EXTENDING THE REACH OF THE LASSO AND ELASTIC NET PENALTIES: METHODOLOGY AND PRACTICE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF STATISTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Jingyi Kenneth Tay

May 2021

Abstract

The lasso and its successor, the elastic net, are two popular regularized regression methods which produce sparse models. We propose two different extensions to these methods that leverage feature information in the model-fitting process to improve predictive accuracy. The first method, the "principal components lasso" ("pcLasso"), combines the lasso penalty with a quadratic penalty that shrinks the coefficient vector toward leading principal components of each group. pcLasso is compatible with both overlapping and non-overlapping groups. We provide some theory for the method and illustrate the method on some simulated and real data examples. Next, we develop a general framework for organizing feature information into an auxiliary matrix and propose a method, called the "feature-weighted elastic net" ("fwelnet"), that uses it to adapt the relative penalties on the feature coefficients in the elastic net penalty. We present connections between this method and the group lasso, and also to Bayesian estimation.

We then switch gears and explore how to relax another limitation of the lasso and the elastic net: the fact that the model's prediction must be a sparse linear combination of the input features. Motivated by reluctant interaction modeling, we propose a multi-stage algorithm, called "reluctant generalized additive modeling" (RGAM), that can fit sparse generalized additive models (GAMs) at scale. While the individual features are allowed to vary non-linearly with the response, the model remains additive in the input features, thus preserving interpretability. RGAM has an explicit bias toward simpler, linear relationships, with non-linearities only included if they improve predictive power. Unlike existing methods for sparse GAMs, RGAM can be extended easily to binary, count and survival data.

Finally, we turn to the practical matter of computation. Earlier versions of the **glmnet** package, which implements the lasso and the elastic net in R, had specialized FORTRAN subroutines for fitting the models for popular model families, including the logistic regression and Poisson regression models. We describe how the elastic net penalty can be applied to any generalized linear model (GLM), and explain the computational approach needed to make model-fitting feasible in practice. We also show how this approach extends to fitting regularized Cox proportional hazards models for (start, stop] data and for stratified data.

Acknowledgments

First, I would like to thank my advisor, Rob Tibshirani, for his invaluable mentorship and guidance. I think it is obvious to everyone that Rob is an outstanding researcher: he has amazing intuition, rapid prototyping skills, and his questions during my meetings with him quickly cut to the heart of the problem. What is perhaps not as obvious but more meaningful to me is that he is also a great encourager: I cannot count the number of times I went into his office feeling down about what I had done that week, only to come out with a sense of optimism and renewed excitement. It is no understatement to say that this thesis is built on his exclamations of “good work!” and “great stuff!” throughout the years.

Next, I would like to thank Art Owen, Guenther Walther, Trevor Hastie and Nima Aghaeepour for devoting time to serve on my thesis committee. I am especially grateful to Trevor for giving me the opportunity to work with him on **glmnet**: through the project I have learned so much about the joys and pains of maintaining high-quality statistical software. I have also gleaned much from sitting in group meetings with him. (It sometimes feels like I got a two-for-one deal when Rob became my PhD advisor.) I have also had the privilege of taking several courses in applied statistics and doing a research rotation with Art. His work has inspired me to see how a wide variety of subfields in applied statistics can be interesting in their own right. Many thanks also to Balasubramanian Narasimhan for his advice and help on all things related to software: TA-ing his STATS 290 class in my first year was a key experience in sparking my love for developing computational tools for statistics.

Thank you to my department mates in Sequoia Hall for walking the PhD journey with me. The students in the Stanford Statistics program are a vibrant, smart and hardworking bunch, and I count myself fortunate to have been part of this community. I have enjoyed our many lunches, teas and happy hours! There are too many names to list here, but I especially treasure my interactions with Paulo, Gene, and Stephen (all three happen to be my quals coaches), Evan R and Rina (for JSM shenanigans), Mona, Claire, and my batch mates Jaime, Mike C, Mike S, Nikos, Youngtak and Zhimei. A special shoutout to Susie Ementon, who makes departmental admin a breeze and who is always up for a random drop-in chat.

I am grateful for family back in Singapore for always supporting me from afar, and for reminding me that their love for me does not depend on how well I do in my career. Special thanks to my in-laws, especially my mother-in-law who came up a number of times to help with childcare. Thank you also to my friends in the Bay Area who have become a second family to me (Ali and Dil, Kelvin and Min, Shyue Ping and Denise, GS and Natalie), and to Dennis for being both a collaborator and a confidant. Not to forget my two daughters, Talia and Eden, who have taught me that no matter how good or bad my day in the office went, it ends at 5pm.

Behind every successful married PhD candidate is a(n extremely) supportive PhD spouse. I owe a huge debt of gratitude to my wife, Karen, who has been my strongest supporter and cheerleader since day one. There are so many things to thank her for: for being willing to take that leap of leaving a secure environment with a baby in tow, for supporting the family financially while taking care of the children, for challenging me to be the best that I can be, for believing in me even when I did not believe in myself. Her choosing to spend the rest of her life with me is the best thing that has happened to me.

Finally, all glory to my Lord and Savior Jesus Christ, for embedding beautiful and mysterious (statistical) treasures in this world for me to find, and for loving me the same even when I don't find them.

"The heavens proclaim the glory of God.
    The skies display his craftsmanship.
Day after day they continue to speak;
    night after night they make him known.
They speak without a sound or word;
    their voice is never heard.
Yet their message has gone throughout the earth,
    and their words to all the world."
            - Psalm 19:1-4

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last two decades, we have witnessed exponential growth in data across several fields, including genomics, finance and internet applications. Datasets have grown both in the number of observations and number of features per observation, presenting data analysts with the opportunity to discover complex and intricate relationships. In particular, having access to a larger pool of features suggests that we can do a better job of capturing the signal in the response variable that we want to predict.

At the same time, this wealth of data has its challenges. In any prediction problem, it is almost certainly the case that a large majority of features have no meaningful relationship to the response (e.g. do we believe that the weather yesterday has any bearing on the number of children born today?). Introducing such features into our prediction models only serves to increase the noise in our dataset, making it harder to uncover features that truly matter for our prediction problem. A secondary problem raised by having access to so many features is that of interpretability: why is our model making the predictions that it is making? Without this understanding, one could argue that our model has done nothing to further scientific knowledge.

One strategy that researchers have taken to deal with these challenges is to develop *regularized* models, that is, models which have some additional constraints imposed on them so that they yield models with desirable properties. One such desirable property is *sparsity*: the property that the model uses only a fraction of the features available to it when making predictions. The *lasso* (Tibshirani, 1996) and its successor, the *elastic net* (Zou and Hastie, 2005), are two popular regularized regression methods which produce sparse models. Their popularity stems from at least three reasons. First, they make few assumptions on the data at hand (the features just have to be real-valued), making it applicable to a wide variety of data settings. Second, their predictions are sparse linear combinations of the input features, making them easy to interpret. Third, the model coefficients are the result of a convex minimization problem which can be solved in an extremely computationally efficient manner.

While having few assumptions about the data makes the lasso and elastic net widely applicable, in some settings they leave potentially useful information on the table. For example, the input features to these algorithms are treated as exchangeable. This is often not the case: when trying to predict the number of children born today, the number of children born yesterday is certainly more useful than the yesterday's weather! Prediction models that are able to leverage such *feature information* in the model-fitting process are likely to be more accurate. In Chapter 2, we consider how the lasso can be modified to incorporate one type of feature information: feature grouping information. The idea is that features in the same group are likely to have a similar relationship with the response. We propose a new method, called the "principal components lasso" ("pcLasso"), that combines the lasso penalty with a quadratic penalty that shrinks the coefficient vector toward leading principal components of each group. pcLasso is compatible with both overlapping and non-overlapping groups. We provide some theory for the method and illustrate the method on some simulated and real data examples. An implementation for pcLasso is provided in the **pcLasso** R package.

In Chapter 3, we develop a more general framework for organizing such feature information to help with modeling. We think of each source of feature information as a "feature of features" and organize them into an auxiliary matrix. We then propose a method, called the "feature-weighted elastic net" ("fwelnet"), that uses this auxiliary matrix to adapt the relative penalties on the feature coefficients in the elastic net penalty. In our simulations, fwelnet outperforms the lasso in terms of test mean squared error and usually gives an improvement in true positive rate or false positive rate for feature selection. We present connections between this method and the group lasso, and also to Bayesian estimation. We also apply this method to early prediction of preeclampsia, where fwelnet outperforms the lasso in terms of 10-fold cross validation (0.84 vs. 0.80) and suggest how fwelnet might be used for multi-task learning. An implementation of the fwelnet method is provided in the **fwelnet** R package.

In Chapter 4, we switch gears and explore how to relax another limitation of the elastic net: the fact that the model's prediction must be a sparse linear combination of the input features, which is a poor approximation of reality in some applications. Motivated by reluctant interaction modeling (Yu et al., 2019), we propose a multi-stage algorithm, called "reluctant generalized additive modeling" (RGAM), that can fit sparse generalized additive models (GAMs) at scale. While the individual features are allowed to vary non-linearly with the response, the model remains additive in the input features, thus preserving interpretability. RGAM is guided by the principle that, if all else is equal, one should prefer a linear feature over a non-linear feature. Hence, RGAM has an explicit bias toward simpler, linear relationships, with non-linearities only included if they improve predictive power. Unlike existing methods for sparse GAMs, RGAM can be extended easily to binary, count and survival data. We demonstrate the method's effectiveness on real and simulated examples. The method is implemented in the **relgam** R package.

In Chapter 5, we turn to the practical matter of computation. For statistical methodology to make real-world impact, it must be accompanied with robust, computationally efficient software. One reason for the popularity of the elastic net is that it has an extremely efficient implementation in the **glmnet** R package (Friedman et al., 2010). Earlier versions of the package had specialized FORTRAN subroutines for fitting the elastic net model for popular model families, including the logistic regression and Poisson regression models. We describe how the elastic net penalty can be applied to any generalized linear model (GLM), and explain the computational approach needed to make model-fitting feasible in practice. We also show how this approach extends to fitting regularized Cox proportional hazards models for (start, stop] data and for stratified data.

A note on provenance

The contents in this dissertation are drawn from four manuscripts. Chapter 2 is drawn with minor modifications from the manuscript "Principal component-guide sparse regression", jointly authored with Jerome Friedman and Robert Tibshirani, to appear in the Canadian Journal of Statistics. Chapter 3 is drawn from a manuscript under review titled "Feature-weighted elastic net: using "features of features" for better prediction" and jointly authored with Nima Aghaeepour, Trevor Hastie and Robert Tibshirani. Chapter 4 is drawn from the manuscript "Reluctant generalized additive modeling", jointly authored with Robert Tibshirani and published in the International Statistical Review (Tay and Tibshirani, 2020). Chapter 5 is drawn with modifications from a manuscript under review titled "Elastic net regularization paths for all generalized linear models" under review, jointly authored with Balasubramanian Narasimhan and Trevor Hastie.

Chapter 2

Principal Components Lasso

2.1 Introduction

We consider the usual linear regression model, where we are given n realizations of p predictors $\mathbf{X} = \{x_{ij}\}$ for $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$, and a response variable $\mathbf{y} = (y_1, \ldots, y_n)^T$. Assume that we have centered $\mathbf{y}$ and the columns of $\mathbf{X}$ so that we can avoid fitting an intercept term. We model the relationship between the response and the predictors by

$$y_i = \sum_j x_{ij}\beta_j + \varepsilon_i,$$

with ε having mean 0 and variance σ^2. The ordinary least squares (OLS) estimates of β_j are obtained by minimizing the residual sum of squares (RSS). There has been much work on regularized estimators that offer an advantage over the OLS estimates, both in terms of accuracy of prediction on future data and interpretation of the fitted model. One such method is the *lasso* (Tibshirani, 1996), which minimizes

$$J(\boldsymbol{\beta}) = \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|_2^2 + \lambda \left\|\boldsymbol{\beta}\right\|_1,$$

where $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)^T$, and the tuning parameter $\lambda \geq 0$ controls the sparsity of the final model. A big reason for the lasso's popularity is that the fitted model is *sparse*, i.e. $\beta_j = 0$ except for a handful of indices j. Besides making the predictions more interpretable, the user can focus on the small subset of selected features for subsequent exploration and research. These advantages are especially pertinent in wide data settings such as genome-wide association studies.

The lasso is an alternative to ridge regression (Hoerl and Kennard, 1970), which minimizes an objective function with an ℓ_2 penalty:

$$\hat{\boldsymbol{\beta}}_R = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}} \quad \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|_2^2 + \lambda \left\|\boldsymbol{\beta}\right\|_2^2.$$

Let $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ be the singular value decomposition (SVD) of $\mathbf{X}$, with $\mathbf{D}$ being a diagonal matrix with diagonal entries being the singular values $d_1 \geq d_2 \geq \cdots \geq d_m > 0$, where $m = \text{rank}(\mathbf{X})$. The ridge regression coefficients and the resulting predictions have an explicit form:

$$\hat{\boldsymbol{\beta}}_R = (\mathbf{X}^T\mathbf{X} + \mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}, \qquad \mathbf{X}\hat{\boldsymbol{\beta}}_R = \sum_{j=1}^{m} \frac{d_j^2}{d_j^2 + \lambda}\mathbf{u}_j\mathbf{u}_j^T\mathbf{y},$$

where $\mathbf{u}_j$ is the jth column of $\mathbf{U}$. Since the OLS predictions have the form

$$\mathbf{X}\hat{\boldsymbol{\beta}}_{OLS} = \sum_{j=1}^{m} \mathbf{u}_j\mathbf{u}_j^T\mathbf{y},$$

and $d_j^2/(d_j^2+\lambda)$ decreases as j increases, we see that ridge regression biases the predictions toward the leading principal components (PCs) of $\mathbf{X}$, which can be effective for improving prediction accuracy. This is because we typically expect variation in the response to correlated well with variation in the features, which is why we collect these features in the first place. However, it has the disadvantage of producing dense models: all the original features have non-zero coefficients in the fitted model. The elastic net (Zou and Hastie, 2005) attempts to combine the lasso and ridge regression by adding an ℓ_2 penalty to the lasso's objective function.

There are two limitations that we found with the elastic net. First, if the predictors come in pre-assigned groups, the elastic net does not make use of this information to guide model-fitting. Such grouping information has the potential to be very useful in some settings. For example, in genomics if we know that a certain genetic pathway has some influence on a response variable, the genes making up that genetic pathway are likely to have an influence as well. Conversely, if a certain genetic pathway is irrelevant to the response variable, the genes in that pathway probably do not add predictive power to our model. Second, in practice we have found that the bias of ridge regression (and hence the elastic net) toward the leading PCs is too weak.

We propose a new method, the *principal components lasso* ("pcLasso"), a penalized regression method that is able to use grouping information in its model-fitting process and strongly biases the solution vector toward the leading principal components while maintaining sparsity of the fitted model. This allows for better prediction accuracy without sacrificing interpretability of the model. pcLasso is applicable to both overlapping and non-overlapping group settings. It is computationally scalable, with the fitted model being a solution to a convex minimization problem that can be solved efficiently by coordinate descent.

The rest of this chapter is organized as follows. In Section 2.2, we briefly review methods comparable to pcLasso and describe the gap that pcLasso fills. We detail the pcLasso method in the two sections following, with Section 2.3 describing pcLasso for the single group case for motivation, and Section 2.4 detailing the method for multiple groups. We illustrate the effectiveness of pcLasso on real data examples in Section 2.5, and give details of our computational approach in Section 2.6.

Section 2.7 derives a formula for the degrees of freedom of the pcLasso fit. In Section 2.8 we discuss a simulation study comparing our proposal with other methods. We study the theoretical properties of the principal components lasso in Section 2.9, showing that, under certain assumptions, it can yield lower ℓ_2 and prediction error than the lasso. We end with a discussion and ideas for future work. The Appendix contains further details and proofs, as well as a full description of the set-up and results for our simulation study in Section 2.8.

2.2 Related work

There are several penalized regression methods that can exploit group structure, the most commonly employed being the group lasso (Yuan and Lin, 2006) for non-overlapping groups and the overlap group lasso (Obozinski et al., 2009) for overlapping groups. These methods induce sparsity at the group level by imposing ℓ_2 penalties for the coefficients in each group. However, they do not induce sparsity within each group: if a group is selected, all of its features will have non-zero coefficients. This may be desirable (e.g. dummy variables for the same categorical variable, or basis expansions of the same original feature) or not (e.g. genetic pathways containing hundreds of genes). pcLasso may be more suitable in the latter case as it achieves sparsity at the level of individual features while using grouping information for obtaining the fit. We demonstrate in Section 2.6.1 that pcLasso can also achieve in sparsity at the group level in some settings. The sparse group lasso (Simon et al., 2013) achieves both sparsity of groups and within the groups by combining the lasso and group lasso penalties, but does not bias the solution toward the leading principal components. It is also computationally much slower than pcLasso (see Appendix A.2 for details).

The idea of biasing the solution vector toward the top PCs of the data matrix is not new. In principal components regression, we compute the PCs of the data matrix, then perform linear regression of the response on the top k PCs, where k is a tuning parameter. The main drawback of this method is that the PCs are dense in the original feature space, meaning the resulting model is not sparse. One way to overcome this is to find "sparse principal components", i.e. sparse linear combinations of the original variables that approximate the PCs. There are a variety of methods to do so, e.g. Jolliffe et al. (2003), Zou et al. (2006), Mackey (2009), Witten et al. (2009), Johnstone and Lu (2009), Cai et al. (2013). The main difficulty in implementing regression on sparse principal components is computational as the sparsity of each PC is typically governed by its own tuning parameter. Sharing of tuning parameters across PCs is not straightforward, even if we enforce the same level of sparsity for each PC via a bound constraint. The algorithms typically solve the optimization problem for the sparse PC in Lagrange form, and this same bound constraint value will map to different dual variable values which we cannot predict in advance. This problem is compounded in situations where we have multiple groups of variables. pcLasso avoids the difficulty of estimating the sparse PCs before running the regression through the use of the pcLasso penalty;

this is not a loss since we are not interesting in the sparse PCs themselves but the resulting regression model.

2.3 Quadratic penalties based on principal components

In generalizing the ℓ_2 penalty, one can imagine a class of penalty functions of the form

$$\boldsymbol{\beta}^T \mathbf{V}\mathbf{Z}\mathbf{V}^T \boldsymbol{\beta},$$

where $\mathbf{Z}$ is a diagonal matrix whose diagonal elements are functions of the squared singular values:

$$Z_{11} = f_1(d_1^2, d_2^2, \ldots, d_m^2), Z_{22} = f_2(d_1^2, d_2^2, \ldots, d_m^2), \ldots$$

The ridge penalty chooses $Z_{jj} = 1$ for all j, leading to the quadratic penalty $\boldsymbol{\beta}^T\boldsymbol{\beta}$. Here we propose a different choice: we minimize

$$J(\boldsymbol{\beta}) = \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|_2^2 + \lambda\left\|\boldsymbol{\beta}\right\|_1 + \frac{\theta}{2}\boldsymbol{\beta}^T \mathbf{V}\mathbf{D}_{d_1^2 - d_j^2}\mathbf{V}^T\boldsymbol{\beta}, \tag{2.1}$$

where $\mathbf{D}_{d_1^2 - d_j^2}$ is an $m \times m$ diagonal matrix with diagonal entries equaling $d_1^2 - d_1^2, d_1^2 - d_2^2, \ldots, d_1^2 - d_m^2$. We call this the "principal components lasso penalty" and define it more generally in the next section. The value $\theta \geq 0$ is a tuning parameter determining the weight given to the second penalty. We see that this penalty term gives no weight (a "free ride") to the component of $\boldsymbol{\beta}$ that aligns with the first right singular vector of $\mathbf{X}$, i.e, the first principal component (PC). Beyond that, the size of the penalty depends on the gap between the squared singular values d_j^2 and d_1^2. If we view the quadratic penalty as representing a Bayesian (log-)prior, it puts infinite prior variance on the leading eigenvector.

It is instructive to compare Equation (2.1) with $\lambda = 0$ (and with $\theta/2$ replaced with θ) to ridge regression which uses the penalty $\theta\boldsymbol{\beta}^T\boldsymbol{\beta}$. Ridge regression produces the fit

$$\mathbf{X}\hat{\boldsymbol{\beta}}_R = \sum_{j=1}^{m} \frac{d_j^2}{d_j^2 + \theta}\mathbf{u}_j\mathbf{u}_j^T\mathbf{y},$$

whereas Equation (2.1) gives

$$\mathbf{X}\hat{\boldsymbol{\beta}}_L = \sum_{j=1}^{m} \frac{d_j^2}{d_j^2 + \theta(d_1^2 - d_j^2)}\mathbf{u}_j\mathbf{u}_j^T\mathbf{y}. \tag{2.2}$$

The latter expression corresponds to a more aggressive form of shrinkage toward the leading singular vectors. To see this, Figure 2.1 shows the shrinkage factors $d_j^2/(d_j^2 + \theta)$ and $d_j^2/[d_j^2 + \theta(d_1^2 - d_j^2)]$ for a 100×20 design matrix $\mathbf{X}$ with a strong rank-1 component. In each panel, we have chosen the value θ (different for each method) to yield the degrees of freedom indicated at the top of the panel. (For each method, the degrees of freedom is equal to the sum of the shrinkage factors.) The

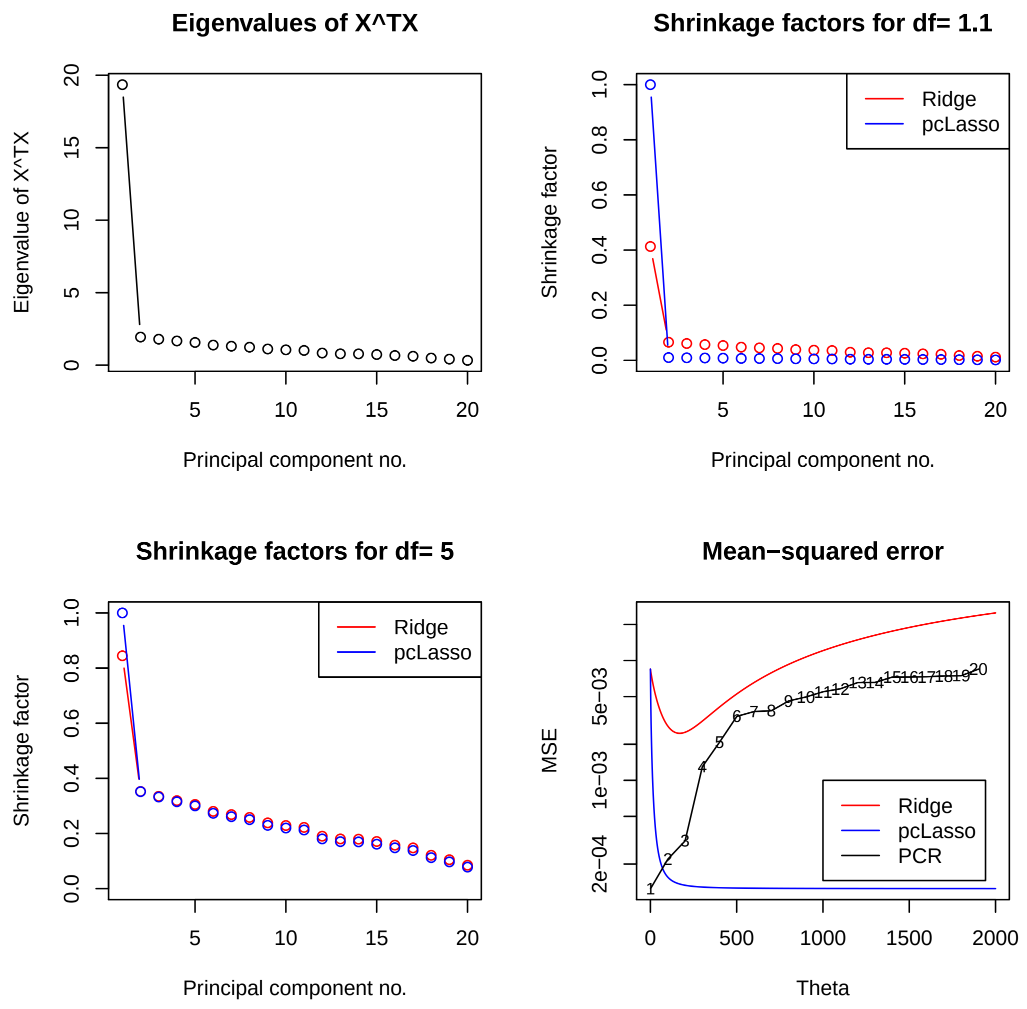

Figure 2.1: *Simulation example: The design matrix* $\mathbf{X}$ *has dimensions* 100×20 *with a strong rank-one component: the eigenvalues of* $\mathbf{X}^T\mathbf{X}$ *are shown in the top-left panel. Top-right and bottom-left panels show the shrinkage factors for the pcLasso penalty (blue) and ridge regression (red) across the PCs, for different degrees of freedom. For small degrees of freedom, pcLasso shrinks more aggressively to the top PCs. For larger degrees of freedom, the two methods have similar shrinkage profiles. The bottom right panel shows the mean-squared error (MSE) of ridge regression and pcLasso as the regularization parameter* θ *varies along the horizontal axis, and that of PC regression for various ranks. pcLasso achieves about the same MSE as PC regression of rank-1, while ridge regression does not.*

eigenvalues of $\mathbf{X}^T\mathbf{X}$ are shown in the top-left panel. We see that when the degrees of freedom is small, ridge regression shrinks the top component about twice as much as does pcLasso (top-right panel). This contrast is less stark when the degrees of freedom is increased to five (bottom-left panel). This aggressiveness can help pcLasso improve prediction accuracy when the signal lines up strongly with the top PCs. More importantly, with pre-specified feature groups, the pcLasso penalty in Equation (2.1) can be modified in such a way that it shrinks the subvector of $\boldsymbol{\beta}$ in each group toward the leading PCs of that group. This exploits the group structure to give potentially better predictions and simultaneously selects groups of features. We give details in the next section.

2.4 Principal components lasso ("pcLasso")

2.4.1 Definition

Suppose that the p predictors are grouped in K non-overlapping groups (we discuss the overlapping groups case later in Section 2.4.2). For example, these groups could be different gene pathways, assays or protein interaction networks. For $k = 1, \dots, K$, let $\mathbf{X}_k \in \mathbb{R}^{n \times p_k}$ denote the p_k columns of $\mathbf{X}$ corresponding to group k, let $m_k = \text{rank}(\mathbf{X}_k)$, and let $(\mathbf{V}_k, d_k)$ denote the right singular vectors and singular values of $\mathbf{X}_k$. pcLasso minimizes

$$J(\boldsymbol{\beta}) = \frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 + \frac{\theta}{2} \sum_k \boldsymbol{\beta}_k^T \Big(\mathbf{V}_k \mathbf{D}_{d_{k1}^2 - d_{kj}^2} \mathbf{V}_k^T \Big) \boldsymbol{\beta}_k. \tag{2.3}$$

Here, $\boldsymbol{\beta}_k$ is the sub-vector of $\boldsymbol{\beta}$ corresponding to group k, $d_k = (d_{k1}, \dots, d_{km_k})$ are the singular values of $\mathbf{X}_k$ is decreasing order, and $\mathbf{D}_{d_{k1}^2 - d_{kj}^2}$ is a diagonal matrix with diagonal entries $d_{k1}^2 - d_{kj}^2$ for $j = 1, 2, \dots, m_k$. Optionally, we might also multiply each penalty term in the sum by a factor $\sqrt{p_k}$, p_k being the group size, as is standard in related methods such as the group lasso (Yuan and Lin, 2006).

Some observations on solving Equation (2.3) are in order:

- The objective function is convex and the non-smooth component is separable. Hence, it can be optimized efficiently by a coordinate descent procedure (see Section 2.6 for details).
- Our approach is to fix a few values of θ (including zero, corresponding to the lasso), and then optimize the objective in Equation (2.3) over a path of λ values. Cross-validation is used to choose $\hat{\theta}$ and $\hat{\lambda}$.
- The parameter θ is not unitless, and is difficult to interpret. In our software, instead of asking the user to specify θ, we ask the user to specify a function of θ, denoted by "ratio". This is the ratio between the shrinkage factors in Equation (2.2) for $k = 2$ and $k = 1$ (the latter being

equal to 1). Explicitly,

$$ratio = \frac{d_2^2}{d_2^2 + \theta(d_1^2 - d_2^2)} \Bigg/ \frac{d_1^2}{d_1^2 + \theta(d_1^2 - d_1^2)} = \frac{d_2^2}{d_2^2 + \theta(d_1^2 - d_2^2)}.$$

This ratio is between 0 and 1, with $ratio = 1$ corresponding to $\theta = 0$ (the lasso) and lower values corresponding to more shrinkage. In practice, we find that using the values $ratio = 0.25, 0.5, 0.75, 0.9, 0.95$ and 1 covers a wide range of possible models.

- A potentially costly part of the computation is the SVD of each $\mathbf{X}_k$. If $\mathbf{X}_k$ large, for computational ease, we can use an SVD of rank $< \text{rank}(\mathbf{X}_k)$ as an approximation without much loss of performance.

For some insight into how pcLasso shrinks coefficients, we consider the contours of the penalty in the case of two predictors. Let ρ denote the correlation between these two predictors. In this case, it is easy to show that the two right singular vectors of $\mathbf{X}$ are $(1,1)^T/\sqrt{2}$ and $(1,-1)^T/\sqrt{2}$, with the former (latter resp.) being the leading singular vector if $\rho > 0$ ($\rho < 0$ resp.). Figure 2.2 presents the penalty contours for pcLasso along with that for ridge, lasso and elastic net regression for comparison. (See Appendix A.1 for detailed computations on the pcLasso contours.) We can see that pcLasso imposes a smaller penalty in the direction of the leading singular vector, and that this penalty is smaller when the correlation between the two predictors is stronger. We also see that as $\theta \to 0$, the pcLasso contours become like those for the lasso, while as $\theta \to \infty$, they become line segments in the direction of the leading singular vector.

Figure 2.3 shows the penalty contours for a particular set of three predictors, with the ratio λ/θ decreasing as we go from left to right. As this ratio goes from ∞ to 0, the contours move from being the ℓ_1 ball (as in the lasso) to becoming more elongated in the direction of the first PC.

2.4.2 The overlapping groups setting

In some settings, individual features might belong to more than one group. For example, one gene can belong to several genetic pathways. As in the group lasso (Yuan and Lin, 2006), there are two approaches one can take to deal with overlapping groups. One approach is to apply the group penalty to the given groups: this however has the undesirable effect of zeroing out a predictor's coefficient if any group to which it belongs is zeroed out. The overlap also causes additional computational challenges as the penalty is no longer separable. A better approach is the "overlap group lasso" penalty (Obozinski et al., 2009), where one replicates columns of $\mathbf{X}$ to account for multiple memberships, hence creating non-overlapping groups. The model is then fit as if the groups are non-overlapping and the coefficients for each original feature are summed to create the estimated coefficient for that feature. We take this same approach for pcLasso.

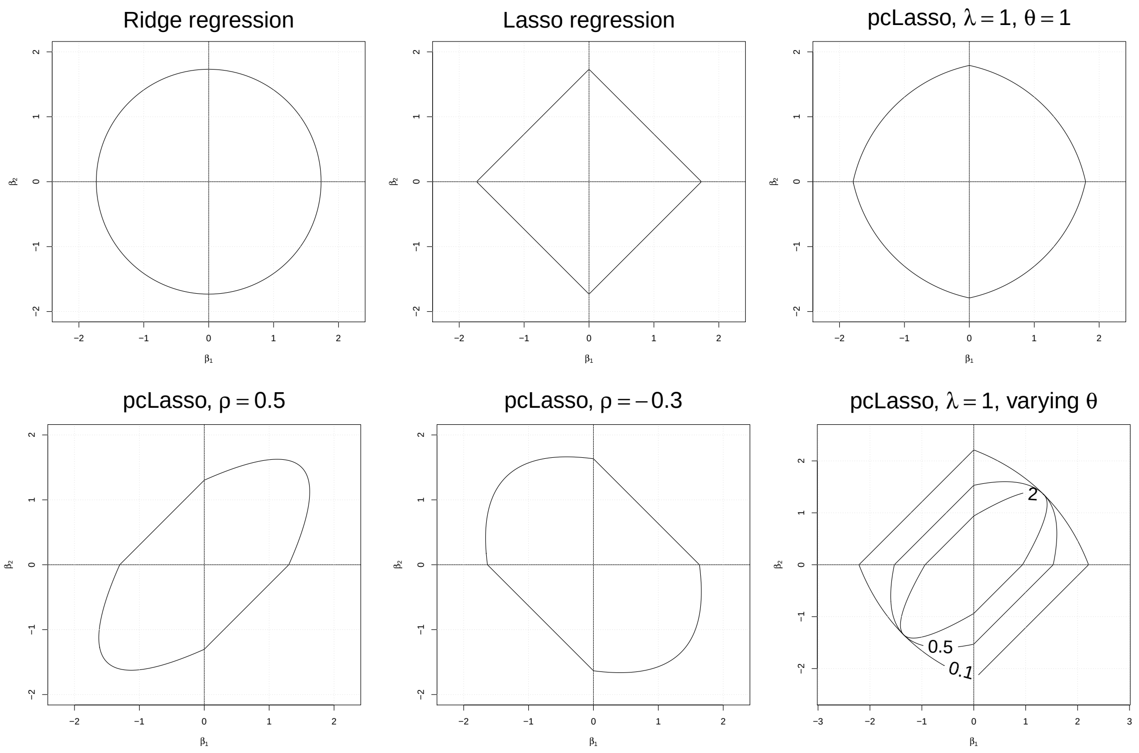

Figure 2.2: *Contours for the pcLasso penalty, compared with those for ridge, lasso and elastic net regression. pcLasso penalizes the coefficient vector less in the direction of the leading singular vector. As $\theta \to 0$, the pcLasso contours become like those for the lasso, while as $\theta \to \infty$, they become line segments in the direction of the leading singular vector.*

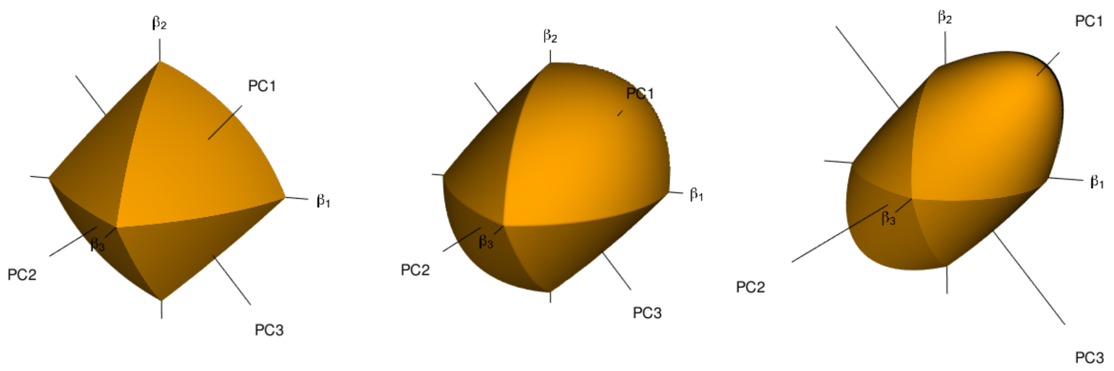

Figure 2.3: *Contours for the pcLasso penalty for three predictors. For a fixed value of θ, λ decreases as we go from left to right. As we give less relative weight to the ℓ_1 penalty, the penalty contours look less and less like that of the lasso and become more elongated in the direction of the first PC.*

2.5 Real data examples

2.5.1 p53 microarray expression data

We analyze the data from a gene expression study, as described in Zeng and Breheny (2016), taken from Subramanian et al. (2005). It involves the mutational status of the gene p53 in cell lines. The study aims to identify pathways that correlate with the mutational status of p53, which regulates gene expression in response to various signals of cellular stress. The data consists of 50 cell lines, 17 of which are classified as normal and 33 of which carry mutations in the p53 gene. To be consistent with the analysis in Subramanian et al. (2005), 308 gene sets that have size between 15 and 500 are included in our analysis. These gene sets contain a total of 4,301 genes and is available in the R package **grpregOverlap**. When the data is expanded to handle the overlapping groups, it contains a total of 13,237 columns. As the response is binary ("does the patient have the mutation or not?"), we fit a penalized logistic regression model where we minimize the sum of the logistic loss and the pcLasso penalty.

We divided the data into 10 cross-validation (CV) folds and applied the lasso, pcLasso and the group lasso using the R package **grpregOverlap**. (The data was too large for the sparse group lasso package, **SGL**.) Figure 2.4 shows that pcLasso achieves higher cross-validated area under the curve (AUC) than the other approaches.

