck and Teboulle (2009), using $x = y$, it follows that

$$P(T_{\text{PGD}}(\beta)) - P(\beta) \leq -\frac{L(f)}{2}||T_{\text{PGD}}(\beta) - \beta||^2,$$

where $L(f)$ is the Lipschitz constant of the gradient of $f(\beta) = \frac{1}{2}||y - X\beta||^2$.

We are now ready to prove that the three assumptions hold.

- Assumption 1 follows from the fact that the level sets of P are compact and from $P(T_{PGD}(\beta)) \leq P(\beta)$ and $P(T_{CD}(\beta)) \leq P(\beta)$.
- Assumption 2 holds since if $\beta \notin \Omega$, it follows that $||T_{\text{PGD}}(\beta) - \beta|| > 0$ and thus $P(T_{\text{PGD}}(\beta)) < P(\beta)$.
- Assumption 3 follows from $P(T_{PGD}(\beta)) \leq P(\beta)$ and $P(T_{CD}(\beta)) \leq P(\beta)$.

Using Theorem 1 from Zangwill (1969), this means that Algorithm 1 converges as stated in the lemma.

A.4 Partial Smoothness of the Sorted ℓ_1 Norm

In this section, we prove that the sorted ℓ_1 norm J is partly smooth (Lewis, 2002). This allows us to apply results about the structure identification of the proximal gradient algorithm.

Definition A.1. Let J be a proper closed convex function and x a point of its domain such that $\partial J(x) \neq \emptyset$. J is said to be partly smooth at x relative to a set $\mathcal{M}$ containing x if:

1. $\mathcal{M}$ is a C^2-manifold around x and J restricted to $\mathcal{M}$ is C^2 around x.
2. The tangent space of $\mathcal{M}$ at x is the orthogonal of the parallel space of $\partial J(x)$.
3. ∂J is continuous at x relative to $\mathcal{M}$.

Because the sorted ℓ_1 norm is a polyhedral, it follows immediately that it is partly smooth (Vaiter et al., 2017, Example 18). But since we believe a direct proof is interesting in and of itself, we provide and prove the following proposition here.

Proposition A.2. *Suppose that the regularization parameter λ is a strictly decreasing sequence. Then the sorted ℓ_1 norm is partly smooth at any point of $\mathbb{R}^p$.*

Proof. Let m be the number of clusters of x and $\mathcal{C}_1, \ldots, \mathcal{C}_m$ be those clusters, and let $c_1 > \cdots > c_m > 0$ be the value of $|x|$ on the clusters.

We define ε_c as in Equation (9) and let $\mathcal{B} = \{u \in \mathbb{R}^p : \|u - x\|_\infty < \varepsilon_c/2\}$. Let $v_k \in \mathbb{R}^p$ for $k \in [m]$ be equal to $\text{sign}(x_{\mathcal{C}_k})$ on $\mathcal{C}_k$ and to 0 outside, such that $x = \sum_{k=1}^m c_k v_k$. We define

$$\mathcal{M} = \begin{cases} \text{span}(v_1, \ldots, v_m) \cap \mathcal{B} & \text{if } c_m \neq 0\,, \\ \text{span}(v_1, \ldots, v_{m-1}) \cap \mathcal{B} & \text{otherwise}\,. \end{cases}$$

We will show that J is partly smooth at x relative to $\mathcal{M}$.

As a first statement, we prove that any $u \in \mathcal{M}$ shares the same clusters as x. For any $u \in \mathcal{M}$ there exists $c' \in \mathbb{R}^m$, $u = \sum_{k=1}^m c'_k v_k$ (with $c'_m = 0$ if $c_m = 0$). Suppose that there exist $k \neq k'$ such that $c'_k = c'_{k'}$. Then since $\|x - u\|_\infty = \max_k |c_k - c'_k|$ and $|c_k - c_{k'}| > \varepsilon_c$, one has:

$$\begin{aligned}\varepsilon_c < |c_k - c_{k'}| &= |c_k - c'_k + c'_{k'} - c_{k'}| \\ &\leq |c_k - c'_k| + |c'_{k'} - c_{k'}| \\ &\leq 2\|x - u\|_\infty \\ &\leq \varepsilon_c \,.\end{aligned}$$

This shows that clusters of any $u \in \mathcal{M}$ are equal to clusters of x. Further, clearly the tangent space of $\mathcal{M}$ at x is $\operatorname{span}(v_1, \ldots, v_m)$ if $c_m \neq 0$ and $\operatorname{span}(v_1, \ldots, v_{m-1})$ otherwise.

1. The set $\mathcal{M}$ is then the intersection of a linear subspace and an open ball, and hence is a $\mathcal{C}^2$ manifold. Since the clusters of any $u \in \mathcal{M}$ are the same as the clusters of x, we can write that

$$J(u) = \sum_{k=1}^{m} \left(\sum_{j \in \mathcal{C}_k} \lambda_j \right) c'_k \;, \tag{16}$$

   and hence J is linear on $\mathcal{M}$ and thus $\mathcal{C}^2$ around x.

2. We let $x_\downarrow$ denote a version of x sorted in non-increasing order and let $R : \mathbb{R}^p \to \mathbb{N}^p$ be the function that returns the ranks of the absolute values of its argument. The subdifferential of J at x (Larsson, Bogdan, and Wallin, 2020, Thm. 1)[6] is the set of all $g \in \mathbb{R}^p$ such that

$$g_{\mathcal{C}_i} \in \mathcal{G}_i \triangleq \left\{ s \in \mathbb{R}^{|C_i|} : \begin{cases} \operatorname{cumsum}(|s|_\downarrow - \lambda_{R(g)_{\mathcal{C}_i}}) \preceq 0 & \text{if } x_{\mathcal{C}_i} = \mathbf{0}\,, \\ \operatorname{cumsum}(|s|_\downarrow - \lambda_{R(g)_{\mathcal{C}_i}}) \preceq 0 & \\ \quad \text{and } \sum_{j \in \mathcal{C}_i} (|s_j| - \lambda_{R(g)_{\mathcal{C}_i}}) = 0 & \\ \quad \text{and } \operatorname{sign}(x_{\mathcal{C}_i}) = \operatorname{sign}(s) & \text{otherwise.} \end{cases} \right\} \tag{17}$$

   Hence, the problem can be decomposed over clusters. We will restrict the analysis to a single $\mathcal{C}_i$ without loss of generality and proceed in $\mathbb{R}^{|\mathcal{C}_i|}$.

   - First we treat the case where $|\mathcal{C}_i| = 1$ and $x_{\mathcal{C}_i} \neq 0$. The set $\mathcal{G}_i$ is then the singleton $\{\operatorname{sign}(x_{\mathcal{C}_i})\lambda_{R(s)_{\mathcal{C}_i}}\}$ and its parallel space is simply $\{0\}$. Hence, $\operatorname{par}(\mathcal{G}_i)^\perp = \mathbb{R} = \operatorname{span}(\operatorname{sign}(x)_{\mathcal{C}_i})$.
   - Then, we study the case where $|\mathcal{C}_i| \neq 1$ and $x_{\mathcal{C}_i} \neq \mathbf{0}$. Since for all $j \in [p]$, $\lambda_j \neq 0$ and λ is a strictly decreasing sequence, we have that for $\varepsilon > 0$ small enough, the $|\mathcal{C}_i| - 1$ points $\lambda_{R(g)_{\mathcal{C}_i}} + \varepsilon[-\operatorname{sign}(x_{\mathcal{C}_i})_1, \operatorname{sign}(x_{\mathcal{C}_i})_2, 0, \ldots, 0]^T$, $\lambda_{R(g)_{\mathcal{C}_i}} + \varepsilon[0, -\operatorname{sign}(x_{\mathcal{C}_i})_2, \operatorname{sign}(x_{\mathcal{C}_i})_3, \ldots, 0]^T$, …, $\lambda_{R(g)_{\mathcal{C}_i}} + \varepsilon[0, 0, 0, \ldots, -\operatorname{sign}(x_{\mathcal{C}_i})_{|\mathcal{C}_i|-1}, \operatorname{sign}(x_{\mathcal{C}_i})_{|\mathcal{C}_i|}]^T$ belong to $\mathcal{G}_i$. Since these vectors are linearly independent, and using the last equality in the feasible set that, we have that

$$\sum_{j \in \mathcal{C}_i} \operatorname{sign}(x_j) s_j = \sum_{j \in \mathcal{C}_i} \lambda_{R(g)_{\mathcal{C}_i}} \;.$$

     Its parallel space is simply the set $\{s \in \mathbb{R}^{|\mathcal{C}_i|} : \sum_{j \in \mathcal{C}_i} \operatorname{sign}(x_j) s_j = 0\}$, that is just $\operatorname{span}(\operatorname{sign}(x_{\mathcal{C}_i}))^\perp$. Hence $\operatorname{par}(\mathcal{G}_i)^\perp = \operatorname{span}(\operatorname{sign}(x_{\mathcal{C}_i}))$.
   - Finally, we study the case where $x_{\mathcal{C}_m} = \mathbf{0}$. Then the ℓ_∞ ball $\{s \in \mathbb{R}^{|\mathcal{C}_m|} : \|s\|_\infty \leq \lambda_p\}$ is contained in the feasible set of the differential, hence the parallel space of $\mathcal{G}_m$ is $\mathbb{R}^{|\mathcal{C}_m|}$ and its orthogonal is reduced to $\{\mathbf{0}\}$.

   We can now prove that $\operatorname{par}(\partial J(x))^\perp$ is the tangent space of $\mathcal{M}$. From the decomposability of ∂J (Equation (17)), one has that $u \in \operatorname{par}(\partial J(x))^\perp$ if and only if $u_{\mathcal{C}_i} \in \operatorname{par}(\mathcal{G}_i)^\perp$ for all $i \in [m]$.

[6] We believe there to be a typo in the definition of the subgradient in (Larsson, Bogdan, and Wallin, 2020, Thm. 1). We believe the argument of R should be g, not s, since otherwise there is a dimension mismatch.

If $c_m > 0$, we have

$$\begin{aligned}\operatorname{par}(\partial J(x))^\perp &= \{u \in \mathbb{R}^p : \forall i \in [m], u_{\mathcal{C}_i} \in \operatorname{par}(\mathcal{G}_i)^\perp\} \\ &= \{u \in \mathbb{R}^p : \forall i \in [m], u_{\mathcal{C}_i} \in \operatorname{span}(\operatorname{sign}(x_{\mathcal{C}_i}))\} \\ &= \operatorname{span}(v_1, \dots, v_m)\,.\end{aligned} \tag{18}$$

If $c_m = 0$, we have

$$\begin{aligned}\operatorname{par}(\partial J(x))^\perp &= \{u \in \mathbb{R}^p : \forall i \in [m], u_{\mathcal{C}_i} \in \operatorname{par}(\mathcal{G}_i)^\perp\} \\ &= \{u \in \mathbb{R}^p : \forall i \in [m-1], u_{\mathcal{C}_i} \in \operatorname{span}(\operatorname{sign}(x_{\mathcal{C}_i})) \quad \& \quad u_{\mathcal{C}_m} = \mathbf{0}\} \\ &= \operatorname{span}(v_1, \dots, v_{m-1})\,.\end{aligned} \tag{19}$$

3. The subdifferential of J is a constant set locally around x along $\mathcal{M}$ since the clusters of any point in the neighborhood of x in $\mathcal{M}$ shares the same clusters with x. This shows that it is continuous at x relative to $\mathcal{M}$.

□

Remark A.3. We believe that the assumption $\lambda_1 > \cdots > \lambda_p$ can be lifted, since for example the ℓ_1 and ℓ_∞ norms are particular instances of J that violate this assumption, yet are still partly smooth. Hence this assumption could probably be lifted in a future work using a slightly different proof.

B ADDITIONAL EXPERIMENTS

B.1 glmnet versus SLOPE Comparison

In this experiment, we ran the glmnet (Friedman et al., 2022) and SLOPE (Larsson et al., 2022) packages on the `bcTCGA` dataset, selecting the regularization sequence λ such that there were 100 nonzero coefficients and clusters at the optimum for glmnet and SLOPE respectively. We used a duality gap of 10^{-6} as stopping criteria. The features were centered by their means and scaled by their standard deviation. The code is available at `github.com/jolars/slopecd`.

B.2 Study on Proximal Gradient Descent Frequency

To study the impact of the frequence at which the PGD step in the `hybrid` solver is used, we performed a comparative study with the `rcv1` dataset. We set this parameter to values ranging from 1 *i.e.*, the `PGD` algorithm, to 9 meaning that a PGD step is taken every 9 epochs. The sequence of λ has been set with the Benjamini-Hochberg method and parametrized with $0.1\lambda_{\max}$.

Figure 8 shows the suboptimality score as a function of the time for the different values of the parameter controlling the frequency at which a PGD step is going to be taken. A first observation is that as long as this parameter is greater than 1 meaning that we perform some coordinate descent steps, we observe a significant speed-up. For all our experiments, this parameter was set to 5. The figure also shows that any choice between 3 and 9 would lead to similar performance for this example.

B.3 Benchmark with Different Parameters for the ADMM Solver

We reproduced the benchmarks setting described in Section 3 for the simulated and real data. We compared the `ADMM` solver with our `hybrid` algorithm for different values of the augmented Lagrangian parameter ρ. We tested three different values $10, 100$ and 1000 as well as the adaptive method (Boyd et al., 2010, Sec. 3.4.1).

We present in Figure 9 and Figure 10 the suboptimality score as a function the time for the different solvers. We see that the best value for ρ depends on the dataset and the regularization strengh. The value chosen for the main benchmark (Section 3) performs well in comparison to other `ADMM` solvers. Nevertheless, our `hybrid` approach is consistently faster than the different `ADMM` solvers.

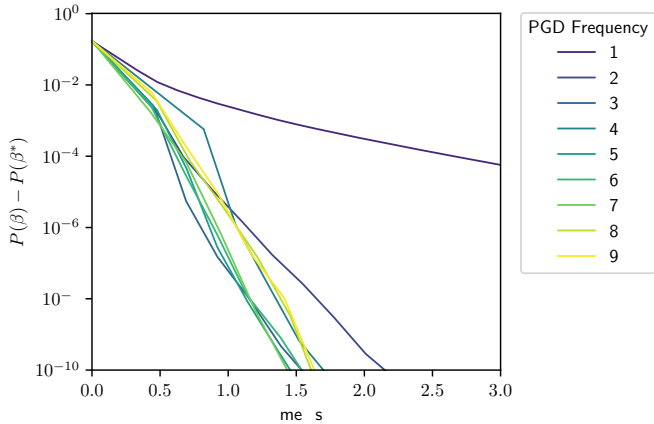


Figure 8: Suboptimality score as a function of the time for different frequencies of the PDG step inside the `hybrid` solver for the `rcv1` dataset

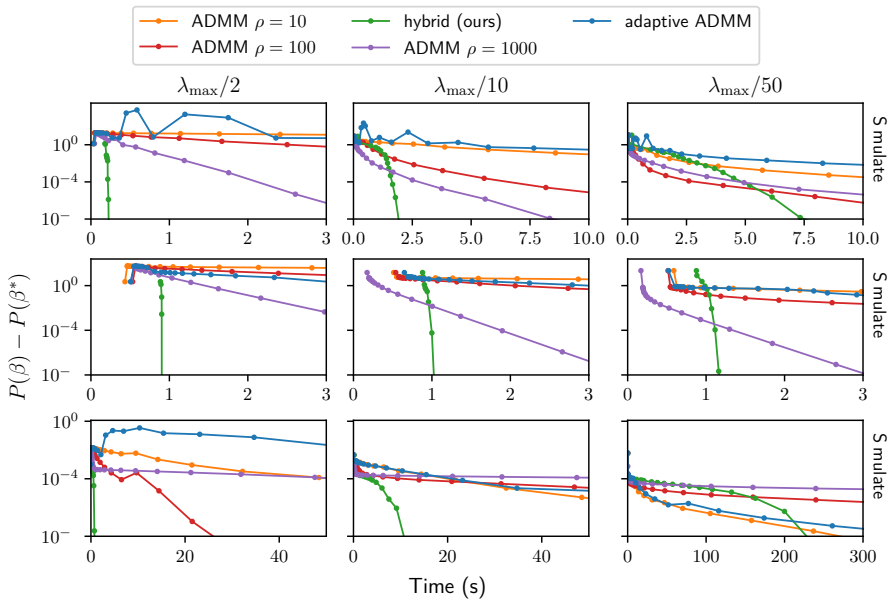


Figure 9: **Benchmark on simulated datasets.** Suboptimality score as a function of time for SLOPE on multiple simulated datasets and for multiple sequence of λ.