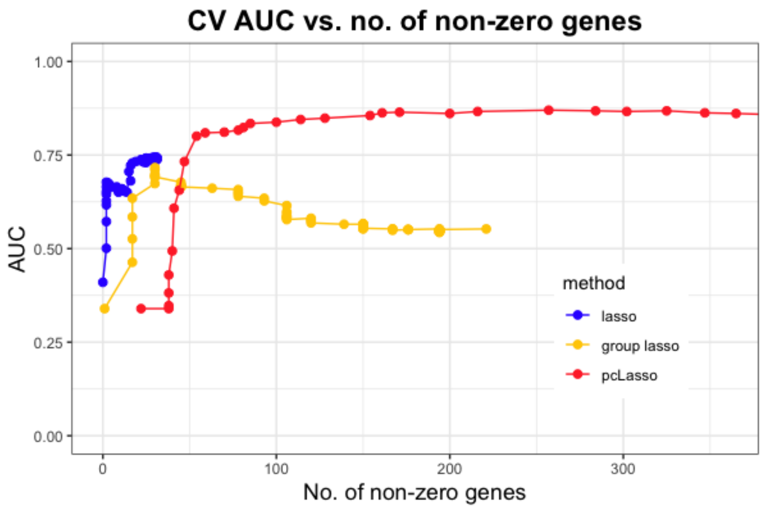

Figure 2.4: *Results for p53 dataset: A plot of cross-validated area under the curve (AUC) vs. the number of non-zero genes in the selected model for the lasso, group lasso and pcLasso. pcLasso achieves the best result here.*

2.5.2 Colorectal array comparative genomic hybridization (CGH) data

In this next example, we analyze data from a colorectal array comparative genomic hybridization (CGH) dataset from Nakao et al. (2004), which can be obtained from the R package **aCGH**. For $n = 39$ samples, the response is the stage of the colorectal cancer: 0, 1, 2 or 3. (We removed one of the samples from the original dataset as the stage of the colorectal cancer was missing.) This response is on the ordinal scale; as a first approximation we treat it as a quantitative response. Array-based CGH is run for each of these samples to detect copy number alterations. Following some data processing and imputation outlined in Nakao et al. (2004), we obtain log2 ratios of copy number changes for $p = 2,031$ clones. We wish to use these copy number changes to predict the stage of the colorectal cancer.

We divided the data into 10 cross-validation (CV) folds and applied the lasso, elastic net, pcLasso and the group lasso using the R package **gglasso**. For elastic net and pcLasso, CV was performed over the hyperparameters for both the ℓ_1 and quadratic penalty terms. For the elastic net, CV chose $\alpha = 1$, thus reducing it to the lasso. For pcLasso and group lasso, we used chromosome number as the grouping information.

The top-left panel in Figure 2.5 shows the CV mean-squared error across the path of models for each algorithm. The number of non-zero coefficients for the lasso solution can never exceed the number of observations, resulting in small and less expressive models. In contrast, the path of pcLasso and group lasso solutions cover a wide range of model sizes, with pcLasso achieving slightly better minimum CV error at a smaller model size.

In the remaining panels of Figure 2.5, we plot the number of non-zero features we see for each of the 23 chromosomes across the path of models for each algorithm. As we go down the path of pcLasso solutions, we can see that the copy number changes in certain chromosomes become more important; the lasso solution path does not offer us any such insight. While the group lasso solution path enables us to zoom in on chromosomes of interest, it does not allow us to distinguish between the importance of different features within the same chromosome.

2.6 Computation of the pcLasso solution

Consider first the simpler case of just one group. Let $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ be the singular value decomposition of $\mathbf{X}$ with $\mathbf{D} = \text{diag}(d_1, \dots, d_m)$, where $m = \text{rank}(\mathbf{X})$, and let $\mathbf{A} = \mathbf{V}\mathbf{D}_{d_1^2 - d_j^2}\mathbf{V}^T$. The objective function for pcLasso is

$$J(\boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 + \frac{\theta}{2}\boldsymbol{\beta}^T\mathbf{A}\boldsymbol{\beta}.$$

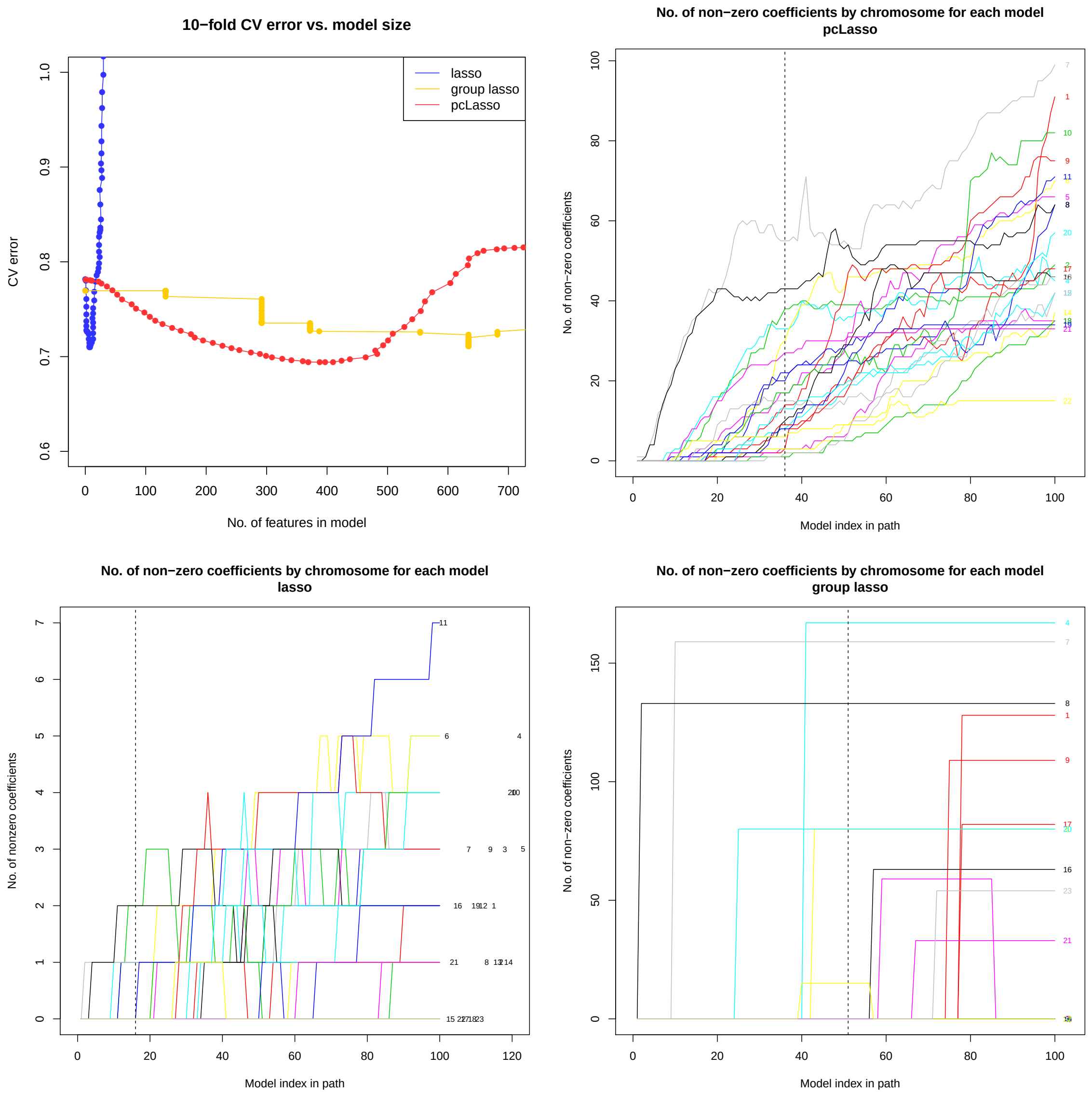

Figure 2.5: *Results for the colorectal array CGH dataset: $n = 39$, $p = 2,031$. The top-left panel shows the CV mean-squared error for the three methods across the solution path. The models along the lasso path are small relative to the total number of features. pcLasso achieves the minimum CV error by a small margin. The rest of the panels plot the number of non-zero features for each chromosome across the solution path for the three methods; vertical lines correspond to the model with smallest CV error.*

Since J is convex in $\boldsymbol{\beta}$ and the non-smooth part of the penalty is separable, we can minimize J by applying coordinate descent. The coordinate-wise update has the form

$$\beta_j \leftarrow \frac{\mathcal{S}\Big(\sum_i x_{ij} r_i^{(j)} - \theta s_j, \lambda\Big)}{\sum_i x_{ij}^2 + \theta A_{jj}},$$

where $r_i^{(j)} = y_i - \sum_{j' \neq j} x_{ij'} \beta_{j'}$ is the partial residual with predictor j removed, $s_j = \sum_\ell A_{j\ell}\beta_\ell - A_{jj}\beta_j$, and $\mathcal{S}$ is the soft-thresholding operator.

These updates can be easily generalized to the general case of K non-overlapping groups (Equation 2.3) and where observations are given different weights (observation i is given weight w_i). Using the notation of Section 2.4, we apply coordinate descent as shown in Algorithm 1. Algorithm 1 is also easily generalized to the logistic regression model for binomial data, using the same Newton-Raphson (iteratively reweighted least squares) framework used in the **glmnet** package (Friedman et al., 2010).

Algorithm 1 *Computation of the pcLasso solution*

Input: Response $\mathbf{y} \in \mathbb{R}^n$, features $\mathbf{X} \in \mathbb{R}^{n \times p}$, K non-overlapping groups of features each of size p_k, fixed value of $\theta \geq 0$. Observation weights $w_i \geq 0$ for $i = 1, \ldots, n$. Grid of hyperparameter values $(\lambda_{max}, \ldots, \lambda_{min})$. Assume that $\mathbf{y}$ and the columns of $\mathbf{X}$ have been centered around their weighted mean.

```
1. Compute the SVD of the columns of X corresponding to each group: $\mathbf{X}_k = \mathbf{U}_k \mathbf{D}_k \mathbf{V}_k^T$ for
   $k = 1, \ldots, K$. Compute $\mathbf{A}^k = \mathbf{V}_k \mathbf{D}_{d_{k1}^2 - d_{kj}^2} \mathbf{V}_k^T$ for each group $k$. Compute $v_j = \sum_i \tilde{w}_i x_{ij}^2$ for
   $j = 1, \ldots, p$, where $\tilde{w}_i = n w_i / \sum_i w_i$. Set $\boldsymbol{\beta} = 0$.
2. For $\lambda \in (\lambda_{max}, \ldots, \lambda_{min})$:
   (a) For $j = 1, 2, \ldots, p, 1, 2, \ldots$ until convergence:
       i. Compute the partial residual $r_i^{(j)} = y_i - \sum_{j' \neq j} x_{ij'} \beta_{j'}$ for $i = 1, \ldots, n$.
       ii. Compute $s_j = \sum_\ell A_{j\ell}^k \beta_\ell - A_{jj}^k \beta_j$, where $k$ is the group containing predictor $j$.
       iii. Update
            $\beta_j \leftarrow \frac{\mathcal{S}\Big(\sum_i \tilde{w}_i x_{ij} r_i^{(j)} - \theta s_j, \lambda\Big)}{v_j + \theta A_{jj}^k}$.
```

We have developed an R package, **pcLasso**, which implements Algorithm 1. For computational efficiency, the coordinate descent step (i.e. Step 2 of Algorithm 1) is implemented as a FORTRAN routine. In a simulation of model-fitting times (see Appendix A.2 for details), the model-fitting time for **pcLasso** is comparable to that of the faster implementation of the group lasso, and is about two orders of magnitude faster than that of the sparse group lasso. We note further that when the cost for the initial SVD is separated out, the speed of **pcLasso** is roughly comparable to that of **glmnet**, which implements the lasso. In practice one can compute the SVD upfront for the given feature matrix, and then use it for subsequent applications of pcLasso. For example, in cross-validation, one could compute just one SVD for the entire feature matrix, rather than one SVD in each fold. Alternatively, one could approximate the pcLasso solution by using a low-rank SVD in place of the full SVD.

2.6.1 pcLasso's grouping effect

In our experiments, we noticed that pcLasso can have a sparse grouping effect, even though it does not use a two-norm penalty. For Figure 2.6, we generated $n = 50$ observations with 750 predictors, arranged in 50 groups of 15 predictors. Within each group, the predictors were either independent (left panel) or had a strong rank-one component (right panel). The response was a linear combination of the features in the first 20 groups with additive Gaussian noise. Within each of these groups, 7 of the features had a coefficient of either 1 or -1 while the remaining 8 features had a coefficient of 0. The figure plots the number of non-zero groups (i.e. groups with at least one zero coefficient) against the number of non-zero coefficients for the lasso, the sparse group lasso (SGL) and pcLasso. The latter two methods were run with different tuning parameters. We see that pcLasso shows no grouping advantage over the lasso in the left panel, but produces moderately strong grouping in the right panel. One might argue that this is reasonable: grouping only occurs when the groups have some strong internal structure. The sparse group lasso, on the other hand, always displays a strong grouping effect.

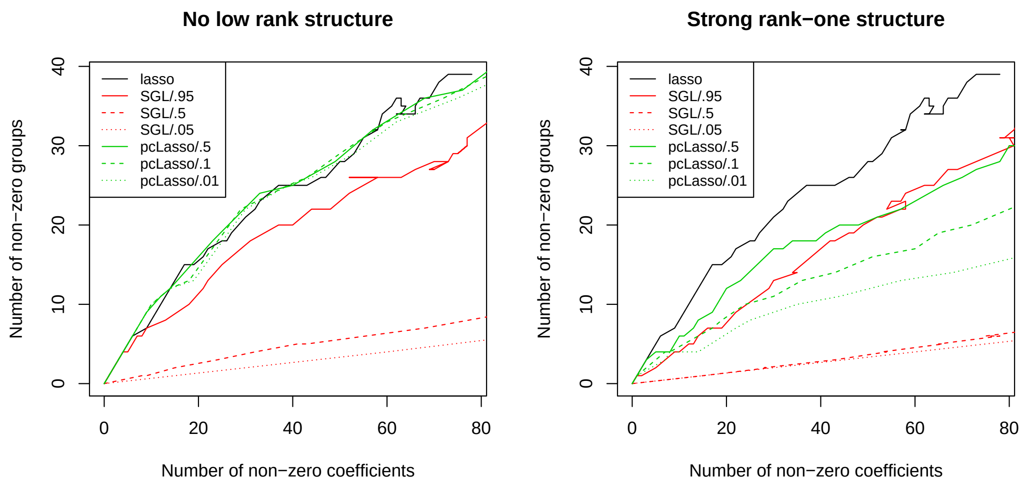

Figure 2.6: *Grouping results for the lasso, sparse group lasso (SGL) and pcLasso. $n = 50$ and $p = 750$, with the predictors coming in 50 groups of size 15 each. Left panel: Predictors were independent. pcLasso has similar grouping behavior as the lasso. Right panel: Within each group, predictors had a strong rank-one component. Here, pcLasso shows a stronger grouping effect than the lasso. SGL has a strong grouping effect in both settings.*

2.6.2 Connection to the group lasso and a generalization of pcLasso

The group lasso and sparse group lasso objective functions are given by

$$\begin{aligned} J_{GL}(\boldsymbol{\beta}) &= \frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda_1 \sum_k \|\boldsymbol{\beta}_k\|_2, \\ J_{SGL}(\boldsymbol{\beta}) &= \frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 + \lambda_1 \sum_k \|\boldsymbol{\beta}_k\|_2, \end{aligned}$$

respectively. In the original group lasso paper (Yuan and Lin, 2006) it was assumed that the $\mathbf{X}_k$, the block of columns in $\mathbf{X}$ corresponding to group k, were orthonormal, i.e. $\mathbf{X}_k^T\mathbf{X}_k = \mathbf{I}$. This makes the computation considerably easier and as pointed out by Simon and Tibshirani (2012), is equivalent to a penalty of the form $\sum_k \|\mathbf{X}_k\boldsymbol{\beta}_k\|_2$. The R package **grpreg** uses this orthogonalization and states that it is essential to its computational performance (Breheny and Huang, 2015). Since $\|\mathbf{X}_k\boldsymbol{\beta}_k\|_2 = \left\|\mathbf{D}_k\mathbf{V}_k^T\boldsymbol{\beta}_k\right\|_2$, this orthogonalization penalizes the top eigenvectors more, in direct contrast to pcLasso which puts *less* penalty on the top eigenvectors. Of course, the group lasso without the ℓ_1 term does not deliver sparsity in the features. We also note that this orthogonalization does not work with the sparse group lasso, as the sparsity among the original features would be lost, and does not work whenever $p_k > n$ for any group k.

One could envision a variation of pcLasso, namely a version of the sparse group lasso that uses an objective function of the form

$$\frac{1}{2}\|\mathbf{y}-\mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 + \lambda_1 \sum_k \left\|\mathbf{R}_k^{1/2}\boldsymbol{\beta}_k\right\|_2,$$

where $\mathbf{R}_k$ is any weight matrix. For example, $\mathbf{R}_k = \mathbf{V}_k\mathbf{D}_{d_{k1}^2-d_{kj}^2}\mathbf{V}_k^T$ as in Equation (2.3) would put less penalty on the higher eigenvectors and induce strong group sparsity. We have not experimented with this yet, as the computation seems challenging due to the presence of ℓ_2 norms.

2.7 Degrees of freedom

Given a vector of response values $\mathbf{y}$ with corresponding fits $\hat{\mathbf{y}}$, Efron (1986) defines the degrees of freedom as

$$\mathrm{df}(\hat{\mathbf{y}}) = \frac{\sum_i \mathrm{Cov}(y_i, \hat{y}_i)}{\sigma^2}.$$

The degrees of freedom measures the flexibility of the fit: the larger the degrees of freedom, the more closely the fit matches the response values. For fits of the form $\hat{\mathbf{y}} = \mathbf{M}\mathbf{y}$, the degrees of freedom is given by $\mathrm{df}(\hat{\mathbf{y}}) = \mathrm{tr}(\mathbf{M})$. For the lasso, the number of non-zero elements in the solution is an unbiased estimate of the degrees of freedom (Zou et al., 2007). Zou (2005) derived an unbiased estimate for the degrees of freedom of the elastic net: if the elastic net solves

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 + \lambda_2 \|\boldsymbol{\beta}\|_2^2,$$

then

$$\widehat{df} = \text{tr}\left\{\mathbf{X}_{\mathcal{A}}(\mathbf{X}_{\mathcal{A}}^T\mathbf{X}_{\mathcal{A}} + \lambda_2 \mathbf{I})^{-1}\mathbf{X}_{\mathcal{A}}^T\right\}$$

is an unbiased estimate for the degrees of freedom, where $\mathcal{A}$ is the active set of the fit.

Using similar techniques as that of Zou (2005), we can derive an unbiased estimate for the degrees of freedom of pcLasso when the number of groups K is equal to one:

Theorem 1 (Degrees of freedom for pcLasso)**.** *Consider the solution to the pcLasso problem*

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 + \theta\boldsymbol{\beta}^T\mathbf{V}\mathbf{D}_{d_1^2-d_j^2}\mathbf{V}^T\boldsymbol{\beta},$$

where $\mathbf{D}_{d_1^2-d_j^2}$ *is a* $p \times p$ *matrix with entries* $d_1^2 - d_j^2$ *(*$m = \mathit{rank}(\mathbf{X}), d_{m+1} = \cdots = d_p = 0$*). Let* $\mathbf{W} = (\mathbf{V}\mathbf{D}_{d_1^2-d_j^2}\mathbf{V}^T)^{1/2}$*. Then an unbiased estimate for the degrees of freedom for pcLasso is*

$$\widehat{df} = \mathit{tr}\left\{\mathbf{X}_{\mathcal{A}}(\mathbf{X}_{\mathcal{A}}^T\mathbf{X}_{\mathcal{A}} + \theta\mathbf{W}_{\mathcal{A}}^T\mathbf{W}_{\mathcal{A}})^{-1}\mathbf{X}_{\mathcal{A}}^T\right\},$$

where $\mathcal{A}$ *is the active set of the fit.*

2.8 A simulation study

We tested the performance of the principal components lasso against other methods in a simulation study. Appendix A.3 contains more figures and details on the data generation processes; we present an illustrative sampling of the results here.

For pcLasso we used the following cross-validation (CV) procedure to select the tuning parameters: For each value of *ratio* $= 0.25, 0.5, 0.75, 0.9, 0.95$ and 1, we ran pcLasso along a path of λ values with `ratio = ratio`. (We found that these values of *ratio* covered a good range of models in practice.) For each run, the value of λ which gave the smallest CV error was selected, i.e. the `lambda.min` value as in the **glmnet** package. We then compared the CV error across the 6 values of *ratio* and selected the value of *ratio* with the smallest CV error and its accompanying λ value. We benchmarked pcLasso against the following methods:

1. The null model, i.e. the mean of the responses in the training data.

2. Elastic net with CV on λ and across $\alpha = 0, 0.2, 0.4, 0.6, 0.8$ and 1.

3. Lasso with CV on λ.

4. Modified principal components (PC) regression, where we perform linear regression with the top k principal components from each group together. (That is, if there are G groups, we perform linear regression on Gk features.) We use CV to choose the rank k.

5. The group lasso with CV on λ.

6. The sparse group lasso with CV on λ and across $\alpha = 0, 0.2, 0.4, 0.6, 0.8$ and 1.

To compare the methods, we considered the mean-squared error (MSE) achieved on 5,000 test points, as well as the true positive rate (TPR) and true negative rate (TNR) for the features selected by the model. Each simulation setting was run 30 times. We ran the simulations for a range of signal-to-noise ratios (SNR) (from 0.5 to 5). (If the response is given by $y = \mu + \epsilon$, where μ and ϵ represent the signal and noise components respectively, then $SNR = Var(\mu)/Var(\epsilon)$.) The noise ϵ was always generated from a mean-zero Gaussian distribution, with variance chosen to attain the desired SNR. We found that the conclusions did not change much across this range. We present results for $SNR = 1$ in the main text; Appendix A.3 includes figures for $SNR = 5$ as well. (For the TPR-TNR plots, a small amount of jitter was added to the points for a better view of the data.)

2.8.1 Setting 1: PC regression and pcLasso's "home court"

In the first setting, $n = 100$ and $p = 200$ with the features coming in 10 groups of 20 predictors each. Each data matrix is generated as an approximately rank-3 matrix: for $k = 1, \ldots, 10$, $\mathbf{X}_k = \sum_{i=1}^3 i \cdot \mathbf{u}_i \mathbf{v}_i^T + \mathbf{E}$, where the entries of the $\mathbf{u}_i$, $\mathbf{v}_i$ and $\mathbf{E}$ are i.i.d. $\mathcal{N}(0,1)$. The response is the sum of the top sparse principal component of the first 5 groups, where the sparse principal component is computed by the **elasticnet** package. Each sparse principal component is constrained to have 3 non-zero loadings each. The performance on test MSE, TPR and TNR is shown in Figure 2.7. In terms of MSE, the modified PC regression procedure performs best, with pcLasso following closely behind. However, PC regression does not provide any selection of features (all features are included in the model) whereas pcLasso sometimes provides a sparse solution.

2.8.2 Setting 2: lasso's "home court"

In this setting, $n = 100$, $p = 50$, with the entries in the design matrix being drawn independently from the $\mathcal{N}(0,1)$ distribution. The response is the sum of the first 5 variables with noise, i.e.

$$\mathbf{y} = \sum_{j=1}^{5} \mathbf{X}_j + \boldsymbol{\epsilon}.$$

The performance on test MSE is shown in Figure 2.8. pcLasso appears to perform on par with the elastic net, the lasso and the sparse group lasso. These 4 methods performed comparably in terms of TPR and TNR, with TPR almost always being 1 even in the $SNR = 1$ case. The modified

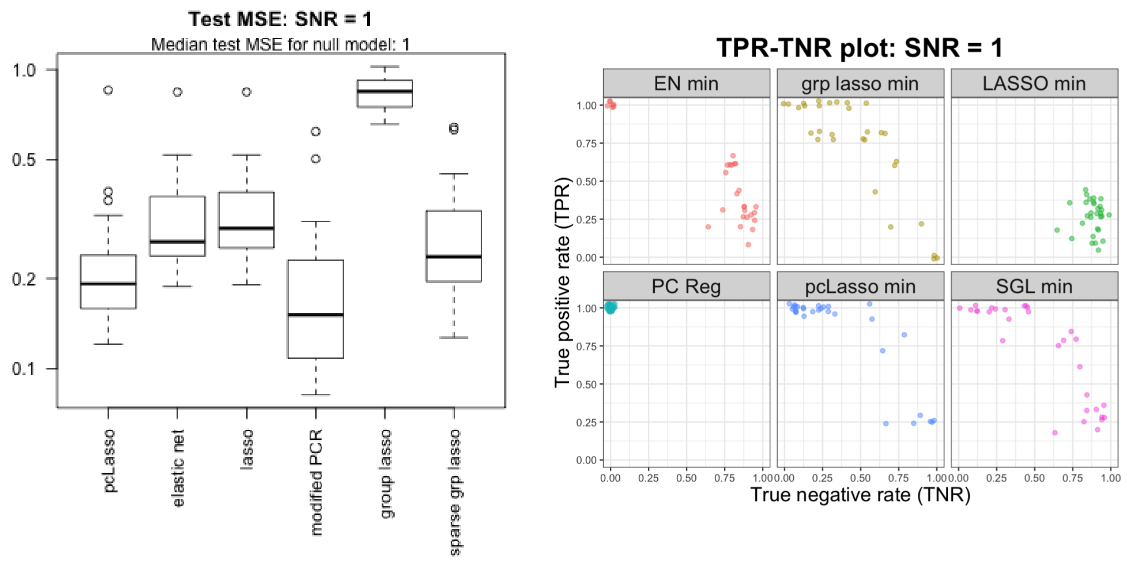

Figure 2.7: *pcLasso's "home court" simulation: $n = 100$, 10 groups of 20 predictors each. Response is the sum of the top sparse PC of the first 5 groups with noise. The figure on the left presents test MSE (note the log scale for the y-axis) while the figure on the right presents TPR and TNR for the features selected by the models (with slight jitter). The modified PC regression procedure performs the best with a slight edge over pcLasso, but does not have any model sparsity.*

PC regression procedure and the group lasso fare much worse because they are unable to capture the sparsity in the true response. pcLasso is able to perform well in this setting because of the tuning parameter *ratio*. The value of *ratio* chosen by CV was almost always $ratio = 1$, which corresponds to the setting where the quadratic penalty is zeroed out, i.e. the lasso.

2.8.3 Setting 3: correlation across groups

Next, we present a setting where the groups are correlated with each other. We have $n = 100$ and $p = 100$ with predictors coming in 5 groups of 20 predictors each. The predictor matrices are constructed to have rank 5 and to share the top principal component. To generate the shared principal component, we draw $d_1^* \sim Unif[0.8, 2]$, and $\mathbf{u}_1^* \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{100})$, normalizing $\mathbf{u}_1^*$ to have norm 1. For each $k = 1, \dots, 5$, we generate $\mathbf{u}_2, \dots, \mathbf{u}_5$ successively by taking draws from $\mathcal{N}(\mathbf{0}, \mathbf{I}_{100})$, orthogonalizing with respect to the $\mathbf{u}_\ell$'s already drawn, and normalizing to have norm 1. We follow the same procedure to get $\mathbf{v}_1, \dots, \mathbf{v}_5$. The kth predictor matrix is then set to be

$$\mathbf{X}_k = d_1^* \mathbf{u}_1^* \mathbf{v}_1 + \sum_{\ell=2}^{5} \frac{\mathbf{u}_\ell \mathbf{v}_\ell}{\sqrt{\ell}}.$$

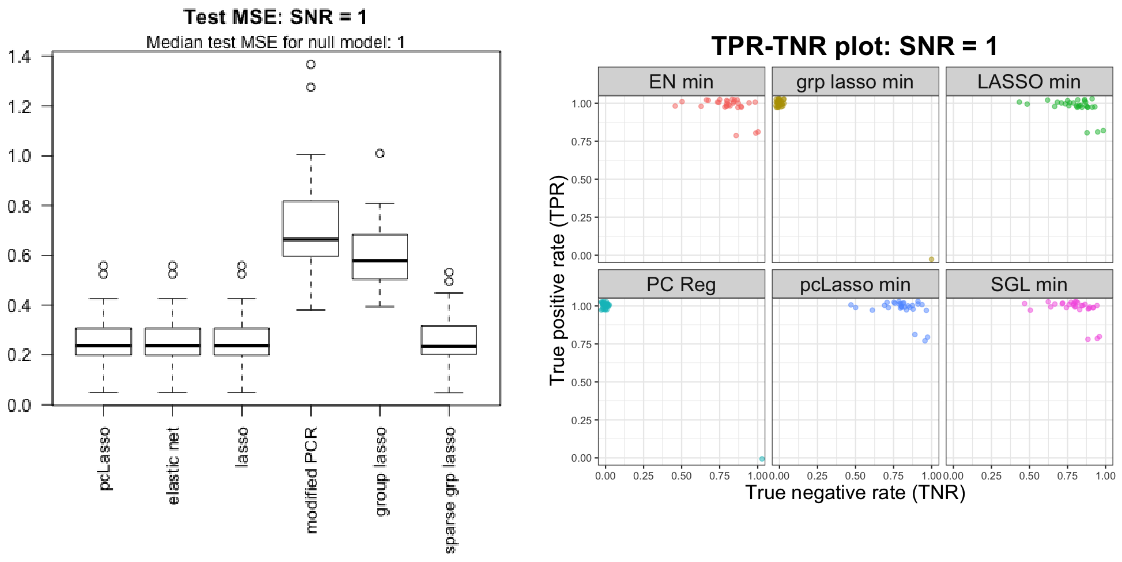

Figure 2.8: *Lasso's "home court" simulation:* $n = 100$*,* $p = 50$*, response is the sum of the first 5 variables. Boxplots are the result of 30 simulation runs. The figure on the left presents test MSE while the figure on the right presents TPR and TNR for the features selected by the models (with slight jitter). pcLasso performs comparably to the elastic net, lasso and sparse group lasso methods, while the modified PC regression procedure and the group lasso perform poorly. Through cross-validation, pcLasso almost always picks ratio* $= 1$ *which corresponds to the lasso, and hence is able to do well.*

The response is this top principal component, $d_1^* \mathbf{u}_1^*$, with additive Gaussian noise. In Figure 2.9, we show how the test MSE varies as a function of the variance explained by the first PC. (We compute the variance explained by the first PC for each group, then take the mean across the 5 groups.)

The left panel of Figure 2.9 presents the test MSE of the various methods. We see that pcLasso outperforms the other methods due to the quadratic penalty that biases the solution in the direction of the top PC of each matrix. The right panel of Figure 2.9 shows the number of times each value of the *ratio* hyperparameter was selected across 30 simulation runs. We see that CV was able to pick small values of *ratio* as the optimal hyperparameter, which corresponds to greater weight on the quadratic penalty term and hence, greater bias toward the leading principal component. (We note that in this setting, the notion of true and null features does not make sense as the signal is represented in each of the groups.)

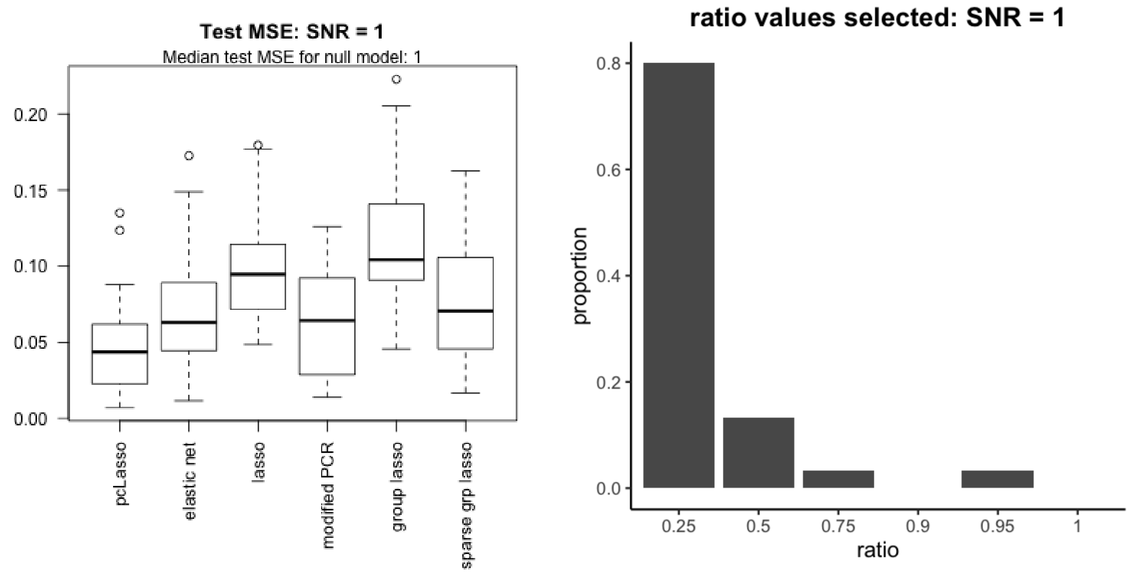

Figure 2.9: *Simulation with correlation across groups: $n = 100$, 5 groups of 20 predictors each. The predictor matrices share the top principal component. The response is the top principal component with additive Gaussian noise. Boxplots are the result of 30 simulation runs. The figure on the left shows the test MSE of various methods; pcLasso performs best. The figure on the right shows the* `ratio` *hyperparameter value chosen by CV across 30 simulation runs. CV tended to choose lower values of* `ratio`*, corresponding to larger quadratic penalties.*

2.8.4 Setting 4: correlation within each group

In this final setting, we examine the impact of within-group correlations on performance. We have $n = 200$ and $p = 50$ with predictors coming in 5 groups of 10 predictors each. For each predictor matrix, the rows are independent draws from the multivariate normal distribution with zero mean and AR(1) covariance, i.e. for $k = 1, \dots, 5$ and $i = 1, \dots, 200$,

$$[(\mathbf{x}_k)_i]^T \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} 1 & \rho & \rho^2 & \dots & \rho^9 \\ \rho & 1 & \rho & \dots & \rho^8 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^9 & \rho^8 & \rho^7 & \dots & 1 \end{pmatrix}\right),$$

where $(\mathbf{x}_k)_i$ denotes the ith row of the kth predictor matrix. The response is the sum of the first principal components of the first two groups with noise: if the SVD of the kth predictor matrix is $\mathbf{X}_k = \mathbf{U}_k\mathbf{D}_k\mathbf{V}_k^T$, the response is

$$\mathbf{y} = (\mathbf{U}_1\mathbf{D}_1)_1 + (\mathbf{U}_2\mathbf{D}_2)_1 + \boldsymbol{\epsilon},$$

where $(\mathbf{U}_k\mathbf{D}_k)_\ell$ represents the ℓth column of the matrix $\mathbf{U}_k\mathbf{D}_k$.

Figure 2.10 compares the test MSE performance for high within-group correlation (correlation parameter $\rho = 0.8$) and low within-group correlation ($\rho = 0.2$). pcLasso is competitive with the best-performing method in both settings. pcLasso's performance gain over the lasso, the elastic net and the group lasso appears to be larger when there is high within-group correlation. This makes intuitive sense as pcLasso utilizes the correlation structure implicitly in its minimization procedure while the other three methods do not. As might be expected, the modified PC regression procedure performs better when the within-group correlation is higher.

2.9 Theoretical properties of the pcLasso solution

In this section, we present some theoretical properties of the principal components lasso when the number of groups K is equal to 1, and compare them with that for the lasso as presented in Chapter 11 of Hastie et al. (2015). In particular, we provide non-asymptotic bounds for ℓ_2 and prediction error for pcLasso which are better than that for the lasso in certain settings. We note that the results in the section can be extended analogously to pcLasso with K non-overlapping groups if the $\mathbf{X}_k$ are orthogonal to each other.

To make the proofs simpler, we consider a slightly different penalty for pcLasso: instead of the second penalty term having $\mathbf{D}_{d_1^2-d_j^2}$ as an $m \times m$ matrix, where $m = \text{rank}(\mathbf{X})$, we have $\mathbf{D}_{d_1^2-d_j^2}$ as a $p \times p$ diagonal matrix with $d_{m+1} = \cdots = d_p = 0$. Let $\mathbf{A} = \mathbf{V}\mathbf{D}_{d_1^2-d_j^2}\mathbf{V}^T$.

As the proofs are rather technical, we state just the results here; proofs can be found in Appendix A.4.