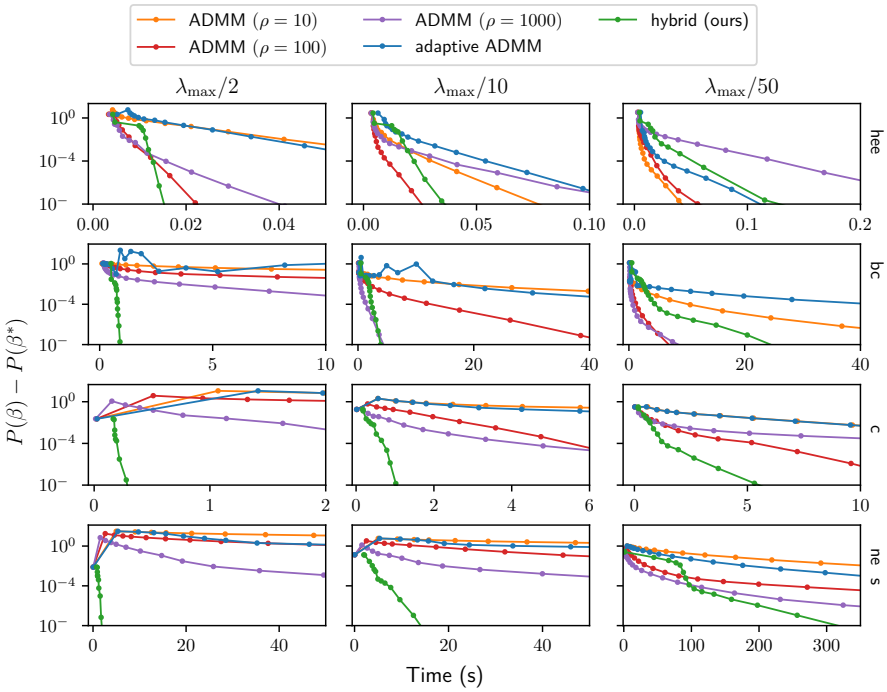


Figure 10: **Benchmark on simulated datasets.** Suboptimality score as a function of time for SLOPE on multiple simulated datasets and for multiple sequence of λ.

B.4 SLOPE Path Benchmarks

The choice of λ sequence in SLOPE is usually made via cross-validation on a training partition of the data across a grid of the q parameter, which controls the shape of the λ sequence, and α, a factor that scales the λ sequence. The α grid is typically a decreasing sequence where the first and last values correspond to the intercept-only and almost-saturated models respectively. The set of models generated from fitting SLOPE across this grid of α values is called the *SLOPE path*. In this section, we report benchmarks on the performance of our algorithm in the case of fitting a SLOPE path to the simulated and real data sets used in Section 3.

We use the same path setup that is used by default for the lasso in the glmnet package (Friedman, Hastie, and Tibshirani, 2010). This entails using a grid of 100 α values spaced evenly on the $\log_e$-scale. The first value, α_1, is chosen such that it leads to the intercept-only model, that is $\alpha_1 \lambda = \lambda_{\max}$ (see Section 3). The last value, α_{100} is set to 10^{-4} if $p > n$ and 10^{-2} otherwise. We terminate the path early if the number of unique nonzero coefficients exceed n[7], if the increase in the coefficient of determination is less than 10^{-4} between two subsequent αs on the path, or if the coefficient of determination equals or exceeds 0.999. The λ sequence setup, preprocessing, and data simulation setup is exactly the same as in Section 3.

For ADMM, we used $\rho = 100$ following the results in Appendix B.3. The Newt-ALM solver is missing from these

[7]Here we deviate from the standard lasso path setup because the lasso can at most select n nonzero coefficients, whilst SLOPE can select at most n nonzero *unique* coefficients

benchmarks because we encountered several issues with convergence.

Our method outperforms the other methods for all of the real datasets (Table 4). In the case of `bcTCGA`, the difference is particularly striking, with our method taking less than one twentieth of the time taken for the runner-up. In the other cases, our method is roughly twice as fast as the second-best method.

Table 4: Time in seconds to fit a full SLOPE path to real data sets. See Table 3 for information about the datasets. For ADMM we set $\rho = 100$.

Dataset	`Rhee2006`	`bcTCGA`	`news20`	`rcv1`
ADMM	47	9252	139 415	6852
Anderson (PGD)	30	9867	29 867	592
FISTA	335	12 682	138 363	1988
hybrid (ours)	14	379	11 574	391

For simulated data (Table 5), our method performs best for the $p > n$ scenarios (1 and 2), being roughly ten and five times faster, respectively, than the runner up. In the case of Scenario 2 where $n > p$, however, Anderson (PGD) instead comes out on top.

Table 5: Time in seconds to fit a full SLOPE path to simulated data sets. See Table 2 for information on what the different scenarios mean. For ADMM we set $\rho = 100$.

Method	Scenario 1	Scenario 2	Scenario 3
ADMM	593	732	649
Anderson (PGD)	539	36	451
FISTA	589	67	279
hybrid (ours)	54	83	49

This experiment was run on a dedicated high-performance computing cluster, using two Intel Xeon E5-2650 v3 (2.3 Ghz, 10-core) CPUs and 64 GB of memory. The computations were enabled by resources provided by LUNARC.

C EXTENSIONS TO OTHER DATAFITS

Our algorithm straightforwardly generalizes to problems where the quadratic datafit $\frac{1}{2}\|y - X\beta\|^2$ is replaced by $F(\beta) = \sum_{i=1}^n f_i(X_{i:}^\top \beta)$, where the f_i's are L smooth (and so F is $L * \|X\|_2^2$-smooth), such as logistic regression.

In that case, one has by the descent lemma applied to $F(\beta(z))$, using $F(\beta) = F(\beta(c_k))$,

$$F(\beta(z)) + H(z) \leq F(\beta) + \sum_{j \in \mathcal{C}_k} \nabla_j F(\beta) \operatorname{sign} \beta_j (z - c_k) + \frac{L\|\tilde{x}\|^2}{2}(z - c_k)^2 + H(z) \tag{20}$$

and so a majorization-minimization approach can be used, by minimizing the right-hand side instead of directly minimizing $F(\beta(z)) + H(z)$. Minimizing the RHS, up to rearranging, is of the form of Problem (6).

D IMPLEMENTATION DETAILS OF SOLVERS

D.1 ADMM

Our implementation of the solver is based on Boyd et al. (2011). For high-dimensional sparse X, we use the numerical LSQR algorithm (Paige and Saunders, 1982) instead of the typical direct linear system solver. We originally implemented the solver using the adaptive step size (ρ) scheme from Boyd et al. (2010) but discovered that it performed poorly. Instead, we used $\rho = 100$ and have provided benchmarks of the alternative configurations in Appendix B.3.

D.2 Newt-ALM

The implementation of the solver is based on the pseudo-code provided in Luo et al. (2019). According to the authors' suggestions, we use the Matrix inversion lemma for high-dimensional and sparse X and the preconditioned conjugate gradient method if, in addition, n is large. Please see the source code for further details regarding hyper-parameter choices for the algorithm.

After having completed our own implementation of the algorithm, we received an implementation directly from the authors. Since our own implementation performed better, however, we opted to use it instead.

E REFERENCES AND SOURCES FOR DATASETS

In Table 6, we list the reference and source (from which the data was gathered) for each of the real datasets used in our experiments.

Table 6: Sources and references for the real data sets used in our experiments.

Dataset	Reference	Source
`bcTCGA`	National Cancer Institute (2022)	Breheny (2022)
`news20`	Keerthi and DeCoste (2005)	Chang and Lin (2022)
`rcv1`	Lewis et al. (2004)	Chang and Lin (2022)
`Rhee2006`	Rhee et al. (2006)	Breheny (2022)

VI

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

Johan Larsson *johan.larsson@stat.lu.se*
Deparment of Statistics
Lund University

Jonas Wallin *jonas.wallin@stat.lu.se*
Department of Statistics
Lund University

Abstract

Many regularized methods, such as the lasso and ridge regression, are sensitive to the scales of the features in the data. As a consequence, it has become standard practice to normalize (center and scale) features such that they share the same scale. For continuous data, the most common strategy is standardization: centering and scaling each feature by its mean and and standard deviation, respectively. For binary data, especially when it is high-dimensional and sparse, the most common strategy, however, is to not scale at all. In this paper, we show that this choice has dramatic effects for the estimated model in the case when the binary features are imbalanced and that these effects, moreover, depend on the type regularization (lasso or ridge) used. In particular, we demonstrate the size of a feature's corresponding coefficient in the lasso is directly related to its class imbalance and that this effect depends on the normalization used. We suggest possible remedies for this problem and also discuss the case when data is mixed, that is, contains both continuous and binary features.

1 Introduction

When the data you want to model is high-dimensional, that is, the number of features p exceed the number of observations n, it is impossible to apply classical statistical models such as standard linear regression since the design matrix $\boldsymbol{X}$ is no longer of full rank. A common remedy to this problem is to *regularize* the model by adding a term to the objective function that punishes models with large coefficients ($\boldsymbol{\beta}$). If we let $g(\boldsymbol{\beta}; \boldsymbol{X}, \boldsymbol{y})$ be the original objective function—which when minimized improves the model's fit to the data $(\boldsymbol{X}, \boldsymbol{y})$—then

$$f(\beta_0, \boldsymbol{\beta}; \boldsymbol{X}, \boldsymbol{y}) = g(\beta_0, \boldsymbol{\beta}; \boldsymbol{X}, \boldsymbol{y}) + h(\boldsymbol{\beta})$$

is a composite function within which we have added a penalty term $h(\boldsymbol{\beta})$. In contrast to g, this penalty depends only on the coefficients ($\boldsymbol{\beta}s$). The intercept, β_0, is not typically penalized.

Some of the most common penalties are the ℓ_1 norm and squared ℓ_2 norm penalties, that is $h(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_1$ or $h(\boldsymbol{\beta}) = \|\boldsymbol{\beta}\|_2^2/2$[1], which, if h is the standard ordinary least-squares objective, represent lasso (Tibshirani, 1996; Santosa & Symes, 1986; Donoho & Johnstone, 1994) and ridge (Tikhonov) regression respectively. Other common penalities include SLOPE (Bogdan et al., 2013; 2015), the minimax-concave penalty (MCP) (Zhang, 2010), hinge loss (used in support vector machines (Cortes & Vapnik, 1995)) and smoothly-clipped absolute deviation (SCAD) (Fan & Li, 2001). Many of these penalities—indeed all of the previously mentioned ones—shrink coefficients in proportion to their sizes.

The issue with this type of shrinkage is that it is typically sensitive to the scales of the features in $\boldsymbol{X}$. A common remedy is to *normalize* the features before fitting the model by translating and dividing each column

[1] Division by two in this case is used only for convenience.

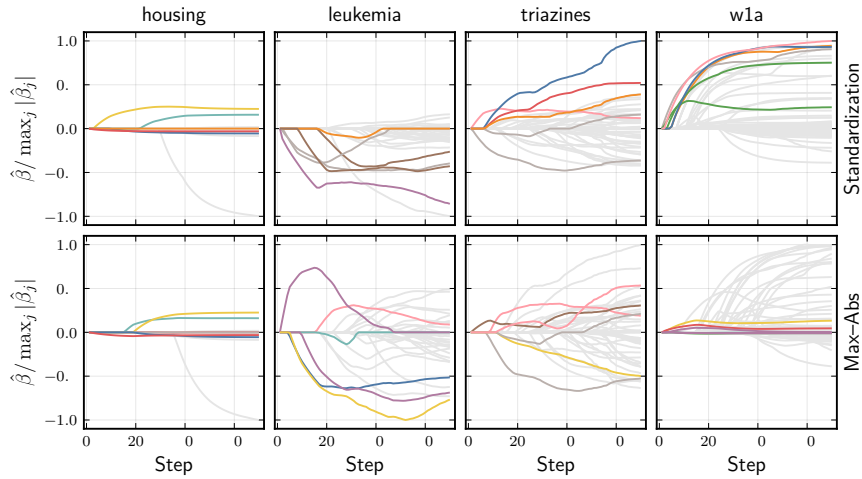


Figure 1: Lasso paths for real datasets using two types of normalization: standardization and maximum absolute value scaling (max–abs). We have fit the lasso path to four different datasets: `housing` (Harrison & Rubinfeld, 1978), `leukemia` (Golub et al., 1999), `triazines` (King), and `w1a` (Platt, 1998). For each dataset, we have colored the coefficients if they were among the first five features to become active in under either of the two types of normalization schemes. We see that the paths differ with regards to the size as well as the signs of the coefficients, and that, in addition, the coefficients to become active first differ between the normalization types.

by respective translation and scaling factors. For some problems, such factors may arise naturally from knowledge of the problem at hand. A researcher may for instance have collected data on coordinates within a limited area and know that the coordinates are measured in meters. Often, however, these scaling factors must be estimated from data. The most popular choices for this type of scaling are based only on the marginal distributions of the features. Some types of normalization, such as that applied in the adaptive lasso[2] (Zou, 2006), however, are based on the conditional distributions of the features and the response. After fitting the model, the estimated coefficients are then usually returned to their original scale. Another reason for normalizing the features is to improve the performance and stability of optimization algorithms used to fit the model. We will not cover this aspect in this paper, but note that it is an important one.

In most sources and discussions on regularized methods, normalization is typically treated as a preprocessing step—separate from modeling. As we will show in this paper, however, the type of normalization used can have a critical effect on the estimated model, sometimes leading to entirely different conclusions with regard to feature importance as well as predictive performance. As a first example of this, consider Figure 1, which displays the lasso paths for four real data sets and two different types of normalization. Each panel shows the union of the first five predictors picked by either type of normalization. The choice of normalization can have a significant impact on the estimated model. In the case of the `leukemia` data set, for instance, the models are starkly different with respect to both the identities of the features selected as well as their signs and magnitudes.

In addition, discussions on the choice of normalization are often focused on computational aspects and data storage requirements, rather than on the statistical properties of the choice of normalization. In our paper,

[2]The adaptive lasso typically uses estimates of the regression coefficients, typically from ordinary-least squares or ridge regression, to scale the features with.

we argue that normalization should rather we considered as an integral part of the model and that it is problematic to base the choice of normalization on the type of data storage, which implicitly encodes the belief that the information in a data set is different if it is stored in a sparse viz-a-viz dense format. At the time of writing, for instance, the popular machine learning library `scikit-learn` (scikit-learn developers, 2024) recommends max–abs scaling in the case of sparse data.