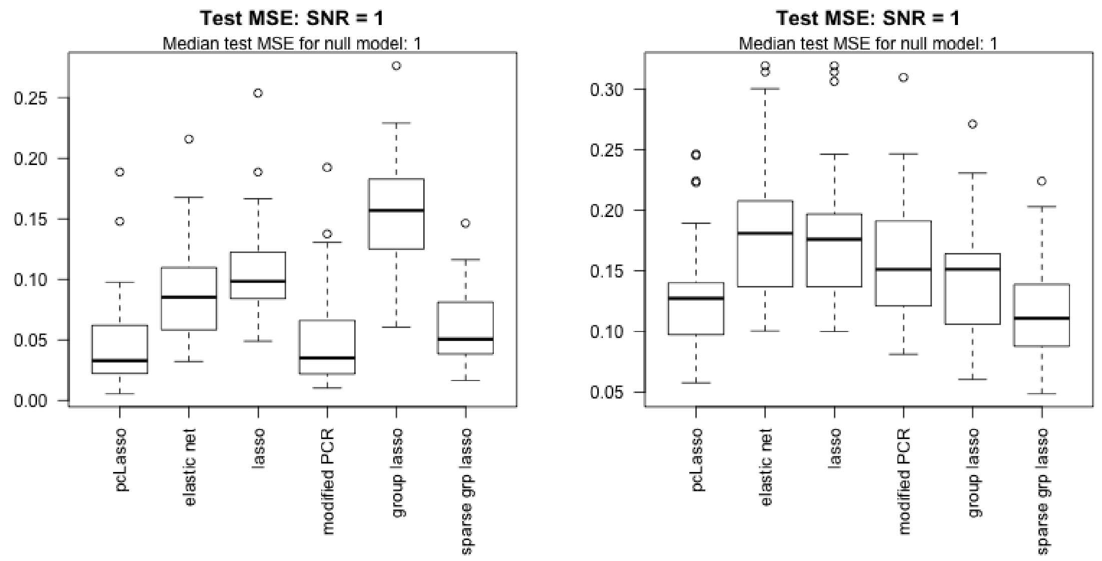

Figure 2.10: *Simulation with within-group correlations: $n = 200$, 5 groups of 10 predictors each. The rows in each predictor matrix are indepedently drawn from the zero-centered multivariate Gaussian distribution with AR(1) correlation matrix. The response is the sum of the top PC of the first two groups with additive Gaussian noise. Boxplots are the result of 30 simulation runs. The figure on the left (right resp.) shows the test MSE when the AR(1) correlation parameter is $\rho = 0.8$ ($\rho = 0.2$ resp.). pcLasso is competitive with the best method in each setting, and its performance gain over the lasso, the elastic net and the group lasso appear to be greater in the high correlation setting.*

2.9.1 Set-up

We assume that the true underlying model is

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta}^* + \mathbf{w},$$

where $\mathbf{X} \in \mathbb{R}^{n \times p}$ is the design matrix, $\mathbf{y}, \mathbf{w} \in \mathbb{R}^n$ are the response and noise vectors respectively, and $\boldsymbol{\beta}^* \in \mathbb{R}^p$ is the true unknown coefficient vector. We denote the support of $\boldsymbol{\beta}^*$ by S. When we solve an optimization problem for $\boldsymbol{\beta}$, we denote the estimate by $\widehat{\boldsymbol{\beta}}$ and the error by $\widehat{\boldsymbol{\nu}} = \widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*$.

In this section, assume that $\theta > 0$ is fixed. (If $\theta = 0$, pcLasso is equivalent to the lasso.) Define

$$\widetilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix}, \quad \widetilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X} \\ \sqrt{n\theta}\sqrt{\mathbf{A}} \end{pmatrix}, \quad \widetilde{\mathbf{w}} = \begin{pmatrix} \mathbf{w} \\ -\sqrt{n\theta}\sqrt{\mathbf{A}}\boldsymbol{\beta}^* \end{pmatrix}.$$

Thus, $\widetilde{\mathbf{y}} = \widetilde{\mathbf{X}}\boldsymbol{\beta}^* + \widetilde{\mathbf{w}}$. The key idea is that with this notation, pcLasso is equivalent to the lasso for the augmented matrices $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{y}}$. Explicitly, the constrained form of pcLasso solves

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \left\| \widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\boldsymbol{\beta} \right\|_2^2 \qquad \text{subject to} \quad \|\boldsymbol{\beta}\|_1 \leq R, \tag{2.4}$$

while the Lagrangian form (Equation (2.1)) solves

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \frac{1}{2n} \left\| \widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\boldsymbol{\beta} \right\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1 . \tag{2.5}$$

(We have changed the fraction in front of the RSS term to $1/(2n)$ so that the results are more directly comparable to those in Hastie et al. (2015).) In the high-dimensional setting, it is standard to impose a *restricted eigenvalue condition* on the design matrix $\mathbf{X}$:

$$\frac{1}{n}\boldsymbol{\nu}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\nu} \geq \gamma \|\boldsymbol{\nu}\|_2^2 \qquad \text{for all nonzero } \boldsymbol{\nu} \in \mathcal{C}, \tag{2.6}$$

with $\gamma > 0$ and $\mathcal{C}$ an appropriately chosen constraint set. We assume additionally that $d_1^2 > n\gamma$. This is not a restrictive assumption: since d_1^2 is the top eigenvalue of $\mathbf{X}^T\mathbf{X}$, we automatically have $d_1^2 \geq n\gamma$. Equality can only happen if $\mathcal{C}$ is a subset of the eigenspace associated with d_1^2.

Since $\mathbf{A}$ is a positive semidefinite matrix, Equation (2.6) holds trivially for the augmented matrix $\widetilde{\mathbf{X}}$ as well:

$$\boldsymbol{\nu}^T\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}}\boldsymbol{\nu} = \boldsymbol{\nu}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\boldsymbol{\nu} \geq n\gamma \|\boldsymbol{\nu}\|_2^2 .$$

The following key lemma shows that we can improve the constant on the RHS of Equation (2.6). This will give us better upper bounds on the rates of convergence for pcLasso.

Lemma 2. *If* $\mathbf{X}$ *satisfies the restricted eigenvalue condition (Equation* (2.6)*), then*

$$\boldsymbol{\nu}^T \widetilde{\mathbf{X}}^T \widetilde{\mathbf{X}} \boldsymbol{\nu} \geq \min\left\{(1-n\theta)n\gamma + n\theta d_1^2, d_1^2\right\} \|\boldsymbol{\nu}\|_2^2 \qquad \text{for all nonzero } \boldsymbol{\nu} \in \mathcal{C}.$$

We note also that the augmented matrix actually satisfies an *unrestricted eigenvalue condition*:

Lemma 3. *For any design matrix* $\mathbf{X}$*, the augmented data matrix* $\widetilde{\mathbf{X}}$ *satisfies*

$$\boldsymbol{\nu}^T \widetilde{\mathbf{X}}^T \widetilde{\mathbf{X}} \boldsymbol{\nu} \geq \min(n\theta, 1)\, d_1^2 \|\boldsymbol{\nu}\|_2^2 \qquad \text{for all } \boldsymbol{\nu} \in \mathbb{R}^p.$$

This enables us to obtain a bound on ℓ_2 error without requiring a restricted eigenvalue condition.

2.9.2 Bounds on ℓ_2 error

The following theorem establishes a bound for ℓ_2 error of the constrained form of pcLasso:

Theorem 4. *Suppose that* $\mathbf{X}$ *satisfies the restricted eigenvalue bound (Equation* (2.6)*) with parameter* $\gamma > 0$ *over the set* $\{\boldsymbol{\nu} \in \mathbb{R}^p : \|\boldsymbol{\nu}_{S^c}\|_1 \leq \|\boldsymbol{\nu}_S\|_1\}$*. Then any estimate* $\widehat{\boldsymbol{\beta}}$ *based on the constrained pcLasso (Equation* (2.4)*) with* $R \leq \|\boldsymbol{\beta}^*\|_1$ *satisfies the bound*

$$\left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\right\|_2 \leq \frac{4\sqrt{|S|}\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty}{\min\left\{(1-n\theta)n\gamma + n\theta d_1^2, d_1^2\right\}}. \tag{2.7}$$

In particular, if $\boldsymbol{\beta}^*$ *is aligned with the first principal component of* $\mathbf{X}$*, then*

$$\left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\right\|_2 \leq \frac{4\sqrt{|S|}\left\|\mathbf{X}^T\mathbf{w}\right\|_\infty}{\min\left\{(1-n\theta)n\gamma + n\theta d_1^2, d_1^2\right\}}, \tag{2.8}$$

which is a ***better*** *upper bound on the rate of convergence than that for the lasso.*

What is interesting is that we can actually obtain a similar (weaker) bound without the restricted eigenvalue condition; the lasso does not have such a bound.

Theorem 5. *For any design matrix* $\mathbf{X}$*, any estimate* $\widehat{\boldsymbol{\beta}}$ *based on the constrained pcLasso (Equation* (2.4)*) with* $R \leq \|\boldsymbol{\beta}^*\|_1$ *satisfies the bound*

$$\left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\right\|_2 \leq \frac{4\sqrt{|S|}\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty}{\min(n\theta, 1)\, d_1^2}.$$

We could use the theorems above to guide the choice of the hyperparameter θ. When $\boldsymbol{\beta}^*$ is aligned with the first PC of $\mathbf{X}$, it is clear that $\theta = 1/n$ gives the tightest possible bound. When $\boldsymbol{\beta}^*$ is not aligned with the first PC, then the best value of the bound depends on unknown quantities, i.e. the true coefficient vector $\boldsymbol{\beta}^*$ and the noise component of the response $\mathbf{w}$ (and, for Theorem 4, the restricted eigenvalue parameter γ). We explore this further in Appendix A.5.

We can obtain similar results for the Lagrangian form of pcLasso:

Theorem 6. *Suppose that* $\mathbf{X}$ *satisfies the restricted eigenvalue bound (Equation* (2.6)*) with parameter* $\gamma > 0$ *over the set* $\{\boldsymbol{\nu} \in \mathbb{R}^p : \|\boldsymbol{\nu}_{S^c}\|_1 \le 3\|\boldsymbol{\nu}_S\|_1\}$*. Then any estimate* $\widehat{\boldsymbol{\beta}}$ *based on the Lagrangian form of pcLasso (Equation* (2.5)*) with* $\lambda \ge 2\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty / n > 0$ *satisfies the bound*

$$\left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\right\|_2 \le \frac{3\lambda\sqrt{|S|}}{\min\left\{(1-n\theta)n\gamma + n\theta d_1^2, d_1^2\right\}/n}. \tag{2.9}$$

If we remove the restricted eigenvalue condition on $\mathbf{X}$*, then the bound* (2.9) *holds but with the denominator of the RHS being* $\min(n\theta, 1)\, d_1^2/n$ *instead.*

2.9.3 Bounds on prediction error

Like the lasso, the principal components lasso has a "slow rate" convergence for prediction error:

Theorem 7. *For any design matrix* $\mathbf{X}$*, consider the Lagrangian form of pcLasso (Equation* (2.5)*) with* $\lambda \ge 2\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty / n$*. We have the following bound on prediction error:*

$$\frac{1}{n}\left\|\mathbf{X}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)\right\|_2^2 \le 12\lambda\|\boldsymbol{\beta}^*\|_1.$$

By using Lemma 3, we can get a smaller bound for the RHS, although the expression is much more complicated:

Theorem 8. *For any design matrix* $\mathbf{X}$*, consider the Lagrangian form of pcLasso (Equation* (2.5)*) with* $\lambda \ge 2\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty / n$*. Let* $\kappa = \min(n\theta, 1)d_1^2$*. We have the following bound on prediction error:*

$$\frac{1}{n}\left\|\mathbf{X}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)\right\|_2^2 \le \frac{3\lambda(-\lambda p + \sqrt{\lambda^2p^2 + 32\lambda\|\boldsymbol{\beta}^*\|_1\kappa p})}{4\kappa}.$$

The preceding two theorems hold for all design matrices $\mathbf{X}$. If we are willing to assume that $\mathbf{X}$ satisfies the restricted eigenvalue condition, we recover the same upper bound for the "fast rate" convergence for prediction error as that for the lasso.

Theorem 9. *Suppose that* $\mathbf{X}$ *satisfies the restricted eigenvalue bound (Equation* (2.6)*) with parameter* $\gamma > 0$ *over the set* $\{\boldsymbol{\nu} \in \mathbb{R}^p : \|\boldsymbol{\nu}_{S^c}\|_1 \le 3\|\boldsymbol{\nu}_S\|_1\}$*. For the Lagrangian form of pcLasso with* $\lambda \ge 2\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty / n$*, we have*

$$\frac{1}{n}\left\|\mathbf{X}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)\right\|_2^2 \le \frac{9|S|\lambda^2}{\min\left\{(1-n\theta)n\gamma + n\theta d_1^2, d_1^2\right\}/n}.$$

2.10 Discussion

Principal components lasso (pcLasso) is a new method for supervised learning that exploits the correlation and group structure of predictors to potentially improve prediction performance. It combines the power of principal components regression with the sparsity of the lasso. There are several interesting avenues for further research:

- *Efficient screening rules.* To speed up the computation of the lasso solution, **glmnet** uses strong rules (Tibshirani et al., 2012) to discard predictors which are likely to have a coefficient of zero. These strong rules can greatly reduce the number of variables entering the optimization, thus speeding up computation. Using similar techniques, we can derive strong rules for the principal components lasso to improve computational efficiency. These rules are provided in Appendix A.6, and merit further investigation.

- *Use of different kinds of correlation or grouping information.* Rather than use the covariance of the observed covariates in a group, one could use other kinds of external information to form this covariance, for example gene or protein pathways. Alternatively, one could use unsupervised clustering to generate the feature groups.

- *Optimality of the quadratic penalty term.* Considering the general class of penalty functions of the form $\boldsymbol{\beta}^T \mathbf{V}\mathbf{Z}\mathbf{V}^T \boldsymbol{\beta}$, one could ask if our choice of $\mathbf{Z} = \mathbf{D}_{d_1^2 - d_j^2}$ is "optimal" in any sense. One could also look for $\mathbf{Z}$ which satisfy certain notions of optimality.

- *Better estimates of the eigenvalues and eigenvectors.* The quadratic penalty involves the sample eigenvalues $\mathbf{D}_k$ and eigenvectors $\mathbf{V}_k$. It is known that in the high-dimensional setting ($p \gg n$), these sample quantities can be poor estimates of their population counterparts (see e.g. Baik and Silverstein (2006), Johnstone and Lu (2009)). Several methods have been developed to obtain better estimates, typically with the assumption of sparsity for the eigenvectors. (See Section 2.2 for references on estimating sparse principal components; for estimation of eigenvalues see e.g. Yata and Aoshima (2012).) The performance of pcLasso could be improved if the sample $\mathbf{D}_k$'s and $\mathbf{V}_k$'s were replaced with better estimates.

  It should be noted, though, that the introduction of such estimates are often accompanied with additional hyperparameters which the user needs to choose, either using prior knowledge or a method such as cross-validation. This makes the method more difficult to use in practice. Additionally, if sparse versions of the $\mathbf{V}_k$'s are estimated, it might be more profitable to fit a model directly on the $\mathbf{X}_k\mathbf{V}_k$'s instead of inserting them into the quadratic penalty and running the pcLasso algorithm.

- *Extension to elastic net penalty.* It is easy to extend pcLasso to incorporate the full elastic net penalty in the pcLasso penalty. That is, instead of minimizing (2.3), we minimize

$$J(\boldsymbol{\beta}) = \frac{1}{2}\left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|_2^2 + \lambda\left(\alpha\left\|\boldsymbol{\beta}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}\right\|_2^2\right) + \frac{\theta}{2}\sum_k \boldsymbol{\beta}_k^T\left(\mathbf{V}_k\mathbf{D}_{d_{k1}^2 - d_{kj}^2}\mathbf{V}_k^T\right)\boldsymbol{\beta}_k,$$

where $\alpha \geq 0$ is an additional tuning parameter. To minimize this, we can reuse Algorithm 1, replacing the update in Step 2(a)iii with

$$\beta_j \leftarrow \frac{\mathcal{S}\left(\sum_i \tilde{w}_i x_{ij} r_i^{(j)} - \theta s_j, \lambda\alpha\right)}{v_j + \theta A_{jj}^k + \lambda(1-\alpha)}.$$

An R language package **pcLasso** implementing pcLasso is available on the CRAN repository.

Chapter 3

Feature-Weighted Elastic Net

3.1 Introduction

As mentioned in the previous chapter, in some supervised learning settings we have external information about the features themselves. For example, in genomics, we know that each gene belongs to one or more genetic pathways, and we may expect genes in the same pathway to have correlated effects on the response. Methods which leverage such information are likely to perform better prediction and inference than methods which ignore it. However, many popular supervised learning methods, including the elastic net Zou and Hastie (2005), do not use such information in the model-fitting process. Other methods, such as principal components lasso introduced in the previous chapter, work only with specific types of feature information (e.g. feature grouping information in the case of pcLasso).

In this chapter, we develop a general framework for organizing such feature information and propose a variant of the elastic net which uses this information in model-fitting. We only assume that the information we have for each feature is quantitative, allowing us to think of each source as a "feature" of the features. For example, in the genomics setting, the kth source of information could be the indicator variable for whether the jth feature belongs to the kth genetic pathway.

We organize these "features of features" into an auxiliary matrix $\mathbf{Z} \in \mathbb{R}^{p \times K}$, where p is the number of features and K is the number of sources of feature information. Let $\mathbf{z}_j \in \mathbb{R}^K$ denote the jth row of $\mathbf{Z}$ as a column vector. To make use of the information in $\mathbf{Z}$, we propose assigning each feature a *score* $\mathbf{z}_j^T \boldsymbol{\theta}$, i.e. a linear combination of its "features of features". We then use these scores to influence the weight given to each feature in the model-fitting procedure. Concretely, we give each feature a different penalty weight in the elastic net objective function based on its score:

$$J_{\lambda,\alpha,\boldsymbol{\theta}}(\boldsymbol{\beta}) = \frac{1}{2} \left\| \mathbf{y} - \mathbf{X}\boldsymbol{\beta} \right\|_2^2 + \lambda \sum_{j=1}^{p} w_j(\boldsymbol{\theta}) \left[\alpha |\beta_j| + \frac{1-\alpha}{2} \beta_j^2 \right],$$

where $w_j(\boldsymbol{\theta}) = f\left(\mathbf{z}_j^T\boldsymbol{\theta}\right)$ for some function f. (Here, $\lambda \geq 0$ and $\alpha \in [0,1]$ are the tuning parameters associated with the elastic net. As in the previous chapter, we assume that $\mathbf{y}$ and the columns of $\mathbf{X}$ are mean-centerd at zero so that we can avoid fitting an intercept.) The variable $\boldsymbol{\theta}$ is a hyperparameter in $\mathbb{R}^K$ which the algorithm needs to select. In the final model, $\mathbf{z}_j^T\boldsymbol{\theta}$ can be thought of as an indication of how influential feature j is on the response, while θ_k represents how important the kth source of feature information is in identifying which features are important for the prediction problem.

The rest of this chapter is organized as follows. In Section 3.2, we survey past work on incorporating "features of features" in supervised learning. In Section 3.3, we propose a method, the *feature-weighted elastic net* ("fwelnet"), which uses the scores in model-fitting. We present connections to the group lasso and Bayesian estimation in Section 3.4, and illustrate fwelnet's performance on simulated data in Section 3.5 and on a real data example in Section 3.6. In Section 3.7, we show how fwelnet can be used in multi-task learning. We end with a discussion and ideas for future work. The appendix contains further details and proofs.

3.2 Related work

The idea of assigning different penalty weights for features in the lasso or elastic net objective is not new. The adaptive lasso (Zou, 2006) assigns feature j a penalty weight $w_j = 1/|\hat{\beta}_j^{OLS}|^\gamma$, where $\hat{\beta}_j^{OLS}$ is the estimated ordinary least squares (OLS) coefficent for feature j and $\gamma > 0$ is some hyperparameter. However, the OLS solution only depends on $\mathbf{X}$ and $\mathbf{y}$ and does not incorporate any external feature information. In the work closest to ours, Bergersen et al. (2011) propose using weights $w_j = 1/|\eta_j(\mathbf{y}, \mathbf{X}, \mathbf{Z})|^q$, where η_j is some function (possibly varying for j) and q is a hyperparameter controlling the shape of the weight function. While the authors present two ideas for what the η_j's could be, they do not give general guidance on how to choose these functions which could drastically influence the model-fitting algorithm.

There is a correspondence between penalized regression estimates and Bayesian maximum a posteriori (MAP) estimates with a particular prior for the coefficients. Within this Bayesian framework, some methods propose using external feature information to guide the choice of prior. For example, van de Wiel et al. (2016) take an empirical Bayes approach to estimate the prior for ridge regression, while Velten and Huber (2018) use variational Bayes to do so for general convex penalties.

Most previous approaches for penalized regression with external information on the features only work with specific types of such information. Several methods has been developed to make use of *feature grouping information*. Popular methods include the group lasso (Yuan and Lin, 2006) and the overlap group lasso (Obozinski et al., 2009). IPF-LASSO (integrative lasso with penalty factors) (Boulesteix et al., 2017) gives each group its own penalty parameter, to be chosen via cross-validation. Tai and Pan (2007) modify the penalized partial least squares (PLS) and nearest shrunken centroids methods to have group-specific penalties.

Other methods have been developed to incorporate "network-like" or feature similarity information. The fused lasso (Tibshirani et al., 2005) adds an ℓ_1 penalty on the successive differences of the coefficients to impose smoothness on the coefficient profile. Structured elastic net (Slawski et al., 2010) generalizes the fused lasso by replacing the ℓ_2-squared penalty in elastic net with $\boldsymbol{\beta}^T \boldsymbol{\Lambda} \boldsymbol{\beta}$, where $\boldsymbol{\Lambda}$ is a symmetric, positive semi-definite matrix chosen to reflect some a priori known structure between the features. Li and Li (2008) is a special case of structured elastic net, where $\boldsymbol{\Lambda}$ is equal to the normalized Laplacian matrix of the feature network graph. Mollaysa et al. (2017) use the feature information matrix $\mathbf{Z}$ to compute a feature similarity matrix, which is used to construct a penalty term in the loss criterion. The regularizer encourages the model to give the same output as long as the total contribution of similar features is the same. We note that this approach implicitly assumes that the sources of feature information are equally relevant, which may or may not be the case.

It is not clear how most of the prior work can be generalized to generic sources of feature information. Our method has the distinction of being able to work directly with real-valued feature information and to integrate multiple sources of feature information. While van de Wiel et al. (2016) claim to be able to handle binary, nominal, ordinal and continuous feature information, the method actually ranks and groups features based on such information and only uses this grouping information in the estimation of the group-specific penalties. Nevertheless, it is able to incorporate more than one source of feature information.

3.3 Feature-weighted elastic net ("fwelnet")

One way to utilize the scores $\mathbf{z}_j^T \boldsymbol{\theta}$ in model-fitting is to give each feature a different penalty weight in the elastic net objective based on its score:

$$J_{\lambda,\alpha,\boldsymbol{\theta}}(\boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^p w_j(\boldsymbol{\theta}) \left[\alpha|\beta_j| + \frac{1-\alpha}{2}\beta_j^2 \right],$$

where $w_j(\boldsymbol{\theta}) = f\left(\mathbf{z}_j^T \boldsymbol{\theta}\right)$ for some function f. Our proposed method, which we call the *feature-weighted elastic net* ("fwelnet"), specifies f:

$$w_j(\boldsymbol{\theta}) = \frac{\sum_{\ell=1}^p \exp\left(\mathbf{z}_\ell^T \boldsymbol{\theta}\right)}{p \exp\left(\mathbf{z}_j^T \boldsymbol{\theta}\right)}. \tag{3.1}$$

The fwelnet algorithm seeks the minimizer of this objective function over $\boldsymbol{\beta}$:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \; \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^p w_j(\boldsymbol{\theta}) \left[\alpha|\beta_j| + \frac{1-\alpha}{2}\beta_j^2 \right]. \tag{3.2}$$

There are a number of reasons for this choice of penalty factors. First, when $\boldsymbol{\theta} = 0$, we have $w_j(\boldsymbol{\theta}) = 1$ for all j, reducing fwelnet to the original elastic net. Second, $w_j(\boldsymbol{\theta}) \geq 1/p$ for all j and $\boldsymbol{\theta}$, ensuring that we do not end up with features having negligible penalty. This allows the fwelnet solution to have a wider range of sparsity across λ hyperparameter values. Third, this formulation provides theoretical connections which we detail in Section 3.4. Finally, a feature's score has a natural interpretation: if $\mathbf{z}_j^T \boldsymbol{\theta}$ is relatively large, then w_j is relatively small, meaning that feature j is more important for the response and hence should have smaller penalty.

We illustrate the last property via a simulated example. In this simulation, we have $n = 200$ observations and $p = 100$ features which come in groups of 10. The response is a linear combination of the first two groups with additive Gaussian noise. The coefficient for the first group is 4 while the coefficient for the second group is -2 so that the first group exhibits stronger correlation to the response compared to the second group. The "features of features" matrix $\mathbf{Z} \in \mathbb{R}^{100 \times 10}$ is grouping information, i.e. $z_{jk} = 1\{\text{feature } j \text{ belongs to group } k\}$. Figure 3.1 shows the penalty factors w_j that fwelnet assigns the features. (The hyperparameter θ was determined using Algorithm 2 described in Section 3.3.1.) As one would expect, the features in the first group have the smallest penalty factor followed by features in the second group. In contrast, the original elastic net algorithm would assign penalty factors $w_j = 1$ for all j.

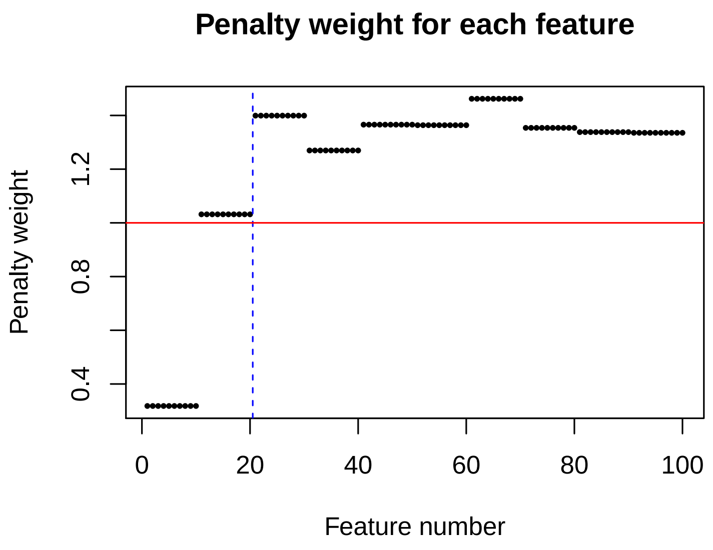

Figure 3.1: *Penalty factors which fwelnet assigns to each feature. $n = 200$, $p = 100$ with features in groups of size 10. The response is a noisy linear combination of the first two groups, with signal in the first group being stronger than that in the second. As expected, fwelnet's penalty weights for the true features (left of blue dotted line) are lower than that for null features. The elastic net would assign all features a penalty factor of 1 (horizontal red line).*

3.3.1 Computing the fwelnet solution

For given values of λ, α and $\boldsymbol{\theta}$, it is easy to solve (3.2): the objective function is convex in $\boldsymbol{\beta}$, and $\hat{\boldsymbol{\beta}}$ can be found efficiently using algorithms such as coordinate descent. However, to deploy fwelnet in practice we need to determine the hyperparameter values $\hat{\lambda} \in \mathbb{R}$, $\hat{\alpha} \in \mathbb{R}$ and $\hat{\boldsymbol{\theta}} \in \mathbb{R}^K$ that give good performance. When K, the number of sources of feature information, is small, one could run the algorithm for a grid of $\boldsymbol{\theta}$ values, then pick the value which gives the smallest cross-validated loss. Unfortunately, this approach is computationally infeasible for even moderate values of K.

To avoid this computational bottleneck, we propose solving the minimization problem:

$$\begin{aligned}
\underset{\boldsymbol{\beta}(\lambda_i), \boldsymbol{\theta}(\lambda_i)}{\text{minimize}} \quad & \frac{1}{m} \sum_{i=1}^m \left[\frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}(\lambda_i)\|_2^2 \right. \\
& \left. \qquad + \lambda_i \sum_{j=1}^p w_j(\boldsymbol{\theta}(\lambda_i)) \left(\alpha |\beta_j(\lambda_i)| + \frac{1-\alpha}{2} \beta_j(\lambda_i)^2 \right) \right] \\
\text{subject to} \quad & \boldsymbol{\theta}(\lambda_1) = \cdots = \boldsymbol{\theta}(\lambda_m),
\end{aligned}$$

where $\lambda_1 > \lambda_2 > \cdots > \lambda_m$ is a path of λ hyperparameter values. Here, we think of $\boldsymbol{\theta}$ as an argument of the objective function J, and we view minimizing J as a joint function of $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ as a heuristic to obtain a good value of $\boldsymbol{\theta}$. However, to maintain the interpretation of θ as a hyperparameter, *we force $\hat{\boldsymbol{\theta}}$ to be the same across all λ values.* We propose an alternating minimization (Algorithm 2) to solve this minimization problem. Step 3(c) finds the optimum solution for $\boldsymbol{\beta}(\lambda_1), \ldots, \boldsymbol{\beta}(\lambda_m)$ for given values of $\boldsymbol{\theta}(\lambda_1), \ldots, \boldsymbol{\theta}(\lambda_m)$, while Steps 3(a) and 3(b) does gradient descent for $\boldsymbol{\theta}(\lambda_1), \ldots, \boldsymbol{\theta}(\lambda_m)$ projected to the constraint set.

Algorithm 2 *Fwelnet algorithm*

```
1. Select a value of $\alpha \in [0,1]$ and a sequence of $\lambda$ values $\lambda_1 > \ldots > \lambda_m$.
2. For $i = 1, \ldots, m$, initialize $\boldsymbol{\beta}^{(0)}(\lambda_i)$ at the elastic net solution for the corresponding $\lambda_i$. Initialize $\boldsymbol{\theta}^{(0)} = \mathbf{0}$.
3. For $k = 0, 1, \ldots$ until convergence:
    (a) Set $\Delta\boldsymbol{\theta}$ to be the component-wise mean of $\left.\frac{\partial J_{\lambda_i,\alpha}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k)}, \boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}}$ over $i = 1, \ldots, m$.
    (b) Set $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\Delta\boldsymbol{\theta}$, where $\eta$ is the step size computed via backtracking line search to ensure that the mean of $J_{\lambda_i,\alpha}\left(\boldsymbol{\beta}^{(k)}, \boldsymbol{\theta}^{(k+1)}\right)$ over $i = 1, \ldots, m$ is less than that for $J_{\lambda_i,\alpha}\left(\boldsymbol{\beta}^{(k)}, \boldsymbol{\theta}^{(k)}\right)$.
    (c) For $i = 1, \ldots, m$, set $\boldsymbol{\beta}^{(k+1)}(\lambda_i)$ = elastic net solution for $\lambda_i$ where the penalty factor for feature $j$ is $w_j(\boldsymbol{\theta}^{(k+1)})$.
```

Because of the backtracking line search in Step 3(b) and the fact that Step 3(c) solves a convex problem, Algorithm 2 is guaranteed to converge, albeit to a stationary point. However, because Step 2 initializes $\hat{\boldsymbol{\beta}}(\lambda_i)$ at the elastic net coefficients, we usually end up with a good solution. In our simulations, convergence was almost always reached within 20 iterations, and often one to three passes gave a sufficiently good solution.

Remark: We also considered an approach where $\boldsymbol{\theta}$ was not constrained to be the same across λ values. While conceptually straightforward, the algorithm was computationally slow and did not perform as well as Algorithm 2 in prediction. A sketch of this approach is in Appendix B.1.

We have developed an R package, **fwelnet**, which implements Algorithm 2. Step 3(c) of Algorithm 2 can be done easily by using the `glmnet` function in the **glmnet** R package and specifying the `penalty.factor` argument. In practice, we use the sequence $\lambda_1 > \cdots > \lambda_m$ provided by **glmnet**'s implementation of the elastic net as this range of λ values covers a sufficiently wide range of models. (In our package, we allow the user to replace the component-wise mean with the component-wise median in Step 3(a), and to replace the mean with the median in Step 3(b). We find these options do not change performance much when the default sequence is used, so we recommend that users stick with the defaults.)

3.3.2 Extending fwelnet to generalized linear models (GLMs)

It is easy to extend the elastic net to generalized linear models (GLMs) by replacing the RSS term with the negative log-likelihood of the data:

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \underset{\beta_0, \boldsymbol{\beta}}{\operatorname{argmin}} \; \sum_{i=1}^n \ell\left(y_i, \beta_0 + \sum_{j=1}^p x_{ij}\beta_j \right) + \lambda \sum_{j=1}^p \left[\alpha|\beta_j| + \frac{1-\alpha}{2}\beta_j^2 \right], \tag{3.3}$$

where $\ell(y_i, \beta_0 + \sum_j x_{ij}\beta_j)$ is the negative log-likelihood contribution of observation i. Fwelnet can be extended to GLMs in a similar fashion:

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\theta}}) = \underset{\beta_0, \boldsymbol{\beta}, \boldsymbol{\theta}}{\operatorname{argmin}} \; \sum_{i=1}^n \ell\left(y_i, \beta_0 + \sum_{j=1}^p x_{ij}\beta_j \right) + \lambda \sum_{j=1}^p w_j(\boldsymbol{\theta}) \left[\alpha|\beta_j| + \frac{1-\alpha}{2}\beta_j^2 \right], \tag{3.4}$$

with $w_j(\boldsymbol{\theta})$ as defined in (3.1). Algorithm 2 can be used as-is to solve (3.4). Because $\boldsymbol{\theta}$ only appears in the penalty term, this extension can be implemented easily. We can rely on the `glmnet` function for Steps 2 and 3(c), Step 3(a) is the same as before, and Step 3(b) simply requires a function that allows us to compute ℓ.

3.4 Theoretical connections

3.4.1 Connection to the group lasso

One common setting where "features of features" arise naturally is when the features come in non-overlapping groups. Assume that the features in $\mathbf{X}$ come in K non-overlapping groups. Let p_k denote the number of features in group k, and let $\boldsymbol{\beta}^{(k)}$ denote the subvector of $\boldsymbol{\beta}$ which belongs to group k. Assume also that $\mathbf{y}$ and the columns of $\mathbf{X}$ are centered so that $\hat{\beta}_0 = 0$. In this setting, Yuan and Lin (2006) introduced the group lasso estimate as the solution to the optimization problem

$$\underset{\boldsymbol{\beta}}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{k=1}^K \left\|\boldsymbol{\beta}^{(k)}\right\|_2.$$

The ℓ_2 penalty on features at the group level ensures that features belonging to the same group are either all included in the model or all excluded from it. Often, the penalty given to group k is modified by a factor of $\sqrt{p_k}$ to take into account varying group sizes:

$$\hat{\boldsymbol{\beta}}_{gl,2}(\lambda) = \underset{\boldsymbol{\beta}}{\text{argmin}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{k=1}^K \sqrt{p_k} \left\|\boldsymbol{\beta}^{(k)}\right\|_2.$$

Theorem 10 below establishes a connection between fwelnet and the group lasso.

Theorem 10. *If the "features of features" matrix $\mathbf{Z} \in \mathbb{R}^{p \times K}$ is given by $z_{jk} = 1\{$feature $j \in$ group $k\}$, then minimizing the fwelnet objective function (3.2) jointly over β_0, $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ reduces to*

$$\begin{aligned} &\underset{\beta}{\textit{argmin}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda' \sum_{k=1}^K \sqrt{p_k \left[\alpha \left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right]} \\ &= \begin{cases} \underset{\beta}{\textit{argmin}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda' \sum_{k=1}^K \sqrt{p_k}\left\|\boldsymbol{\beta}^{(k)}\right\|_2 & \textit{if } \alpha = 0, \\ \underset{\beta}{\textit{argmin}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda' \left(\sum_{k=1}^K \sqrt{p_k \left\|\boldsymbol{\beta}^{(k)}\right\|_1}\right)^2 & \textit{if } \alpha = 1, \end{cases} \end{aligned}$$

for some $\lambda' \geq 0$.

We recognize the $\alpha = 0$ case as minimizing the residual sum of squares (RSS) and the group lasso penalty, while the $\alpha = 1$ case is minimizing the RSS and the ℓ_1 version of the group lasso penalty. The proof of Theorem 10 can be found in Appendix B.2.

3.4.2 Connection to Bayesian estimation

Regularized estimators can often be thought of as the Bayes posterior mode for a given prior distribution. For example, it is well-known that if the prior and likelihood are given by

$$\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}),$$
$$\mathbf{y} \mid \mathbf{X}, \boldsymbol{\beta} \sim \mathcal{N}\left(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}\right),$$

for some $\tau^2, \sigma^2 > 0$, then the posterior distribution for $\boldsymbol{\beta}$ is minimized at the ridge regression solution for $\lambda = \sigma^2/(2\tau^2)$. If feature information is available, a better prior might be one where the β_j are exchangeable conditional on the $\mathbf{z}_j$'s, i.e. $\beta_j \overset{i.i.d.}{\sim} G(\cdot \mid \mathbf{z}_j)$ for some prior distribution G. One possible choice is

$$\beta_j \overset{ind.}{\sim} \mathcal{N}\left(0, v_j^2 \tau^2\right), \qquad v_j^2 = \frac{p \exp\left(\mathbf{z}_j^T \boldsymbol{\theta}\right)}{\sum_{\ell=1}^p \exp\left(\mathbf{z}_\ell^T \boldsymbol{\theta}\right)}, \tag{3.5}$$

for some fixed $\boldsymbol{\theta}$. With this prior, τ^2 is the average prior variance for the β_j's, while the v_j^2's modulate the prior variance for each coefficient based on its feature information. The expression for the v_j^2's is simply softmax applied to the $\mathbf{z}_j^T \boldsymbol{\theta}$'s (scaled by p), a commonly used function to convert a vector of real values to a probability vector. Features with larger scores $\mathbf{z}_j^T \theta$ have correspondingly larger v_j^2, meaning they are more likely to have larger coefficients in the model. Straightforward computation shows that the posterior mode for $\boldsymbol{\beta}$ is the fwelnet solution (3.2) with $\alpha = 0$ and $\lambda = \sigma^2/(2\tau^2)$. Algorithm 2 can be viewed as an empirical Bayes approximation to estimate $\boldsymbol{\theta}$. For other values of α, the fwelnet solution corresponds to the posterior mode for the following prior on $\boldsymbol{\beta}$:

$$p(\beta_j) \propto \exp\left[-v_j^2 \tau^2 \left(\alpha |\beta_j| + \frac{1-\alpha}{2} \beta_j^2\right)\right], \qquad v_j^2 = \frac{p \exp\left(\mathbf{z}_j^T \boldsymbol{\theta}\right)}{\sum_{\ell=1}^p \exp\left(\mathbf{z}_\ell^T \boldsymbol{\theta}\right)}.$$

This connection also presents a way to incorporate feature information in a fully Bayesian framework: instead of estimating θ from the data, we could impose a prior on it. This direction also gives us an explicit way to encode beliefs about the relative importance of the sources of side information for the predictive model.