2 Preliminaries

Throughout this paper, we assume that the data is generated from a linear model, that is,

$$y_i = \beta_0^* + \boldsymbol{x}_i^\intercal \boldsymbol{\beta}^* + \varepsilon_i \quad \text{for} \quad i \in \{1, 2, \dots, n\},$$

where we use β_0^* and $\boldsymbol{\beta}^*$ to denote the true intercept and coefficients, respectively, and ε_i to denote measurement noise. $\boldsymbol{X}$ is the $n \times p$ design matrix with columns $\boldsymbol{x}_j$ and $\boldsymbol{y}$ the $n \times 1$ response vector. Furthermore, we use $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}$ to denote our estimates of the intercept and coefficients and use β_0 and β to refer to corresponding variables in the optimization problem. Unless otherwise stated, we assume $\boldsymbol{X}$, β_0^*, and $\boldsymbol{\beta}^*$ to be fixed.

There is ambiguity regarding many of the key terms in the field of normalization. *Scaling*, *standardization*, and *normalizaton* are for instance used interchangeably throughout the literature. Here, we define *normalization* as the process of centering and scaling the feature matrix, which we formalize in Definition 2.1.

Definition 2.1 (Normalization)**.** 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 $\boldsymbol{X}$ and c_j and s_j are the *centering* and *scaling* factors respectively.

Some authors refer to this procedure as *standardization*, but here we define standardization only as the case when centering with the arithmetic mean and scaling with the (uncorrected) standard deviation. Also note that normalization is sometimes defined as the process of scaling the *samples*, rather than the features. We will not consider this type of normalization in this paper.

2.1 Types of Normalization

There are many different strategies for normalizing the design matrix. We list a few of the most common choices in Table 1.

Table 1: Common ways to normalize a matrix of features

Normalization	Centering (c_j)	Scaling (s_j)
Standardization	$\frac{1}{n}\sum_{i=1}^n x_{ij}$	$\sqrt{\frac{1}{n}\sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}$
Max–Abs	0	$\max_i(\lvert x_{ij}\rvert)$
Min–Max	$\min_i(x_{ij})$	$\max_i(x_{ij}) - \min_i(x_{ij})$
Norm Scaling	0	$\lVert\boldsymbol{x}_j\rVert_p$, $p \in \{1, 2, \dots\}$
Adaptive Lasso	0	β_j^{OLS}

Standardization is perhaps the most common type of normalization, at least in the field of statistics. It is sometimes known as *z-scoring* or *z-transformation*. One of the benefits of using standardization is that it simplifies certain aspects of fitting the model. For instance, the intercept term $\hat{\beta}_0$ is equal to the mean of the response $\boldsymbol{y}$. For regularized methods, it is typically the case that we standardize with the uncorrected sample standard deviation (division by n). The downside of standardization is that it involves centering by

the mean, which typically destroys sparsity in the data structure. This is not a problem when the data is stored as a dense matrix; but when the data is sparse, it can lead to a significant increases in memory usage and processing time.

A common alternative to standardization, particularly when data is sparse, is to scale the features by their maximum absolute value (max–abs normalization). This method has no impact on binary data[3], and therefore retains sparsity. For other types of data, it scales the features to take values in the range $[-1, 1]$. Since the scaling is determined by a single value for each feature, the method is naturally sensitive to outliers. In addition, it is for many types of continuous data, such as normally distributed data, the case that the sample maximum depends on the sample size, which makes the method problematic for much continuous data. In Theorem A.1 (Appendix A), we study how this effect comes into play in the case when the feature is normally distributed.

Min-max normalization scales the data to lie in $[0, 1]$. As with maximum absolute value scaling, min-max normalization retains sparsity and also shares its sensitivity to outliers and sample size. Unlike max–abs scaling, min–max scaling is not sensitive to the *location* of the data, only its *spread*. Norm-scaling, scaling by a norm, is seldom used in practice and more often encountered in theoretical work. The norm can be any p-norm, and the choice of p will determine the scaling. Standard choices are $p = 1$, when the scaling is the sum of the absolute values of the features, and $p = 2$, where it is the Euclidean norm. A special case of normalization is the adaptive lasso (Zou, 2006), which is a two-step procedure. In the first step, a model, often ordinary least-squares regression (OLS) or ridge regression, is fit to the data. The estimated coefficients from the model are then used to scale the features.

2.2 The Lasso and Ridge Regression

From now on, we will direct our focus on ridge regression and the lasso. Both of these models are special cases of the elastic net (Zou & Hastie, 2005), which is the ordinary-least squares regression objective regularized by a combination of the ℓ_1 and squared ℓ_2 norms. For the normalized feature matrix $\tilde{\boldsymbol{X}}$, the elastic net is represented by the following convex optimization problem:

$$\underset{\beta_0 \in \mathbb{R}, \boldsymbol{\beta} \in \mathbb{R}^p}{\text{minimize}} \left(f(\beta_0, \boldsymbol{\beta}; \boldsymbol{X}, \boldsymbol{y}, \lambda_1, \lambda_2) = \frac{1}{2} \|\boldsymbol{y} - \beta_0 - \tilde{\boldsymbol{X}}\boldsymbol{\beta}\|_2^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2 \right). \tag{1}$$

We define $(\hat{\beta}_0^{(n)}, \hat{\boldsymbol{\beta}}^{(n)})$ as a solution to the optimization problem in Equation (1). When $\lambda_1 > 0$ and $\lambda_2 = 0$, the elastic net is equivalent to the lasso, and when $\lambda_1 = 0$ and $\lambda_2 > 0$, it is equivalent to ridge regression. Expanding f in Equation (1), we have

$$\frac{1}{2} \left(\boldsymbol{y}^\intercal \boldsymbol{y} - 2(\tilde{\boldsymbol{X}}\boldsymbol{\beta} + \beta_0)^\intercal \boldsymbol{y} + (\tilde{\boldsymbol{X}}\boldsymbol{\beta} + \beta_0)^\intercal (\tilde{\boldsymbol{X}}\boldsymbol{\beta} + \beta_0) \right) + \lambda_1 \|\boldsymbol{\beta}\|_1 + \frac{\lambda_2}{2} \|\boldsymbol{\beta}\|_2^2.$$

Taking the subdifferential with respect to $\boldsymbol{\beta}$ and β_0, the KKT stationarity condition yields the following system of equations:

$$\begin{cases} \tilde{\boldsymbol{X}}^\intercal (\tilde{\boldsymbol{X}}\boldsymbol{\beta} + \beta_0 - \boldsymbol{y}) + \lambda_1 g + \lambda_2 \boldsymbol{\beta} \ni \boldsymbol{0}, \\ n\beta_0 + (\tilde{\boldsymbol{X}}\boldsymbol{\beta})^\intercal \boldsymbol{1} - \boldsymbol{y}^\intercal \boldsymbol{1} = 0, \end{cases} \tag{2}$$

where g is a subgradient of the ℓ_1 norm that has elements g_i such that

$$g_i \in \begin{cases} \{\operatorname{sign} \beta_i\} & \text{if } \beta_i \neq 0, \\ [-1, 1] & \text{otherwise.} \end{cases}$$

2.3 Orthogonal Features

If the features of the normalized design matrix are orthogonal, that is, $\tilde{\boldsymbol{X}}^\intercal \tilde{\boldsymbol{X}} = \operatorname{diag}\left(\tilde{\boldsymbol{x}}_1^\intercal \tilde{\boldsymbol{x}}_1, \ldots, \tilde{\boldsymbol{x}}_p^\intercal \tilde{\boldsymbol{x}}_p\right)$, then Equation (2) can be decomposed into a set of $p + 1$ conditions:

$$\begin{cases} \tilde{\boldsymbol{x}}_j^\intercal \tilde{\boldsymbol{x}}_j \beta_j + \tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{1} \beta_0 - \tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{y} + \lambda_2 \beta_j + \lambda_1 g \ni 0, \quad j = 1, \ldots, p, \\ n\beta_0 + (\tilde{\boldsymbol{X}}\boldsymbol{\beta})^\intercal \boldsymbol{1} - \boldsymbol{y}^\intercal \boldsymbol{1} = 0. \end{cases}$$

[3]Except in the extreme case when all values are 0.

The inclusion of an intercept, β_0, ensures that the locations of the features (their means) does not affect the solution (except for the intercept itself). Therefore, we will from now on assume that the features are mean-centered, that is, $c_j = \bar{\boldsymbol{x}}_j$ for all j and therefore $\tilde{\boldsymbol{x}}_j^\intercal \mathbf{1} = 0$. A solution to the system of equations is then given by the following set of equations (Donoho & Johnstone, 1994):

$$\hat{\beta}_j^{(n)} = \frac{\mathrm{S}_{\lambda_1}\left(\tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{y}\right)}{\tilde{\boldsymbol{x}}_j^\intercal \tilde{\boldsymbol{x}}_j + \lambda_2}, \qquad \hat{\beta}_0^{(n)} = \frac{\boldsymbol{y}^\intercal \mathbf{1}}{n},$$

where S is the soft-thresholding operator, defined as

$$\mathrm{S}_\lambda(z) = \operatorname{sign}(z)\max(|z| - \lambda, 0) = \mathrm{I}_{|z|>\lambda}\left(z - \operatorname{sign}(z)\lambda\right).$$

2.4 Rescaling Regression Coefficients

Normalization changes the optimization problem and therefore its solution, the coefficients, which will now be on the scale of the normalized features. We, however, are interested in $\hat{\boldsymbol{\beta}}$: the coefficients on the scale of the original problem. To obtain these, we transform the coefficients from the normalized poblem, $\hat{\beta}_j^{(n)}$, back via

$$\hat{\beta}_j = \frac{\hat{\beta}_j^{(n)}}{s_j} \quad \text{for} \quad j = 1, 2, \ldots, p. \tag{3}$$

There is a similar transformation for the intercept which we omit here since we are not interested in it.

3 Bias and Variance of the Elastic Net Estimator

Now, assume that $\boldsymbol{X}$ and $\boldsymbol{\beta}$ are fixed and that $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, where ε_i is identically and independently distributed noise with mean zero and finite variance σ_ε^2. As in the previous section, we assume that the feature vectors are orthogonal. We are interested in the expected value of Equation (3), $\mathrm{E}\,\hat{\beta}_j$. Let

$$Z = \tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{y} = \tilde{\boldsymbol{x}}_j^\intercal(\boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}), \qquad d_j = s_j(\tilde{\boldsymbol{x}}_j^\intercal \tilde{\boldsymbol{x}}_j + \lambda_2)$$

so that $\hat{\beta}_j = \mathrm{S}_{\lambda_1}(Z)/d_j$. Since d_j is fixed under our assumptions, we will direct most of our focus towards $S_{\lambda_1}(Z)$. First observe that

$$\begin{aligned}
\mathrm{E}\,Z = \mu &= \mathrm{E}\left(\tilde{\boldsymbol{x}}_j^\intercal(\boldsymbol{x}_j\beta_j + \boldsymbol{\varepsilon})\right) = \tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{x}_j \beta_j, \\
\operatorname{Var} Z = \sigma^2 &= \operatorname{Var}\left(\tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{\varepsilon}\right) = \sigma_\varepsilon^2 \|\tilde{\boldsymbol{x}}_j\|_2^2.
\end{aligned}$$

The expected value of the soft-thresholding estimator is

$$\begin{aligned}
\mathrm{E}\,\mathrm{S}_\lambda(Z) &= \int_{-\infty}^{\infty} \mathrm{S}_\lambda(z) f_Z(z)\,\mathrm{d}z \\
&= \int_{-\infty}^{\infty} \mathrm{I}_{|z|>\lambda}(z - \operatorname{sign}(z)\lambda) f_Z(z)\,\mathrm{d}z \\
&= \int_{-\infty}^{-\lambda} (z+\lambda) f_Z(z)\,\mathrm{d}z + \int_{\lambda}^{\infty} (z-\lambda) f_Z(z)\,\mathrm{d}z.
\end{aligned}$$

And then the bias of $\hat{\beta}_j$ with respect to the true coefficient β_j^* is

$$\mathrm{E}\,\hat{\beta}_j - \beta_j^* = \frac{1}{d_j}\mathrm{E}\,\mathrm{S}_\lambda(Z) - \beta_j^*.$$

Finally, we note that the variance of the soft-thresholding estimator is

$$\operatorname{Var} S_\lambda(Z) = \int_{-\infty}^{-\lambda} (z+\lambda)^2 f_Z(z)\,\mathrm{d}z + \int_{\lambda}^{\infty} (z-\lambda)^2 f_Z(z)\,\mathrm{d}z - \left(\mathrm{E}\,\mathrm{S}_\lambda(Z)\right)^2 \tag{4}$$

and that the variance of the elastic net estimator is therefore

$$\operatorname{Var}\hat{\beta}_j = \frac{1}{d_j^2}\operatorname{Var}\mathrm{S}_\lambda(Z). \tag{5}$$

3.1 Normally Distributed Noise

Next, we add the additional assumption that $\boldsymbol{\varepsilon}$ is normally distributed. Then

$$Z \sim \text{Normal}\left(\tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{x}_j \beta_j, \sigma_\varepsilon^2 \|\tilde{\boldsymbol{x}}_j\|_2^2\right).$$

Let $\theta = -\mu - \lambda_1$ and $\gamma = \mu - \lambda_1$. Then the expected value of soft-thresholding of Z is

$$\begin{aligned}
\mathrm{E}\, S_{\lambda_1}(Z) &= \int_{-\infty}^{\frac{\theta}{\sigma}} (\sigma u - \theta)\, \phi(u)\, \mathrm{d}u + \int_{-\frac{\gamma}{\sigma}}^{\infty} (\sigma u + \gamma)\, \phi(u)\, \mathrm{d}u \\
&= -\theta\, \Phi\left(\frac{\theta}{\sigma}\right) - \sigma\, \phi\left(\frac{\theta}{\sigma}\right) + \gamma\, \Phi\left(\frac{\gamma}{\sigma}\right) + \sigma\, \phi\left(\frac{\gamma}{\sigma}\right) \qquad (6)
\end{aligned}$$

where $\phi(u)$ and $\Phi(u)$ are the probability density and cumulative distribution functions of the standard normal distribution, respectively.

Next, we consider what the variance of the elastic net estimator looks like. Starting with the first term on the left-hand side of Equation (4), we have

$$\begin{aligned}
\int_{-\infty}^{-\lambda_1} (z + \lambda_1)^2 f_Z(z)\, \mathrm{d}z &= \sigma^2 \int_{-\infty}^{\frac{\theta}{\sigma}} y^2\, \phi(y)\, \mathrm{d}y + 2\theta\sigma \int_{-\infty}^{\frac{\theta}{\sigma}} y\, \phi(y)\, \mathrm{d}y + \theta^2 \int_{-\infty}^{\frac{\theta}{\sigma}} \phi(y)\, \mathrm{d}y \\
&= \frac{\sigma^2}{2}\left(\operatorname{erf}\left(\frac{\theta}{\sigma\sqrt{2}}\right) - \frac{\theta}{\sigma}\sqrt{\frac{2}{\pi}} \exp\left(-\frac{\theta^2}{2\sigma^2}\right) + 1\right) + 2\theta\sigma\, \phi\left(\frac{\theta}{\sigma}\right) + \theta^2\, \Phi\left(\frac{\theta}{\sigma}\right). \qquad (7)
\end{aligned}$$

Similar computations for the second term on the left-hand side of Equation (4) yield

$$\int_{\lambda_1}^{\infty} (z - \lambda_1)^2 f_Z(z)\, \mathrm{d}z = \frac{\sigma^2}{2}\left(\operatorname{erf}\left(\frac{\gamma}{\sigma\sqrt{2}}\right) - \frac{\gamma}{\sigma}\sqrt{\frac{2}{\pi}} \exp\left(-\frac{\gamma^2}{2\sigma^2}\right) + 1\right) + 2\gamma\sigma\, \phi\left(\frac{\gamma}{\sigma}\right) + \gamma^2\, \Phi\left(\frac{\gamma}{\sigma}\right). \qquad (8)$$

Plugging Equations (6) to (8) into Equation (5) yields the variance of the estimator. Consequently, we can also compute the mean-squared error via the bias-variance decomposition

$$\mathrm{MSE}(\hat{\beta}_j, \beta_j^*) = \operatorname{Var} \hat{\beta}_j + \left(\mathrm{E}\, \hat{\beta}_j - \beta_j^*\right)^2.$$

3.2 Binary Features

The main focus in this paper is the case when $\boldsymbol{x_j}$ is a binary feature with class balance $q = \bar{\boldsymbol{x}}_j$, that is, $x_{ij} \in \{0, 1\}$ for all i and $\sum_{i=1}^n x_{ij} = nq$. In this case, we observe that

$$\begin{aligned}
\tilde{\boldsymbol{x}}_j^\intercal \tilde{\boldsymbol{x}}_j &= \frac{1}{s_j^2}(\boldsymbol{x}_j - \mathbf{1}c_j)^\intercal (\boldsymbol{x}_j - \mathbf{1}c_j) = \frac{1}{s_j^2}(nq - 2nq^2 + nq^2) = \frac{nq(1-q)}{s_j^2}, \\
\tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{x}_j &= \frac{1}{s_j}(\boldsymbol{x}_j^\intercal \boldsymbol{x}_j - \boldsymbol{x}_j^\intercal \mathbf{1}c_j) = \frac{nq(1-q)}{s_j}.
\end{aligned}$$

And consequently

$$\mu = \frac{\beta_j^* nq(1-q)}{s_j}, \qquad \sigma^2 = \frac{\sigma_\varepsilon^2 nq(1-q)}{s_j^2}, \qquad d_j = \frac{nq(1-q)}{s_j} + \lambda_2 s_j.$$

We will allow ourselves to abuse notation and overload the definitions of μ, σ^2, and d_j as functions of q. Then, an expression for the expected value of the elastic net estimate with respect to q can be obtained by plugging in μ and σ into Equation (6).

The presence of the factor $q - q^2$ in μ, σ^2, and d_j means that there is a relationship between class balance and the elastic net estimator and that this relationship is mediated by the scaling factor s_j. To achieve some

initial intuition for this relationship, we begin by considering the noiseless case ($\sigma_\varepsilon = 0$) in which, inserting μ, σ, and d_j into Equation (3) yields

$$\hat{\beta}_j = \frac{\mathrm{S}_{\lambda_1}(\tilde{\boldsymbol{x}}_j^\intercal \boldsymbol{y})}{s_j\left(\tilde{\boldsymbol{x}}_j^\intercal \tilde{\boldsymbol{x}}_j + \lambda_2\right)} = \frac{\mathrm{S}_{\lambda_1}\left(\frac{\beta_j^* n(q-q^2)}{s_j}\right)}{s_j\left(\frac{n(q-q^2)}{s_j^2} + \lambda_2\right)}. \tag{9}$$

This expression shows that the class balance, q, directly affects the estimator. For values of q close to 0 or 1, the input into the soft-thresholding part of the estimator will diminish and consequently force the estimate to zero, that is, unless we use the scaling factor $s_j = (q - q^2)$, in which case the soft-thresholding part will be unaffected by class imbalance. This choice will not, however, mitigate the impact of class imbalance on the ridge part of the estimator, for which we would instead need $s_j = \sqrt{q - q^2}$. For any other choices of δ, such as $\delta = 0$, q will affect the estimator through both the ridge and lasso parts.

Based on these facts, we will consider the scaling parameterization $s_j = (q - q^2)^\delta$, $\delta \geq 0$. This includes the cases that we are primarily interested in, that is, $\delta = 0$ (no scaling), $\delta = 1/2$ (standard-deviation scaling), and $\delta = 1$ (variance scaling). Note that the last of these types, variance scaling, is not a standard type of normalization; yet, as we have already seen, it has some interesting properties in the context of binary features.

Another interesting fact about Equation (9), which holds also in the noisy situation, is that even when the binary feature is balanced ($q = 1/2$), normalization will still have an effect on the estimator. Using $\delta = 0$, for instance, leads the true coefficient β_j^* in the input to S_λ to be scaled by $n(q - q^2) = n/4$. For $\delta = 1$, there would be, in contrast, be no scaling in the class-balanced case. And for $\delta = 1/2$, the scaling factor is $n/2$. Generalizing this, we see that to achieve equivalent scaling in the class-balanced case for all types of normalization, under our parameterization, we would need to use

$$s_j = 4^{\delta-1}(q - q^2)^\delta.$$

This only resolves the issue for the lasso. To achieve a similar effect for ridge regression, we would need another (but similar) modification. Since all features are binary under our current assumptions, however, we will for now just assume that we scale λ_1 and λ_2 to account for this effect[4], which is equivalent to modifying s_j. We will, however, return to this issue later in Section 3.3, since it has important implications in the situation when $\boldsymbol{X}$ contains both binary and continuous features.

We now leave the noise-less scenario and proceed to consider how class balance affects the probability of selection, bias, and variance of the elastic net estimator, starting with the first of these. A consequence of the normal error distribution and consequent normal distribution of Z is that the probability of selection in the elastic net problem is given analytically by

$$\begin{aligned}
\Pr\left(\hat{\beta}_j \neq 0\right) &= \Pr\left(\mathrm{S}_{\lambda_1}(Z) \neq 0\right) \\
&= \Pr\left(Z > \lambda_1\right) + \Pr\left(Z < -\lambda_1\right) \\
&= \Phi\left(\frac{\mu - \lambda_1}{\sigma}\right) + \Phi\left(\frac{-\mu - \lambda_1}{\sigma}\right). \\
&= \Phi\left(\frac{\beta_j^* n(q-q^2)^{1/2} - \lambda_1(q-q^2)^{\delta-1/2}}{\sigma_\varepsilon\sqrt{n}}\right) + \Phi\left(\frac{-\beta_j^* n(q-q^2)^{1/2} - \lambda_1(q-q^2)^{\delta-1/2}}{\sigma_\varepsilon\sqrt{n}}\right).
\end{aligned}$$

Letting $\theta = -\mu - \lambda_1$ and $\gamma = \mu - \lambda_1$, we can express the probability of selection in the limit as $q \to 1^+$ as

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

[4]We do this in all of the following examples.

In Figure 2, we plot this probability for various settings of δ for a single feature. Our intuition from the noise-less case holds: δ mitigates the influence of class imbalance on selection probability. The lower the value of δ, the larger the effect of class imbalance becomes. Note that the probability of selection initially decreases also in the case when $\delta \geq 1$. This is a consequence of increased variance of Z dues to the scaling factor that scales the measurement noise σ_ε^2 upwards. Then, as q approaches 1, the probability picks up again and eventually approaches 1 for these $\delta \in \{1, 1.5\}$. The reason for this is that the variance of Z eventually explodes (again due to the scaling), which ultimately removes the soft-thresholding effect altogether. Note that the selection probability is unaffected by λ_2 (the ridge penalty), so these results hold for any value of it.

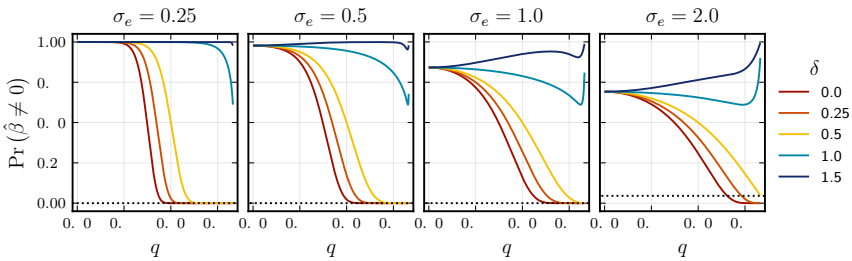


Figure 2: Probability of selection in the lasso given a measurement noise level σ_ε, a regularization parameter λ_1, and a class balance q. The scaling factor is parameterized by $s_j = (q - q^2)^\delta$, $\delta \geq 0$. The dotted line represents the asymptotic limit for the standardization case, $\delta = 1/2$.

Now we turn to the impact of class imbalance on bias and variance of the elastic net estimator. We begin, in Theorem 3.1, by considering the expected value of the elastic net estimator in the limit as $q \to 1^+$.

Theorem 3.1. *If $\boldsymbol{x}_j$ is a binary feature with class balance $q \in (0, 1)$, $\lambda_1 \in (0, \infty)$, $\lambda_2 \in [0, \infty)$, $\sigma_\varepsilon > 0$, and $s_j = (q - q^2)^\delta$, $\delta \geq 0$ then*

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

Theorem 3.1 shows that the bias of the elastic net estimator when $0 \leq \delta < 1/2$ approaches $-\beta_j^*$ as $q \to 1^+$. Interestingly, when $\delta = 1/2$ (standardization), the estimate does not in fact tend to zero. Instead, it approaches the true coefficient scaled by the probability that a standard normal variable is smaller than $\beta_j^* \sqrt{n} \sigma_\varepsilon^{-1}$. For $\delta > 1/2$, the estimate is unbiased asymptotically, which is related to the scaled variance of the error term. Note that this unbiasedness is paralleled by a surge in variance and therefore also a rise in mean-squared error, and only serves to demonstrate that the cost of the decoupling of q is unbearable in the large noise–large imbalance scenario. In Theorem 3.2, we continue by studying the variance in the limit as $q \to 1^+$.

Theorem 3.2. *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*

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

Corollary 3.2.1 (Variance in Ridge Regression)**.** *Assume the conditions of Theorem 3.2 hold, except that $\lambda_1 = 0$. Then*

$$\lim_{q \to 1^+} \operatorname{Var} \hat{\beta}_j = \begin{cases} 0 & \text{if } 0 \leq \delta < 1/4, \\ \frac{\sigma_\varepsilon^2 n}{\lambda_2^2} & \text{if } \delta = 1/4, \\ \infty & \text{if } \delta > 1/4. \end{cases}$$

Theorem 3.2 formally proves the asymptotic variance effects of our scaling parameter s_j which we have already discussed in the context of selection probability and bias. Taken together with the results from Theorem 3.1, this suggests that the choice of scaling parameter, at least in the case of our specific parameterization, introduces a bias–variance tradeoff with respect to δ: to reduce bias (with respect to q), we need to pay the cost of increased variance.

In Figure 3, we now visualize bias, variance, and mean-squared error for ranges of class balance and various noise-level settings for a lasso problem. The figure demonstrates the bias–variance tradeoff that our asymptotic results suggested and indicates that the optimal choice of δ is related to the noise level in the data. Since this level is unknown for most data sets, it suggests there might be value in selecting δ through hyper-optimization as is typically done for the other hyper-parameters in the elastic net (λ_1, λ_2)

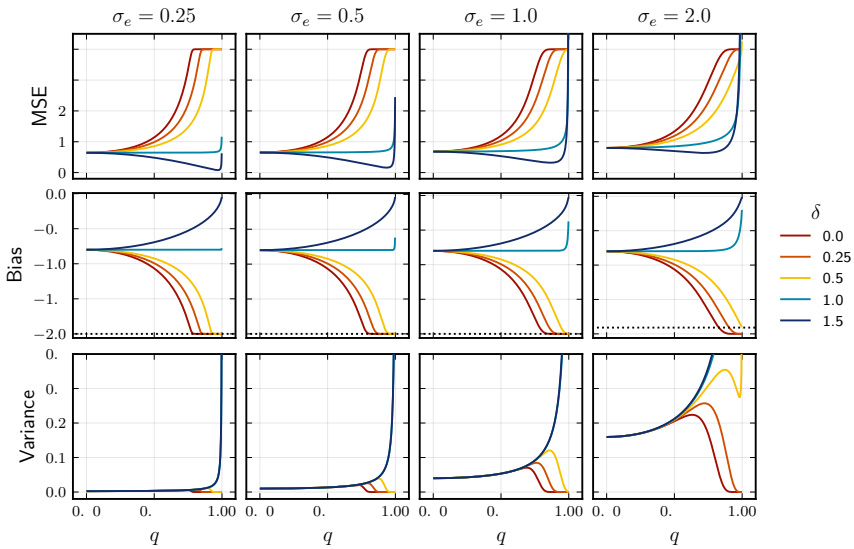


Figure 3: Bias, variance, and mean-squared error for a one-dimensional lasso problem. We show these measures for various noise levels (σ_ε), class balances (q), and scaling factors (δ). The dotted lines represent the asymptotic bias of the lasso estimator in the case of $\delta = 1/2$.

So far, we have only considered a single binary feature. But under the assumption of orthogonal features, it is straightforward to introduce multiple binary features. In a first example, we study how the power of correctly detecting $k = 10$ signals under q linearly spaced in $[0.5, 0.99]$ (Figure 5a). We set $\beta_j^* = 2$ for each of the signals, use $n = 100\,000$, and let $\sigma_\varepsilon = 1$. As we can see, the power is directly related to q and for unbalanced features stronger the higher the choice of δ is.

We also consider a version of the same setup, but with p linearly spaced in $[20, 100]$ to compute the normalized mean-squared error (NMSE) and false discovery rate (FDR) (Figure 5b). As before, we let $k = 10$ and consider three different levels of class imbalance. The remaining $p - k$ features have class balances spaced evenly on a logarithmic scale from 0.5 to 0.99. Unsurprisingly, the increase in power gained from selecting $\delta = 1$ imposes increased false discovery rates. The mean-squared error depends on the class balance. For class-balanced signals, $\delta \in \{0, 1/2\}$ proves to b the best choice, while for unbalanced signals, $\delta = 1$ is the best choice. In the case when $q = 0.99$, the model is altogether unable to detect any of the true signals, instead picking up on the noisy, but better-balanced, features.

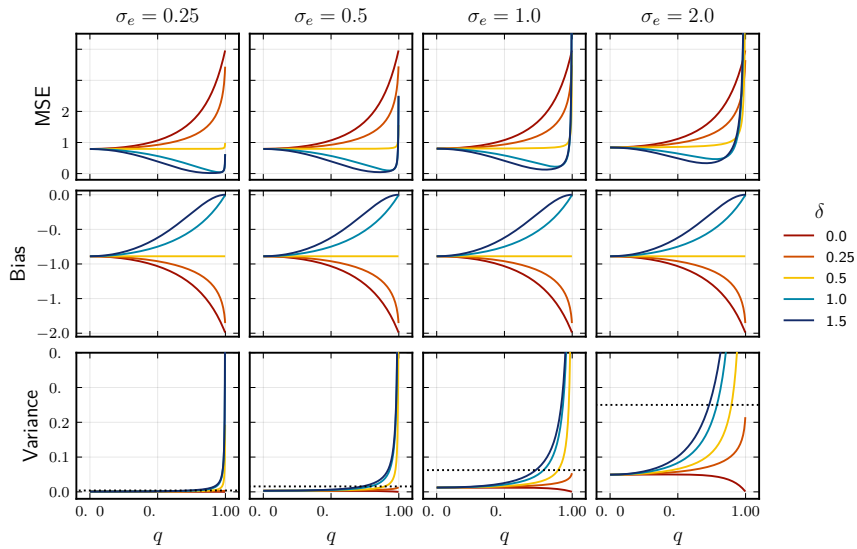
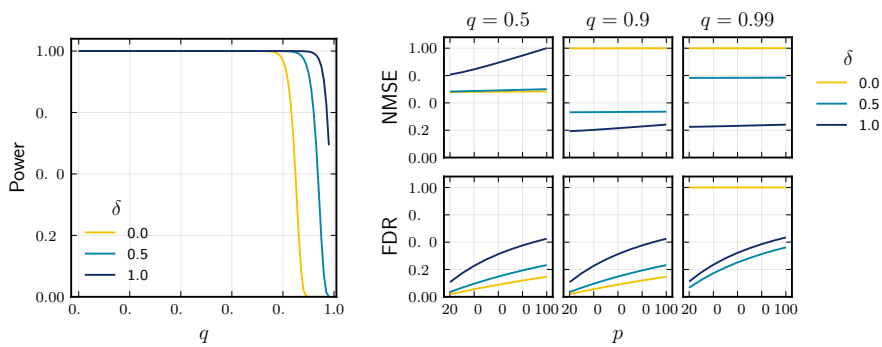


Figure 4: Bias, variance, and mean-squared error for one-dimensional ridge regression. We show these measures for various noise levels (σ_ε), class balances (q), and scaling factors (δ). The dotted lines represent the asymptotic bias of the lasso estimator in the case of $\delta = 1/2$.