3.5 A simulation study

We tested the performance of fwelnet against other methods in a simulation study. In the three settings studied, the true signal is a linear combination of the columns of $\mathbf{X}$, with the true coefficient vector $\boldsymbol{\beta}$ being sparse. The response $\mathbf{y}$ is the signal corrupted by additive Gaussian noise. In each setting, we gave different types of feature information to fwelnet to determine the method's effectiveness.

For all methods, we used cross-validation (CV) to select the tuning parameter λ. Unless otherwise stated, the α hyperparameter was set to 1 (i.e. no ℓ_2 squared penalty). To compare methods, we considered the mean squared error (MSE) $MSE = \mathbb{E}[(\hat{y} - \mu)^2]$ achieved on 10,000 test points, as well as the true positive rate (TPR) and false positive rate (FPR) of the fitted models. ($\hat{y}$ denotes the model's prediction while μ denotes the true underlying signal. The oracle model which knows the true coefficient vector β can compute μ exactly and hence has a test MSE of 0.) We ran each simulation 30 times to get estimates for these quantities. (See Appendix B.3 for details of the simulations.)

3.5.1 Setting 1: Noisy version of the true $|\boldsymbol{\beta}|$

In this setting, we have $n = 100$ observations and $p = 50$ features, with the true signal being a linear combination of just the first 10 features. The feature information matrix $\mathbf{Z}$ has a single column: a noisy version of $|\boldsymbol{\beta}|$.

We compared fwelnet against the lasso (using the **glmnet** package) and the adaptive lasso (using the OLS solution as the pilot estimator) across a range of signal-to-noise ratios (SNR) in both the response $\mathbf{y}$ and the feature information matrix $\mathbf{Z}$ (see details in Appendix B.3.1). The results are shown in Figure 3.2. As we would expect, the test MSE figures for the methods decreased as the SNR in the response increased. Fwelnet performed the best, with improvement over the other methods increasing as the SNR in $\mathbf{Z}$ increased, up to a point. In terms of feature selection, fwelnet appeared to have similar TPR but smaller FPR.

3.5.2 Setting 2: Grouped data setting

In this setting, we have $n = 100$ observations and $p = 150$ features, with the features coming in 15 groups of size 10. The feature information matrix $\mathbf{Z} \in \mathbb{R}^{150 \times 15}$ contains group membership information for the features: $z_{jk} = 1\{\text{feature } j \in \text{group } k\}$. We compared fwelnet against the lasso, the adaptive lasso, and the group lasso (using the **grpreg** package) across a range of signal-to-noise ratios (SNR) in the response $\mathbf{y}$. (For the adaptive lasso, we used the lasso solution with λ chosen by CV as the pilot estimator, since the OLS solution is unidentified in this setting.)

We considered two different responses in this setting. The first response was a linear combination of the features in the first group only, with additive Gaussian noise. The results are depicted in Figure 3.3. In terms of test MSE, fwelnet was competitive with the group lasso in the low SNR scenario and edged out the group lasso slightly for the highest SNR setting. In terms of feature selection, fwelnet had comparable TPR as the group lasso (except in the lowest SNR setting) but drastically smaller FPR. Fwelnet had better TPR and FPR than the lasso in this case.

The second response was not as sparse in the features: the true signal was a linear combination of the first 4 feature groups. The results are shown in Figure 3.4. In this case, fwelnet with α fixed at 1 lags the group lasso slightly in test MSE. It is worth noting that fwelnet with $\alpha = 1$ performs appreciably better than the lasso when the SNR is higher. Selecting α via CV improved the test

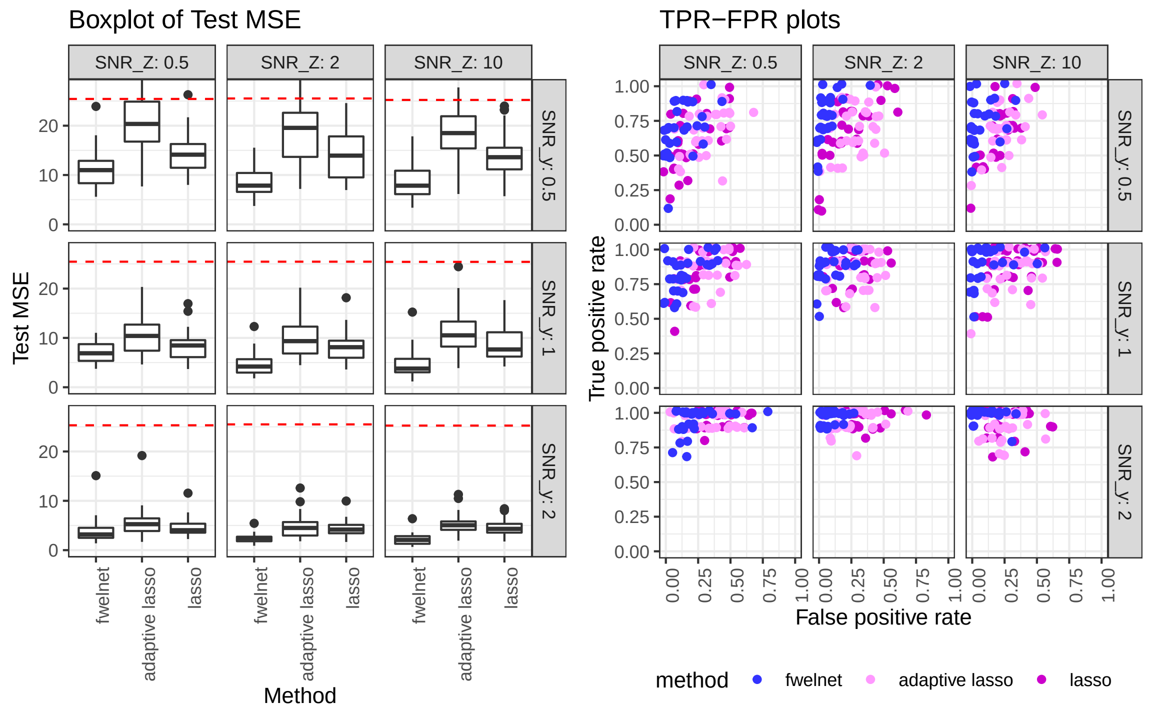

Figure 3.2: *"Feature of features": noisy version of the true* $|\beta|$*.* $n = 100$*,* $p = 50$*. The response is a linear combination of the first 10 features. As we go from left to right, the signal-to-noise ratio (SNR) for* $\mathbf{y}$ *increases; as we go from top to bottom, the SNR in* $\mathbf{Z}$ *increases. The left panel shows the test mean squared error (MSE) figures with the red dotted line indicating the median null test MSE. In the figure on the right, each point depicts the true positive rate (TPR) and false positive rate (FPR) of the fitted model for one of 30 simulation runs. Fwelnet performs the best in test MSE, with the improvement getting larger as the SNR in* $\mathbf{Z}$ *increases, up to a point. Fwelnet appears to have similar TPR but significantly smaller FPR.*

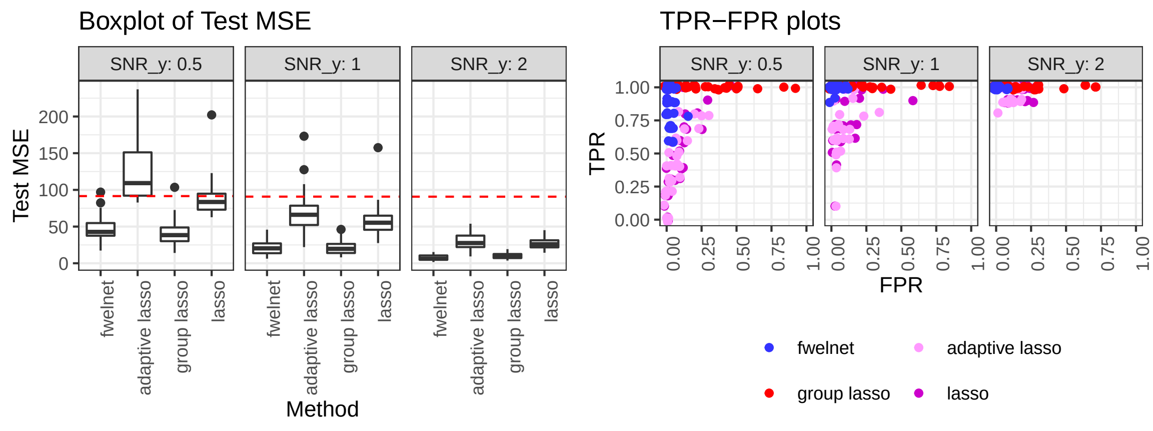

Figure 3.3: *"Feature of features": grouping data.* $n = 100$, $p = 150$. *The features come in groups of 10, with the response being a linear combination of the features in the first group. As we go from left to right, the signal-to-noise ratio (SNR) for* $\mathbf{y}$ *increases. The figure on the left shows the test mean squared error (MSE) results with the red dotted line indicating the median null test MSE. In the figure on the right, each point depicts the true positive rate (TPR) and false positive rate (FPR) of the fitted model for one of 30 simulation runs. Fwelnet performs comparably to the group lasso in terms of test MSE. Fwelnet has higher TPR than the lasso, and lower FPR than the group lasso.*

MSE performance of fwelnet slightly but not enough to outperform the group lasso; it also came at the cost of very high FPR.

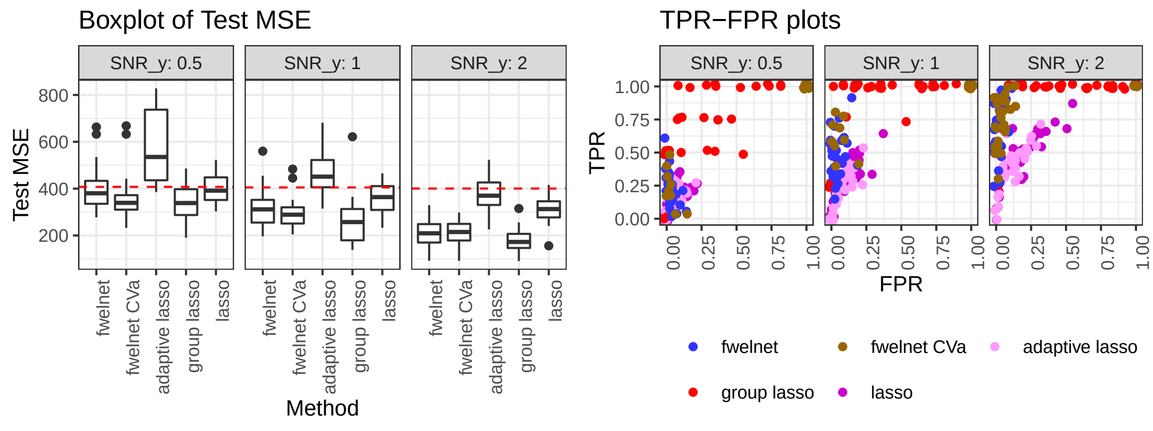

Figure 3.4: *"Feature of features": grouping data.* $n = 100$, $p = 150$. *The features come in groups of 10, with the response being a linear combination of the first 4 groups. As we go from left to right, the SNR for* $\mathbf{y}$ *increases. The left figure shows the test MSE results with the red dotted line indicating the median null test MSE. Fwelnet sets* $\alpha = 1$ *while fwelnet CVa selects* α *via cross-validation. In the figure on the right, each point depicts the TPR and FPR of the fitted model for one of 30 simulation runs. Group lasso performs best here. CV for* α *improves test MSE performance slightly but at the expense of having very high FPR.*

3.5.3 Setting 3: Noise variables

In this setting, we have $n = 100$ observations and $p = 80$ features, with the true signal being a linear combination of just the first 10 features. The feature information matrix $\mathbf{Z}$ consists of 10 noise variables that have nothing to do with the response. Since fwelnet is adapting to these features, we expect it to perform worse than comparable methods.

We compare fwelnet against the lasso and the adaptive lasso (using the OLS solution as the pilot estimator): the results are depicted in Figure 3.5. As expected, fwelnet has higher test MSE than the lasso, but the decrease in performance is not drastic. The adaptive lasso performs much more poorly than the other methods. This is likely due to unstable least squares estimates for the weights due to p being close to n. Fwelnet attained similar FPR and TPR to the lasso.

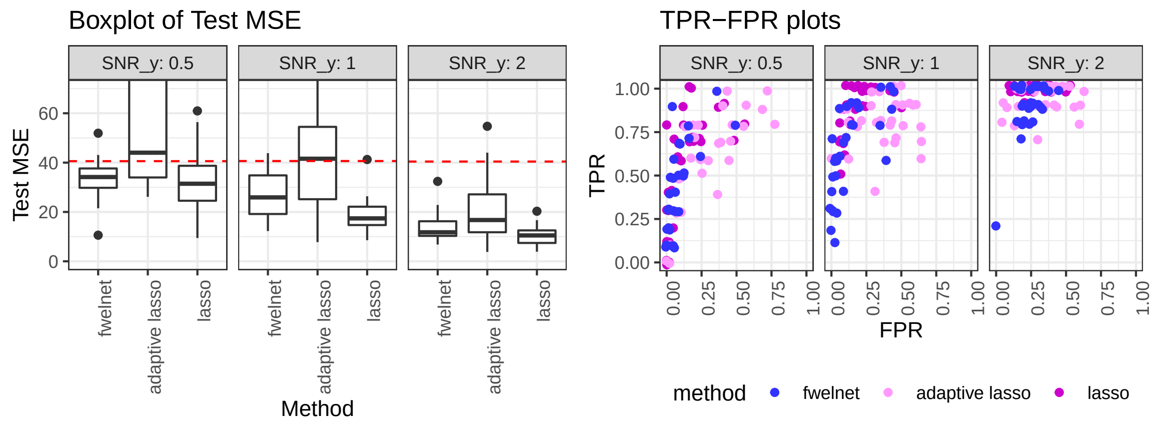

Figure 3.5: *"Feature of features": 10 noise variables. $n = 100$, $p = 80$. The response is a linear combination of the first 10 features. Going from left to right, the SNR for $\mathbf{y}$ increases. The left figure shows the test MSE results, with the red dotted line indicating the median null test MSE. In the right figure, each point depicts the TPR and FPR of the fitted model for one of 30 simulation runs. Fwelnet only performs slightly worse than the lasso in test MSE, and has similar TPR and FPR as the lasso.*

3.6 Application: Early prediction of preeclampsia

Preeclampsia is a leading cause of maternal and neonatal morbidity and mortality, affecting 5 to 10 percent of all pregnancies. The biological and phenotypical signals associated with late-onset preeclampsia strengthen during the course of pregnancy, often resulting in a clinical diagnosis after 20 weeks of gestation (Zeisler et al., 2016). An earlier test for late-onset preeclampsia has substantially higher clinical value as it enables interventions for improvement of maternal and neonatal outcomes (Jabeen et al., 2011). In this example, we leverage protein data collected in late pregnancy, which is closer to the onset of preeclampsia but of lower clinical utility, to learn about the proteins most helpful for this prediction task, then use this information to build a model using protein measurements from early in the pregnancy. It is important to note that data from late pregnancy is only used to train the model: for prediction on new patients, only the samples collected during early pregnancy will be needed.

We used a dataset of 1,125 plasma proteins measured during various gestational ages of pregnancy (Erez et al., 2017). The SOMAScan platform used in this dataset produces targeted measurements of a broad range of proteins that are broadly related to various aspects of human biology. To maintain the exploratory nature of the study, we did not select specific proteins that are expected to be related to preeclampsia based on prior studies. We considered time points ≤ 20 weeks "early" and time points > 20 weeks as "late". The dataset consisted of 166 patients each having 2 to 6 time points, for a total of 666 time point observations. Protein measurements were log-transformed to reduce skewness. We used the following procedure to build a predictive model based on early time point data only:

1. Patients were split randomly into two equal-sized buckets. For patients in the first bucket, we only used their late time points (83 patients with 219 time points) while for patients in the second bucket, we only used their early time points (83 patients with 116 time points).

2. We trained an elastic net logistic regression model on the late time points for patients in the first bucket to predict whether the patient would have preeclampsia (using the log-transformed protein measurements for predictors). The hyperparameter α was set to 0.5 and λ was selected by CV. We extracted model coefficients at the λ value which gave the highest CV area under the curve (AUC).

3. We trained a fwelnet logistic regression model on the early time points for patients in the second bucket, using the absolute values of the late time point model coefficients as feature information. The hyperparameter α was set to 1 and we computed the 10-fold CV AUC for the entire path of λ values.

When performing CV in Steps 2 and 3, we made sure that observations from one patient all belonged to the same CV fold to avoid "contamination" of the held-out fold. One can also run the fwelnet model with additional sources of feature information for each of the proteins.

Figure 3.6 shows a plot of 10-fold CV AUC for the fwelnet model in Step 3 and the baseline lasso model against the number of features in the model. The lasso obtains a maximum CV AUC of 0.80, while fwelnet obtains the largest CV AUC of 0.84.

In running the workflow several times, we noted that the results were somewhat dependent on (i) how the patients were split into the two buckets in Step 1, and (ii) how patients were split into CV folds when training the models in Steps 2 and 3. We found that if the late time point model had few non-zero coefficients, then the fwelnet model for early time point data was very similar to that for the lasso. This matches our intuition: few non-zero coefficients means injecting very little

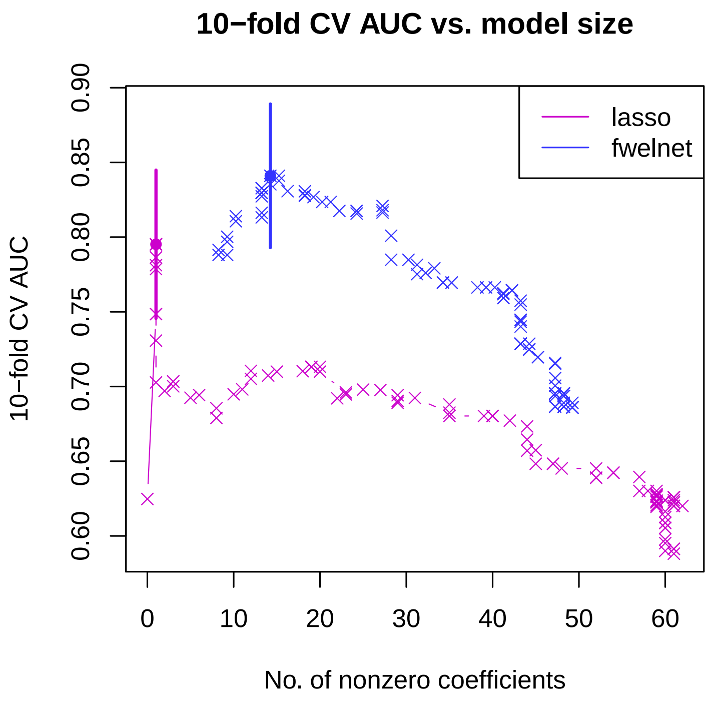

Figure 3.6: *Early prediction of preeclampsia: Plot of 10-fold cross-validated (CV) area under the curve (AUC). 10-fold CV AUC is plotted against the number of non-zero coefficients for each model trained on just early time point data. For each method, the model with highest CV AUC is marked by a dot. To reduce clutter in the figure, the ±1 standard error bars are drawn for just these models. Fwelnet achieved higher CV AUC for the same model size, i.e. number of features with non-zero coefficients.*

additional information through fwelnet's relative penalty factors. Nevertheless, we did not encounter cases where running fwelnet gave worse CV AUC than the lasso.

3.7 Using fwelnet for multi-task learning

We turn now to an application of fwelnet to *multi-task learning*. Here, we have a single model matrix $\mathbf{X}$ but are interested in multiple responses $\mathbf{y}_1, \dots, \mathbf{y}_B$. If there is some common structure between the signals in the responses, it can be advantageous to fit models for them simultaneously. This is especially useful if the responses have low SNR.

We demonstrate how fwelnet can be used to learn better models in the setting with two responses, $\mathbf{y}_1$ and $\mathbf{y}_2$. The idea is to use the absolute value of coefficients of one response as the external information for the other response. That way, a feature which has larger influence on one response is likely to be given a correspondingly lower penalty weight when fitting the other response. Algorithm 3 presents one possible way of doing so.

Algorithm 3 *Using fwelnet for multi-task learning*

```
1. Initialize $\boldsymbol{\beta}_1^{(0)}$ and $\boldsymbol{\beta}_2^{(0)}$ at the lambda.min elastic net solutions for $(\mathbf{X}, \mathbf{y}_1)$ and $(\mathbf{X}, \mathbf{y}_2)$ respectively, that is, the value of the hyperparameter $\lambda$ which minimizes cross-validated error.
2. For $k = 0, 1, \dots$ until convergence:
    (a) Set $\mathbf{Z}_2 = \left|\boldsymbol{\beta}_1^{(k)}\right|$. Run fwelnet with $(\mathbf{X}, \mathbf{y}_2, \mathbf{Z}_2)$ and set $\boldsymbol{\beta}_2^{(k+1)}$ to be the lambda.min solution.
    (b) Set $\mathbf{Z}_1 = \left|\boldsymbol{\beta}_2^{(k+1)}\right|$. Run fwelnet with $(\mathbf{X}, \mathbf{y}_1, \mathbf{Z}_1)$ and set $\boldsymbol{\beta}_1^{(k+1)}$ to be the lambda.min solution.
```

We tested the effectiveness of Algorithm 3 (with step 2 run for 3 iterations) on simulated data. We generate 150 observations with 50 independent features. The signal in response 1 is a linear combination of features 1 to 10, while the signal in response 2 is a linear combination of features 1 to 5 and 11 to 15. The coefficients are set such that those for the common features (i.e. features 1 to 5) have larger absolute value than those for the features specific to one response. The SNRs in response 1 and response 2 are 0.5 and 1.5 respectively. (See Appendix B.4 for more details of the simulation.)

We compared Algorithm 3 against: (i) the *individual lasso (ind_lasso)*, where the lasso is run separately for $\mathbf{y}_1$ and $\mathbf{y}_2$; and (ii) the *multi-response lasso (mt_lasso)* (Obozinski et al., 2010), where coefficients belonging to the same feature across the responses are given a joint ℓ_2 penalty. Because of the ℓ_2 penalty, a feature is either included or excluded in the model for all the responses at the same time.

Figure 3.7 shows the results for 50 simulation runs. Fwelnet outpeforms the other two methods in test MSE as evaluated on 10,000 test points. The individual lasso performs well for the higher SNR response but poorly for the lower SNR response. The multi-response lasso is able to borrow strength from the higher SNR response to obtain good performance on the lower SNR response. However, because the models for both responses are forced to have the same set of features, performance suffers on the higher SNR response. Fwelnet has the ability to borrow strength across responses without being hampered by this restriction.

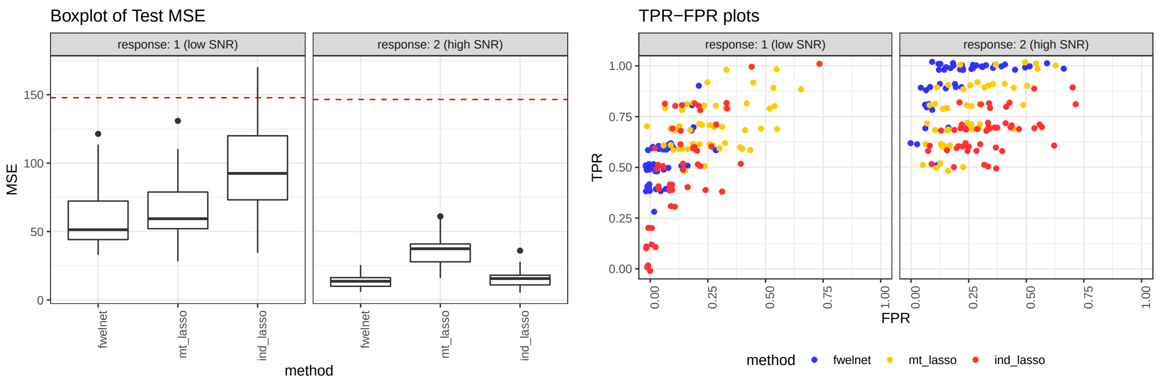

Figure 3.7: *Application of fwelnet to multi-task learning.* $n = 150$, $p = 50$. *Response 1 is a linear combination of features 1 to 10, while response 2 is a linear combination of features 1 to 5 and 11 to 15. The SNR for the responses are 0.5 and 1.5 respectively. The left figure shows the test MSE figures with the red dotted line indicating the median null test MSE. The right figure shows the TPR and FPR of the fitted model (each point being one of 50 simulation runs). Fwelnet outperforms the individual lasso and the multi-response lasso in test MSE for both responses. Fwelnet also appears to have better FPR than the other methods and better TPR than the individual lasso.*

3.8 Discussion

We have proposed organizing external information on features, or "features of features", as a matrix $\mathbf{Z} \in \mathbb{R}^{p \times K}$, and modifying model-fitting algorithms by assigning each feature a score, $\mathbf{z}_j^T \boldsymbol{\theta}$, based on this auxiliary information. The *feature-weighted elastic net* (*"fwelnet"*) is one such method: it imposes a penalty modification factor $w_j(\boldsymbol{\theta}) = \left[\sum_{\ell=1}^p \exp\left(\mathbf{z}_\ell^T \boldsymbol{\theta}\right)\right] / \left[p \exp\left(\mathbf{z}_j^T \boldsymbol{\theta}\right)\right]$ for the elastic net algorithm.

Fwelnet is widely applicable in that there are no restrictions on the type of feature information that can be incorporated into $\mathbf{Z}$ as long as it is real-valued. As such, we recommend using fwelnet whenever feature information is available (e.g. grouping information, prior guesses on feature importance). When the feature information is relevant to the prediction problem, in that it has some signal on how important a feature is to predicting the response, we expect fwelnet to outperform competitor methods. At the same time, simulation setting 3 (Section 3.5.3) shows that using irrelevant feature information can be detrimental to the fit. In practice we recommend using domain knowledge to guide selection of side information for the model. We also recommend fitting the vanilla elastic net and comparing the CV error of the two methods: this comparison will show if the feature information was relevant to the prediction problem or not.

There is much scope for future work:

- *Interpretation of $\mathbf{z}_j^T\boldsymbol{\theta}$ and $\boldsymbol{\theta}$.* As noted in the Introduction, $\mathbf{z}_j^T\boldsymbol{\theta}$ can be thought of as an indication of how influential feature j is on the response, since a larger $\mathbf{z}_j^T\boldsymbol{\theta}$ corresponds to a smaller penalty weight $w_j(\boldsymbol{\theta})$ (see Equations (3.1) and (3.2)).

  The interpretation for $\boldsymbol{\theta}$ is not as straightforward. When $\mathbf{Z} \in \mathbb{R}^{p\times K}$ is orthonormal, we can interpret θ_k as the relative importance of the kth source of feature information for identifying important features for the prediction problem. However, this interpretation becomes less clear when there are correlations between the columns of $\mathbf{Z}$. In the extreme case where there is multicollinearity in $\mathbf{Z}$, $\boldsymbol{\theta}$ is not identified even though $\mathbf{z}_j^T\boldsymbol{\theta}$ is unique. These are the same issues one faces when interpreting OLS coefficients in the presence of feature correlations.

- *Different choices of side information* $\mathbf{Z}$. We have explored a few different choices of side information, including prior coefficient estimates and group membership. It would be interesting to evaluate fwelnet's effectiveness when using other types of side information. One natural extension of group membership is probabilistic group membership, where each feature is assigned a probability distribution across the K groups. Another extension is overlapping groups, where each row of $\mathbf{Z}$ need not sum to 1. Setting $\mathbf{Z}$ as a $p \times p$ similarity matrix is another option which can be thought of as a combination of the two extensions above, with group j being associated with feature j, and the degree of group membership measured by how similar each feature is to feature j.

- *Whether $\boldsymbol{\theta}$ should be treated as a parameter or a hyperparameter, and how to determine its value.* We introduced $\boldsymbol{\theta}$ as a hyperparameter for (3.2). This gives us a clear interpretation for $\boldsymbol{\theta}$ described above. However, grid search computation to find its optimal value grows exponentially with the number of sources of feature information. To avoid this growth, we suggested a descent algorithm for $\boldsymbol{\theta}$ based on its gradient with respect to the fwelnet objective function. There are other methods for hyperparameter optimization such as random search (e.g. Bergstra and Bengio (2012)) or Bayesian optimization (e.g. Snoek et al. (2012)) that could be applied.

  One could consider $\boldsymbol{\theta}$ as an argument of the fwelnet objective function to be minimized over jointly with $\boldsymbol{\beta}$. This approach gives us a theoretical connection to the group lasso (Section 3.4.1). However, we will obtain different estimates of $\boldsymbol{\theta}$ for each value of the hyperparameter λ, which may be undesirable for interpretation. The objective function is also not jointly convex in $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$, so different minimization algorithms could end up at different local minima. Our attempts to make this approach work (see Appendix B.1) did not fare as well in prediction performance and was computationally expensive.

- *Choice of penalty modification factor.* While the penalty modification factor $w_j(\boldsymbol{\theta})$ we have proposed works well in practice and has several desirable properties, we make no claim about its optimality.

- *Extending the use of scores beyond the elastic net.* The use of feature scores $\mathbf{z}_j^T\boldsymbol{\theta}$ in modifying feature weights is a general idea that could apply to any supervised learning algorithm. More work needs to be done on how such scores can be incorporated, with particular focus on how θ can be learned through the algorithm.

An R language package **fwelnet** which implements fwelnet is available at `https://www.github.com/kjytay/fwelnet`.

Chapter 4

Reluctant Generalized Additive Models

4.1 Introduction

In the previous two chapters, we proposed modifications of the lasso and the elastic net that leveraged external information we might have on the features for better prediction and inference. In this chapter, we propose a modification along a different dimension: the functional form of the model's output.

Assume that the response $\mathbf{y}$ and the columns of design matrix $\mathbf{X}$ are mean-centered at zero so that we need not fit an intercept term. In the generalized linear model (GLM) setting, the elastic net assumes that the relationship between the response and the features is

$$y_i \sim P_{\mu_i}, \quad \mu_i = g^{-1}(\eta_i), \quad \eta_i = \sum_{j=1}^{p} \beta_j x_{ij}, \tag{4.1}$$

where g is a link function, $\{P_\mu\}$ is a family of probability distributions indexed by μ, and $\beta_1, \dots, \beta_p \in \mathbb{R}$ are parameters of the model. The elastic net determines these parameters by minimizing the sum of the negative log-likelihood (NLL) and the elastic net penalty. While the resulting model is highly interpretable, it makes the (possibly) unrealistic assumption that each feature influences the transformed response $\boldsymbol{\eta}$ in a linear fashion.

Generalized additive models (GAMs), introduced by Hastie and Tibshirani (1986), model the relationship more flexibily as

$$y_i \sim P_{\mu_i}, \quad \mu_i = g^{-1}(\eta_i), \quad \eta_i = \sum_{j=1}^{p} f_j(x_{ij}), \tag{4.2}$$

where the $f_j(\cdot)$'s are unknown component functions, assumed to be smooth or to have low complexity. Even though the transformed response $\boldsymbol{\eta}$ can vary with the individual features in a non-linear fashion, GAMs remain interpretable since, letting $\mathbf{X}_j \in \mathbb{R}^n$ denote the values of the jth feature, the effect of $\mathbf{X}_j$ on $\boldsymbol{\eta}$ (and hence, on $\mathbf{y}$) does not depend on any $\mathbf{X}_k$ with $k \neq j$.

One drawback of GAMs is that they assume that the response is influenced by every feature available to the data analyst. When p is large, this seems to be an unreasonable assumption. In fact, once $p \geq n$, GAMs are unidentifiable: we can find two different fits $\hat{f}_1(\cdot), \dots, \hat{f}_p(\cdot)$ and $\hat{f}'_1(\cdot), \dots, \hat{f}'_p(\cdot)$ such that $\sum_j \hat{f}_j(x_j) = \sum_j \hat{f}'_j(x_j)$ for all possible $(x_1, \dots, x_p) \in \mathbb{R}^p$. This causes GAMs to lose their interpretability. An additional problem in this setting is that GAMs will tend to overfit to the noise in the data. As a result, there has been demand for additive models which are *sparse*, i.e. consisting of just a handful of the features available to the data analyst. Previous methods for estimating sparse additive models, detailed in Section 4.2, have cast model-fitting as an optimization problem

$$\underset{f_1, \dots, f_p \in \mathcal{F}}{\text{minimize}} \quad \ell(\mathbf{y}; f_1, \dots, f_p) + \sum_{j=1}^{p} J(f_j), \tag{4.3}$$

where ℓ is the negative log-likelihood of the data, J is some penalty function, and $\mathcal{F}$ is some space of allowable functions for the f_j's.

When building a sparse additive model, the algorithm needs to make a choice: for some signal in the response, should we attribute it to a linear term in some feature $\mathbf{X}_j$, or should we attribute it to a non-linear term in some (possibly other) feature $\mathbf{X}_k$? Some of the earlier sparse additive methods ignore this choice: the f_j's are all modeled as non-linear functions. This may result in needlessly complex models when having some of the f_j's as linear functions would have sufficed. Later methods recognized this deficiency and have the flexibility to model each f_j as either a linear or non-linear function through clever choices of penalty functions. However, the tradeoff between having a linear or non-linear function is often implicit and controlled via a tuning parameter.

Inspired by "reluctant interaction modeling" (Yu et al., 2019), we propose a new algorithm for fitting sparse generalized additive models that has an explicit preference toward linear relationships over non-linear ones. As a guiding principle, we prefer a model to contain only effects that are linear in the original set of features: it is "reluctant" to included non-linearities in the model unless they add to predictive performance. To operationalize this, we first construct a sparse model for $\eta(y)$ with just linear features. Next, we use the residual from the first step to construct new non-linear features. Finally, we fit another sparse model for $\eta(y)$ utilizing both the linear and non-linear features.

The rest of the paper is organized as follows: in the next section, we review previous methods which have sought to estimate sparse additive models from the given data. In Section 4.3, we give a brief review of the ideas in Yu et al. (2019) and introduce our method, called *reluctant generalized additive modeling* (RGAM), in greater detail. In Section 4.4, we point to parameter choices that a practitioner should be cognizant of when using RGAM, as well as the computational advantages of the method. We demonstrate the method on synthetic and real data examples in Section 4.5, briefly discuss RGAM's effective degrees of freedom in Section 4.6 and end off with a discussion in Section 4.7.

4.2 Related work

This review closely follows that in Chouldechova and Hastie (2015) and Petersen and Witten (2019). As mentioned in the introduction, previous methods for fitting sparse additive models involve solving an optimization problem

$$\underset{f_1,\ldots,f_p\in\mathcal{F}}{\text{minimize}} \quad \ell(\mathbf{y}; f_1,\ldots,f_p) + \sum_{j=1}^{p} J(f_j),$$

with different methods choosing different penalty functions $J(\cdot)$ and different family of functions $\mathcal{F}$. The *Component Selection and Smoothing Operator* (COSSO) (Lin and Zhang, 2006) models the f_j's as belonging to a reproducing kernel Hilbert space (RKHS) and penalizes the sum of the RKHS norms of the component functions. Ravikumar et al. (2007) proposed the *Sparse Additive Model* (SpAM), which is essentially a functional version of the group lasso (Yuan and Lin, 2006). For each j, f_j is modeled as a linear combination of d basis functions $f_j = \beta_{j1}g_{j1} + \cdots + \beta_{jd}g_{jd}$. Letting $\mathbf{B}_j$ denote the $n \times d$ matrix with $(B_j)_{k\ell} = g_{j\ell}(x_{kj})$, SpAM penalizes the sum of ℓ_2 norms of the $\mathbf{B}_j\boldsymbol{\beta}_j$, i.e. $J(f_j) = \lambda \left\|\mathbf{B}_j\boldsymbol{\beta}_j\right\|_2$ for some hyperparameter $\lambda \geq 0$. Meier et al. (2009) parametrize each f_j in a similar fashion, and propose a penalty which is the quadratic mean of the component function norm and a second derivative smoothness penalty, summed over the component functions. Sadhanala and Tibshirani (2017) proposed *Additive Models with Trend Filtering*, where the penalty for f_j is the (discrete) total variation of its kth (discrete) derivative, k being an integer hyperparameter chosen by the user. The fit for each variable is restricted to piecewise polynomials of degreee k. Additive models with trend filtering are a generalization of the *Fused Lasso Additive Model* (FLAM) (Petersen et al., 2016), where each f_j is either all zero or piecewise constant with a small number of adaptively chosen knots.

While these earlier methods are able to capture non-linear fits between the features and the response, they will continue to do so even when a linear fit would have sufficed. (Additive models with trend filtering will only give a linear fit when $k = 1$.) In these cases, the methods above may overfit to the data, resulting in less interpretable models with possibly worse predictive performance. To address this issue, more recent methods have the ability to decide whether to model a feature linearly or non-linearly, given that it is included in the model. This ability is achieved with the use of more complex penalty functions. For example, the *Sparse Partially Linear Additive Model* (SPLAM) (Lou et al., 2016) does so using a hierarchical group lasso penalty (Yan and Bien, 2017), while *Generalized Additive Model Selection* (GAMSEL) (Chouldechova and Hastie, 2015) does so with an overlap group lasso penalty (Obozinski et al., 2009). Most recently, Petersen and Witten (2019) introduced *Sparse Partially Linear Additive Trend Filtering* (SPLAT), which allows the knots for the non-linear fits to be adaptively chosen. It does so using a three-term penalty for each f_j that is a combination of ℓ_1 and ℓ_2 norms of different quantities.