(a) The power (probability of detecting all true signals) of the lasso. In our orthogonal setting, power is constant over p, which is why we have omitted the parameter in the plot.

(b) NMSE and FDR: the rate of coefficients incorrectly set to non-zero (false discoveries) to the total number of estimated coefficients that are nonzero (discoveries).

Figure 5: Normalized mean-squared error (NMSE), false discovery rate (FDR), and power for a lasso problem with $k = 10$ true signals (nonzero β_j^*), varying p, and $q \in [0.5, 0.99]$. The noise level is set at $\sigma_\varepsilon = 1$ and $\lambda_1 = 0.02$.

In Section 4, we will continue to study binary features in simulated experiments. For now, however, we will turn to the case of mixed data.

3.3 Mixed Data

In this section, we consider the case where the features are made up of a mix of continuous and and binary features. Throughout the section, we will continue to assume that $\boldsymbol{X}$ is fixed and that the features are orthogonal to one another. As in our theoreticalt results, we will also restrict our focus to the case where the continuous features are normally distributed.

A fundamental problem in the context of mixed data is how to put the binary and normal features on the same scale, which we need to do in order for regularization to be, roughly speaking, "fair", given that the solution is sensitive to the scale of the features. In essence, we need to say something about how an effect associated with a one-unit change in the binary feature (a flip) relates to a one-unit change in the continuous feature. Since we assume our continuous feature to be normal, however, we will instead reason about change in terms of standard deviations of the normal feature.

To setup this situation more formally, we will say that the effect of a binary feature $\boldsymbol{x}_1$ and a normal feature $\boldsymbol{x}_2$ are *comparable* if

$$\beta_1^* = \kappa\sigma_2\beta_2^*,$$

where σ_2 is the standard deviation of $\boldsymbol{x}_2$ and $\kappa > 0$ is a scaling factor that represents the number of standard deviations (of the continuous feature) we consider achieves comparability between the features' effects. (Note that $\sigma_2\beta_2^*$ is just the standardized coefficient for the normal feature.) We illustrate this notion of comparability by a couple of examples.

Example 3.1. Assume $\kappa = 2$. If $\boldsymbol{x}_2$ is sampled from Normal$(\mu, 1/2^2)$, then the effects of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are comparable if $\beta_1^* = \beta_2^*$.

Example 3.2. Assume $\kappa = 1$. If $\boldsymbol{x}_2$ is sampled from Normal$(\mu, 2^2)$, then the effects of $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are comparable if $\beta_1^* = 2\beta_2^*$.

Note that this definition refers to the data-generating mechanism, and not the regularized estimates. What we ultimately want for comparability, however, is for the following relationship to hold:

$$\hat{\beta}_1 = \kappa\sigma_2\hat{\beta}_2.$$

Put plainly, we want the effects of regularization to be distributed evenly across the estimates. The crux of the problem is how to choose the scaling factor s_j for the binary features in order to achieve this effect for a given κ. Let us assume that we have two features, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, where $\boldsymbol{x}_1$ is binary and $\boldsymbol{x}_2$ is normally distributed and that their effects are comparable in the sense given above. Then it should hold that

$$\begin{aligned}
\hat{\beta}_1 = \kappa\sigma_2\hat{\beta}_2 \qquad &\implies \\
\frac{\mathrm{S}_{\lambda_1}(\tilde{\boldsymbol{x}}_1^\intercal \boldsymbol{y})}{s_1\left(\tilde{\boldsymbol{x}}_1^\intercal\tilde{\boldsymbol{x}}_1 + \lambda_2\right)} = \frac{\kappa\sigma_2\,\mathrm{S}_{\lambda_1}(\tilde{\boldsymbol{x}}_2^\intercal \boldsymbol{y})}{s_2\left(\tilde{\boldsymbol{x}}_2^\intercal\tilde{\boldsymbol{x}}_2 + \lambda_2\right)} &\implies \\
\frac{\mathrm{S}_{\lambda_1}\left(\frac{n\beta_1^*(q-q^2)}{s_1}\right)}{s_1\left(\frac{n(q-q^2)}{s_1^2} + \lambda_2\right)} = \frac{\kappa\,\mathrm{S}_{\lambda_1}\left(\frac{n\beta_1^*}{\kappa}\right)}{n + \lambda_2}. &
\end{aligned} \tag{10}$$

For the lasso ($\lambda_2 = 0$) and ridge regression ($\lambda_1 = 0$), we observe that $s_1 = \kappa(q - q^2)$ and $s_1 = (q - q^2)^{1/2}$, respectively, are the values for which Equation (10) hold. In other words, we can achieve comparability in the lasso by scaling each binary feature with its variance times κ, the number of standard deviations we consider achieves comparability between the features' effects. And for ridge regression, we can achieve comparability by scaling with standard deviation, irrespective of κ.

For any other choices of s_1, equality can only hold for a specific level of class balance.. If we let this level be called q_0, then, to achieve equality for $\lambda_2 = 0$, we need $s_1 = \kappa(q_0 - q_0^2)^{1-\delta}(q - q^2)^\delta$. Similarly, for $\lambda_1 = 0$, we

need $s_1 = (q_0 - q_0^2)^{1-2\delta}(q - q^2)^\delta$. In the sequel, we will assume that $q_0 = 1/2$, to have effects be equivalent for the class-balanced case.

Note that this also means that there is an implicit relationship between the strength of penalization for binary and normal features, which depends on the level of class balance and normalization type. This means, for instance, that even in the class-balanced case ($q = 1/2$), we have to account for the type of normalization if we want binary and normal features to be treated equally

For the rest of this paper, we will use $\kappa = 2$. That is, we will say that the effects are comparable if the effect of a flip in the binary feature equals the effect of a two-standard deviation change in the normal feature. We base this argument on the discussion by Gelman (2008), who argues that the classical approach of comparing standardized coefficients[5] awards effects of continuous features undue strength for most real data, since a change from, for instance, the lower to the upper 16% of the distribution will equal approximately twice the effect of a change in the binary feature. Using two standard deviations as a comparability factor would, in contrast, equivocate this change with the flip of the binary feature, which we believe is a better default. We want to stress that the choice of κ should, if possible, in general be made on a case-by-case (feature-by-feature) basis, using all available knowledge about the data at hand. But, irrespective of this, we also want to emphasize that the choice should be made. If you do not make it explicitly, then it will be implicitly dictated through the combination of normalization and penalization types you use.

Finally, note that the reasoning of comparability above rests on the assumption of no noise. And we are, in fact, in general instead more interested in the expected value of the estimators, which depend on the noise level. In the case of large class-imbalances and large noise, for instance, our previous results (see Figure 3 for instance), suggest that the estimators for normally distributed and binary features will not be comparable in this case.

4 Experiments

In the following sections, we present the results of our experiments. We begin by examining the variability and bias in the estimates of the regression coefficients. We then move on to predictive performance and hyperparameter selection. We also consider the effect of class imbalance on the estimates of the regression coefficients. Finally, we look at the effect of interactions between features on the estimates of the regression coefficients.

In all cases where we use simulated data, we generate our response vector according to

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^* + \boldsymbol{\varepsilon},$$

with $\boldsymbol{\varepsilon} \sim \text{Normal}(\boldsymbol{0}, \sigma_\varepsilon^2 \boldsymbol{I})$, where $\boldsymbol{X}$ is the design matrix, $\boldsymbol{\beta}^*$ is the vector of true regression coefficients, and σ_ε^2 is the noise level.

We consider two types of features: binary and quasi-normal features. To generate binary vectors, we sample $\lceil qn \rceil$ indexes uniformly at random without replacement from $\{1, 2, \dots, n\}$ and set the corresponding elements to one and the remaining ones to zero. To generate quasi-normal features, we generate a linear sequence $\boldsymbol{w}$ with n values from 10^{-4} to $1 - 10^{-4}$, and set

$$x_{ij} = \Phi^{-1}(w_i)$$

and then shuffle the elements of $\boldsymbol{x}_j$ uniformly at random.

In each case, we fit either the lasso (the elastic net with $\lambda_1 = \alpha\lambda$) or ridge (the elastic net with $\lambda_2 = (1-\alpha)\lambda$). To normalize the data, we use standardization for all quasi-normally distributed features and otherwise

$$s_j = (q - q^2)^\delta,$$

which is equivalent to the (uncorrected) sample variance raised to the power of δ.

[5] Coefficients multiplied by the standard deviation of the respective feature.

Throughout the experiments, we have used the Lasso.jl package (Kornblith, 2024) to fit lasso or ridge regression, which implements the coordinate descent algorithm by citetfriedman2010. All experiments were coded using the Julia programming language (Bezanson et al., 2017) and the code is availabe at `https://github.com/jolars/normreg`.

4.1 Variability and Bias in Estimates

In our first experiment, we consider fitting the lasso to a simulated data set with $n = 500$ observations and $p = 1000$ features, out of which the first 20 features correspond to signals, with β_j^* decreasing linearly from 1 to 0.1. We introduce dependence between the features by copying the first $\lceil \rho n/2 \rceil$ values from the first feature to each of the following features. In addition, we set the class balance of the first 20 features so that it decreases linearly on a log-scale from 0.5 to 0.99. We estimate the regression coefficients using the lasso and ridge regression and compare the estimates to the true coefficients. We run the experiment for 50 iterations in each case and aggregate the results by reporting means and standard deviations.

The results (Figure 6) show that there is a considerable effect of class balance, particularly in the case of no scaling ($\delta = 0$), which corroborates our theoretical results from Section 3.2. At $q = 0.99$, for instance, the estimate $\hat{\beta}_{20}$ is consistently zero when $\delta = 0$. There is an effect also in the case of standardization ($\delta = 1/2$), but it is less pronounced. For $\delta = 1$ (variance scaling), we see that the effect of class balance on the estimates is, if anything, the reverse when the class imbalance is severe. What is also clear is that the variance of the estimates increase with class imbalance and that this effected is augmented with larger values of δ. The level of correlation between the features introduces additional variance in the estimates but also seems to increase the effect of class imbalance in the cases when $\delta = 0$ or $1/2$.

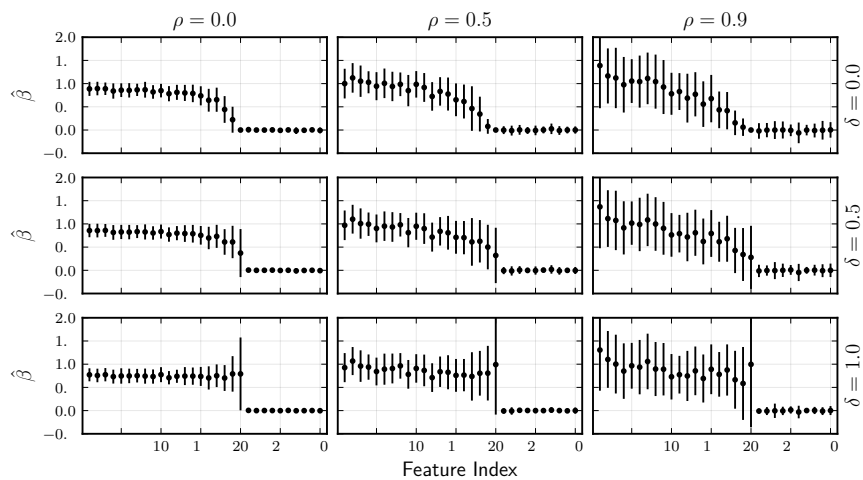


Figure 6: Estimates of the regression coefficients from the lasso, $\hat{\boldsymbol{\beta}}$, for the first 30 coefficients in the experiment. All of the features are binary and the first 20 features correspond to true signals with $\beta_j^* = 2$ and geometrically decreasing class balance from 0.5 to 0.99. The remaining features have a class balance $q_j \in [0.5, 0.99]$, distributed linearly among the features. The plot shows means and standard deviations averaged over 50 iterations.

4.2 Predictive Performance

In this experiment, we consider predictive performance in terms of mean-squared error of the lasso given different levels of class balance ($qin\{0.5, 0.9, 0.99\}$), signal-to-noise ratio, and normalization (δ). As in the previous section, all of the features are binary, but here we have used $n = 300$, $p = 1000$. The $k = 10$ first features correspond to true signals with $\beta_j^* = 1$ and all have class balance (q). To set signal-to-noise ratio levels, we rely on the same choice as in Hastie et al. (2020) and use a log-spaced sequence of values from 0.05 to 6.

To estimate prediction performance, we use a standard hold-out validation method equal splits for the training, validation, and test sets. We fit a full lasso path, parameterized by a log-spaced grid of 100 values[6], from $\lambda_{\max}$ (the value of λ at which the first feature enters the model) to $10^{-2}\lambda_{\max}$ on the training set and pick a λ based on validation set error. Then we compute the hold-out test set error and aggregate the results across 100 iterations.

The results (Figure 7) show that the optimal normalization type in terms of prediction power depends on the class balance of the true signals. If the imbalance is severe, then we gain from using $\delta = 1/2$ or 1, which gives a chance of recovering the true signals. If everything is balanced, however, then we do better by not scaling at all. In general, $\delta = 1/2$ works well for these specific combinations of settings.

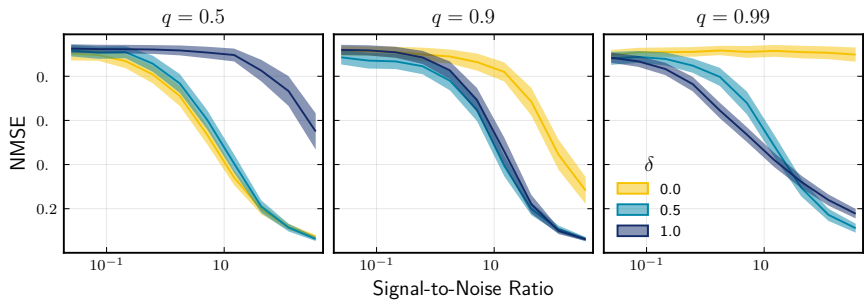


Figure 7: Normalized mean-squared prediction error in a lasso model for different types of normalization (δ), types of class imbalances (q), and signal-to-noise ratios (0.05 to 6) in a data set with $n = 300$ observations and $p = 1000$ features. The error is aggregated test-set error from hold-out validation with 100 observations in each of the training, validation, and test sets. The plot shows means and Student's t-based 95% confidence intervals.