4.3 Reluctant generalized additive modeling ("RGAM")

Our method, which we call *Reluctant Generalized Additive Modeling* (RGAM), was inspired by the ideas behind *Reluctant Interaction Modeling* (Yu et al., 2019). We give a brief overview of reluctant interaction modeling here.

4.3.1 Reluctant interaction modeling

Yu et al. (2019) considers the following interaction model:

$$\mathbf{y} = \sum_{j=1}^{p} \beta_j \mathbf{X}_j + \sum_{k=1}^{p^2} \gamma_k \mathbf{Z}_k + \varepsilon, \tag{4.4}$$

where the $\mathbf{Z}_k$'s index the $q = (p^2 + p)/2$ two-way interaction terms $\mathbf{X}_j * \mathbf{X}_{j'}$, $1 \leq j \leq j' \leq p$. The key difficulty in fitting such a model is to pick a small but relevant subset of the q interaction terms. Instead of the commonly used *hierarchical principle*, where one includes an interaction only if the corresponding main effects are also included, Yu et al. (2019) propose a new guiding principle:

> **The reluctant interaction selection principle:** *One should prefer a main effect over an interaction if all else is equal.*

One way to interpret this principle is to fit the response as well as possible using only the main effects; only after that do we include interaction terms to capture signal in the response which could not be captured by the main effects. Yu et al. (2019)'s full algorithm, implemented in the **sprinter** R package, is detailed in Algorithm 4. The authors note that the lasso (Tibshirani, 1996) in Steps 1 and 3 could be substituted by other regression methods.

4.3.2 Reluctant generalized additive modeling

We adapt the reluctant interaction selection principle for GAMs:

> **The reluctant non-linear selection principle:** *One should prefer a linear term over a non-linear term if all else is equal.*

Algorithm 4 *Reluctant interaction model algorithm*

Input: Design matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, response $\mathbf{y} \in \mathbb{R}^n$, screening hyperparameter $\eta > 0$.

1. Fit the lasso of $\mathbf{y}$ on $\mathbf{X}$ to get coefficients $\hat{\boldsymbol{\beta}}$. Compute the residuals $\mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$, using the λ hyperparameter selected by cross-validation.
2. For the hyperparameter $\eta > 0$, screen the interaction terms based on the residual:

$$\hat{\mathcal{I}}_\eta = \left\{ k \in \{1, \ldots, q\} : \overline{\mathrm{sd}}(\mathbf{r}) |\overline{\mathrm{cor}}(\mathbf{Z}_k, \mathbf{r})| > \eta \right\}, \tag{4.5}$$

   where $\overline{\mathrm{sd}}(\mathbf{r})$ denotes the sample standard deviation of r and $\overline{\mathrm{cor}}(\mathbf{Z}_k, r)$ denotes the sample correlation between $\mathbf{Z}_k$ and r.
3. Fit the lasso of $\mathbf{y}$ on $\mathbf{X}$ and $\mathbf{Z}_{\hat{\mathcal{I}}_\eta}$.

To operationalize this principle, we mimic the three-step process of reluctant interaction modeling. In Step 1, we fit the response as well as we can using only the main effects, and in Step 3, we re-fit the response on all the main effects and the additional features which were constructed in Step 2. Where our proposal differs from reluctant interaction modeling is in the construction of the additional features in Step 2. Given a hyperparameter $d \in \mathbb{N}$, for each $j \in \{1, \ldots, p\}$, we build a smoothing spline with d degrees of freedom of $\mathbf{r}$, the residual from Step 1, on $\mathbf{X}_j$. This new feature, which we denote by $\hat{f}_j$, captures signal in the residual using a non-linear relationship with $\mathbf{X}_j$. Full details of our proposal, which we call *Reluctant Generalized Additive Modeling (RGAM)*, can be found in Algorithm 5. While the lasso in Steps 1 and 3 could be substituted with a different regression method, we recommend it strongly in this context as it performs variable selection, giving us the sparsity we want for the final model.

Algorithm 5 *Reluctant generalized additive model algorithm*

Input: Design matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$, response $\mathbf{y} \in \mathbb{R}^n$, degrees of freedom hyperparameter $d \in \mathbb{N}$, scaling hyperparameter $\gamma \in [0, 1]$ and a path of lasso hyperparameters $\lambda_1 > \cdots > \lambda_m \geq 0$. (Note that the lasso hyperparameters will only be used in Step 3.)

1. Fit the lasso of $\mathbf{y}$ on $\mathbf{X}$ to get coefficients $\hat{\boldsymbol{\beta}}$. Compute the residuals $\mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$, using the λ hyperparameter selected by cross-validation.
2. For each $j \in \{1, \ldots, p\}$, fit a smoothing spline with d degrees of freedom of $\mathbf{r}$ on $\mathbf{X}_j$ which we denote by $\hat{f}_j$. Rescale $\hat{f}_j$ so that $\overline{\mathrm{sd}}(\hat{f}_j)/\mathrm{mean}(\overline{\mathrm{sd}}(\mathbf{X}_j)) = \gamma$. Here, $\overline{\mathrm{sd}}(\hat{f}_j)$ denotes the sample standard deviation of $\hat{f}_j$, while $\mathrm{mean}(\overline{\mathrm{sd}}(\mathbf{X}_j))$ denotes the mean of the sample standard deviations of the $\mathbf{X}_j$'s across $j = 1, \ldots, p$. (When $\mathbf{X}$ is standardized, the denominator is equal to 1.) Let $\mathbf{F} \in \mathbb{R}^{n \times p}$ denote the matrix whose columns are the $\hat{f}_j$'s.
3. Fit the lasso of $\mathbf{y}$ on $\mathbf{X}$ and $\mathbf{F}$ for the path of tuning parameters $\lambda_1 > \cdots > \lambda_m \geq 0$.

Our proposal is "reluctant" to include non-linearities in a few ways. First, as with reluctant interaction modeling, the non-linear features are only allowed to model signal which the main effects were unable to capture in Step 1. Second, by rescaling the non-linear features so that their sample standard deviation is just a fraction γ compared to that of the main effects, it means that the non-linearity must be strong enough so that its associated coefficient is important enough to survive variable selection by the lasso in Step 3. Third, if we think of the non-linear feature for variable j as a linear combination of spline basis functions for variable j, the construction in Step 2 forces this linear combination to be fixed up to a global scaling factor. As such, we expect RGAM to have smaller effective degrees of freedom than methods which allow these coefficients to vary independently of each other. We explore this last point in more detail in Section 4.6.

We note that in Algorithm 5, we construct non-linear counterparts for all p features in Step 2. In some settings, we may wish to be conservative in allowing non-linear features into the model. One can tweak Step 2 of Algorithm 5 to achieve this outcome. For example, let $\mathcal{A} = \{j : \hat{\beta}_j \neq 0\}$ be the active set of features after Step 1, i.e. the set of features which were selected by the lasso on the main effects. We could constrain Step 2 to compute non-linear features only for $j \in \mathcal{A}$. This version of RGAM, which we call RGAM_SEL, weakly assumes a hierarchical principle where we expect a non-linear version of a variable to have an effect only if we expect the variable to have a linear effect in the first place. (The hierarchy is not strictly enforced as it is still possible for the non-linear version of variable j to be selected without variable j itself being selected in Step 3.) As a side benefit, RGAM_SEL is more computationally efficient than RGAM since Step 3 involves computing the lasso solution for $p + |\mathcal{A}|$, rather than $2p$, features. In our simulations, RGAM_SEL is 1.5 to 3 times as fast as RGAM, but does not appear to perform as well in terms of test error.

4.4 Computation

We have developed an R package, **relgam**, which implements our proposal. Steps 1 and 3 of the RGAM algorithm are implemented using the `cv.glmnet` and `glmnet` functions from the **glmnet** package (Friedman et al., 2010), while Step 2 is implemented with the `smooth.spline()` function in the **stats** package.

The `rgam` function (which fits our model) has an argument `init_nz` which admits a vector of indices. For a given feature index j, a non-linear feature is computed for $\mathbf{X}_j$ if it appears in `init_nz` or if it appears in the active set $\mathcal{A}$. The default behavior is to compute non-linear features for all p variables, i.e. `init_nz = 1:p`. To compute non-linear features for just the active set of Step 1, the user can set `init_nz` to the empty vector: `init_nz = c()`. Hence, this argument allows `rgam` to compute the solutions to both RGAM (as in Algorithm 5) and RGAM_SEL (defined in the previous section). The `init_nz` argument is also useful if the user has some prior information on the relevance of the variables to the response: variables with high relevance can always have non-linear features included in Step 3 by including them in the vector passed to `init_nz`.

With respect to hyperparameters, the user can specify the λ, γ and d values using the `lambda`, `gamma` and `df` options respectively. `rgam` selects a path of λ values in the same manner as the `glmnet` function in the **glmnet** package; we recommend that the user stick with this choice of λ values. The default value for `gamma` is 0.8 if `init_nz = c()` (i.e. RGAM_SEL), and is 0.6 otherwise. We recommend that the user perform cross-validation (CV) to pick an optimal value of `gamma`. In our simulations, we find that values of `gamma` below 0.5 usually result in models without any non-linearities. The default value for `df` is set conservatively at 4. We recommend using CV to pick an optimal value of `df` but over just a handful of values as the model is not that sensitive to this choice.

4.4.1 Extension to other likelihood functions

In Section 4.2, we noted that previous methods for fitting sparse GAMs solve an optimization problem of the form

$$\underset{f_1,\dots,f_p \in \mathcal{F}}{\text{minimize}} \quad \ell(\mathbf{y}; f_1, \dots, f_p) + \sum_{j=1}^p J(f_j).$$

The optimization is typically performed via an iterative algorithm such as coordinate descent or block coordinate descent. In theory, these methods work with a large class of likelihoods ℓ. For example, in the case of GLMs, ℓ is repeatedly approximated by a quadratic term ℓ' and the sum $\ell' + \sum_{j=1}^p J(f_j)$ is minimized until convergence is attained. Implementing this procedure in practice, however, can be tedious. This is also the case for extending the methods to Cox regression models, where ℓ is the partial likelihood of the data.

Unlike previous methods, the RGAM algorithm (Algorithm 5) can be extended easily to different likelihood functions. As long as the likelihood can be handled by the `glmnet` function in the **glmnet** package through its `family` Argument, Steps 1 and 3 of the RGAM algorithm can be adapted immediately by passing that `family` argument to `glmnet`. The only remaining work is to compute the analog of the residual in Step 2, which is much easier than solving a modified optimization problem. At the time of writing, apart from the Gaussian likelihood for continuous responses, we have working software implementing the logistic, Poisson and Cox regression models for binary, count and survival data respectively.

4.4.2 Timing comparison

Since RGAM uses k-fold CV of the lasso in Step 1 (our software sets $k = 5$ as a default) and the lasso on all the linear and non-linear features in Step 2, we expect RGAM to take at least $k + 1$ times as long as `glmnet`, which implements the lasso. Nevertheless, we find that RGAM is very competitive with other sparse additive modeling techniques in terms of computational efficiency.

Figure 4.1 presents the absolute time taken to fit the models for various values of n (number of observations) and p (number of features), while Figure 4.2 presents these times relative to that for RGAM. (Recall that RGAM refers to procedure in Algorithm 5 while RGAM_SEL refers to the procedure where non-linear features are only constructed for features in the active set from Step 1.) Each point or bar is the mean of 5 simulation runs. We see that GAMSEL takes anywhere from 1.5 to 8 times as long as RGAM for model fitting, with the factors being bigger for larger simulation settings. The computation burden for SpAM and RGAM are comparable, while RGAM_SEL can often be faster than these two methods.

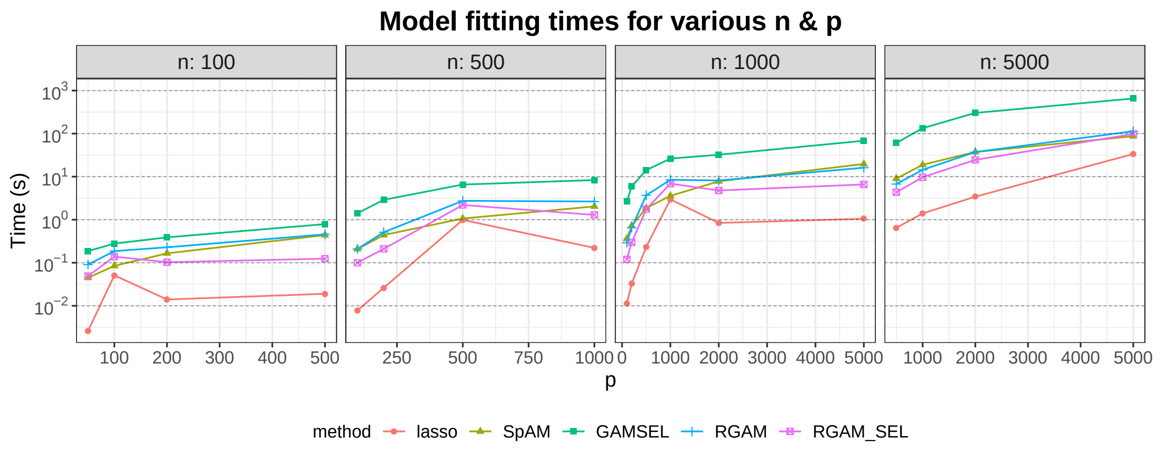

Figure 4.1: *Model fitting times (in seconds) for sparse additive methods, for different combinations of n (number of observations) and p (number of features). Each point is the mean of 5 simulation runs. Note that the y-axis is on a logarithmic scale.*

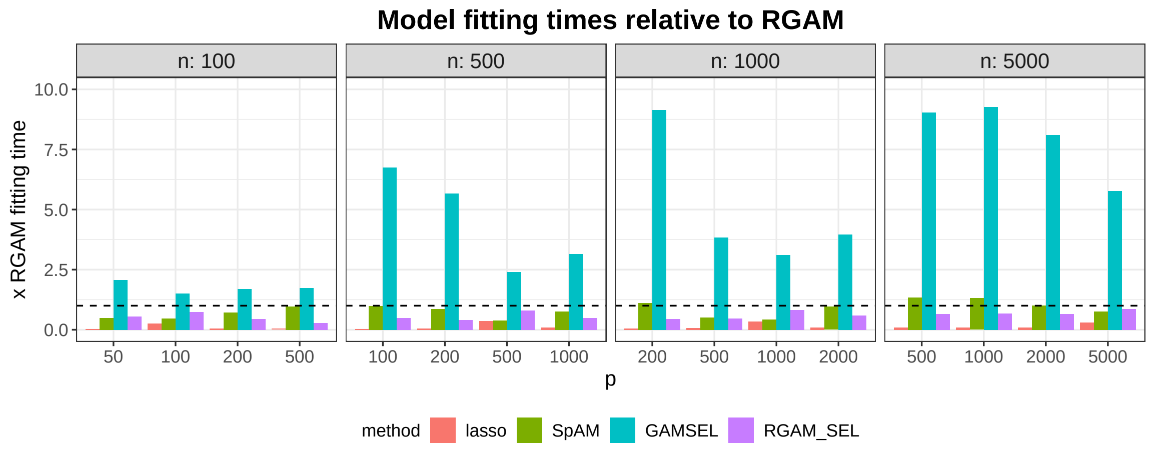

Figure 4.2: *Model fitting times for sparse additive methods for different combinations of n (number of observations) and p (number of features), expressed as a multiple of the model fitting time for RGAM. Each bar is the mean of 5 simulation runs. The dotted horizontal line indicates the fitting time for RGAM.*

4.5 Simulated and real data examples

We conducted an extensive simulation study comparing our method with the lasso and GAMSEL. (We did not compare RGAM with SPLAM (Lou et al., 2016) and SPLAT (Petersen and Witten, 2019) as we did not find R packages implementing these methods at the time of writing.) We use the null model, i.e. mean of the responses in the training dataset, as a baseline. Appendix C.1 contains full details and results on the simulation; we present some illustrative snippets here. For RGAM, we considered two versions: one where non-linear features were constructed for all p main effects, and one where non-linear features were constructed for only the main effects in the active set from Step 1.

In all the simulations that follow, the feature values X_{ij} are independent draws from the Unif$[-1, 1]$ distribution. The response is $y_i = \mu_i + \varepsilon_i = f(X_{i1}, \ldots, X_{ip}) + \varepsilon_i$, where f is a function that depends on the simulation and the ε_i's are independent $\mathcal{N}(0, \sigma^2)$ draws. The data generating process for the signal is such that the linear and non-linear components are orthogonal. The noise variance σ^2 is set so that the data has the desired signal-to-noise ratio (SNR).

For all methods, 5-fold CV was performed to select the hyperparameter λ only: default values were used for all other hyperparameters. Each boxplot in Figures 4.3, 4.4 and 4.5 is the result of 30 simulation runs. The test error metric is mean-squared error where the target is the true signal value, i.e. $\mathbb{E}[(\hat{y}_{test} - \mu_{test})^2]$ instead of $\mathbb{E}[(\hat{y}_{test} - y_{test})^2]$. (With this test error metric, the oracle which knows the data generating model would have a test error of 0.) Test error is estimated using 5,000 test points.

4.5.1 Simulation 1: Small n and p, hierarchical non-linear signal

In this setting, we have 100 observations and 200 features with the signal being a function of the first five features. The setting is "hierarchical" in the sense that all the features that make up the non-linear component of the signal also have a linear component. More explicitly, the signal is $f(\mathbf{X}_1, \ldots, \mathbf{X}_p) = \sum_{j=1}^5 [\mathbf{X}_j + 2(3\mathbf{X}_j^2 - 1)/3]$. The SNR of the overall response is set to 2, with roughly equal SNR in each of the non-linear and linear components.

The results are shown in Figure 4.3. Both versions of RGAM outperform the other methods, with RGAM_SEL being the best. This makes intuitive sense: since the signal is hierarchical, the main effects selected by RGAM_SEL for Step 2 will be smaller than p, yet will very likely include the true main effects. Thus, in Step 3, the true non-linear features only have to compete with a smaller set of features to enter the final model as opposed to RGAM, where they have to compete with all p non-linear features.

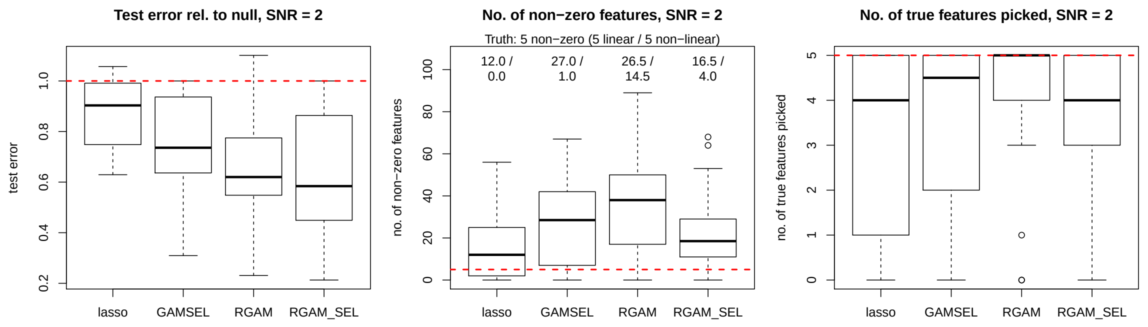

Figure 4.3: *Hierarchical setting: $n = 100$, $p = 200$. The left-most plot presents test error as a fraction of the null model's test error. The middle plot presents the number of features each model selected, with the two numbers on top of the boxplots being the median number of linear components and non-linear components selected. The right-most plot presents the number of true features each method selected, with the total number of true features indicated by the dotted red line. RGAM_SEL performs best in test error due to the hierarchical nature of the non-linearities.*

4.5.2 Simulation 2: Small n and p, purely non-linear signal

In this setting, we again have 100 observations and 200 features, with the signal being a function of the first five features. However, the signal, $f(\mathbf{X}_1, \ldots, \mathbf{X}_p) = \sum_{j=1}^{5} 2(5\mathbf{X}_j^3 - 3\mathbf{X}_j)$, only depends on non-linear functions of the features which are orthogonal to the feature itself. Since the $\mathbf{X}_j$'s are drawn from a Unif$[-1, 1]$ distribution, we have $\text{Cov}(5\mathbf{X}_j^3 - 3\mathbf{X}_j, \mathbf{X}_j) = 0$. The SNR of the response is set to 2.

The results are shown in Figure 4.4. Only RGAM is able to outperform the null model. This is expected for the lasso since it only captures linear effects. GAMSEL can include a non-linear effect in a particular variable only if its corresponding linear effect is included too, and thus does not perform well either. Without linear effects in the signal, Step 1 of the RGAM algorithm cannot pick out the true features reliably. RGAM_SEL thus cannot reliably pick out the correct non-linear features for Step 3 of the algorithm. RGAM circumvents this problem by constructing non-linear features for all p features.

4.5.3 Simulation 3: Large n and p, hierarchical and non-hierarchical non-linear signal

In this setting, we have 1,000 observations and 500 features. The signal is $f(\mathbf{X}_1, \ldots, \mathbf{X}_p) = \sum_{j=1}^{20} \mathbf{X}_j + \sum_{j=1}^{20} 3(5\mathbf{X}_j^3 - 3\mathbf{X}_j)/4 + \sum_{j=21}^{28}(3\mathbf{X}_j^2 - 1)$. The first 20 features have both linear and non-linear components featuring in the signal, while the next 8 features only have non-linear components in the signal. The SNR of the overall response is set to 1, with each of the three sums in

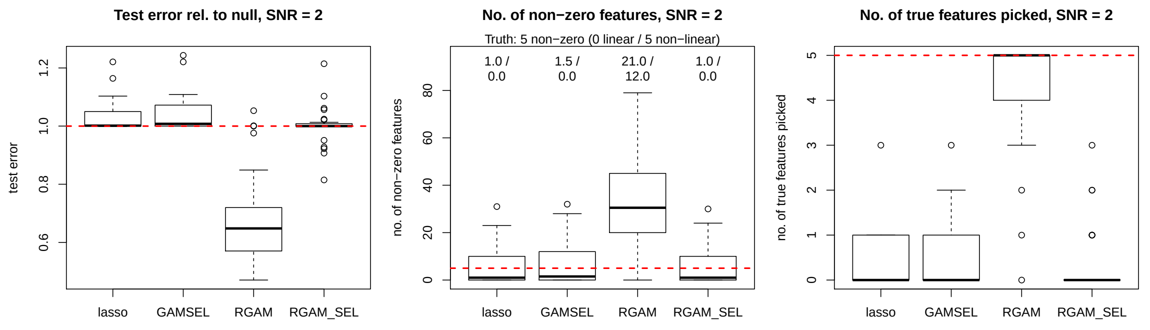

Figure 4.4: *Fully non-linear setting: $n = 100$, $p = 200$, $SNR = 2$. The left-most plot presents test error as a fraction of the null model's test error. The middle plot presents the number of features each model selected, with the two numbers on top of the boxplots being the median number of linear components and non-linear components selected. The right-most plot presents the number of true features each method selected, with the total number of true features indicated by the dotted red line. Only RGAM is able to outperform the null model as there are no linear effects in the true signal.*

the expression above having roughly equal SNR.

The results are shown in Figure 4.5. In this setting RGAM clearly outperforms all the other methods, and performs well despite low SNR. This is partially due to its ability to pick out the non-linear features that do not have a corresponding linear component in the signal. RGAM_SEL exhibits roughly the same test error performance as GAMSEL, but selects much fewer linear and non-linear components in its predictive model.

4.5.4 Prostate cancer dataset

We apply RGAM to a microarray dataset from a prostate cancer study carried out by Singh et al. (2002), and which was analyzed in Efron (2012). The data consists of expression levels for 6,033 genes for 102 men. 50 men were normal control subjects while the remaining 52 men were prostate cancer patients. The goal is to predict which subjects had prostate cancer based on the gene expression levels.

We compare RGAM's cross-validated performance with that of the lasso and GAMSEL. Each of these methods was run on a path of λ values, with other hyperparameters set to their default values. The fitting times for GAMSEL, RGAM and RGAM_SEL were 72, 32 and 3.5 seconds respectively. The results are shown in Figure 4.6. For the same model size, both versions of RGAM outperform the lasso and GAMSEL in terms of both cross-validated deviance and cross-validated area under the curve (AUC).

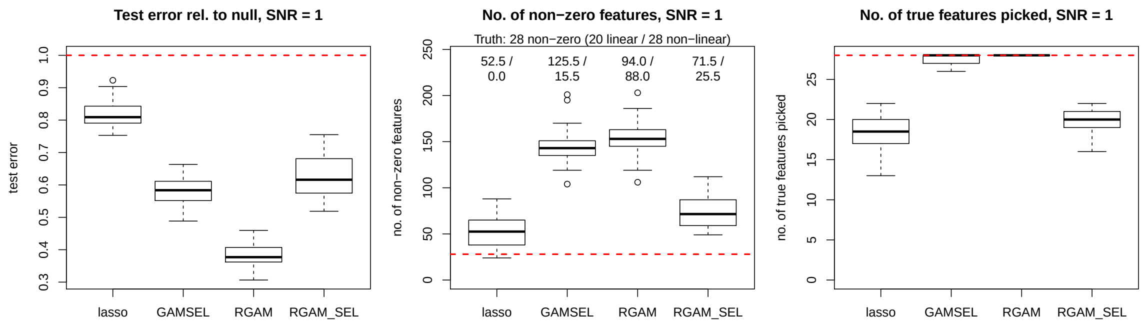

Figure 4.5: *Large setting, with hierarchical and non-hierarchical non-linear signals: $n = 1,000$, $p = 500$, $SNR = 1$. The left-most plot presents test error as a fraction of the null model's test error. The middle plot presents the number of features each model selected, with the two numbers on top of the boxplots being the median number of linear components and non-linear components selected. The right-most plot presents the number of true features each method selected, with the total number of true features indicated by the dotted red line. RGAM performs best in terms of test error; RGAM_SEL is comparable to GAMSEL but selects much fewer linear and non-linear components.*

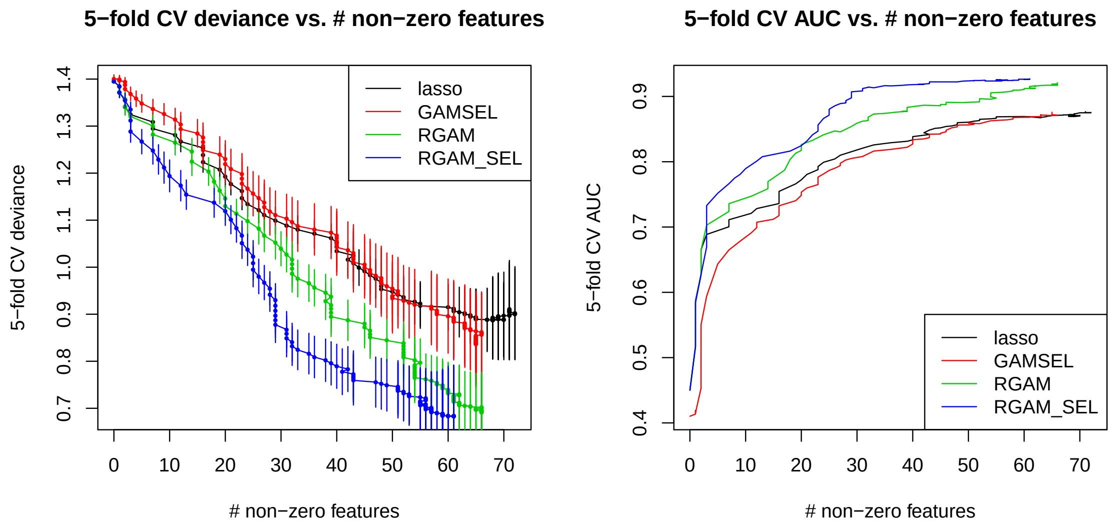

Figure 4.6: *5-fold cross-validation results for the prostate cancer dataset. The left panel shows the cross-validated deviance (with error bars showing ±1 standard deviation) while the right panel shows the cross-validated area under the curve (AUC). The x-axis represents the number of features selected at each value in the lambda path.*

4.6 Degrees of freedom

When introducing RGAM, we claimed that, intuitively, the way in which the non-linear features are constructed in Step 2 gives the non-linear components less degrees of freedom than giving Step 3 d spline basis functions for each $\mathbf{X}_j$. The degrees of freedom measures the flexibility of the fit: the larger the degrees of freedom, the more closely the fit matches the response values. We explore this claim in more detail here.

Given a vector of response values $\mathbf{y}$ with corresponding fits $\hat{\mathbf{y}}$, Efron (1986) defines the degrees of freedom as

$$\text{df}(\hat{y}) = \frac{\sum_{i=1}^n \text{Cov}(y_i, \hat{y}_i)}{\sigma^2}.$$

Here, n is the number of observations, and y_i and $\hat{y}_i$ denote the ith entry of $\mathbf{y}$ and $\hat{\mathbf{y}}$ respectively. We can estimate this quantity via Monte Carlo simulation under different data generating processes. For each simulation setting, assume that we have p features $\mathbf{X}_1, \dots, \mathbf{X}_p \in \mathbb{R}^n$ whose values we consider to be fixed (i.e. not random), and assume the true data generating process for the response is

$$\mathbf{y}^* = \boldsymbol{\mu} + \sigma \mathbf{z}, \qquad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{n \times n}), \tag{4.6}$$

where $\boldsymbol{\mu} \in \mathbb{R}^n$ is a function of the $\mathbf{X}_j$'s, and hence is considered fixed for that simulation setting. (Simulation settings differ in how $\boldsymbol{\mu}$ is generated from the $\mathbf{X}_j$'s and in the value of σ.) For each simulation setting, we generate response vectors $\mathbf{y}^{*b}$ according to (4.6) for $b = 1, \dots, B$, with B representing the number of Monte Carlo samples we are taking. For each $\mathbf{y}^{*b}$, we fit a predictive model to this data, generating predictions $\hat{\mathbf{y}}^{*b}$. This gives us the Monte Carlo estimate

$$\text{df} \approx \sum_{i=1}^n \widehat{\text{Cov}}(\hat{y}_i^*, y_i^*)/\sigma^2,$$
$$\widehat{\text{Cov}}(\hat{y}_i^*, y_i^*) = \frac{1}{B} \sum_{b=1}^B [\hat{y}_i^{*b} - a_i][y_i^{*b} - \mu_i],$$

where the a_i's can be any fixed known constants (usually taken to be 0).

We compare the estimated degrees of freedom for unpenalized versions of RGAM and GAMSEL, i.e. setting the hyperparameter $\lambda = 0$. Figure 4.7 shows the estimated degrees of freedom for the unpenalized procedures (with degrees of freedom $d = 4$) and OLS of $\mathbf{y}$ on the $\mathbf{X}_j$'s for three different settings. (For GAMSEL, each feature was given 6 basis functions; the default value for `gamsel` in the **gamsel** package is 10.) As predicted in theory, OLS on the $\mathbf{X}_j$'s (with intercept) has $p + 1$ degrees of freedom. The degrees of freedom for unpenalized GAMSEL seems to be relatively constant at roughly p times the value of the degrees of freedom hyperparameter, even as the non-linear component's contribution to the SNR of the signal changes. Unpenalized RGAM has roughly the same degrees of freedom when the true underlying signal is completely linear. As the proportion of SNR in the true underlying signal coming from the non-linear component increases, RGAM's degrees of freedom decreases. We currently do not have a good explanation for this phenomenon. The degrees of freedom for unpenalized RGAM_SEL is substantially lower than both that of unpenalized GAMSEL and unpenalized RGAM.

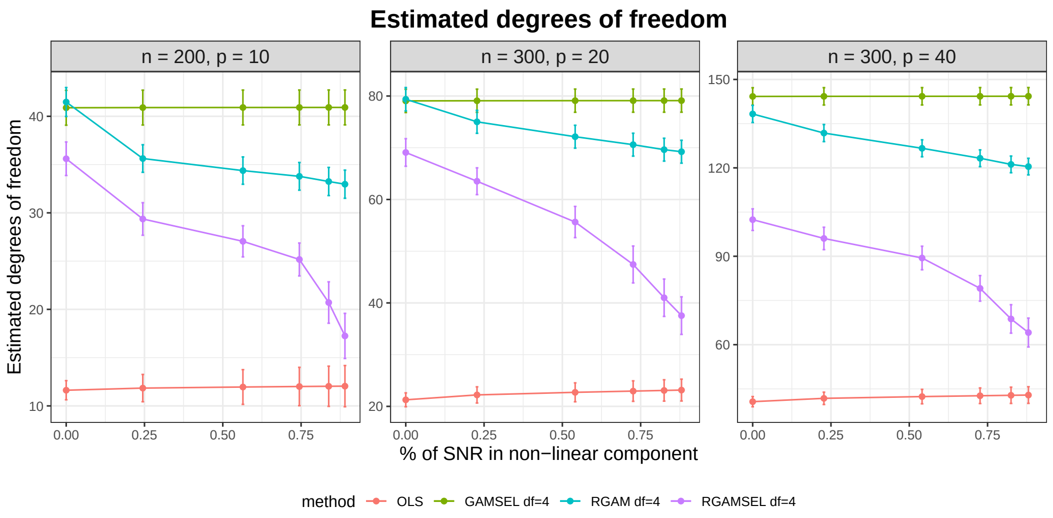

Figure 4.7: *Estimated degrees of freedom across three different* (n, p) *settings. The points are the Monte Carlo estimates from* $B = 100$ *simulation runs, and the error bars are* ± 1 *standard deviation for the sample mean. For unpenalized GAMSEL, each feature was given 6 basis functions. Unpenalized RGAM has smaller degrees of freedom than unpenalized GAMSEL. As the non-linear component of the true signal increases (in terms of SNR), unpenalized RGAM's degrees of freedom appears to decrease.*

4.7 Discussion

Reluctant generalized additive modeling (RGAM) is a three-step algorithm for fitting sparse GAMs. The model's prediction is allowed to vary linearly or non-linearly with each input variable. RGAM is guided by the reluctant non-linear selection principle, preferring linear effects over non-linear effects, only including the latter if they add to predictive performance. Unlike existing methods for sparse GAMs, RGAM can be extended easily to binary, count and survival data.

The three-step framework of Algorithm 5 is extremely flexible. As previously noted, one may replace the lasso method in Steps 1 and 3 with a regression method of one's choosing. We note that Step 2 is highly customizable as well: we seek to model the residual in this step, and we can use any method to do so. In our software implementation we model the residual with a cubic smoothing spline for each $\mathbf{X}_j$; other spline methods could be used. If we believe that there are discontinuities in the relationship between the response $\mathbf{y}$ and $\mathbf{X}_j$, we could model the residual as a piece-wise constant function of $\mathbf{X}_j$. There are even more possibilities if we are willing to allow interactions between different input variables. For example, we could combine RGAM with reluctant interaction modeling by adding the interaction terms chosen by reluctant interaction modeling in Step 3 of Algorithm 5. Another possibility is to fit the residual to random trees, much like a random forest (Breiman, 2001); Step 3 then selects the most appropriate linear effects and trees for the final model. We leave the implementation and exploration of these more complex methods for future work.

An R language package **relgam** which implements RGAM is available on the CRAN repository.

Chapter 5

Extending the glmnet Package

5.1 Introduction

In this chapter we turn to the practical matter of computing the solution for the elastic net penalty. Recall that our data is of the form $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$, where $y_i \in \mathbb{R}$ is the target and $\mathbf{x}_i = (x_{i,1}, \dots, x_{i,p})^T \in \mathbb{R}^p$ is a vector of potential predictors. (In this chapter, we do not assume that the features and responses are mean-centered at zero.) The elastic net model assumes that the response can be modeled as a linear combination of the covariates, i.e. $y_i = \beta_0 + \mathbf{x}_i^T \boldsymbol{\beta}$ for some coefficient vector $\boldsymbol{\beta} \in \mathbb{R}^p$ and intercept $\beta_0 \in \mathbb{R}$, which are the solution to the following minimization problem:

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \underset{(\beta_0, \boldsymbol{\beta}) \in \mathbb{R}^{p+1}}{\operatorname{argmin}} \left[\frac{1}{2n} \sum_{i=1}^n (y_i - \beta_0 - \mathbf{x}_i^T \boldsymbol{\beta})^2 + \lambda \left(\frac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2 + \alpha \|\boldsymbol{\beta}\|_1 \right) \right], \tag{5.1}$$

where $\lambda \geq 0$ is a tuning parameter and $\alpha \in [0, 1]$ is a higher level hyperparameter[1]. We always fit a path of models in λ, but set a value of α depending on the type of prediction model we want. For example, if we want ridge regression (Hoerl and Kennard, 1970) we set $\alpha = 0$ and if we want the lasso (Tibshirani, 1996) we set $\alpha = 1$. If we want a sparse model but are worried about correlations between features, we might set α close to but not equal to 1. The final value of λ is usually chosen via cross-validation: we select the coefficients corresponding to the λ value giving smallest cross-validated error as the final model.

The elastic net can be extended easily to generalized linear models (GLMs) (Nelder and Wedderburn, 1972) and Cox proportional hazards models (Cox, 1972). Instead of solving the minimization problem (5.1), the RSS term in the objective function is replaced with a negative log-likelihood term

[1]If the square were removed from the ℓ_2-norm penalty, it would be more natural to have $1-\alpha$ instead of $(1-\alpha)/2$ as its mixing parameter. The factor of $1/2$ compensates for the fact that a squared ℓ_2-norm penalty is used, in the sense that the gradient of the penalty with respect to β can be seen as a convex combination of the ℓ_1 and ℓ_2 penalty terms. We note also that there is a one-to-one correspondence between these two parameterizations for the penalty.

or a negative log partial likelihood term respectively.

The **glmnet** R package (Friedman et al., 2010) contains efficient functions for computing the elastic net solution for an entire path of values $\lambda_1 > \cdots > \lambda_m$. The minimization problems are solved via cyclic coordinate descent (van der Kooij, 2007), with the core routines programmed in Fortran for computational efficiency. Earlier versions of the package contained specialized Fortran subroutines for a handful of popular GLMs and the Cox model for right-censored survival data. The package includes functions for performing K-fold cross-validation (CV), plotting coefficient paths and CV errors, and predicting on future data. The package can also accept the predictor matrix in sparse matrix format: this is especially useful in certain applications where the predictor matrix is both large and sparse. In particular, this means that we can fit unpenalized GLMs with sparse predictor matrices, something the `glm` function in the **stats** package cannot do.

From version 4.1 and later, **glmnet** is able to compute the elastic net regularization path for all GLMs, Cox models with (start, stop] data and strata, and a simplified version of the relaxed lasso (Hastie et al., 2020).

Several other packages currently exist in R for computing the lasso and elastic net regularization paths, although it is not always the main focus of the package. The model families and penalties supported varies widely from package to package: see Tables 5.1 and 5.2 for a summary. The **glmnet** package effectively supersedes the **elasticnet** (Zou and Hastie, 2020), **glmpath** (Park and Hastie, 2018) and **lars** (Hastie and Efron, 2013) packages. The **lasso2** package (Lokhorst et al., 2020) solves the constrained version of the lasso for a single constraint value (as opposed to a path), and is the only other package that supports GLMs other than the Gaussian, binomial and Poisson GLMs with canonical links. The **ncvreg** package (Breheny and Huang, 2011) can compute the solution paths for the lasso penalty, minimax concave penalty (MCP) (Zhang, 2010) and smoothly clipped absolute deviation (SCAD) penalty (Fan and Li, 2001), but not for the elastic net penalty. The package which is closest to **glmnet** in functionality is the **penalized** package (Goeman et al., 2018). It has the ability to use R's formula syntax to define the model and it can compute the fused lasso solution (Tibshirani et al., 2005) as well. However, it is more limited in the model families it supports. The **biglasso** package (Zeng and Breheny, 2017) specializes in fitting the elastic net regularization path for big data that cannot be loaded into memory. The **bmrm** package (Prados, 2019) computes the solution for a wide variety of loss functions with ℓ_1 or ℓ_2 regularization, but not both ℓ_1 and ℓ_2 regularization at the same time and only for a single point on the regularization path. For survival analysis, the **ahaz** package (Gorst-Rasmussen and Scheike, 2012) can fit semiparametric additive hazards models, which the Cox model is a special case of, with the lasso penalty but not the elastic net penalty. It works with both right-censored and (start, stop] data, but not with tied survival times. Finally, the **relaxo** package (Meinshausen, 2012) computes the solution path for the original relaxed lasso (Meinshausen, 2007) for the OLS model only.

Friedman et al. (2010) gives details on how the **glmnet** package computes the elastic net solution

	GLMs				Cox models		
	Gaussian	Binomial	Poisson	All other GLMs	Right-censored data	(start, stop] data	Strata
glmnet	✓	✓	✓	✓	✓	✓	✓
ahaz					✓	✓	
biglasso	✓	✓			✓		
bmrm	✓	✓					
elasticnet	✓						
glmpath	✓	✓	✓		✓		
lars	✓						
lasso2	✓	✓	✓	✓			
ncvreg	✓	✓	✓		✓		
penalized	✓	✓	✓		✓		
relaxo	✓						

Table 5.1: Table of model families that each R package can fit. Only the **glmnet** and **lasso2** packages can fit GLMs other than the Gaussian, binomial and Poisson GLMs with canonical links. Only the **glmnet** and **ahaz** packages can fit Cox models with (start, stop] data, and only **glmnet** can fit stratified Cox models.

	Lasso (ℓ_1)	Ridge (ℓ_2-squared)	Elastic net	Other penalties	Formula syntax?
glmnet	✓	✓	✓		
ahaz	✓			SCAD	
biglasso	✓	✓	✓		
bmrm	✓	✓			
elasticnet	✓	✓	✓		
glmpath	✓				
lars	✓				
lasso2	✓				✓
ncvreg	✓			MCP, SCAD	
penalized	✓	✓	✓	Fused lasso	✓
relaxo	✓				

Table 5.2: Table of penalties that each R package can fit, as well as whether the package works with R's formula syntax. MCP refers to the minimax concave penalty, and SCAD refers to the smoothly clipped absolute deviation penalty. **glmnet** does not support formula syntax as the primary use case for regularized models is when the design matrix X is "wide", i.e., many more features than observations. Formula syntax also does not mix well with column-specific function arguments such as `exclude`, `penalty.factor`, `lower.limits` and `upper.limits`.

for ordinary least squares regression, logistic regression and multinomial logistic regression, while Simon et al. (2011) explains how the package fits regularized Cox models for right-censored data. This chapter builds on these two earlier works. In Section 5.2, we provide a brief overview of GLMs, explain how the elastic net penalty can be applied to all GLMs and give details on the software implementation. In Section 5.3, we describe extensions to Cox models with (start, stop]

GLM family / Regression type	Response type	Representation in R
Gaussian	$\mathbb{R}$	`gaussian()`
Logistic	$\{0, 1\}$	`binomial()`
Probit	$\{0, 1\}$	`binomial(link = "probit")`
Quasi-Binomial	$\{0, 1\}$	`quasibinomial()`
Poisson	$\mathbb{N}_0 = \{0, 1, \dots\}$	`poisson()`
Quasi-Poisson	$\mathbb{N}_0$	`quasipoisson()`
Negative binomial	$\mathbb{N}_0$	`MASS::negative.binomial(theta = 3)`
Gamma	$\mathbb{R}_+ = [0, \infty)$	`Gamma()`
Inverse Gaussian	$\mathbb{R}_+$	`inverse.gaussian()`
Tweedie	Depends on variance power parameter	`statmod::tweedie()`

Table 5.3: Examples of generalized linear models (GLMs) and their representations in R.

data and strata. We conclude with a summary. For more code examples and greater detail on how to use functions in the **glmnet** package, see the vignettes on the official **glmnet** website `https://glmnet.stanford.edu/`.

5.2 Regularized generalized linear models

5.2.1 Overview of generalized linear models

Generalized linear models (GLMs) (Nelder and Wedderburn, 1972) are a simple but powerful extension of OLS. A GLM consists of 3 parts:

- A linear predictor: $\eta_i = \mathbf{x}_i^T \boldsymbol{\beta}$,
- A link function: $\eta_i = g(\mu_i)$, and
- A variance function as a function of the mean: $V = V(\mu_i)$.

The user gets to specify the link function g and the variance function V. For one-dimensional exponential families, the family determines the variance function, which, along with the link, are sufficient to specify a GLM. More generally, modeling can proceed once the link and variance functions are specified via a quasi-likelihood approach (see McCullagh and Nelder (1983) for details); this is the approach taken by the quasi-binomial and quasi-Poisson models. The OLS model is a special case, with link $g(x) = x$ and constant variance function $V(\mu) = \sigma^2$ for some constant σ^2. More examples of GLMs are listed in Table 5.3.

The GLM parameter $\boldsymbol{\beta}$ is determined by maximum likelihood estimation. Unlike OLS, there is no closed form solution for $\hat{\boldsymbol{\beta}}$. Rather, it is typically computed via an iteratively reweighted least squares (IRLS) algorithm known as *Fisher scoring*. In each iteration of the algorithm we make a quadratic approximation to the negative log-likelihood (NLL), reducing the minimization problem to a weighted least squares (WLS) problem. For GLMs with canonical link functions, the negative log-likelihood is convex in $\boldsymbol{\beta}$, Fisher scoring is equivalent to the Newton-Raphson method and is guaranteed to converge to a global minimum. For GLMs with non-canonical links, the negative log-likelihood is not guaranteed to be convex[2]. Also, Fisher scoring is no longer equivalent to the Newton-Raphson method and is only guaranteed to converge to a local minimum.

It is easy to fit GLMs in R using the `glm` function from the **stats** package; the user can specify the GLM to be fit using `family` objects. These objects capture details of the GLM such as the link function and the variance function. For example, the code below shows the `family` object associated with probit regression model:

```
R> class(binomial(link = "probit"))

[1] "family"

R> str(binomial(link = "probit"))

List of 12
 $ family    : chr "binomial"
 $ link      : chr "probit"
 $ linkfun   : function (mu)
 $ linkinv   : function (eta)
 $ variance  : function (mu)
 $ dev.resids: function (y, mu, wt)
 $ aic       : function (y, n, mu, wt, dev)
 $ mu.eta    : function (eta)
 $ initialize: language ... # code to set up objects needed for the family
 $ validmu   : function (mu)
 $ valideta  : function (eta)
 $ simulate  : function (object, nsim)
 - attr(*, "class")= chr "family"
```

The `linkfun`, `linkinv`, `variance` and `mu.eta` functions are used in fitting the GLM, and the `dev.resids` function is used in computing the deviance of the resulting model. By passing a class `"family"` object to the `family` argument of a `glm` call, `glm` has all the information it needs to fit the model. Here is an example of how one can fit a probit regression model in R:

```
R> library("glmnet")
```

[2]It is not true that the negative log-likelihood is always non-convex for non-canonical links. For example, it can be shown via direct computation that the negative log-likelihood for probit regression is convex in $\boldsymbol{\beta}$.

```
R> data("BinomialExample")
R> fit <- glm(y ~ x, data =  BinomialExample,
+    family = binomial(link = "probit"))
```

5.2.2 Extending the elastic net to all GLM families

To extend the elastic net to GLMs, we replace the RSS term in (5.1) with a negative log-likelihood term:

$$(\hat{\beta}_0, \hat{\boldsymbol{\beta}}) = \underset{(\beta_0, \boldsymbol{\beta}) \in \mathbb{R}^{p+1}}{\text{argmin}} \left[-\frac{1}{n} \sum_{i=1}^n \ell \left(y_i, \beta_0 + \mathbf{x}_i^T \boldsymbol{\beta} \right) + \lambda \left(\frac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2 + \alpha \|\boldsymbol{\beta}\|_1 \right) \right], \tag{5.2}$$

where $\ell \left(y_i, \beta_0 + \mathbf{x}_j^T \boldsymbol{\beta} \right)$ is the log-likelihood term associated with observation i. We can apply the same strategy as for GLMs to minimize this objective function. The key difference is that instead of solving a WLS problem in each iteration, we solve a penalized WLS problem.

The algorithm for solving (5.2) for a path of λ values is described in Algorithm 6. Note that in Step 2(a), we initialize the solution for $\lambda = \lambda_k$ at the solution obtained for $\lambda = \lambda_{k-1}$. This is known as a *warm start*: since we expect the solution at these two λ values to be similar, the algorithm will likely require fewer iterations than if we initialized the solution at zero.

Algorithm 6 *Fitting GLMs with elastic net penalty*

```
1. Select a value of $\alpha \in [0,1]$ and a sequence of $\lambda$ values $\lambda_1 > \ldots > \lambda_m$.
2. For $k = 1, \ldots, m$:
   (a) Initialize $(\hat{\beta}_0^{(0)}(\lambda_k), \hat{\boldsymbol{\beta}}^{(0)}(\lambda_k)) = (\hat{\beta}_0(\lambda_{k-1}), \hat{\boldsymbol{\beta}}(\lambda_{k-1}))$. For $k = 1$, initialize $(\hat{\beta}_0^{(0)}(\lambda_k), \hat{\boldsymbol{\beta}}^{(0)}(\lambda_k)) = (0, \mathbf{0})$. (Here, $(\hat{\beta}_0(\lambda_k), \hat{\boldsymbol{\beta}}(\lambda_k))$ denotes the elastic net solution at $\lambda = \lambda_k$.)
   (b) For $t = 0, 1, \ldots$ until convergence:
       i. For $i = 1, \ldots, n$, compute $\eta_i^{(t)} = \hat{\beta}_0^{(t)}(\lambda_k) + \hat{\boldsymbol{\beta}}^{(t)}(\lambda_k)^T \mathbf{x}_i$ and $\mu_i^{(t)} = g^{-1}\left(\eta_i^{(t)}\right)$.
       ii. For $i = 1, \ldots, n$, compute working responses and weights
           $z_i^{(t)} = \eta_i^{(t)} + \left(y_i - \mu_i^{(t)}\right) / \frac{d\mu_i^{(t)}}{d\eta_i^{(t)}}, \quad w_i^{(t)} = \left(\frac{d\mu_i^{(t)}}{d\eta_i^{(t)}}\right)^2 / V\left(\mu_i^{(t)}\right).$   (5.3)
       iii. Solve the penalized WLS problem
           $(\hat{\beta}_0^{(t+1)}(\lambda_k), \hat{\boldsymbol{\beta}}^{(t+1)}(\lambda_k)) = \underset{(\beta_0, \boldsymbol{\beta}) \in \mathbb{R}^{p+1}}{\text{argmin}} \left[ \frac{1}{2n} \sum_{i=1}^n w_i^{(t)} \left( z_i^{(t)} - \beta_0 - \mathbf{x}_i^T \boldsymbol{\beta} \right)^2 + \lambda_k \left( \frac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2 + \alpha \|\boldsymbol{\beta}\|_1 \right) \right].$   (5.4)
```

5.2.3 Implementation details

There are two main approaches we can take in implementing Algorithm 6. In the original implementation of **glmnet**, the entire algorithm was implemented in Fortran for specific GLM families. In version 4.0 and later, we added a second implementation which implemented just the computational bottleneck, the penalized WLS problem in Step 2(b)iii, in Fortran, with the rest of the algorithm implemented in R. Here are the relative merits and disadvantages of the second approach compared to the first:

- ✓ Because the formulas for the working weights and responses in (5.3) are specific to each GLM, the first approach requires a new Fortran subroutine for each GLM family. This is tedious to manage, and also means that users cannot fit regularized models for their bespoke GLM families. The second approach allows the user to pass a class `"family"` object to `glmnet`: the working weights and responses can then be computed in R before the Fortran subroutine solves the resulting penalized WLS problem.

- ✓ As written, Algorithm 6 is a proximal Newton algorithm with a constant step size of 1, and hence it may not converge in certain cases. To ensure convergence, we can implement step-size halving after Step 2(b)iii: as long as the objective function (5.2) is not decreasing, set $\hat{\boldsymbol{\beta}}^{(t+1)}(\lambda_k) \leftarrow \hat{\boldsymbol{\beta}}^{(t)}(\lambda_k) + \left[\hat{\boldsymbol{\beta}}^{(t+1)}(\lambda_k) - \hat{\boldsymbol{\beta}}^{(t)}(\lambda_k)\right]/2$ (with a similar formula for the intercept). Since the objective function involves a log-likelihood term, the formula for the objective function differs across GLMs, and the first approach has to maintain different subroutines for step-size halving. For the second approach, we can write a single function that takes in the class `"family"` object (along with other necessary parameters) and returns the objective function value.

- × It is computationally less efficient than the first approach because (i) R is generally slower than Fortran, and (ii) there is overhead associated with constant switching between R and Fortran. Some timing comparisons for Gaussian and logistic regression with the default parameters are presented in Figure 5.1. The second approach is 10 to 15 times as slow than the first approach.

- × Since each GLM family has its own set of Fortran subroutines in the first approach, it allows for special computational tricks to be employed in each situation. For example, with `family = "gaussian"`, the predictors can be centered once upfront to have zero mean and Algorithm 6 can be run ignoring the intercept term.

We stress that both approaches have been implemented in **glmnet**. Users should use the first implementation for the most popular GLM families including OLS (Gaussian regression), logistic regression and Poisson regression (see `glmnet`'s documentation for the full list of such families), and use the second implementation for all other GLM families. For example, the code below shows two equivalent ways to fit a regularized Poisson regression model:

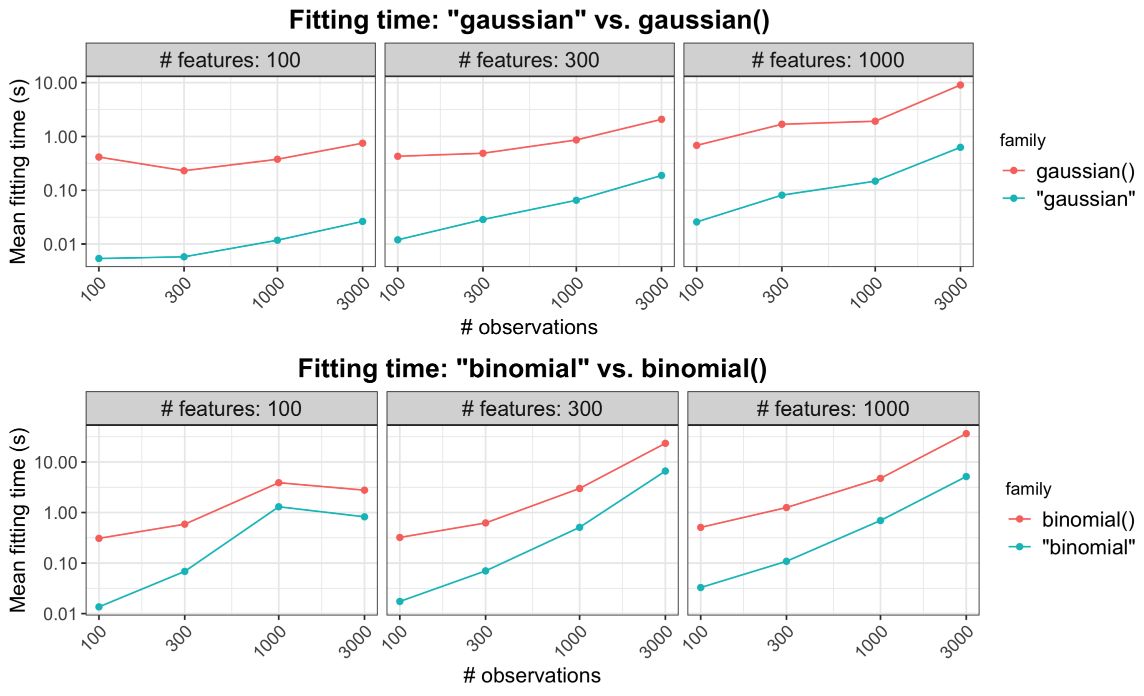

Figure 5.1: The top plot compares model fitting times for `family = "gaussian"` and `family = gaussian()` for a range of problem sizes, while the plot below compares that for `family = "binomial"` and `family = binomial()`. Each point is the mean of 5 simulation runs. Note that both the x and y axes are on the log scale.

```
R> data("PoissonExample")
R> x <- PoissonExample$x
R> y <- PoissonExample$y
R> fit1 <- glmnet(x, y, family = "poisson")
R> fit2 <- glmnet(x, y, family = poisson())
R> cbind(coef(fit1, s = 0.1), coef(fit2, s = 0.1))

21 x 2 sparse Matrix of class "dgCMatrix"
                      1            1
(Intercept)  0.097117743  0.096014186
V1           0.600969943  0.601083898
V2          -0.963561440 -0.963849601
...
V18          .            .
V19         -0.016139939 -0.016215082
V20          0.011030660  0.010915409
```

The first call specifies the GLM family as a character string to the `family` argument, invoking the first implementation. The second call passes a class `"family"` object to the `family` argument instead of a character string, invoking the second implementation. One would never run the second call in practice though, as it returns the same result as the first call but takes longer to fit. (We note that the coefficients returned for the two models differ slightly because they use different fitting algorithms and have different convergence criteria. We can tighten the convergence criteria by lowering the `thresh` argument: when we do this we will have greater agreement between the two models.)

The example below fits a regularized quasi-Poisson model that allows for overdispersion, a family that is only available via the second approach:

```
R> glmnet(x, y, family = quasipoisson())
```

5.2.4 Details on the penalized WLS subroutine

Since the penalized WLS problem in Step 2(b)iii of Algorithm 6 is the computational bottleneck, we elected to implement it in Fortran. Concretely, the subroutine solves the problem

$$\underset{(\beta_0,\boldsymbol{\beta})\in\mathbb{R}^{p+1}}{\text{minimize}} \quad \frac{1}{2n}\sum_{i=1}^n w_i\left(z_i - \beta_0 - \mathbf{x}_i^T\boldsymbol{\beta}\right)^2 + \lambda_k \sum_{j=1}^p \gamma_j\left(\frac{1-\alpha}{2}\beta_j^2 + \alpha|\beta_j|\right) \tag{5.5}$$

$$\text{subject to} \quad L_j \le \beta_j \le U_j, \quad j = 1, \dots, p. \tag{5.6}$$

This is the same problem as (5.4) except for two things. First, the penalty placed on each coefficient β_j has its own multiplicative factor γ_j. (Note that (5.6) reduces to (5.4) if $\gamma_j = 1$ for all j, which is the default value for the `glmnet` function.) This allows the user to place different penalty weights on the coefficients. An instance where this is especially useful is when the user always wants to include feature j in the model: in that case the user could set $\gamma_j = 0$ so that β_j is unpenalized. Second, the coefficient β_j is constrained to lie in the interval $[L_j, U_j]$. (`glmnet`'s default is $L_j = -\infty$ and $U_j = \infty$ for all j, i.e. no constraints on the coefficients.) One example where these constraints are useful is when we want a certain β_j to always be non-negative or always non-positive.

The Fortran subroutine solves (5.6) by cyclic coordinate descent: see Friedman et al. (2010) for details. Here we describe one major computational trick that was not covered in that paper: the application of *strong rules* (Tibshirani et al., 2012).

In each iteration of cyclic coordinate descent, the solver has to loop through all p features to update the corresponding model coefficients. This can be time-consuming if p is large, and is potentially wasteful if the solution is sparse: most of the β_j would remain at zero. If we know a priori which predictors will be "active" at the solution (i.e. have $\beta_j \neq 0$), we could perform cyclic coordinate descent on just those coefficients and leave the others untouched. The set of "active" predictors is known as the *active set*. Strong rules are a simple yet powerful heuristic for guessing what the active set is, and can be combined with the Karush-Kuhn-Tucker (KKT) conditions to ensure that we get the exact solution. (The set of predictors determined by the strong rules is known as the *strong set*.) We describe the use of strong rules in solving (5.6) fully in Algorithm 7.

Algorithm 7 *Solving penalized WLS* (5.6) *with strong rules*

Assume that we are trying to solve for $\hat{\boldsymbol{\beta}}(\lambda_k)$ for some $k = 1, \ldots, m$, and that we have already computed $\hat{\boldsymbol{\beta}}(\lambda_{k-1})$. (If $k = 1$, set $\hat{\boldsymbol{\beta}}(\lambda_{k-1}) = \mathbf{0}$.)

1. Initialize the strong set $\mathcal{S}_{\lambda_k} = \{j : \hat{\beta}(\lambda_{k-1})_j \neq 0\}$.
2. Check the strong rules: for $j = 1, \ldots, p$, include j in $\mathcal{S}_{\lambda_k}$ if
$$\left| \mathbf{x}_j^T \left\{ \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}(\lambda_{k-1}) \right\} \right| > \alpha \left[\lambda_k - (\lambda_{k-1} - \lambda_k) \right] \gamma_j.$$
3. Perform cyclic coordinate descent only for features in $\mathcal{S}_{\lambda_k}$.
4. Check that the KKT conditions hold for each $j = 1, \ldots, p$. If the conditions hold for all j, we have the exact solution. If the conditions do not hold for some features, include them in the strong set $\mathcal{S}_{\lambda_k}$ and go back to Step 3.

Finally, we note that in some applications, the design matrix $\mathbf{X}$ is sparse. In these settings, computational savings can be reaped by representing $\mathbf{X}$ in a sparse matrix format and performing matrix manipulations with this form. To leverage this property of the data, we have a separate Fortran subroutine that solves (5.6) when $\mathbf{X}$ is in sparse matrix format.

5.2.5 Other useful functionality

In this section, we mention other use functionality that the **glmnet** package provides for fitting elastic net models.

For fixed α, `glmnet` solves (5.2) for a path of λ values. While the user has the option of specifying this path of values using the `lambda` option, it is recommended that the user let `glmnet` compute the sequence on its own. `glmnet` uses the arguments passed to it to determine the value of λ_{max}, defined to be the smallest value of λ such that the estimated coefficients would be all equal to zero[3]. The program then computes λ_{min} such that the ratio $\lambda_{min}/\lambda_{max}$ is equal to `lambda.min.ratio` (default 10^{-2} if the number of variables exceeds the number of observations, 10^{-4} otherwise). Model (5.2) is then fit for `nlambda` λ values (default 100) starting at λ_{max} and ending at λ_{min} which are equally spaced on the log scale.

In practice, it common to choose the value of λ via cross-validation (CV). The `cv.glmnet` function is a convenience function that runs CV for the λ tuning parameter. The returned object has class `"cv.glmnet"`, which comes equipped with `plot`, `coef` and `predict` methods. The `plot` method produces a plot of CV error against λ (see Figure 5.2 for an example.) As mentioned earlier, we prefer to think of α as a higher level hyperparameter whose value depends on the type of prediction model we want. Nevertheless, the code below shows how the user can perform CV for α manually using a for loop and extract the value of α giving the smallest CV error. Care must be taken to ensure that the same CV folds are used across runs for the CV errors to be comparable.

```
R> data("QuickStartExample")
R> x <- QuickStartExample$x
R> y <- QuickStartExample$y
R> alphas <- c(1, 0.8, 0.5, 0.2, 0)
R> fits <- list()
R> set.seed(1)
R> fits[[1]] <- cv.glmnet(x, y, keep = TRUE)
R> foldid <- fits[[1]]$foldid
R> for (i in 2:length(alphas)) {
+    fits[[i]] <- cv.glmnet(x, y, alpha = alphas[i], foldid = foldid)
+  }
R> best_cvm <- unlist(lapply(fits, function(fit) min(fit$cvm)))
R> alphas[which.min(best_cvm)]

[1] 1
```

[3]We note that when $\alpha = 0$, λ_{max} is infinite, i.e. all coefficients will always be non-zero for finite λ. To avoid such extreme values of λ_{max}, if $\alpha < 0.001$ we return the λ_{max} value for $\alpha = 0.001$.

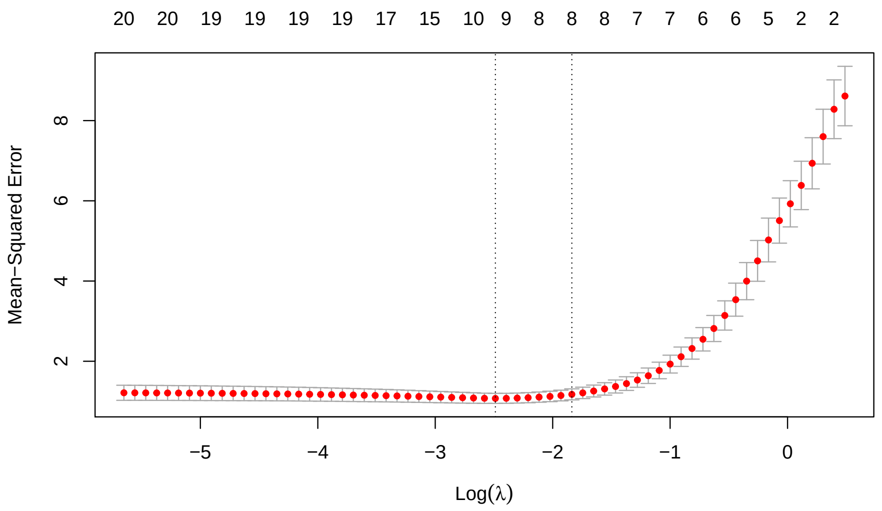

Figure 5.2: Example output for plotting a `cv.glmnet` object: a plot of CV error against $\log(\lambda)$. The error bars correspond to ± 1 standard error. The left vertical line corresponds to the minimum error while the right vertical line corresponds to the largest value of λ such that the CV error is within one standard error of the minimum. The top of the plot is annotated with the size of the models, i.e. the number of predictors with non-zero coefficient.

The returned `cv.glmnet` object contains estimated standard errors for the model CV error at each λ value. (We note that the method for obtaining these estimates is crude, and the estimates are generally too small due to correlations across CV folds; see Bates et al. (2021).) By default, the `predict` method returns predictions for the model at the `"lambda.1se"` value, i.e. the value of λ that gives the most regularized model such that the CV error is within one standard error of the minimum. To get predictions at the λ value which gives the minimum CV error, the `s = "lambda.min"` argument is passed.

```
R> set.seed(1)
R> cfit <- cv.glmnet(x, y)
R> predict(cfit, x)
```

```
                  1
  [1,] -1.33820168
  [2,]  2.50786936
  [3,]  0.56371947
...
 [98,] -2.59959545
 [99,]  4.68516614
[100,] -0.75782622
```

```
R> predict(cfit, x, s = "lambda.min")
```

```
                 1
  [1,] -1.36474902
  [2,]  2.56860130
  [3,]  0.57058790
...
 [98,] -2.74205637
 [99,]  5.12461067
[100,] -1.07513903
```

The `glmnet` and `cv.glmnet` functions have an `exclude` argument which accepts a vector of indices, indicating which variables should be excluded from the fit, i.e., get zeros for their coefficients. More intriguingly, the `exclude` argument can also accept a function. The idea is that varibles can be filtered based on some property before any models are fit. One use case for this is in genomics, where we often want to exclude features which are too sparse, being non-zero for the vast majority of observations. The code below shows the two different ways of using the `exclude` argument. They return the same result except for the function call that produced the fit.

```
R> set.seed(1)
R> x[sample(seq(length(x)), 1600)] <- 0
R> filter <- function(x, ...) which(colMeans(x == 0) > 0.8)
R> exclude <- filter(x)
R> fit1 <- glmnet(x, y, exclude = exclude)
R> fit2 <- glmnet(x, y, exclude = filter)
R> all.equal(fit1, fit2)
```

```
[1] "Component "call": target, current do not match when deparsed"
```

The real power of assigning a function to the `exclude` argument is in performing CV. If a filtering function is given as the `exclude` argument, `cv.glmnet` will apply that function separately to each training fold, hence accounting for any bias that may be incurred by the variable filtering. This is not possible when assigning a vector of indices to `exclude`: in that case, the excluded variables are forced to be the same across folds.

In large data settings, it may take some time to fit the entire sequence of elastic net models. `glmnet` and `cv.glmnet` come equipped with a progress bar which can be displayed with the argument `trace.it = TRUE`. This gives the user a sense of how model fitting is progressing.

The **glmnet** package provides a convenience function `bigGlm` for fitting a single *unpenalized* GLM but allowing all the options of `glmnet`. In particular, the user can set upper and/or lower bounds on the coefficients, and can provide the `x` matrix in sparse matrix format: options that are not available for the `stats::glm` function.

```
R> data("BinomialExample")
R> x <- BinomialExample$x
R> y <- BinomialExample$y
R> fit <- bigGlm(x, y, family = "binomial")
R> which(coef(fit) < -1)

[1]  4  5  6  7  9 11 12 13 14 18 19 20 21 22 25 27 28 30

R> fit <- bigGlm(x, y, family = "binomial", lower.limits = -1)
R> which(coef(fit) < -1)

integer(0)
```

5.3 Regularized Cox proportional hazards models

We assume the usual survival-analysis framework. Instead of having $y_i \in \mathbb{R}$ as a response, we have instead $(y_i, \delta_i) \in \mathbb{R}_+ \times \{0, 1\}$. Here y_i is the observed time for observation i, and $\delta_i = 1$ if y_i is the failure time and $\delta_i = 0$ if it is the right-censoring time. The Cox proportional hazards model (Cox, 1972) is a commonly used model for the relationship between the predictor variables and survival time. It assumes a semi-parametric form for the hazard function

$$h_i(t) = h(t)e^{\mathbf{x}_i^T \boldsymbol{\beta}},$$

where $h_i(t)$ is the hazard for observation i at time t, h is the baseline hazard for the entire population of observations, and $\boldsymbol{\beta} \in \mathbb{R}^p$ is the vector of coefficients to be estimated. Let $t_1 < \cdots < t_m$ denote the unique failure times and let $j(i)$ denote the index of the observation failing at time t_i. (Assume for the moment that the y_i's are unique.) If $y_j \geq t_i$, we say that observation j is *at risk* at time t_i. Let R_i denote the *risk set* at time t_i. The coefficient vector $\boldsymbol{\beta}$ is estimated by maximizing the partial likelihood

$$L(\boldsymbol{\beta}) = \prod_{i=1}^m \frac{e^{\mathbf{x}_{j(i)}^T \boldsymbol{\beta}}}{\sum_{j \in R_i} e^{\mathbf{x}_j^T \boldsymbol{\beta}}}. \tag{5.7}$$

It is the conditional likelihood that the failure occurs for observation $j(i)$ given all the observations at risk. Maximizing the partial likelihood is equivalent to minimizing the negative log partial likelihood

$$-\ell(\boldsymbol{\beta}) = \frac{2}{n}\sum_{i=1}^{m}\left[-\mathbf{x}_{j(i)}^T\boldsymbol{\beta} + \log\left(\sum_{j\in R_i} e^{\mathbf{x}_j^T\boldsymbol{\beta}}\right)\right]. \tag{5.8}$$

We put a negative sign in front of ℓ so that ℓ denotes the log partial likelihood, and the scale factor $2/n$ is included for convenience. Note also that the model does not have an intercept term β_0, as it cancels out in the partial likelihood. Simon et al. (2011) proposed an elastic-net regularization path version for the Cox model, as well as Algorithm 8 for solving the minimization problem.

Algorithm 8 *Fitting Cox models with elastic net penalty*

```
1. Select a value of $\alpha \in [0,1]$ and a sequence of $\lambda$ values $\lambda_1 > \ldots > \lambda_m$. Define $\hat{\boldsymbol{\beta}}(\lambda_0) = \mathbf{0}$.
2. For $\ell = 1,\ldots,m$:
   (a) Initialize $\hat{\boldsymbol{\beta}}(\lambda_\ell) = \hat{\boldsymbol{\beta}}(\lambda_{\ell-1})$.
   (b) For $t = 0,1,\ldots$ until convergence (outer loop):
       i. For $k = 1,\ldots,n$, compute $\eta_k^{(t)} = \hat{\boldsymbol{\beta}}(\lambda_\ell)^T\mathbf{x}_k$.
       ii. For $k = 1,\ldots,n$, compute
```

$$\ell'\left(\eta^{(t)}\right)_k = \delta_k - e^{\eta_k^{(t)}}\sum_{i\in C_k}\left(\frac{1}{\sum_{j\in R_i} e^{\eta_j^{(t)}}}\right), \tag{5.9}$$

$$\ell''\left(\eta^{(t)}\right)_{k,k} = \sum_{i\in C_k}\left[\frac{e^{\eta_k^{(t)}}\sum_{j\in R_i} e^{\eta_j^{(t)}} - \left(e^{\eta_k^{(t)}}\right)^2}{\left(\sum_{j\in R_i} e^{\eta_j^{(t)}}\right)^2}\right], \tag{5.10}$$

$$w_k^{(t)} = -\ell''\left(\eta^{(t)}\right)_{k,k}, \tag{5.11}$$

$$z_k^{(t)} = \eta_k^{(t)} + \frac{\ell'\left(\eta^{(t)}\right)_k}{\ell''\left(\eta^{(t)}\right)_{k,k}}, \tag{5.12}$$

```
          where $C_k$ is the set of failure times $i$ such that $t_i < y_k$ (i.e. times for which observation $k$ is still at risk.)
       iii. Solve the penalized WLS problem (inner loop):
```

$$\hat{\boldsymbol{\beta}}(\lambda_\ell) = \underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\operatorname{argmin}}\left[\frac{1}{2}\sum_{k=1}^{n} w_k^{(t)}\left(z_k^{(t)} - \mathbf{x}_k^T\boldsymbol{\beta}\right)^2 + \lambda_\ell\left(\frac{1-\alpha}{2}\|\boldsymbol{\beta}\|_2^2 + \alpha\|\boldsymbol{\beta}\|_1\right)\right].$$

Algorithm 8 has the same structure as Algorithm 6 except for different formulas for computing the working responses and weights. (We note that these formulas implicitly approximate the Hessian of the log partial likelihood by a diagonal matrix with the Hessian's diagonal entries.) This means that we can leverage the fast implementation of the penalized WLS problem in Section 5.2.4 for an efficient implementation of Algorithm 8. (As a small benefit, it also means that we can fit regularized Cox models when the design matrix $\mathbf{X}$ is sparse.) Such a model can be fit with **glmnet** by specifying `family = "cox"`. The response provided needs to be a `Surv` object from the **survival** package (Therneau, 2020).

```
R> data("CoxExample")
R> glmnet(CoxExample$x, CoxExample$y, family = "cox")


Call:  glmnet(x = CoxExample$x, y = CoxExample$y, family = "cox")

   Df %Dev   Lambda
1   0 0.00 0.236800
2   1 0.18 0.215700
3   3 0.60 0.196600
...
47 24 5.87 0.003279
48 24 5.87 0.002988
49 24 5.87 0.002722
```

The computation of these w_k's and z_k's can be a computational bottleneck if not implemented carefully: since the C_k and R_i have $O(n)$ elements, a naive implementation takes $O(n^2)$ time. Simon et al. (2011) exploit the fact that, once the observations are sorted in order of the observed times y_i, the risk sets are nested ($R_{i+1} \subseteq R_i$ for all i) and the w_k's and z_k's can be computed in $O(n)$ time.