4.3 Normalization as a Hyperparamter

Our previous results (particularly those from Section 4.2) suggest that the choice of normalization matters for predictive performance. These results have relied on knowledge of the measurement error (signal-to-noise ratio), which we do not have reliable estimates of in practice (at least not in the high-dimensional context). An alterative that, however, comes naturally as a consequence of our particular parameterization using δ, is to treat the choice of normalization as a hyperparameter and optimize over it. This is the approach we take in this experiment.

We set up a grid of λ values as in Section 4.2 and, in addition, also create a linearly spaced grid of δ values in $[0, 1]$. We split the data into a 50/50 training/validation set split and for each point in this two-dimensional grid fit the lasso or ridge to the training set and compute a hold-out validation set error. We do this for three data sets: `a1a` (Becker & Kohavi, 1996), `rhee2006` (Rhee et al., 2006), and `w1a` (Platt, 1998).

[6]This is a standard choice of grid, used for instance by Friedman et al. (2010)

Table 2: Details of the real datasets used in the experiments

Dataset	n	p	Response
`w1a`	2477	300	Binary
`a1a`	1605	123	Binary
`rhee2006`	842	361	Continuous

We show estimated level-curves of validation set error, in terms of normalized mean-squared error (NMSE), in Figure 8. For `a1a`, the lasso is generally quite insensitive to the type of normalization, even if the optimal value is around 0.2. For ridge regression, lower values of δ clearly work better. With the `w1a` data set, however, the relationship is flipped in the case of ridge regression and the optimal value is approximately 0.8. In the case of the lasso (for `w1a`), a value around 0.5 is optimal and low values (little scaling) yield worse prediction errors. Finally, for `rhee2006`, the lasso is again insensitive to normalization type. This is not the case for ridge, however, where a value around 0.2 is optimal and high values of δ yield worse prediction errors.

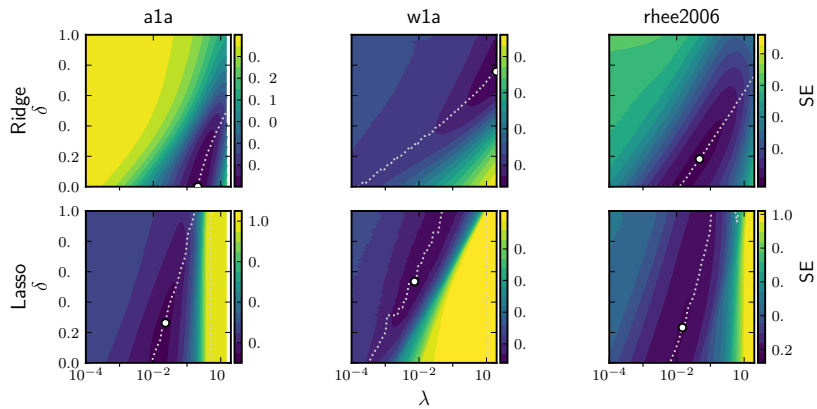


Figure 8: Contour plots of normalized mean-squared error (NMSE) for the hold-out validation set across a grid of δ and λ values for ridge regression and the lasso. The dotted path shows the smallest NMSE as a function of λ. The dot marks the combination with the smallest error.

We would like to point out that there is a dependency between λ and δ here that make it difficult to interpret the relationship between them and the error. This comes fro mthe fact that scaling with a smaller value (as in $\delta = 1$) increases the sizes of the vectors, which means that the level of penalization is relaxed, relative speaking.

In Figure 9, we have, in addition to NMSE on the validation set, also plotted the size of the support of the lasso (cardinality of the set of features that have corresponding nonzero coefficients). Here, however, we only show results for $\delta \in \{0, 1/2, 1\}$. It is clear that $\delta = 1/2$ works quite well for all of these three data sets, being able to attain a value close to the mininum for each of the three data sets. This is not the case for $\delta \in \{0, 1\}$, for which the best possible prediction error is considerably worse. This is particularly the case with $\delta = 0$ and the `w1a` data set. The dependency between λ and δ is also visible here by looking at the support size.

4.4 Mixed Data

In Section 3.3, we discovered that extra care needs to be taken when normalizing mixed data. In this experiment, we construct a quasi-normal feature with mean zero and standard deviation 1/2 and a binary

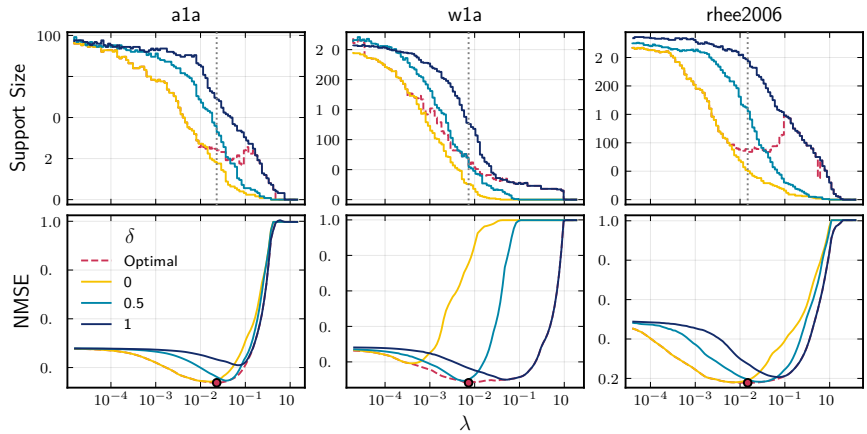


Figure 9: Support size and normalized mean-squared error (NMSE) for the validation set for the lasso fit to datasets `a1a`, `w1a`, and `rhee2006` across combinations of δ and λ. The optimal δ is marked with dashed red lines and the best combination of δ (among 0, 1/2, and 1) and λ is shown with a dot.

feature with varying class balance q. We set the signal-to-noise ratio to 0.5 and generate our response vector $\boldsymbol{y}$ as before, with $n = 1000$. These features are constructed so that their effects are comparable under the notion of comparability that we introduce in Section 3.3, using $\kappa = 2$.

The results (Figure 10) reflect our theoretical results from Section 3. In the case of the lasso, we need $\delta = 1$ to avoid the effect of class imbalance, whereas for ridge we instead need $\delta = 1/2$ (standardization). As our theory suggests, this extra scaling mitigates this class-balance dependency at the cost of added variance.

Note that we do not see the bias reduction that we observed in our theoretical results for high q values and $\delta \geq 1/2$ in Figure 10. This is related to the error term (signal-to-noise ratio) and level of q. Typically, we would need stronger class imbalance and larger error for the effect to show up in our experiments.

4.5 Interactions

In our final experiment, we study the effect of normalization and class balance on interactions when using the lasso. Our example consists of a two-feature problem with an added interaction term given by $x_{i3} = x_{i1}x_{i2}$. The first feature is binary with class balance $q = 0.9$ and the second quasi-normal with standard deviation 0.5. We set $n = 1000$. Note that we perform normalization *after* the interaction term is added.

The results show, as before, that class balance (which, recall, is set to 0.9 here) has a dramatic effect on estimates of the binary feature when $\delta \in \{0, 1/2\}$. Somewhat surprisingly, however, the interaction term does not seem to be affected by the normalization type for any of the cases in which it is present.

Note that the interaction in this experiment naturally introduces correlation between the features and that this has an effect on the lasso estimates since we, for instance, can penalize the main effect whilst still retaining information about it in the interaction term.

5 Discussion

In this paper, we have studied the effects of normalization in ridge regression and the lasso for features that are binary—an issue that has so far been treated with disregarded in the literature. We have discovered the

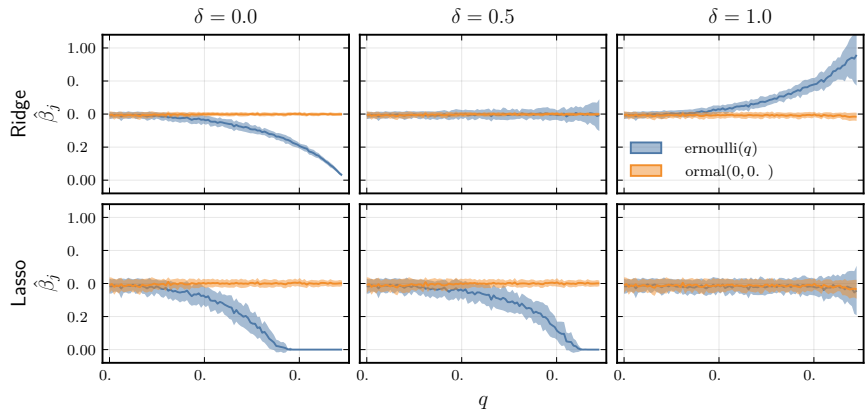


Figure 10: Lasso and ridge estimates for a two-dimensional problem where one feature is a binary feature with class balance q, Bernoulli(q), and the other is a quasi-normal feature with standard deviation $1/2$, Normal$(0, 0.5)$. Here, we have $n = 1000$ observations. The signal-to-noise ratio is 0.5 In every case, we standardize the normal feature. The binary feature, meanwhile, is centered by its mean and scaled by $(q - q^2)^\delta$. The experiment is run for 50 iterations and we aggregate and report means and standard deviations of the estimates.

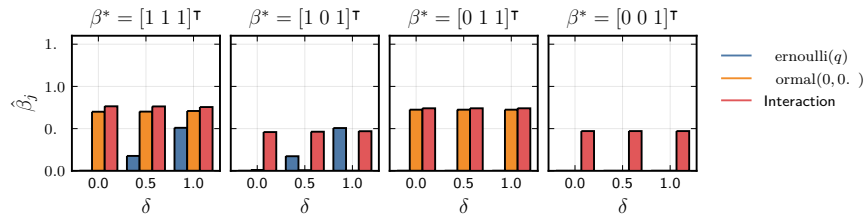


Figure 11: Lasso estimates for a three-feature problem where the third feature is an interaction term between the first two features. The first feature is binary with class balance $q = 0.9$ and the second is quasi-normal with standard deviation 0.5. The signal-to-noise ratio is 0.5. The experiment is run for 50 iterations and we aggregate and report means across all iterations.

class imbalance of binary features—the proportion of ones and zeros in the features—have a pronounced effect on both lasso and ridge estimates, and that this effect depends on the type of normalization used. For the lasso, for instance, our results show that features with large class imbalances will be regularized heavily, and provided that λ is large enough might stand little chance of being selected, even if the true effect of the feature on the response is large.

We have, however, found that scaling binary features with standard deviation in the case of ridge regression and variance in the case of the lasso mitigates this effect, but that doing so comes at the price of increased variance. This effectively means that the choice of normalization constitutes a bias–variability trade-off with respect to imbalanced binary features.

To study these effects theoretically and in practice, we have introduced the scaling parameterization

$$s_j = (q - q^2)^\delta,$$

which, for instance, includes the cases $\delta = 0$ (no scaling), $\delta = 1/2$ (standard deviation scaling), and $\delta = 1$ (variance scaling). These, in turn, correspond to standard choices of normalization types for this kind of data. The common variants max–abs and min–max normalization, for instance, in practice correspond to $\delta = 0$ in the case of binary data, whilst standardization corresponds to $\delta = 1/2$. As far as we know, scaling with $\delta = 1$ have previously not been considered in the literature nor to any extent that we are aware of in practice.

Our results demonstrate, however, that the choice of δ affects the lasso and ridge estimates heavily in many cases. This is particularly true with respect to selective inference, in which case $\delta = 0$ scaling will reduce the chances of finding the true model via the lasso in class-imbalanced settings (Section 4.1). But it will also bias the regression coefficients in both the lasso and ridge, which may also lead to suboptimal predictive performance (Section 4.2).

Both our theoretical results (Section 3.2) and experiments (Section 4.1) show that the optimal choice of δ may depend on the error in the data-generating process, which is typically unknown. As an alternative, we investigated choosing δ in a data-driven manner by optimizing over δ as if it were a hyperparameter (Section 4.3).

We have also studied the case of mixed data: designs that consist of both binary and normally distributed features. In this setting, our first finding is that there is an implicit relationship between the choice of normalization and the manner in which regularization affects binary viz-a-viz normally distributed features. For instance, the choice of max–abs normaliation carries a specific assumption about how the effect of a binary feature should be compared to that of a normally distributed feature. There is still much uncertainty about how to best handle the mixed data case and no ground truth given that a binary feature can mean any number of things—few of which are directly comparable to a continuous feature.

In our experimental results, we touch briefly on the case of interactions. In this case, it seems that the interaction term between a normal feature and a binary one is more-or-less unaffected by the class balance of the latter (Section 4.5). An interesting avenue for future research could be to study this in more detail, both theoretically and empirically. One particular problem with interactions is that the interaction term depends on the location, and not just the scale, of the normal feature (in this two-feature setting), which may call for conditional normalization strategies. Much remain to be explored in this area.

Finally, note that our theoretical results are limited by several assumptions: 1) a fixed feature matrix $\boldsymbol{X}$, 2) orthogonality between the features, and 3) normal and idependent errors. Future work could relax these assumptions to study the effects of normalization in more general settings. For instance, the assumption of orthogonality could be relaxed to allow for correlated features, which is often the case in practice. This would allow for a more general understanding of the effects of normalization in regularized regression models. We have also limited ourselves to the case of the lasso and ridge regression. Investigating to which extent, if any, the effects we observe generalize to other models as well would yield valuable insights. We have also focused on the case of binary and continuous features here, but we are convinced that the case of categorical features is also of interest and might raise additional challenges with respect to normalization.

References

Barry Becker and Ronny Kohavi. Adult, 1996.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, February 2017. ISSN 0036-1445. doi: 10.1137/141000671.

Małgorzata Bogdan, Ewout van den Berg, Weijie Su, and Emmanuel J. Candès. Statistical estimation and testing via the sorted L1 norm, October 2013.

Małgorzata Bogdan, Ewout van den Berg, Chiara Sabatti, Weijie Su, and Emmanuel J. Candès. SLOPE – adaptive variable selection via convex optimization. *The annals of applied statistics*, 9(3):1103–1140, September 2015. ISSN 1932-6157. doi: 10.1214/15-AOAS842.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, September 1995. ISSN 1573-0565. doi: 10.1007/BF00994018.

David L. Donoho and Iain M. Johnstone. Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3): 425–455, August 1994. ISSN 0006-3444. doi: 10.2307/2337118.

Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456):1348–1360, December 2001. ISSN 0162-1459. doi: 10/fd7bfs.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, January 2010. doi: 10.18637/jss.v033.i01.

Andrew Gelman. Scaling regression inputs by dividing by two standard deviations. *Statistics in Medicine*, 27 (15):2865–2873, July 2008. ISSN 02776715, 10970258. doi: 10.1002/sim.3107.

T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, October 1999. ISSN 0036-8075. doi: 10.1126/science.286.5439.531.

David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, March 1978. ISSN 0095-0696. doi: 10.1016/0095-0696(78)90006-2.

Trevor Hastie, Robert Tibshirani, and Ryan Tibshirani. Best subset, forward stepwise or lasso? Analysis and recommendations based on extensive comparisons. *Statistical Science*, 35(4):579–592, November 2020. ISSN 0883-4237. doi: 10.1214/19-STS733.

Ross King. Qualitative structure activity relationships.

Simon Kornblith. Lasso.jl, March 2024.