If our data contains tied observed times, `glmnet` uses the Breslow approximation of the partial likelihood for ties (Breslow, 1972) and maximizes the elastic net-regularized version of this approximation instead. See Simon et al. (2011) for details.

5.3.1 Extending regularized Cox models to (start, stop] data

Instead of working with right-censored responses, the Cox model can be extended to work with responses which are a pair of times (called the "start time" and "stop time"), with the possibility of the stop time being censored. This is an instantiation of the counting process framework proposed by Andersen and Gill (1982), and the right-censored data set-up is a special case with the start times all being equal to zero.

As noted in Therneau and Grambsch (2000), (start, stop] responses greatly increase the flexibility of the Cox model, allowing for

- Time-dependent covariates,
- Time-dependent strata,
- Left truncation,
- Multiple time scales,
- Multiple events per subject,
- Independent increment, marginal, and conditional models for correlated data, and
- Various forms of case-cohort models.

From a data analysis viewpoint, this extension amounts to requiring just one more variable: the `time` variable is replaced by `(start, stop]` variables, with `(start, stop]` indicating the interval where the unit is at risk. The **survival** package provides the function `tmerge` to aid in the creation of such datasets.

For this more general setup, inference for $\boldsymbol{\beta}$ can proceed as before. The formulas for the partial likelihood and negative log partial likelihood (Equations (5.7) and (5.8)) remain the same; what changes is the definition of what it means for an observation to be at risk at time t_i. If we let $(y_{1j}, y_{2j}]$ denote the (start, stop] times for observation j, then observation j is at risk at time t_i if and only if $t_i \in (y_{1j}, y_{2j}]$. Similarly, the elastic net-regularized version of the Cox model for (start, stop] data can be fitted using Algorithm 8 with this new definition of what it means for an observation to be at risk at a failure time.

With (start, stop] data, it is no longer true that the risk sets are nested. For example, if $t_i < y_{1j} < t_{i+1} < y_{2j}$, then $j \in R_{i+1}$ but $j \notin R_i$. However, as Algorithm 9 shows, it is still possible to compute the working responses and weights in $O(n \log n)$ time. In fact, only the ordering of observations (Step 1) requires $O(n \log n)$ time: the rest of the algorithm requires just $O(n)$ time. Since the ordering of observations never changes, the results of Step 1 can be cached, meaning that only the first run of Algorithm 9 requires $O(n \log n)$ time, and future runs just need $O(n)$ time.

The differences between right-censored data and (start, stop] data for Cox models are hidden from the user, in that the function call for (start, stop] data is exactly the same as that for right-censored data. The difference is in the type of `Surv` object that is passed for the response `y`. `glmnet` checks for the `Surv` object type before routing to the correct internal subroutine.

We illustrate this new functionality on the `bladder2` dataset from the **survival** package. We will fit a regularized Cox model for the time to recurrence based on the treatment type (`rx`), the initial number of tumors (`number`) and the size of the largest initial tumor (`size`).

Algorithm 9 *Computing working responses and weights for Algorithm 8*

Input: $\eta_j = \mathbf{x}_j^T \hat{\boldsymbol{\beta}}$ (where $\hat{\boldsymbol{\beta}}$ is the current estimate for $\boldsymbol{\beta}$), $(y_{1j}, y_{2j}]$, δ_j for $j = 1, \dots, n$. For simplicity, assume that the observations are ordered by ascending stop time, i.e. $y_{21} < \cdots < y_{2n}$. As before, let $t_1 < \cdots < t_m$ denote the failure times in increasing order.

```
1. Get the ordering for the observations according to start times. Let $start(j)$ denote the index
   for the observation with the $j$th earliest start time.
2. Compute the risk set sums $RSS_i = \sum_{j \in R_i} e^{\eta_j}$, $i = 1, \dots, m$ using the following steps:
   (a) For $j = 1, \dots, n$, set $RSS_j \leftarrow \sum_{\ell=j}^{n} e^{\eta_\ell}$.
   (b) Set $curr \leftarrow 0$, $i \leftarrow m$, $start\_idx \leftarrow n$.
   (c) While $i > 0$ and $start\_idx > 0$:
       i. If $y_{1start(start\_idx)} < t_i$, set $RSS_{j(i)} \leftarrow RSS_{j(i)} - curr$ and $i \leftarrow i - 1$.
       ii. If not, set $curr \leftarrow curr + e^{\eta_{start(start\_idx)}}$ and $start\_idx \leftarrow start\_idx - 1$.
   (d) Take just the elements of $RSS$ corresponding to death times, i.e. set $RSS_i \leftarrow RSS_{j(i)}$.
3. Compute the partial sums $RSK_k = \sum_{i \in C_k} 1/RSS_i$, $k = 1, \dots, n$ using the following steps:
   (a) For $i = 1, \dots, m$, set $RD_i \leftarrow \sum_{\ell=1}^{i} 1/RSS_i$. Set $RD_0 \leftarrow 0$.
   (b) For $k = 1, \dots, n$, set $D_k \leftarrow \sum_{\ell=1}^{k} \delta_\ell$.
   (c) For $k = 1, \dots, n$, set $RSK_k \leftarrow RD_{D_k}$.
   (d) Set $curr \leftarrow 0$, $i \leftarrow 1$, $start\_idx \leftarrow 1$.
   (e) While $i \leq m$ and $start\_idx \leq n$:
       i. If $y_{1start(start\_idx)} < t_i$, set $RSK_{start(start\_idx)} \leftarrow RSK_{start(start\_idx)} - curr$ and
          $start\_idx \leftarrow start\_idx + 1$.
       ii. If not, set $curr \leftarrow curr + 1/RSS_i$ and $i \leftarrow i + 1$.
4. Compute the partial sums $RSKSQ_k = \sum_{i \in C_k} 1/RSS_i^2$, $k = 1, \dots, n$ in a similar manner as
   Step 3.
5. Compute $\ell'(\eta)_k$ amd $\ell''(\eta)_{k,k}$ using the formulas (5.9) and (5.10):
       $\ell'(\eta)_k = \delta_k - e^{\eta_k} \cdot RSK_k, \qquad \ell''(\eta)_{k,k} = (e^{\eta_k})^2 \cdot RSKSQ_k - e^{\eta_k} \cdot RSK_k.$
6. Compute the working responses and weights using the formulas (5.11) and (5.12).
```

```
R> library("survival")
R> head(bladder2)

  id rx number size start stop event enum
1  1  1      1    3     0    1     0    1
2  2  1      2    1     0    4     0    1
3  3  1      1    1     0    7     0    1
4  4  1      5    1     0   10     0    1
5  5  1      4    1     0    6     1    1
6  5  1      4    1     6   10     0    2
```

We start by creating the (start, stop] response with `survival::Surv`:

```
R> y <- with(bladder2, Surv(start, stop, event))
R> head(y)

[1] (0, 1+] (0, 4+] (0, 7+] (0,10+] (0, 6]  (6,10+]
```

The code for fitting the regularized Cox model is then exactly the same as the code we would have used if the response was right-censored:

```
R> x <- as.matrix(bladder2[, 2:4])
R> glmnet(x, y, family = "cox")


Call:  glmnet(x = x, y = y, family = "cox")

   Df %Dev   Lambda
1   0 0.00 0.194800
2   1 0.34 0.177500
3   1 0.61 0.161700
...
41  3 2.67 0.004715
42  3 2.67 0.004296
43  3 2.68 0.003914
```

5.3.2 Stratified Cox models

An extension of the Cox model is to allow for strata. These strata divide the units into disjoint groups, with each group having its own baseline hazard function but having the same values of β.

Specifically, if the units are divided into K strata, then the stratified Cox model assumes that a unit in stratum k has the hazard function

$$h_i(t) = h_k(t) e^{\mathbf{x}_i^T \boldsymbol{\beta}},$$

where $h_k(t)$ is the shared baseline hazard for all units in stratum k. In several applications, allowing different subgroups to have different baseline hazards approximates reality more closely. For example, it might be reasonable to have different baseline hazards based on gender in clinical trials, or a separate baseline for each center in multi-center trials.

In this setting, the negative log partial likelihood is

$$\ell(\boldsymbol{\beta}) = \sum_{k=1}^{K} \ell_k(\boldsymbol{\beta}),$$

where $\ell_k(\boldsymbol{\beta})$ is exactly (5.8) but considering just the units in stratum k. Since the negative log partial likelihood decouples across strata (conditional on $\boldsymbol{\beta}$), regularized versions of stratified Cox models can be fit using a slightly modified version of Algorithm 8.

To fit an unpenalized stratified Cox model, the **survival** package has a special `strata` function that allows users to specify the strata variable in formula syntax. Since `glmnet` does not work with formulas, we needed a different approach for specifying strata. To fit regularized stratified Cox models in **glmnet**, the user needs to add a `strata` attribute to the response `y`. `glmnet` checks for the presence of this attribute and if it is present, it fits a stratified Cox model. We note that the user cannot simply add the attribute manually because R drops attributes when subsetting vectors. Instead, the user should use the `stratifySurv` function to add the `strata` attribute. (`stratifySurv` creates an object of class `"stratifySurv"` that inherits from the class `"Surv"`, ensuring that **glmnet** can reassign the `strata` attribute correctly after any subsetting.) The code below shows an example of how to fit a regularized stratified Cox model with **glmnet**; there are a total of 1,000 observations, with the first 500 belonging to the first strata and the rest belonging to the second strata.

```
R> data("CoxExample")
R> x <- CoxExample$x
R> y <- CoxExample$y
R> strata <- c(rep(1, 500), rep(2, 500))
R> y2 <- stratifySurv(y, strata)
R> glmnet(x, y2, family = "cox")


Call:  glmnet(x = x, y = y2, family = "cox")
```

```
   Df %Dev   Lambda
1   0 0.00 0.235500
2   2 0.23 0.214600
3   3 0.69 0.195500
...
47 24 6.57 0.003262
48 24 6.57 0.002972
49 25 6.58 0.002708
```

5.3.3 Plotting survival curves

The beauty of the Cox partial likelihood is that the baseline hazard, $h_0(t)$, is not required for inference on the model coefficients $\boldsymbol{\beta}$. However, the estimated hazard is often of interest to users. The **survival** package already has a well-established `survfit` method that can produce estimated survival curves from a fitted Cox model. **glmnet** implements a `survfit` method for regularized Cox models fit by `glmnet` by creating the `coxph` object corresponding to the model and calling `survival::survfit`.

The code below is an example of calling `survfit` for `coxnet` objects for a particular value of the λ tuning parameter (in this case, $\lambda = 0.05$). Note that we had to pass the original design matrix `x` and response `y` to the `survfit` call: they are needed for `survfit.coxnet` to reconstruct the required `coxph` object. The survival curves are computed for the individuals represented in `newx`: we get one curve per individual, as seen in Figure 5.3.

```
R> set.seed(1)
R> nobs <- 100; nvars <- 15
R> x <- matrix(rnorm(nobs * nvars), nrow = nobs)
R> ty <- rep(rexp(nobs / 5), each = 5)
R> tcens <- rbinom(n = nobs, prob = 0.3, size = 1)
R> y <- Surv(ty, tcens)

R> fit <- glmnet(x, y, family = "cox")
R> sf_obj <- survfit(fit, s = 0.05, x = x, y = y, newx = x[1:2, ])
R> plot(sf_obj, col = 1:2, mark.time = TRUE, pch = "12")
```

The `survfit` method is available for Cox models fitted by `cv.glmnet` as well. By default, the survival curves are computed for the `lambda.1se` value of the λ hyperparameter. The user can use the code below to compute the survival curve at the `lambda.min` value:

```
R> set.seed(1)
R> cfit <- cv.glmnet(x, y, family = "cox", nfolds = 5)
```

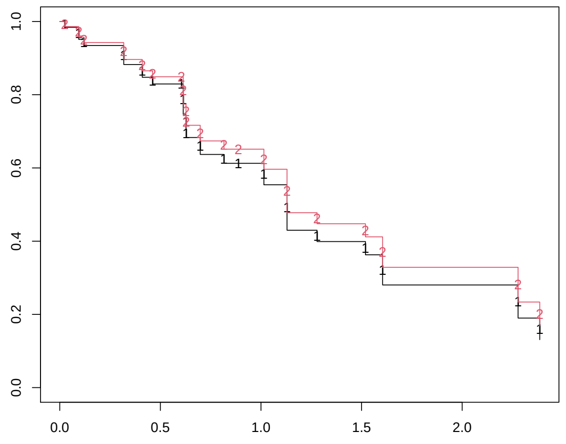

Figure 5.3: An illustration of the plotted **`survfit`** object. One survival curve is plotted for each individual represented in the **`newx`** argument.

```
R> survfit(cfit, s = "lambda.min", x = x, y = y, newx = x[1:2, ])

Call: survfit.cv.glmnet(formula = cfit, s = "lambda.min", x = x, y = y,
    newx = x[1:2, ])

    n events median
1 100     33    1.6
2 100     33    1.6
```

5.4 Summary

In this chapter, we have shown how to extend the use of the elastic net penalty to all GLM model families, as well as Cox models with (start, stop] data and with strata. These new capabilities are available in version 4.1 and later of the **glmnet** package on CRAN.

Appendix A

Appendix for Principal Components Lasso

A.1 pcLasso penalty contours for two predictors

Assume that we only have two predictors which are standardized to have mean zero and sum of squares one. Then $\mathbf{X}^T\mathbf{X} = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$, where ρ is the sample correlation between the two predictors. Let $\mathbf{A} = \mathbf{V}\mathbf{D}_{d_1^2 - d_j^2}\mathbf{V}^T$. If $\rho > 0$, the expressions for the singular value decomposition (SVD) of $\mathbf{X}$ and $\mathbf{A}$ are

$$\mathbf{X} = \mathbf{U} \cdot \begin{pmatrix} 1+\rho & 0 \\ 0 & 1-\rho \end{pmatrix} \cdot \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}, \qquad \mathbf{A} = 2\rho \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

If $\rho < 0$, the corresponding expressions are

$$\mathbf{X} = \mathbf{U} \cdot \begin{pmatrix} 1-\rho & 0 \\ 0 & 1+\rho \end{pmatrix} \cdot \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}, \qquad \mathbf{A} = -2\rho \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

With these expressions, the pcLasso penalty can be written explicitly as

$$\begin{cases} \lambda(|\beta_1| + |\beta_2|) + 2\theta\rho(\beta_1 - \beta_2)^2 & \text{if } \rho > 0, \\ \lambda(|\beta_1| + |\beta_2|) - 2\theta\rho(\beta_1 + \beta_2)^2 & \text{if } \rho < 0. \end{cases}$$

(Here, the penalty is that in Equation (2.1) but with $\theta/2$ replaced with θ.) Considering the signs of β_1 and β_2, we can get an even more explicit description of the contours. We provide the description for $\rho > 0$ below:

- $\beta_1 \geq 0, \beta_2 \geq 0$: The parabola $\beta_2 = -\frac{2\sqrt{2}\theta\rho}{\lambda}\beta_1^2 + \frac{C}{\sqrt{2}\lambda}$, rotated 45° clockwise.
- $\beta_1 \geq 0, \beta_2 \leq 0$: The line $\beta_2 = \beta_1 + \frac{\lambda - \sqrt{\lambda^2 + 8\theta\rho C}}{4\theta\rho}$.
- $\beta_1 \leq 0, \beta_2 \leq 0$: The parabola $\beta_2 = \frac{2\sqrt{2}\theta\rho}{\lambda}\beta_1^2 - \frac{C}{\sqrt{2}\lambda}$, rotated 45° clockwise.
- $\beta_1 \leq 0, \beta_2 \geq 0$: The line $\beta_2 = \beta_1 - \frac{\lambda - \sqrt{\lambda^2 + 8\theta\rho C}}{4\theta\rho}$.

A.2 Comparison of model-fitting times

Table A.1 shows a comparison of model-fitting times for pcLasso against that for other methods, all available as R packages, for different values of n and p. The competitors include the lasso **glmnet**, sparse group lasso **SGL** and group lasso packages **gglasso** and **grpreg**. We note that **grpreg** uses orthogonalization within groups to speed up the computation. Among these competitors, only **glmnet** and **SGL** provides sparsity at the level of individual features. All functions were called with default settings. We see that **SGL** and **gglasso** start to slow down considerably as the problem size gets moderately large, while the times for **grpreg** are similar to that for pcLasso.

						pcLasso		
n	p	**glmnet**	**SGL**	**gglasso**	**grpreg**	SVD	Rest	Total
100	100	0.05	3.59	3.31	0.02	0.00	0.02	0.02
100	200	0.02	7.14	0.39	0.03	0.01	0.01	0.01
100	500	0.02	29.65	2.40	0.08	0.03	0.02	0.05
500	1000	0.19		14.89	1.12	0.25	0.13	0.38
1000	2000	0.67			6.38	1.50	0.54	2.04
1000	5000	1.10			17.26	11.16	2.35	13.51
2000	5000	2.49			36.57	24.15	3.81	27.96
2000	10000	4.50			139.63	106.56	12.96	119.52

Table A.1: *Comparison of timings for pcLasso against various algorithms. The predictors are pre-assigned to 10 groups. Time in seconds, average over three runs for an entire path of solutions. The last three columns show the pcLasso model-fitting times broken up by the initial SVD(s) and the rest of the computation.* ***SGL*** *implements the sparse group lasso.* ***gglasso*** *and* ***grpreg*** *are group lasso packages; the latter does orthogonalization of predictors.*

A.3 Full details of simulation study in Section 2.8

In this appendix, we present the full range of simulation settings that we tested the methods on, along with greater detail for the data generation processes.

The methods which we compared are listed in the main text. We present boxplots of the test mean-squared error (MSE) on $5,000$ test points, i.e. $MSE = \mathbb{E}[(\widehat{y}_{test} - signal_{test})^2]$. We also present plots of the true positive rate (TPR) vs. the true negative rate (TNR) (which is also equal to $1-$ the false positive rate) for each method regarding feature selection. That is, TPR is the proportion of true features the model selected, and TNR is the proportion of null features that the model does not select.

For each simulation setting, we run the simulation 30 times. As such, each boxplot is a summary of 30 data points, and there are 30 data points on each TPR-TNR plot. A small amount of jitter has been added to the points in the TPR-TNR plots for a better view of the data.

A.3.1 Setting 1: PC regression and pcLasso's "home court"

1. $n = 100$, 10 groups of 20 predictors each.
2. The kth data matrix is generated as an approximately rank-3 matrix: $\mathbf{X}_k = \sum_{i=1}^{3} i \cdot \mathbf{u}_i \mathbf{v}_i^T + \mathbf{E}$, where the entries of the $\mathbf{u}_i$, $\mathbf{v}_i$ and $\mathbf{E}$ are i.i.d. $\mathcal{N}(0,1)$.
3. The true signal is the sum of the first sparse PC for the first 5 groups. The sparse PCs are computed using the **elasticnet** package and are constrained to have 3 non-zero loadings each. The signal is scaled to have mean 0 and variance 1. The response is the true signal plus Gaussian noise such that the target SNR is reached.

The results are shown in Figure A.1.

A.3.2 Setting 2: lasso's "home court"

1. $n = 100$, $p = 50$.
2. The entries of the data matrix are i.i.d. $\mathcal{N}(0,1)$.
3. The true signal is the sum of the first 5 features. The response is the true signal plus Gaussian noise such that the target SNR is reached.

The results are shown in Figure A.2.

A.3.3 Setting 3: correlation across groups

1. $n = 100$, 5 groups of 20 predictors each.
2. Generate $\mathbf{u}_1^* \in \mathbb{R}^{100}$ with entries being i.i.d. $\mathcal{N}(0,1)$ draws. Normalize $\mathbf{u}_1^*$ such that $\|\mathbf{u}_1\|_2 = 1$. Generate d_1^* as a random draw from $Unif[0.8, 2]$.
3. For each $k = 1, \ldots, 5$:

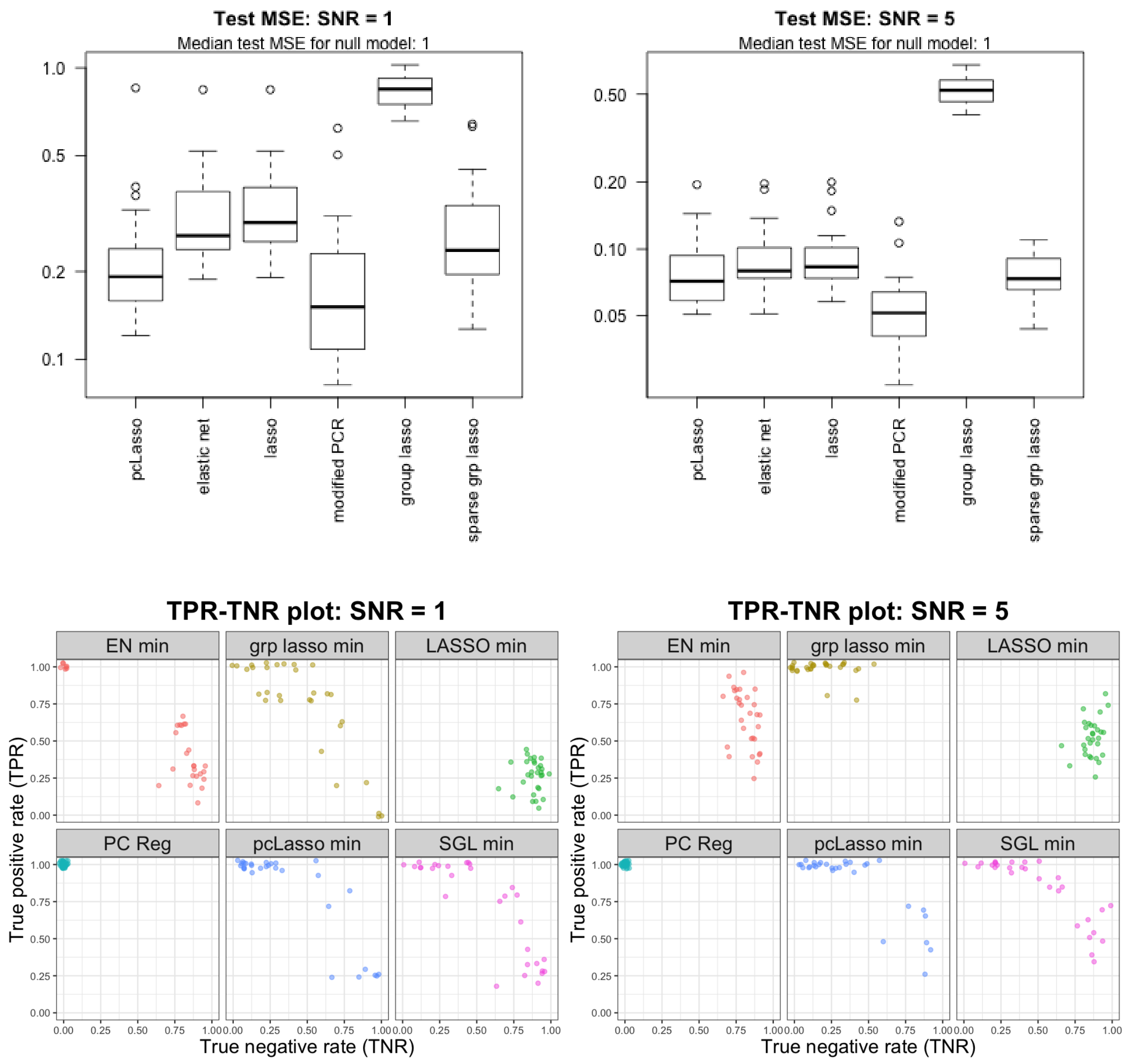

Figure A.1: *Results for setting 1: pcLasso's "home court" simulation. Note the log scale on the y-axis for the MSE plots.*

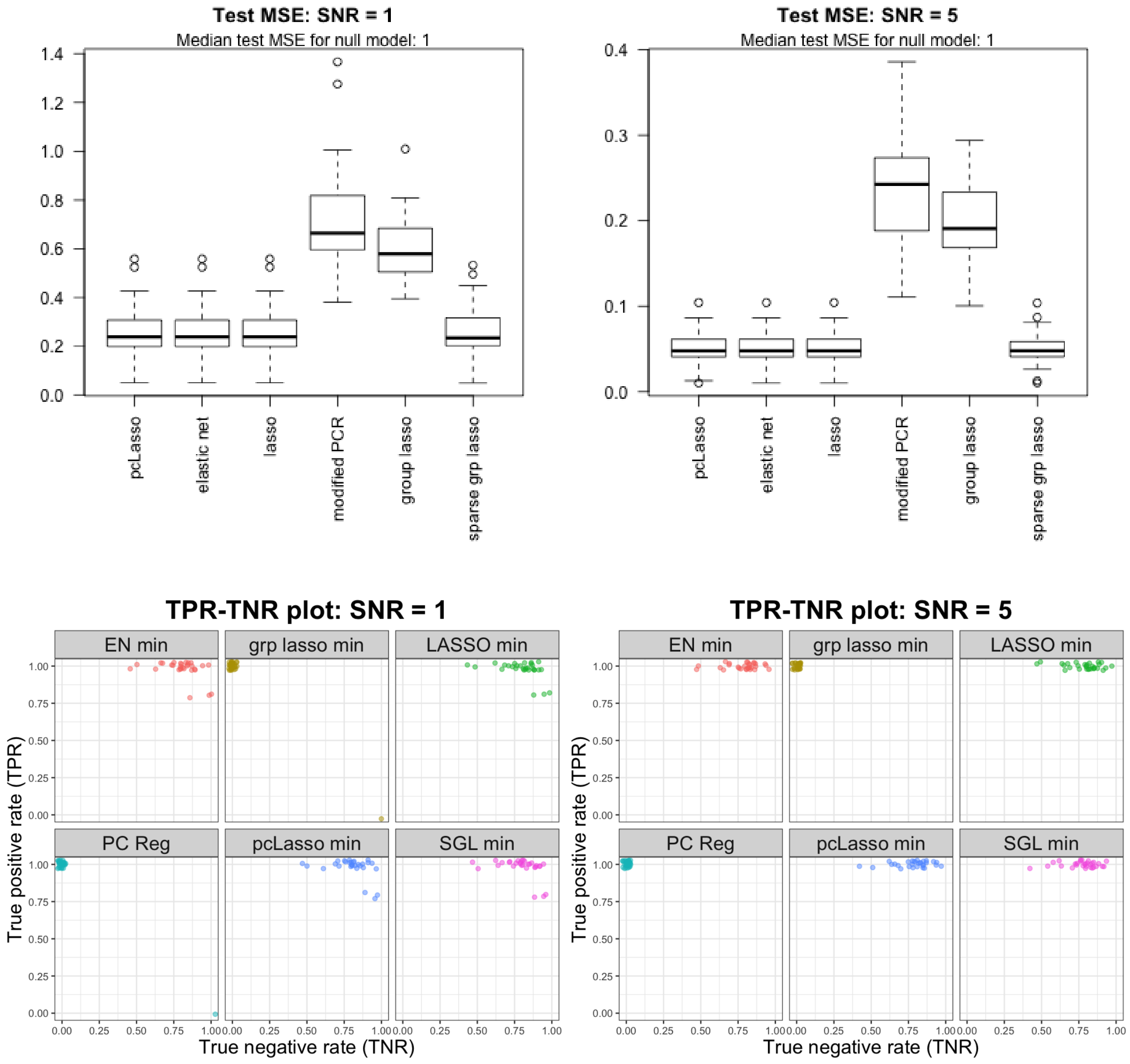

Figure A.2: *Results for setting 2: lasso's "home court" simulation.*

(a) For $\ell = 2, \ldots, 5$, generate $\mathbf{u}_\ell \in \mathbb{R}^{100}$ with entries being i.i.d. $\mathcal{N}(0,1)$ draws. Project out the component of $\mathbf{u}_\ell$ that lies in span$\{\mathbf{u}_1^*, \mathbf{u}_2, \ldots, \mathbf{u}_{\ell-1}\}$, then normalize the remaining vector to have norm 1. (Call this resulting vector $\mathbf{u}_\ell$.)

(b) Generate $\mathbf{v}_1, \ldots, \mathbf{v}_5 \in \mathbb{R}^{20}$ with entries being i.i.d. $\mathcal{N}(0,1)$ draws. Normalize $\mathbf{v}_1$ to have norm 1. For $\ell = 2, \ldots, 5$, project out the component of $\mathbf{v}_\ell$ that lies in span$\{\mathbf{v}_1, \ldots, \mathbf{v}_{\ell-1}\}$ and normalize the remaining vector to have norm 1. (Call the vectors resulting from this procedure $\mathbf{v}_1, \ldots, \mathbf{v}_5$.)

(c) Set $d_\ell = 1/\sqrt{\ell}$ for $\ell = 2, \ldots, 5$.

(d) Set $\mathbf{X}_k = d_1^* \mathbf{u}_1^* \mathbf{v}_1 + \sum_{\ell=2}^5 d_\ell \mathbf{u}_\ell \mathbf{v}_\ell$.

4. The true signal is the first PC $d_1^* \mathbf{u}_1^*$, and the response is the true signal plus Gaussian noise such that the target SNR is reached.

The results are shown in Figure A.3. We do not show TPR-TNR plots for this setting as the signal is present in each of the groups, so true and null features does not make sense in this setting.

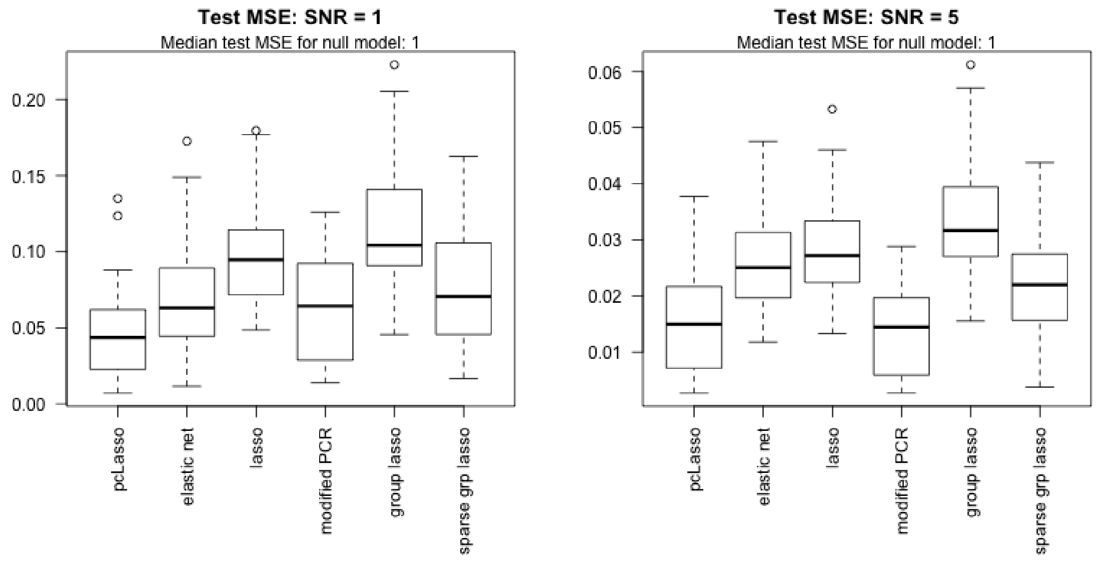

Figure A.3: *Results for setting 3: correlation between groups. We do not show TPR-TNR plots as the notion of true and null features does not make sense in this setting: the signal is present in each of the groups.*

A.3.4 Setting 4: correlation within each group, power decay correlation structure

1. $n = 200$, 5 groups of 10 predictors each.

2. For the kth data matrix, the rows are independent draws from the multivariate normal distribution having mean $\mathbf{0}$ and covariance equal to the AR(1) correlation matrix with parameter ρ.

3. The true signal is the sum of the first PCs of the first two groups. The response is the true signal plus Gaussian noise such that the target SNR is reached.

We ran the set-up above for two values of ρ: $\rho = 0.8$ for the high correlation setting and $\rho = 0.2$ for the low correlation setting. The results for the high and low correlation settings are presented in Figure A.4 and Figure A.5 respectively.

A.3.5 Setting 5: correlation within each group, equicorrelation correlation structure

1. $n = 200$, 5 groups of 10 predictors each.

2. For the kth data matrix, the rows are independent draws from the multivariate normal distribution having mean $\mathbf{0}$ and covariance $\rho\mathbf{1}\mathbf{1}^T + (1-\rho)\mathbf{I}$ for some parameter ρ.

3. The true signal is the sum of the first PCs of the first two groups. The response is the true signal plus Gaussian noise such that the target SNR is reached.

We ran the set-up above for two values of ρ: $\rho = 0.8$ for the high correlation setting and $\rho = 0.2$ for the low correlation setting. The results for the high and low correlation settings are presented in Figure A.6 and Figure A.7 respectively.

A.4 Proofs for Section 2.9

Before presenting proofs for Section 2.9, we first prove two technical lemmas which show that the pcLasso solution lies in the constraint set which we will use for the restricted eigenvalue condition (Equation (2.6)).

Lemma 11. *If we solve the constrained form of pcLasso (Equation* (2.4)*) with $R \le \|\boldsymbol{\beta}^*\|_1$, then $\|\widehat{\boldsymbol{\nu}}_{S^c}\|_1 \le \|\widehat{\boldsymbol{\nu}}_S\|_1$.*

Proof. Since $\widehat{\boldsymbol{\beta}}$ is feasible for Equation (2.4), by the triangle inequality,

$$\|\boldsymbol{\beta}^*_S\|_1 \ge R \ge \|\boldsymbol{\beta}^* + \widehat{\boldsymbol{\nu}}\|_1 = \|\boldsymbol{\beta}^*_S + \widehat{\boldsymbol{\nu}}_S\|_1 + \|\widehat{\boldsymbol{\nu}}_{S^c}\|_1 \ge \|\boldsymbol{\beta}^*_S\|_1 - \|\widehat{\boldsymbol{\nu}}_S\|_1 + \|\widehat{\boldsymbol{\nu}}_{S^c}\|_1 .$$

□

Lemma 12. *If we solve the Lagrangian form of pcLasso (Equation* (2.5)*) with $\lambda \ge 2\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty / n$, then $\|\widehat{\boldsymbol{\nu}}_{S^c}\|_1 \le 3\,\|\widehat{\boldsymbol{\nu}}_S\|_1$.*

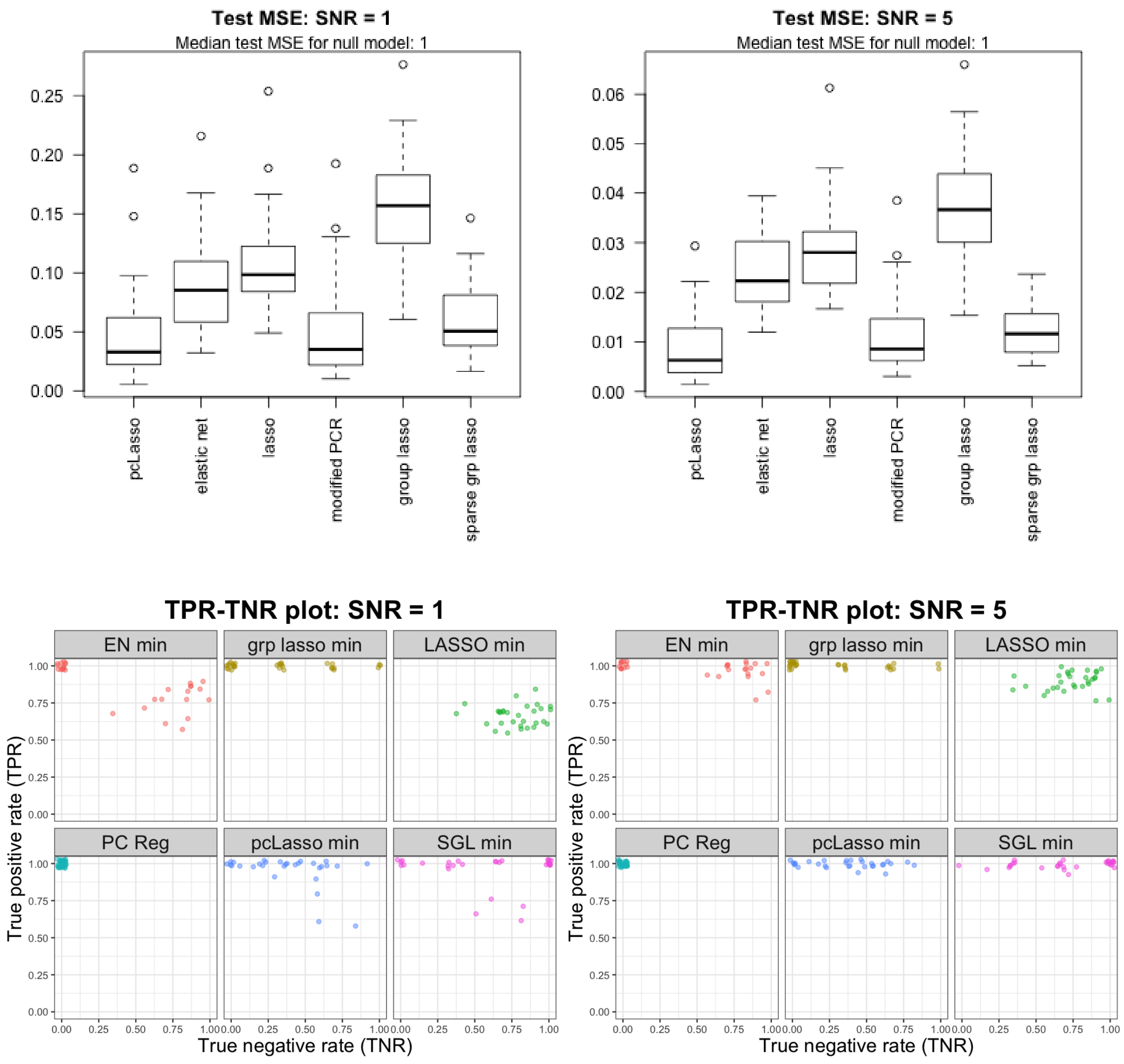

Figure A.4: *Results for setting 4: high correlation within each group. The correlation matrix is the AR(1) correlation matrix with* $\rho = 0.8$.