Haikady N. Nagaraja and Herbert A. David. *Order Statistics*. Wiley Series in Probability and Statistics. John Wiley & Sons Inc, Hoboken, N.J, 3 edition, July 2003. ISBN 978-0-471-38926-2.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola (eds.), *Advances in Kernel Methods: Support Vector Learning*, pp. 185–208. MIT Press, Boston, MA, USA, 1 edition, January 1998. ISBN 978-0-262-28319-9. doi: 10.7551/mitpress/1130.003.0016.

Soo-Yon Rhee, Jonathan Taylor, Gauhar Wadhera, Asa Ben-Hur, Douglas L. Brutlag, and Robert W. Shafer. Genotypic predictors of human immunodeficiency virus type 1 drug resistance. *Proceedings of the National Academy of Sciences*, 103(46):17355–17360, November 2006. doi: 10.1073/pnas.0607274103.

Fadil Santosa and William W. Symes. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1307–1330, October 1986. ISSN 0196-5204. doi: 10.1137/0907087.

scikit-learn developers. 6.3. Preprocessing data. https://scikit-learn/stable/modules/preprocessing.html, February 2024.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B*, 58(1):267–288, 1996. ISSN 0035-9246. doi: 10.1111/j.2517-6161.1996.tb02080.x.

Cun-Hui Zhang. Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 38(2):894–942, April 2010. ISSN 0090-5364. doi: 10/bp22zz.

Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101 (476):1418–1429, December 2006. ISSN 0162-1459. doi: 10.1198/016214506000000735.

Hui Zou and Trevor Hastie. Regularization and variable selection via the Elastic Net. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 67(2):301–320, 2005. ISSN 1369-7412.

A Additional Theory

A.1 Why Maximum–Absolute and Min–Max Scaling are Unsuitable for Normally Distributed Data

In Theorem A.1, we show that the scaling factor in the max–abs method converges in distribution to a Gumbel distribution.

Theorem A.1. *Let $X_1, X_2, \ldots, X_n$ be a sample of normally distributed random variables, each with mean μ and standard deviation σ. Then*

$$\lim_{n\to\infty} \Pr\left(\max_{i\in[n]} |X_i| \le x\right) = G(x),$$

where G is the cumulative distribution function of a Gumbel distribution with parameters

$$b_n = F_Y^{-1}(1 - 1/n) \quad \textit{and} \quad a_n = \frac{1}{n f_Y(\mu_n)},$$

where f_Y and F_Y^{-1} are the probability distribution function and quantile function, respectively, of a folded normal distribution with mean μ and standard deviation σ.

The gist of Theorem A.1 is that the limiting distribution of $\max_{i\in[n]} |X_i|$ has expected value $b_n + \gamma a_n$, where γ is the Euler-Mascheroni constant. This indicates that the scaling factor strongly dependent on the sample size. In Figure 12a, we observe empirically that the limiting distribution agrees well with the empirical distribution in expected value even for small values of n.

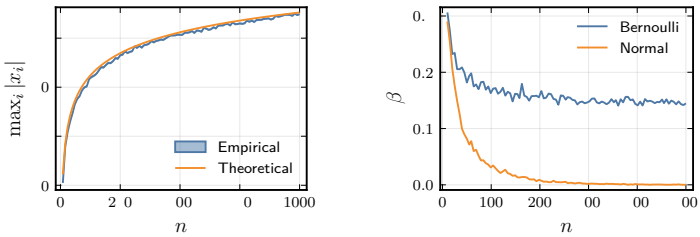
In Figure 12b we show the effect of increasing the number of observations, n, in a two-feature lasso model with max-abs normalization applied to both features. The coefficient corresponding to the Normally distributed feature shrinks as the number of observation n increases. Since the expected value of the Gumbel distribution diverges with n, this means that there's always a large enough n to force the coefficient in a lasso problem to zero with high probability.

For min–max scaling, the situation is similar and we omit the details here. The main point is that the scaling factor is strongly dependent on the sample size, which makes it unsuitable for normally distributed data in several situations, such as on-line learning (where sample size changes over time) or model validation with uneven data splits.

B Proofs

B.1 Proof of Theorem A.1

If $X_i \sim \text{Normal}(\mu, \sigma)$, then $|X_i| \sim \text{FoldedNormal}(\mu, \sigma)$. By the Fisher–Tippett–Gnedenko theorem, we know that $(\max_i |X_i| - b_n)/a_n$ converges in distribution to either the Gumbel, Fréchet, or Weibull distribution,



(a) Theoretical versus empirical distribution of the maximum absolute value of normally distributed random variables.

(b) Estimation of mixed features under maximum absolute value scaling

Figure 12: Effects of maximum absolute value scaling.

given a proper choice of $a_n > 0$ and $b_n \in \mathbb{R}$. A sufficient condition for convergence to the Gumbel distribution for a absolutely continuous cumulative distribution function (Nagaraja & David, 2003, Theorem 10.5.2) is

$$\lim_{x \to \infty} \frac{d}{dx} \left(\frac{1 - F(x)}{f(x)} \right) = 0.$$

We have

$$\begin{aligned} \frac{1 - F_Y(x)}{f_Y(x)} &= \frac{1 - \frac{1}{2} \operatorname{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right) - \frac{1}{2} \operatorname{erf}\left(\frac{x+\mu}{\sqrt{2\sigma^2}}\right)}{\frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} + \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x+\mu)^2}{2\sigma^2}}} \\ &= \frac{2 - \Phi\left(\frac{x-\mu}{\sigma}\right) - \Phi\left(\frac{x+\mu}{\sigma}\right)}{\frac{1}{\sigma}\left(\phi\left(\frac{x-\mu}{\sigma}\right) + \phi\left(\frac{x+\mu}{\sigma}\right)\right)} \\ &\to \frac{\sigma(1 - \Phi(x))}{\phi(x)} \text{ as } n \to n, \end{aligned}$$

where ϕ and Φ are the probability distribution and cumulative density functions of the standard normal distribution respectively. Next, we follow Nagaraja & David (2003, example 10.5.3) and observe that

$$\frac{d}{dx} \frac{\sigma(1 - \Phi(x))}{\phi(x)} = \frac{\sigma x (1 - \Phi(x))}{\phi(x)} - \sigma \to 0 \text{ as } x \to \infty$$

since

$$\frac{1 - \Phi(x)}{\phi(x)} \sim \frac{1}{x}.$$

In this case, we may take $b_n = F_Y^{-1}(1 - 1/n)$ and $a_n = \left(n f_Y(b_n)\right)^{-1}$.

B.2 Proof of Theorem 3.1

Since $s_j = (q - q^2)^\delta$, we have

$$\begin{aligned}
\mu &= \beta_j^* n (q-q^2)^{1-\delta} & \frac{\theta}{\sigma} &= -a\sqrt{q(1-q)} - b(q-q^2)^{\delta-1/2},\\
\sigma &= \sigma_\varepsilon \sqrt{n}(q-q^2)^{1/2-\delta}, & \frac{\gamma}{\sigma} &= a\sqrt{q(1-q)} - b(q-q^2)^{\delta-1/2},\\
d_j &= n(q-q^2)^{1-\delta} + \lambda_2 (q-q^2)^\delta, & \frac{\theta}{d_j} &= -\beta_j^* - \frac{\lambda_1 (q-q^2)^{\delta-1}}{n},\\
\theta &= -\beta_j^* n (q-q^2)^{1-\delta} - \lambda_1, & \frac{\gamma}{d_j} &= \beta_j^* - \frac{\lambda_1 (q-q^2)^{\delta-1}}{n},\\
\gamma &= \beta_j^* n (q-q^2)^{1-\delta} - \lambda_1,
\end{aligned}$$

with

$$a = \frac{\beta_j^* \sqrt{n}}{\sigma_\varepsilon} \qquad \text{and} \qquad b = \frac{\lambda_1}{\sigma_\varepsilon \sqrt{n}}.$$

We are interested in

$$\lim_{q\to 1^+} \mathrm{E}\,\hat{\beta}_j = \lim_{q\to 1^+} \frac{1}{d}\left(-\theta\,\Phi\left(\frac{\theta}{\sigma}\right) - \sigma\,\phi\left(\frac{\theta}{\sigma}\right) + \gamma\,\Phi\left(\frac{\gamma}{\sigma}\right) + \sigma\,\phi\left(\frac{\gamma}{\sigma}\right)\right). \tag{11}$$

Before we proceed, note the following limits, which we will make repeated use of throughout the proof.

$$\lim_{q\to 1^+} \frac{\theta}{\sigma} = \lim_{q\to 1^+} \frac{\gamma}{\sigma} = \begin{cases} -\infty & \text{if } 0 \le \delta < \frac{1}{2},\\ -b & \text{if } \delta = \frac{1}{2},\\ 0 & \text{if } \delta > \frac{1}{2}, \end{cases} \tag{12}$$

Starting with the terms involving Φ inside the limit in Equation (11), for now assuming that they are well-defined and that the limits of the remaining terms also exist seperately, we have

$$\begin{aligned}
&\lim_{q\to 1^+} \left(-\frac{\theta}{d}\,\Phi\left(\frac{\theta}{\sigma}\right) + \frac{\gamma}{d_j}\,\Phi\left(\frac{\gamma}{\sigma}\right)\right)\\
&= \lim_{q\to 1^+} \left(\left(\frac{\beta_j^* n}{n + \lambda_2 (q-q^2)^{2\delta-1}} + \frac{\lambda_1}{n(q-q^2)^{1-\delta} + \lambda_2 (q-q^2)^\delta}\right)\Phi\left(\frac{\theta}{\sigma}\right)\right.\\
&\qquad \left. + \left(\frac{\beta_j^* n}{n + \lambda_2 (q-q^2)^{2\delta-1}} - \frac{\lambda_1}{n(q-q^2)^{1-\delta} + \lambda_2 (q-q^2)^\delta}\right)\Phi\left(\frac{\gamma}{\sigma}\right)\right)\\
&= \lim_{q\to 1^+} \frac{\beta_j^* n}{n + \lambda_2 (q-q^2)^{2\delta-1}} \left(\Phi\left(\frac{\theta}{\sigma}\right) + \Phi\left(\frac{\gamma}{\sigma}\right)\right)\\
&\quad + \lim_{q\to 1^+} \frac{\lambda_1}{n(q-q^2)^{1-\delta} + \lambda_2 (q-q^2)^\delta} \left(\Phi\left(\frac{\theta}{\sigma}\right) - \Phi\left(\frac{\gamma}{\sigma}\right)\right).
\end{aligned} \tag{13}$$

Considering the first term in Equation (13), we see that

$$\lim_{q\to 1^+} \frac{\beta_j^* n}{n + \lambda_2 (q-q^2)^{2\delta-1}} \left(\Phi\left(\frac{\theta}{\sigma}\right) + \Phi\left(\frac{\gamma}{\sigma}\right)\right) = \begin{cases} 0 & \text{if } 0 \le \delta < 1/2,\\ \frac{2n\beta_j^*}{n+\lambda_2}\,\Phi(-b) & \text{if } \delta = 1/2,\\ \beta_j^* & \text{if } \delta > 1/2. \end{cases}$$

For the second term in Equation (13), we start by observing that if $\delta = 1$, then $q(1-q)^{\delta-1} = 1$, and if $\delta > 1$, then $\lim_{q\to 1^+} (q-q^2)^{\delta-1} = 0$. Moreover, the arguments of Φ approach 0 in the limit for $\delta \ge 1$, which means that the entire term vanishes in both cases ($\delta \ge 1$).

For $0 \leq \delta < 1$, the limit is indeterminite of the form $\infty \times 0$. We define

$$f(q) = \Phi\left(\frac{\theta}{\sigma}\right) - \Phi\left(\frac{\gamma}{\sigma}\right) \qquad \text{and} \qquad g(q) = n(q-q^2)^{1-\delta} + \lambda_2(q-q^2)^{\delta},$$

such that we can express the limit as $\lim_{q\to 1^+} f(q)/g(q)$. The corresponding derivatives are

$$\begin{aligned} f'(q) &= \left(-\frac{a}{2}(1-2q)(q-q^2)^{-1/2} - b(\delta-1/2)(1-2q)(q-q^2)^{\delta-3/2}\right)\phi\left(\frac{\theta}{\sigma}\right) \\ &\quad - \left(-\frac{a}{2}(1-2q)(q-q^2)^{-1/2} - b(\delta-1/2)(1-2q)(q-q^2)^{\delta-3/2}\right)\phi\left(\frac{\gamma}{\sigma}\right), \\ g'(q) &= n(1-\delta)(1-2q)(q-q^2)^{-\delta} + \lambda_2\delta(1-2q)(q-q^2)^{\delta-1} \end{aligned}$$

Note that $f(q)$ and $g(q)$ are both differentiable and $g'(q) \neq 0$ everywhere in the interval $(1/2, 1)$. Now note that we have

$$\begin{aligned} \frac{f'(q)}{g'(q)} = &\frac{1}{n(1-\delta)(q-q^2)^{1/2-\delta} + \lambda_2\delta(1-2q)(q-q^2)^{\delta-1/2}} \\ &\times \left(\left(-\frac{a}{2} - b(\delta-1/2)(q-q^2)^{\delta-1}\right)\phi\left(\frac{\theta}{\sigma}\right) - \left(\frac{a}{2} - b(\delta-1/2)(q-q^2)^{\delta-1}\right)\phi\left(\frac{\gamma}{\sigma}\right)\right). \end{aligned} \tag{14}$$

For $0 \leq \delta < 1/2$, $\lim_{q\to 1^+} f'(q)/g'(q) = 0$ since the exponential terms of ϕ in Equation (14) dominate in the limit.

For $\delta = 1/2$, we have

$$\lim_{q\to 1^+} \frac{f'(q)}{g'(q)} = -\frac{a}{n+\lambda_2} \lim_{q\to 1^+} \left(\phi\left(\frac{\theta}{\sigma}\right) + \phi\left(\frac{\gamma}{\sigma}\right)\right) = -\frac{a}{n+\lambda_2}\,\phi(-b)$$

so that we can use L'Hôpital's rule to show that the second term in Equation (13) becomes

$$-\frac{2\beta_j^*\lambda_1\sqrt{n}}{\sigma_\varepsilon(n+\lambda_2)}\,\phi\left(\frac{-\lambda_1}{\sigma_\varepsilon\sqrt{n}}\right). \tag{15}$$

For $\delta > 1/2$, we have

$$\begin{aligned} \lim_{q\to 1^+} \frac{f'(q)}{g'(q)} &= \lim_{q\to 1^+} \frac{-\frac{a}{2}\left(\phi\left(\frac{\theta}{\sigma}\right) + \phi\left(\frac{\gamma}{\sigma}\right)\right)}{n(1-\delta)(q-q^2)^{1/2-\delta} + \lambda_2\delta(1-2q)(q-q^2)^{\delta-1/2}} \\ &\quad + \lim_{q\to 1^+} \frac{b(\delta-1/2)\left(\phi\left(\frac{\gamma}{\sigma}\right) - \phi\left(\frac{\theta}{\sigma}\right)\right)}{n(1-\delta)(q-q^2)^{3/2-2\delta} + \lambda_2\delta(1-2q)(q-q^2)^{1/2}} \\ &= 0 + \lim_{q\to 1^+} \frac{b(\delta-1/2)e^{-\frac{1}{2}\left(a^2(q-q^2)+b^2(q-q^2)^{2\delta-1}\right)}\left(e^{-ab(q-q^2)^{\delta}} - e^{ab(q-q^2)^{\delta}}\right)}{\sqrt{2\pi}\left(n(1-\delta)(q-q^2)^{3/2-2\delta} + \lambda_2\delta(1-2q)(q-q^2)^{1/2}\right)} \\ &= 0 \end{aligned}$$

since the exponential term in the numerator dominates.

Now we proceed to consider the terms involving ϕ in Equation (11). We have

$$\lim_{q\to 1^+} \frac{\sigma}{d}\left(\phi\left(\frac{\gamma}{\sigma}\right) - \phi\left(\frac{\theta}{\sigma}\right)\right) = \sigma_\varepsilon\sqrt{n} \lim_{q\to 1^+} \frac{\phi\left(\frac{\gamma}{\sigma}\right) - \phi\left(\frac{\theta}{\sigma}\right)}{n(q-q^2)^{1/2} + \lambda_2(q-q^2)^{2\delta-1/2}} \tag{16}$$

For $0 \leq \delta < 1/2$, we observe that the exponential terms in ϕ dominate in the limit, and so we can distribute the limit and consider the limits of the respective terms individually, which both vanish.

For $\delta \geq 1/2$, the limit in Equation (16) has an indeterminate form of the type $\infty \times 0$. Define

$$u(q) = \phi\left(\frac{\gamma}{\sigma}\right) - \phi\left(\frac{\theta}{\sigma}\right) \qquad \text{and} \qquad v(q) = n(q-q^2)^{1/2} + \lambda_2(q-q^2)^{2\delta-1/2}$$

which are both differentiable in the interval $(1/2, 1)$ and $v'(q) \neq 0$ everywhere in this interval. The derivatives are

$$\begin{aligned} u'(q) = &- \phi\left(\frac{\gamma}{\sigma}\right)\frac{\gamma}{\sigma}\left(\frac{1}{2}\left(a(1-2q)(q-q^2)^{-1/2}\right) - b(\delta-1/2)(1-2q)(q-q^2)^{\delta-3/2}\right) \\ &+ \phi\left(\frac{\theta}{\sigma}\right)\frac{\theta}{\sigma}\left(-\frac{1}{2}\left(a(1-2q)(q-q^2)^{-1/2}\right) - b(\delta-1/2)(1-2q)(q-q^2)^{\delta-3/2}\right), \\ v'(q) = &\frac{n}{2}(1-2q)(q-q^2)^{-1/2} + \lambda_2(2\delta-1/2)(1-2q)(q-q^2)^{2\delta-3/2}. \end{aligned}$$

And so

$$\begin{aligned} \frac{u'(q)}{v'(q)} = \frac{1}{n + \lambda_2(4\delta-1)(q-q^2)^{2\delta-1}}\Bigg(&- \left(a - b(2\delta-1)(q-q^2)^{\delta-1}\right)\phi\left(\frac{\gamma}{\sigma}\right)\frac{\gamma}{\sigma} \\ &- \left(a + b(2\delta-1)(q-q^2)^{\delta-1}\right)\phi\left(\frac{\theta}{\sigma}\right)\frac{\theta}{\sigma}\Bigg). \end{aligned} \tag{17}$$

Taking the limit, rearranging, and assuming that the limits of the separate terms exist, we obtain

$$\begin{aligned} \lim_{q\to 1^+}\frac{u'(q)}{v'(q)} = &-a\lim_{q\to 1^+}\frac{1}{n + \lambda_2(4\delta-1)(q-q^2)^{2\delta-1}}\left(\phi\left(\frac{\gamma}{\sigma}\right)\frac{\gamma}{\sigma} + \phi\left(\frac{\theta}{\sigma}\right)\frac{\theta}{\sigma}\right) \\ &+ b(2\delta-1)\lim_{q\to 1^+}\frac{1}{n + \lambda_2(4\delta-1)(q-q^2)^{2\delta-1}}\Bigg(\phi\left(\frac{\gamma}{\sigma}\right)\left(a(q-q^2)^{\delta-1/2} - b(q-q^2)^{2\delta-3/2}\right) \\ &- \phi\left(\frac{\theta}{\sigma}\right)\left(-a(q-q^2)^{\delta-1/2} - b(q-q^2)^{2\delta-3/2}\right)\Bigg). \end{aligned} \tag{18}$$

For $\delta = 1/2$, we have

$$\lim_{q\to 1^+}\frac{u'(q)}{v'(q)} = -\frac{a}{n+\lambda_2}\left(-b\,\phi(-b) - b\,\phi(-b)\right) + 0 = 2ab\,\phi(-b) = \frac{2\beta_j^*\lambda_1}{\sigma_\varepsilon^2(n+\lambda_2)}\,\phi\left(\frac{-\lambda_1}{\sigma_\varepsilon\sqrt{n}}\right).$$

Using L'Hôpital's rule, the second term in Equation (16) must consequently be

$$\frac{2\beta_j^*\lambda_1\sqrt{n}}{\sigma_\varepsilon(n+\lambda_2)}\,\phi\left(\frac{-\lambda_1}{\sigma_\varepsilon\sqrt{n}}\right).$$

which cancels with Equation (15).

For $\delta > 1/2$, we first observe that the first term in Equation (18) tends to zero due to Equation (12) and the properties of the standard normal distribution. For the second term, we note that this is essentially of the same form as Equation (14) and that the limit is therefore 0 here.

B.3 Proof of Theorem 3.2

The variance of the elastic net estimator is given by

$$\begin{aligned} \operatorname{Var}\hat{\beta}_j = \frac{1}{d^2}\Bigg(&\frac{\sigma^2}{2}\left(2 + \operatorname{erf}\left(\frac{\theta}{\sigma\sqrt{2}}\right) - \frac{\theta}{\sigma}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\theta^2}{2\sigma^2}\right) + \operatorname{erf}\left(\frac{\gamma}{\sigma\sqrt{2}}\right) - \frac{\gamma}{\sigma}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\gamma^2}{2\gamma^2}\right)\right) \\ &+ 2\theta\sigma\,\phi\left(\frac{\theta}{\sigma}\right) + \theta^2\,\Phi\left(\frac{\theta}{\sigma}\right) + 2\gamma\sigma\,\phi\left(\frac{\gamma}{\sigma}\right) + \gamma^2\,\Phi\left(\frac{\gamma}{\sigma}\right)\Bigg) - \left(\frac{1}{d}\operatorname{E}\hat{\beta}_j\right)^2. \end{aligned} \tag{19}$$

We start by noting the following identities:

$$\begin{aligned}
\theta^2 &= \left(\beta_j^* n\right)^2 (q-q^2)^{2-2\delta} + \lambda_1^2 + 2\lambda_1\beta_j^* n(q-q^2)^{1-\delta},\\
d^2 &= n^2(q-q^2)^{2-2\delta} + 2n\lambda_2(q-q^2) + \lambda_2^2(q-q^2)^{2\delta},\\
\theta\sigma &= -\sigma_\varepsilon\left(\beta_j^* n^{3/2}(q-q^2)^{3/2-2\delta} + \sqrt{n}\lambda_1(q-q^2)^{1/2-\delta}\right),\\
\frac{\theta^2}{\sigma^2} &= a^2(q-q^2) + b^2(q-q^2)^{2\delta-1} + 2ab(q-q^2)^{\delta},\\
\frac{\sigma}{d} &= \frac{\sigma_\varepsilon\sqrt{n}}{n(q-q^2)^{\frac{1}{2}} + \lambda_2(q-q^2)^{2\delta-1/2}}.
\end{aligned}$$

Expansions involving γ, instead of θ, have identical expansions up to sign changes of the individual terms. Also recall the definitions provided in the proof of Theorem 3.1.

Starting with the case when $0 \le \delta < 1/2$, we write the limit of Equation (19) as

$$\begin{aligned}
&\lim_{q\to} \operatorname{Var}\hat{\beta}_j\\
&= \sigma_\varepsilon^2 n \lim_{q\to 1^+} \frac{1}{\left(n(q-q^2)^{1/2} + \lambda_2(q-q^2)^{2\delta-1/2}\right)^2}\left(1 + \operatorname{erf}\left(\frac{\theta}{\sigma\sqrt{2}}\right) - \frac{\theta}{\sigma}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\theta^2}{2\sigma^2}\right)\right)\\
&\quad + \sigma_\varepsilon^2 n \lim_{q\to 1^+} \frac{1}{\left(n(q-q^2)^{1/2} + \lambda_2(q-q^2)^{2\delta-1/2}\right)^2}\left(1 + \operatorname{erf}\left(\frac{\gamma}{\sigma\sqrt{2}}\right) - \frac{\gamma}{\sigma}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\gamma^2}{2\sigma^2}\right)\right)\\
&\quad + \lim_{q\to 1^+}\frac{2\theta\sigma}{d^2}\,\phi\left(\frac{\theta}{\sigma}\right) + \lim_{q\to 1^+}\frac{\theta^2}{d^2}\,\Phi\left(\frac{\theta}{\sigma}\right) + \lim_{q\to 1^+}\frac{2\gamma}{d^2}\sigma\,\phi\left(\frac{\gamma}{\sigma}\right) + \lim_{q\to 1^+}\frac{\gamma^2}{d^2}\,\Phi\left(\frac{\gamma}{\sigma}\right)\\
&\quad - \left(\lim_{q\to 1^+}\frac{1}{d}\,\mathrm{E}\,\hat{\beta}_j\right)^2,
\end{aligned}$$

assuming, for now, that all limits exist. Next, let

$$\begin{aligned}
f_1(q) &= 1 + \operatorname{erf}\left(\frac{\theta}{\sigma\sqrt{2}}\right) - \frac{\theta}{\sigma}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\theta^2}{2\sigma^2}\right),\\
f_2(q) &= 1 + \operatorname{erf}\left(\frac{\gamma}{\sigma\sqrt{2}}\right) - \frac{\gamma}{\sigma}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\gamma^2}{2\sigma^2}\right),\\
g(q) &= \left(n^2(q-q^2) + 2n\lambda_2(q-q^2)^{2\delta} + \lambda_2^2(q-q^2)^{4\delta-1}\right)^2.
\end{aligned}$$

And

$$\begin{aligned}
f_1'(q) &= \frac{\theta^2}{\sigma^2}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\theta^2}{2\sigma^2}\right),\\
f_2'(q) &= \frac{\gamma^2}{\sigma^2}\sqrt{\frac{2}{\pi}}\exp\left(-\frac{\gamma^2}{2\sigma^2}\right),\\
g'(q) &= (1-2q)\left((q-q^2)^{-1} + 4n\delta\lambda_2(q-q^2)^{2\delta-1} + \lambda_2^2(4\delta-1)(q-q^2)^{4\delta-2}\right).
\end{aligned}$$

f_1, f_1 and g are differentiable in $(1/2, 1)$ and $g'(q) \neq 0$ everywhere in this interval. f_1/g and f_2/g are indeterminate of the form $0/0$. And we see that

$$\lim_{q\to 1^+}\frac{f_1'(q)}{g'(q)} = \lim_{q\to 1^+}\frac{f_2'(q)}{g'(q)} = 0$$

due to the dominance of the exponential terms as θ/σ and γ/σ both tend to $-\infty$. Thus f_1/g and f_2/g also tend to 0 by L'Hôpital's rule.

Similar reasoning shows that

$$\lim_{q\to 1^+} \frac{2\theta\sigma}{d^2}\,\phi\left(\frac{\theta}{\sigma}\right) = \lim_{q\to 1^+} \frac{\theta^2}{d^2}\,\Phi\left(\frac{\theta}{\sigma}\right) = 0.$$

The same result applies to the respective terms involving γ.

And since we in Theorem 3.1 showed that $\lim_{q\to 1^+} \frac{1}{d}\,\mathrm{E}\,\hat{\beta}_j = 0$, the limit of Equation (19) must be 0.

For $\delta = 1/2$, we start by establishing that

$$\lim_{q\to 1^+} \int_{-\infty}^{-\lambda} (z+\lambda)^2 f_Z(z)\,\mathrm{d}z = \lim_{q\to 1^+} \left(\sigma^2 \int_{-\infty}^{\frac{\theta}{\sigma}} y^2\,\phi(y)\,\mathrm{d}y + 2\theta\sigma \int_{-\infty}^{\frac{\theta}{\sigma}} y\,\phi(y)\,\mathrm{d}y + \theta^2 \int_{-\infty}^{\frac{\theta}{\sigma}} \phi(y)\,\mathrm{d}y\right)$$

is a positive constant since $\theta/\sigma \to -b$, $\sigma = \sigma_\varepsilon\sqrt{n}$, $\theta \to -\lambda$, and $\theta\sigma \to -\sigma_\varepsilon\sqrt{n}\lambda$. An identical argument can be made in the case of

$$\lim_{q\to 1^+} \int_{\lambda}^{\infty} (z-\lambda)^2 f_Z(z)\,\mathrm{d}z.$$

We then have

$$\lim_{q\to 1^+} \frac{1}{d^2} \int_{-\infty}^{-\lambda} (z+\lambda)^2 f_Z(z)\,\mathrm{d}z = \frac{C^+}{\lim_{q\to 1^+} d^2} = \frac{C^+}{0} = \infty,$$

where C^+ is some positive constant. And because $\lim_{q\to 1^+} \frac{1}{d}\,\mathrm{E}\,\hat{\beta}_j = \beta_j^*$ (Theorem 3.1), the limit of Equation (19) must be ∞.

Finally, for the case when $\delta > 1/2$, we have

$$\begin{aligned}
\lim_{q\to 1^+} \frac{1}{d^2} &\left(\sigma^2 \int_{-\infty}^{\frac{\theta}{\sigma}} y^2\,\phi(y)\,\mathrm{d}y + 2\theta\sigma \int_{-\infty}^{\frac{\theta}{\sigma}} y\,\phi(y)\,\mathrm{d}y + \theta^2 \int_{-\infty}^{\frac{\theta}{\sigma}} \phi(y)\,\mathrm{d}y\right) \\
&= \lim_{q\to 1^+} \left(\frac{n\sigma^2}{\left(n(q-q^2)^{1/2} + \lambda_2(q-q^2)^{2\delta-1/2}\right)^2} \int_{-\infty}^{\frac{\theta}{\sigma}} y^2\,\phi(y)\,\mathrm{d}y \right. \\
&\quad - \frac{2\sigma_\varepsilon\sqrt{n}\left(\beta_j^* n(q-q^2)^{1-\delta} - \lambda_1\right)}{\left(n(q-q^2)^{3/4-\delta/2} + \lambda_2(q-q^2)^{3\delta/2-1/4}\right)^2} \int_{-\infty}^{\frac{\theta}{\sigma}} y\,\phi(y)\,\mathrm{d}y \\
&\quad \left. + \left(\frac{-\beta_j^* n(q-q^2)^{1-\delta} - \lambda_1}{n(q-q^2)^{1-\delta} + \lambda_2(q-q^2)^{\delta}}\right)^2 \int_{-\infty}^{\frac{\theta}{\sigma}} \phi(y)\,\mathrm{d}y\right).
\end{aligned}$$

Inspection of the exponents involving the factor $(q-q^2)$ shows that the first term inside the limit will dominate. And since the upper limit of the integrals, $\theta/\sigma \to 0$ as $q \to 1^+$, the limit must be ∞.

B.4 Proof of Corollary 3.2.1

We have

$$\lim_{q\to 1^+} \mathrm{Var}\,\hat{\beta}_j = \lim_{q\to 1^+} \frac{\sigma^2}{d_j^2} \left(\frac{\sigma_\varepsilon\sqrt{n}(q-q^2)^{1/2-\delta}}{n(q-q^2)^{1-\delta} + \lambda_2(q-q^2)^{\delta}}\right)^2 = \frac{\sigma_\varepsilon^2 n}{\lambda_2^2} \lim_{q\to 1^+} (q-q^2)^{1-4\delta},$$

from which the result follows directly.