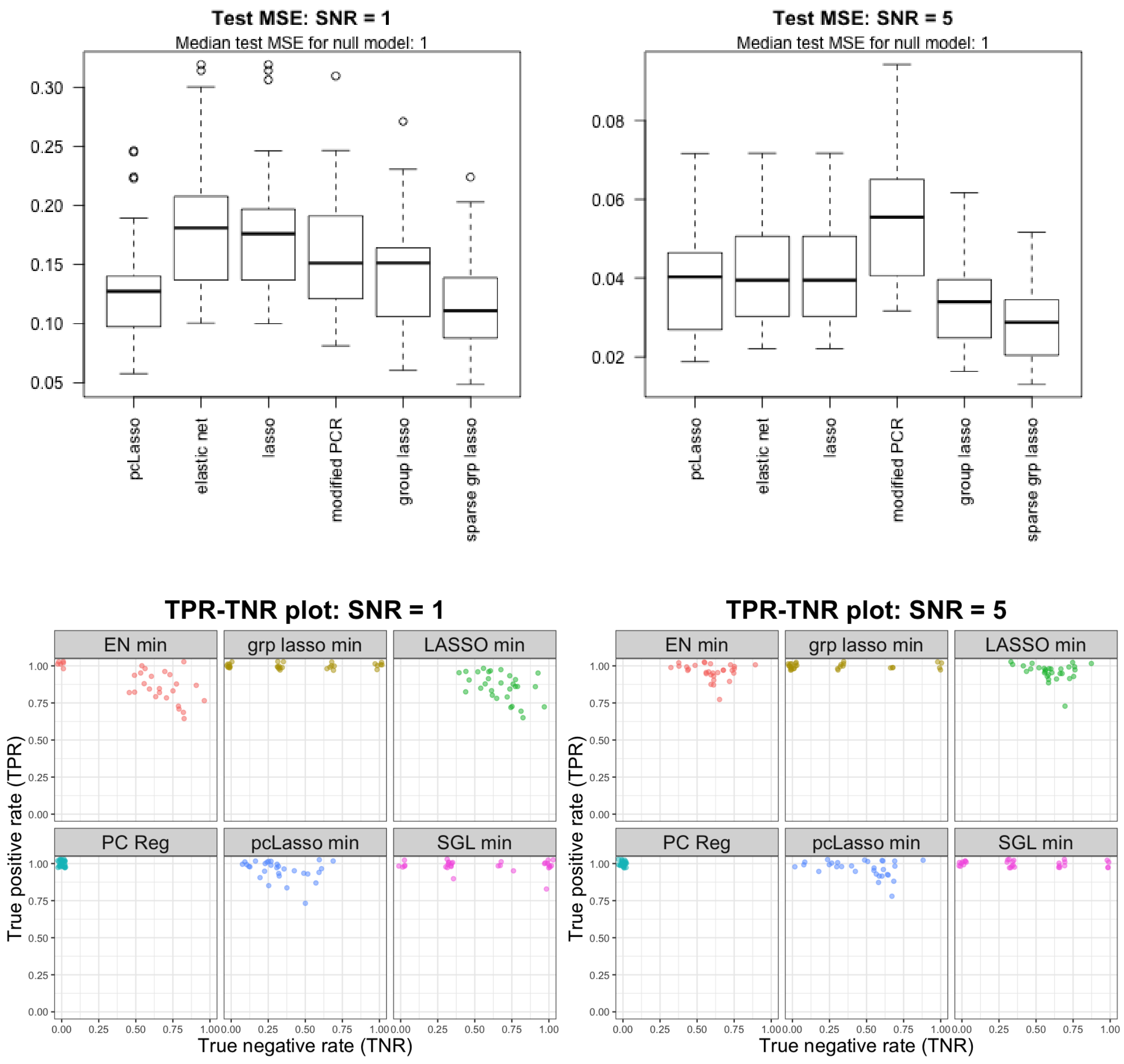

Figure A.5: *Results for setting 4: low correlation within each group. The correlation matrix is the AR(1) correlation matrix with* $\rho = 0.2$.

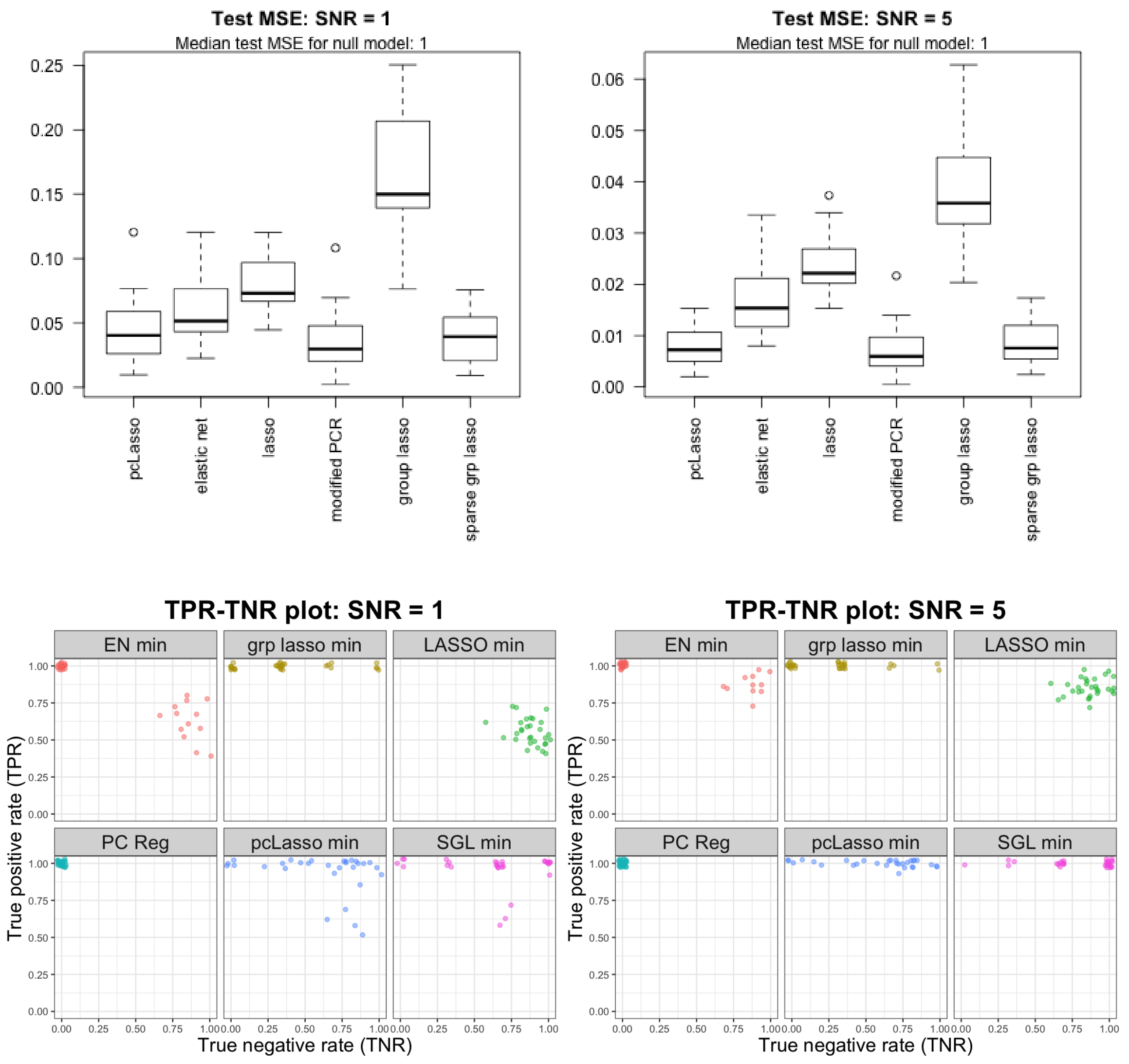

Figure A.6: *Results for setting 5: high correlation within each group. The correlation matrix is the equicorrelation matrix with* $\rho = 0.8$.

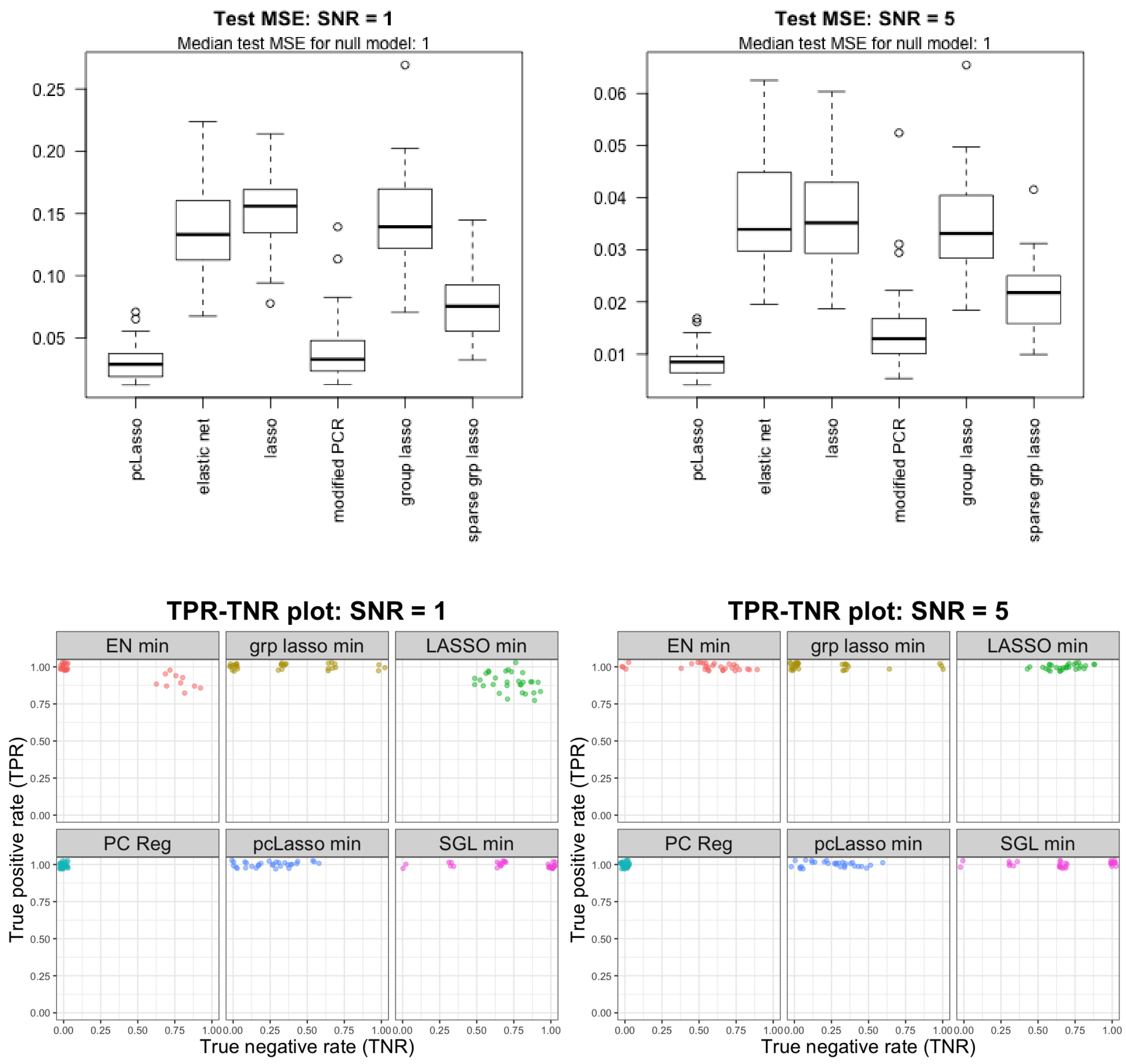

Figure A.7: *Results for setting 5: low correlation within each group. The correlation matrix is the equicorrelation matrix with* $\rho = 0.2$.

Proof. Define $G(\boldsymbol{\nu}) = \frac{1}{2n}\left\|\widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}(\boldsymbol{\beta}^* + \boldsymbol{\nu})\right\|_2^2 + \lambda\left\|\boldsymbol{\beta}^* + \boldsymbol{\nu}\right\|_1$. By definition, $G(\widehat{\boldsymbol{\nu}}) \leq G(0)$, so

$$\begin{aligned}\frac{1}{2n}\left\|\widetilde{\mathbf{w}} - \widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}\right\|_2^2 + \lambda\left\|\boldsymbol{\beta}^* + \widehat{\boldsymbol{\nu}}\right\|_1 &\leq \frac{1}{2n}\left\|\widetilde{\mathbf{w}}\right\|_2^2 + \lambda\left\|\boldsymbol{\beta}^*\right\|_1, \\ \frac{1}{2n}\widehat{\boldsymbol{\nu}}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\widehat{\boldsymbol{\nu}} &\leq \frac{\widetilde{\mathbf{w}}^T\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}}{n} + \lambda(\left\|\boldsymbol{\beta}^*\right\|_1 - \left\|\boldsymbol{\beta}^* + \widehat{\boldsymbol{\nu}}\right\|_1). \qquad \text{(A.1)}\end{aligned}$$

Note that $\left\|\boldsymbol{\beta}^* + \widehat{\boldsymbol{\nu}}\right\|_1 = \left\|\boldsymbol{\beta}_S^* + \widehat{\boldsymbol{\nu}}_S\right\|_1 + \left\|\widehat{\boldsymbol{\nu}}_{S^c}\right\|_1 \geq \left\|\boldsymbol{\beta}_S^*\right\|_1 - \left\|\widehat{\boldsymbol{\nu}}_S\right\|_1 + \left\|\widehat{\boldsymbol{\nu}}_{S^c}\right\|_1$; substituting this in the above gives

$$\frac{1}{2n}\widehat{\boldsymbol{\nu}}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\widehat{\boldsymbol{\nu}} \leq \frac{\widetilde{\mathbf{w}}^T\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}}{n} + \lambda(\left\|\widehat{\boldsymbol{\nu}}_S\right\|_1 - \left\|\widehat{\boldsymbol{\nu}}_{S^c}\right\|_1). \qquad \text{(A.2)}$$

By Hölder's inequality and our choice of λ, we have

$$\frac{\widetilde{\mathbf{w}}^T\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}}{n} \leq \frac{1}{n}\left\|\widetilde{\mathbf{X}}^T\widetilde{\mathbf{w}}\right\|_\infty\left\|\widehat{\boldsymbol{\nu}}\right\|_1 = \frac{1}{n}\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty\left\|\widehat{\boldsymbol{\nu}}\right\|_1 \leq \frac{1}{2}\lambda\left\|\widehat{\boldsymbol{\nu}}\right\|_1. \qquad \text{(A.3)}$$

Substituting this inequality into Equation (A.2) and noting that the LHS of Equation (A.2) is always non-negative, we obtain the desired conclusion. □

A.4.1 Proof for Lemma 2

Let $\boldsymbol{\nu} \in \mathcal{C}$ be expressed as $\boldsymbol{\nu} = \sum_{j=1}^p a_j\mathbf{v}_j$, where the $\mathbf{v}_j$'s are the columns of $\mathbf{V}$. Direct computation gives

$$\|\boldsymbol{\nu}\|_2^2 = \sum_{j=1}^p a_j^2, \qquad \boldsymbol{\nu}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\nu} = \sum_{j=1}^p d_j^2a_j^2, \qquad \boldsymbol{\nu}^T(n\theta\mathbf{A})\boldsymbol{\nu} = \sum_{j=2}^p n\theta(d_1^2 - d_j^2)a_j^2. \qquad \text{(A.4)}$$

Thus, for any $\alpha \in [0, 1]$,

$$\begin{aligned}\boldsymbol{\nu}^T(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})\boldsymbol{\nu} &= \boldsymbol{\nu}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\boldsymbol{\nu} \\ &= \sum_{j=1}^p d_j^2a_j^2 + \sum_{j=2}^p n\theta(d_1^2 - d_j^2)a_j^2 \\ &= \alpha\sum_{j=1}^p d_j^2a_j^2 + (1-\alpha)d_1^2a_1^2 + \sum_{j=2}^p\left[n\theta d_1^2 + (1-\alpha-n\theta)d_j^2\right]a_j^2 \\ &\geq \alpha n\gamma\sum_{j=1}^p a_j^2 + (1-\alpha)d_1^2a_1^2 + \min\left[n\theta d_1^2, n\theta d_1^2 + (1-\alpha-n\theta)d_2^2\right]\sum_{j=2}^p a_j^2, \\ \boldsymbol{\nu}^T(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})\boldsymbol{\nu} &\geq \left[\alpha n\gamma + \min\left((1-\alpha)d_1^2, n\theta d_1^2, n\theta d_1^2 + (1-\alpha-n\theta)d_2^2\right)\right]\|\boldsymbol{\nu}\|_2^2. \qquad \text{(A.5)}\end{aligned}$$

If $n\theta \leq 1$, setting $\alpha = 1 - n\theta$ in Equation (A.5) gives

$$\begin{aligned}\boldsymbol{\nu}^T(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})\boldsymbol{\nu} &\geq \left[(1-n\theta)n\gamma + \min\left(n\theta d_1^2, n\theta d_1^2, n\theta d_1^2\right)\right]\|\boldsymbol{\nu}\|_2^2 \\ &= \left[(1-n\theta)n\gamma + n\theta d_1^2\right]\|\boldsymbol{\nu}\|_2^2.\end{aligned}$$

If $n\theta \geq 1$, setting $\alpha = 0$ in Equation (A.5) gives

$$\begin{aligned}\boldsymbol{\nu}^T(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})\boldsymbol{\nu} &\geq \min\left(d_1^2, n\theta(d_1^2 - d_2^2) + d_2^2\right)\|\boldsymbol{\nu}\|_2^2 \\ &\geq \min\left(d_1^2, (d_1^2 - d_2^2) + d_2^2\right)\|\boldsymbol{\nu}\|_2^2 \\ &= d_1^2\|\boldsymbol{\nu}\|_2^2.\end{aligned}$$

A.4.2 Proof for Lemma 3

Let $\boldsymbol{\nu} \in \mathbb{R}^p$ be expressed as $\boldsymbol{\nu} = \sum_{j=1}^p a_j \mathbf{v}_j$, where the $\mathbf{v}_j$'s are the columns of $\mathbf{V}$. Using the formulas in Equation (A.4),

$$\begin{aligned}\boldsymbol{\nu}^T(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})\boldsymbol{\nu} &= \boldsymbol{\nu}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\boldsymbol{\nu} \\ &= \sum_{j=1}^p d_j^2 a_j^2 + \sum_{j=2}^p n\theta(d_1^2 - d_j^2)a_j^2 \\ &= \sum_{j=1}^p \left[n\theta d_1^2 + (1-n\theta)d_j^2\right]a_j^2 \\ &\geq \min_j \left[n\theta d_1^2 + (1-n\theta)d_j^2\right]\sum_{j=1}^p a_j^2 \\ &\begin{cases}\geq n\theta d_1^2\|\boldsymbol{\nu}\|_2^2 & \text{if } n\theta \leq 1, \\ \geq d_1^2\|\boldsymbol{\nu}\|_2^2 & \text{if } n\theta \geq 1\end{cases} \\ &= \min(n\theta, 1)d_1^2\|\boldsymbol{\nu}\|_2^2.\end{aligned}$$

A.4.3 Proofs for Theorems 4 and 5

Let $\widehat{\boldsymbol{\nu}} = \widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*$. By definition of $\widehat{\boldsymbol{\beta}}$ and using the relation $\widetilde{\mathbf{y}} = \widetilde{\mathbf{X}}\boldsymbol{\beta}^* + \widetilde{\mathbf{w}}$,

$$\begin{aligned}\left\|\widetilde{\mathbf{y}} - \widetilde{\mathbf{X}}\widehat{\boldsymbol{\beta}}\right\|_2^2 &\leq \left\|\widetilde{y} - \widetilde{\mathbf{X}}\boldsymbol{\beta}^*\right\|_2^2, \\ \left\|\widetilde{\mathbf{w}} - \widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}\right\|_2^2 &\leq \|\widetilde{\mathbf{w}}\|_2^2, \\ \widehat{\boldsymbol{\nu}}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\widehat{\boldsymbol{\nu}} &\leq 2(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{w}})^T\widehat{\boldsymbol{\nu}}.\end{aligned} \tag{A.6}$$

By Lemma 2, the LHS is $\geq n\eta\|\widehat{\boldsymbol{\nu}}\|_2^2$, where $n\eta = \min\left[(1-n\theta)n\gamma + n\theta d_1^2, d_1^2\right]$. We can get an upper bound on the RHS:

$$\begin{aligned}
(\widetilde{\mathbf{X}}^T \widetilde{\mathbf{w}})^T \widehat{\boldsymbol{\nu}} &\leq \left\| \widetilde{\mathbf{X}}^T \widetilde{\mathbf{w}} \right\|_\infty \|\widehat{\boldsymbol{\nu}}\|_1 && \text{(Hölder's inequality)} \\
&= \left\| \mathbf{X}^T \mathbf{w} - n\theta \mathbf{A}\boldsymbol{\beta}^* \right\|_\infty (\|\widehat{\boldsymbol{\nu}}_S\|_1 + \|\widehat{\boldsymbol{\nu}}_{S^c}\|_1) \\
&\leq 2 \left\| \mathbf{X}^T \mathbf{w} - n\theta \mathbf{A}\boldsymbol{\beta}^* \right\|_\infty \|\widehat{\boldsymbol{\nu}}_S\|_1 && \text{(Lemma 11)} \\
&\leq 2 \left\| \mathbf{X}^T \mathbf{w} - n\theta \mathbf{A}\boldsymbol{\beta}^* \right\|_\infty \sqrt{|S|}\, \|\widehat{\boldsymbol{\nu}}\|_2 . && \text{(Cauchy-Schwarz)}
\end{aligned}$$

Hence,

$$\begin{aligned}
n\eta \|\widehat{\boldsymbol{\nu}}\|_2^2 &\leq 4 \left\| \mathbf{X}^T \mathbf{w} - n\theta A\boldsymbol{\beta}^* \right\|_\infty \sqrt{|S|}\, \|\widehat{\boldsymbol{\nu}}\|_2 , \\
\|\widehat{\boldsymbol{\nu}}\|_2 &\leq \frac{4\sqrt{|S|} \left\| \mathbf{X}^T \mathbf{w} - n\theta \mathbf{A}\boldsymbol{\beta}^* \right\|_\infty}{n\eta}.
\end{aligned}$$

When $\boldsymbol{\beta}^*$ is aligned with the first principal component of $\mathbf{X}$, $\mathbf{A}\boldsymbol{\beta}^* = 0$. Hence, the first inequality in Equation (2.7) reduces to Equation (2.8).

The proof of Theorem 5 is exactly the same as that for Theorem 4, except that we bound the LHS of Equation (A.6) from below by $\min(n\theta, 1)\, d_1^2 \|\widehat{\boldsymbol{\nu}}\|_2^2$. We can do this due to Lemma 3.

A.4.4 Proof of Theorem 6

As in the proof of Lemma 12, we have Equation (A.2):

$$\frac{1}{2n} \widehat{\boldsymbol{\nu}}^T (\mathbf{X}^T \mathbf{X} + n\theta \mathbf{A}) \widehat{\boldsymbol{\nu}} \leq \frac{\widetilde{\mathbf{w}}^T \widetilde{\mathbf{X}} \widehat{\boldsymbol{\nu}}}{n} + \lambda(\|\widehat{\boldsymbol{\nu}}_S\|_1 - \|\widehat{\boldsymbol{\nu}}_{S^c}\|_1).$$

By Lemma 2, the LHS is $\geq \eta \|\widehat{\boldsymbol{\nu}}\|_2^2 /2$, where $n\eta = \min\left[(1-n\theta)n\gamma + n\theta d_1^2, d_1^2\right]$. By our choice of λ, we may use Equation (A.3) to obtain

$$\begin{aligned}
\frac{\eta}{2} \|\widehat{\boldsymbol{\nu}}\|_2^2 &\leq \frac{1}{2}\lambda \|\widehat{\boldsymbol{\nu}}\|_1 + \lambda(\|\widehat{\boldsymbol{\nu}}_S\|_1 - \|\widehat{\boldsymbol{\nu}}_{S^c}\|_1) \\
&\leq \frac{3\lambda}{2} \|\widehat{\boldsymbol{\nu}}_S\|_1 \leq \frac{3\lambda}{2} \sqrt{|S|}\, \|\widehat{\boldsymbol{\nu}}\|_2 , \\
\|\widehat{\boldsymbol{\nu}}\|_2 &\leq \frac{3\lambda\sqrt{|S|}}{\eta},
\end{aligned}$$

as required.

A.4.5 Proof of Theorem 7

By Equation (A.1),

$$\begin{aligned}
0 &\le \frac{1}{2n}\widehat{\boldsymbol{\nu}}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\widehat{\boldsymbol{\nu}} \\
&\le \frac{\widetilde{\mathbf{w}}^T\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}}{n} + \lambda(\|\boldsymbol{\beta}^*\|_1 - \|\boldsymbol{\beta}^* + \widehat{\boldsymbol{\nu}}\|_1) \\
&\le \frac{1}{n}\left\|\widetilde{\mathbf{X}}^T\widetilde{\mathbf{w}}\right\|_\infty \|\widehat{\boldsymbol{\nu}}\|_1 + \lambda(\|\boldsymbol{\beta}^*\|_1 - \|\boldsymbol{\beta}^* + \widehat{\boldsymbol{\nu}}\|_1) && \text{(Hölder's inequality)} \\
&\le \left(\frac{1}{n}\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty - \lambda\right)\|\widehat{\boldsymbol{\nu}}\|_1 + 2\lambda\|\boldsymbol{\beta}^*\|_1 && \text{(triangle inequality)} \\
&\le -\frac{\lambda}{2}\|\widehat{\boldsymbol{\nu}}\|_1 + 2\lambda\|\boldsymbol{\beta}^*\|_1, && \text{(by choice of } \lambda\text{)},
\end{aligned}$$

so $\|\widehat{\boldsymbol{\nu}}\|_1 \le 4\|\boldsymbol{\beta}^*\|_1$. Using Equation (A.1) again,

$$\begin{aligned}
\frac{1}{2n}\left\|\mathbf{X}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*)\right\|_2^2 = \frac{1}{2n}\widehat{\boldsymbol{\nu}}^T\mathbf{X}^T\mathbf{X}\widehat{\boldsymbol{\nu}} &\le \frac{1}{2n}\widehat{\boldsymbol{\nu}}^T(\mathbf{X}^T\mathbf{X} + n\theta\mathbf{A})\widehat{\boldsymbol{\nu}} \\
&\le \frac{1}{n}\left\|\widetilde{\mathbf{X}}^T\widetilde{\mathbf{w}}\right\|_\infty \|\widehat{\boldsymbol{\nu}}\|_1 + \lambda\|\widehat{\boldsymbol{\nu}}\|_1 \\
&\le \frac{3\lambda}{2}\|\widehat{\boldsymbol{\nu}}\|_1 \\
&\le 6\lambda\|\boldsymbol{\beta}^*\|_1.
\end{aligned}$$

Multiplying both sides by 2 establishes the claim.

A.4.6 Proof of Theorem 8

Let $\kappa = \min(n\theta, 1)d_1^2$. Mimicking the first part of the proof of Theorem 7 and using Lemma 3, we have

$$\begin{aligned}
-\frac{\lambda}{2}\|\widehat{\boldsymbol{\nu}}\|_1 + 2\lambda\|\boldsymbol{\beta}^*\|_1 &\ge \frac{1}{2n}\widehat{\boldsymbol{\nu}}^T(\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})\widehat{\boldsymbol{\nu}} \ge \kappa\|\widehat{\boldsymbol{\nu}}\|_2^2 \\
&\ge \kappa\frac{\|\widehat{\boldsymbol{\nu}}\|_1^2}{p}, && \text{(Cauchy-Schwarz)}, \\
\frac{\kappa}{p}\|\widehat{\boldsymbol{\nu}}\|_1^2 + \frac{\lambda}{2}\|\widehat{\boldsymbol{\nu}}\|_1 - 2\lambda\|\boldsymbol{\beta}^*\|_1 &\le 0, \\
\|\widehat{\boldsymbol{\nu}}\|_1 &\le \frac{-\lambda/2 + \sqrt{\lambda^2/4 + 8\lambda\|\boldsymbol{\beta}^*\|_1\kappa/p}}{2\kappa/p} \\
&= \frac{-\lambda p + \sqrt{\lambda^2p^2 + 32\lambda\|\boldsymbol{\beta}^*\|_1\kappa p}}{4\kappa}.
\end{aligned}$$

Thus, mimicking the second part of the proof of Theorem 7,

$$\begin{aligned}
\frac{1}{2n}\left\|\mathbf{X}(\widehat{\boldsymbol{\beta}}-\boldsymbol{\beta}^*)\right\|_2^2 &\le \frac{3\lambda}{2}\left\|\widehat{\boldsymbol{\nu}}\right\|_1 \\
&\le \frac{3\lambda}{2}\frac{-\lambda p+\sqrt{\lambda^2p^2+32\lambda\left\|\boldsymbol{\beta}^*\right\|_1\kappa p}}{4\kappa}, \\
\frac{1}{n}\left\|\mathbf{X}(\widehat{\boldsymbol{\beta}}-\boldsymbol{\beta}^*)\right\|_2^2 &\le \frac{3\lambda(-\lambda p+\sqrt{\lambda^2p^2+32\lambda\left\|\boldsymbol{\beta}^*\right\|_1\kappa p})}{4\kappa},
\end{aligned}$$

as required. To show that the RHS above is indeed a better bound than that in Theorem 7:

$$\begin{aligned}
& & 12\lambda\left\|\boldsymbol{\beta}^*\right\|_1 &\ge \frac{3\lambda(-\lambda p+\sqrt{\lambda^2p^2+32\lambda\left\|\boldsymbol{\beta}^*\right\|_1\kappa p})}{4\kappa} \\
&\Leftrightarrow & 16\kappa\left\|\boldsymbol{\beta}^*\right\|_1 &\ge -\lambda p+\sqrt{\lambda^2p^2+32\lambda\left\|\boldsymbol{\beta}^*\right\|_1\kappa p} \\
&\Leftrightarrow & (\lambda p+16\kappa\left\|\boldsymbol{\beta}^*\right\|_1)^2 &\ge \lambda^2p^2+32\lambda\left\|\boldsymbol{\beta}^*\right\|_1\kappa p \\
&\Leftrightarrow & 256\kappa^2\left\|\boldsymbol{\beta}^*\right\|_1^2 &\ge 0,
\end{aligned}$$

which is obviously true.

A.4.7 Proof of Theorem 9

We have

$$\begin{aligned}
\frac{1}{2n}\widehat{\boldsymbol{\nu}}^T(\mathbf{X}^T\mathbf{X}+n\theta A)\widehat{\boldsymbol{\nu}} &\le \frac{\widetilde{\mathbf{w}}^T\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}}{n}+\lambda(\left\|\widehat{\boldsymbol{\nu}}_S\right\|_1-\left\|\widehat{\boldsymbol{\nu}}_{S^c}\right\|_1) && \text{((A.2) in Lemma 12)} \\
&\le \frac{1}{n}\left\|\widetilde{\mathbf{X}}^T\widetilde{\mathbf{w}}\right\|_\infty\left\|\widehat{\boldsymbol{\nu}}\right\|_1+\lambda\left\|\widehat{\boldsymbol{\nu}}\right\|_1 && \text{(Hölder's inequality)} \\
&\le \frac{3\lambda}{2}\left\|\widehat{\boldsymbol{\nu}}\right\|_1 && \text{(choice of } \lambda\text{)} \\
&\le \frac{3\lambda}{2}\sqrt{|S|}\left\|\widehat{\boldsymbol{\nu}}\right\|_2. && \text{(Cauchy-Schwarz inequality)}
\end{aligned}$$

By Lemma 12, $\widehat{\boldsymbol{\nu}}$ lies in the set $\{\boldsymbol{\nu}\in\mathbb{R}^p : \left\|\boldsymbol{\nu}_{S^c}\right\|_1 \le 3\left\|\boldsymbol{\nu}_S\right\|_1\}$, and so the restricted eigenvalue condition and Lemma 2 apply, i.e. $\left\|\widehat{\boldsymbol{\nu}}\right\|_2^2 \le \left\|\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}\right\|_2^2/(n\eta)$, where $n\eta = \min\left[(1-n\theta)n\gamma+n\theta d_1^2, d_1^2\right]$. Hence,

$$\begin{aligned}
\frac{1}{2n}\widehat{\boldsymbol{\nu}}^T(\mathbf{X}^T\mathbf{X}+n\theta\mathbf{A})\widehat{\boldsymbol{\nu}} &\le \frac{3\lambda}{2}\sqrt{|S|}\left\|\widehat{\boldsymbol{\nu}}\right\|_2 \le \frac{3\lambda}{2}\sqrt{|S|}\sqrt{\frac{1}{n\eta}}\left\|\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}\right\|_2, \\
\left\|\mathbf{X}\widehat{\boldsymbol{\nu}}\right\|_2 \le \left\|\widetilde{\mathbf{X}}\widehat{\boldsymbol{\nu}}\right\|_2 &\le 3n\lambda\sqrt{|S|}\sqrt{\frac{1}{n\eta}}, \\
\frac{\left\|\mathbf{X}\widehat{\boldsymbol{\nu}}\right\|_2^2}{n} &\le \frac{9|S|\lambda^2}{\eta},
\end{aligned}$$

as required.

A.5 Effect of θ on bound in Theorem 5

In this Appendix, we investigate the effect that θ has on the bound on the RHS of Theorem 5, i.e. $4\sqrt{|S|}\left\|\mathbf{X}^T\mathbf{w} - n\theta\mathbf{A}\boldsymbol{\beta}^*\right\|_\infty / [\min(n\theta, 1)\, d_1^2]$. Other than θ, this bound depends on a number of different quantities which makes it difficult to perform an exhaustive study. Instead, we perform a numerical investigation with data generated according to Algorithm 10. In this set-up, $\mathbf{X}$ is assumed to have rank 2. We investigate how the value of the bound changes with θ across a range of signal-to-noise ratios (SNR) in the response (denoted by SNR), the proportion of $\boldsymbol{\beta}^*$ that aligns with the first PC of $\mathbf{X}$ (denoted by b), and the relative strength of the first and second PCs of $\mathbf{X}$ (denoted by d_2).

Algorithm 10 *Data generating process for numerical simulation in Appendix A.5*

```
1. Input: Second singular value $d_2 \in [0, 1]$, $b \in [0, 1]$ representing the proportion of $\boldsymbol{\beta}^*$ that is aligned with the first PC of $\mathbf{X}$, and SNR ratio $SNR > 0$. Set $n = 50$, $p = 100$.
2. Generate random vectors $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^n$ and $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^p$ such that their entries are i.i.d. $\mathcal{N}(0, 1)$. Orthogonalize them so that they all have norm 1 and $\mathbf{u}_1^T\mathbf{u}_2 = \mathbf{v}_1^T\mathbf{v}_2 = 0$. Set $\mathbf{X} = \mathbf{u}_1\mathbf{v}_1^T + d_2\mathbf{u}_2\mathbf{v}_2^T$.
3. Compute the SVD of $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. Let $\mathbf{D}_{d_1^2 - d_j^2}$ be the $p \times p$ diagonal matrix with diagonal entries being $d_1^2 - d_j^2$, and compute $\mathbf{A} = \mathbf{V}\mathbf{D}_{d_1^2 - d_j^2}\mathbf{V}^T$.
4. Define $\boldsymbol{\beta}^* = [b\mathbf{v}_1 + (1-b)\mathbf{v}_2] / \|b\mathbf{v}_1 + (1-b)\mathbf{v}_2\|_2$.
5. Compute true signal $\mu = \mathbf{X}\boldsymbol{\beta}^*$. Generate noise vector $\mathbf{w} \in \mathbb{R}^n$ such that its entries are i.i.d. $\mathcal{N}(0, 1)$. Define the respones to be $\mathbf{y} = \mu + \sigma\mathbf{w}$, where $\sigma$ is chosen so that $Var(\mu)/Var(\sigma\mathbf{w}) = SNR$.
```

For each choice of SNR, b and d_2, we generate data according to Algorithm 10 100 times and compute the value of the bound for a grid of $n\theta$ values from 0.3 to 1.25. The mean value of the bound is shown in Figure A.8. We see that the bound generally decreases as the SNR increases, which makes sense as pcLasso should be able to estimate $\boldsymbol{\beta}^*$ more accurately when the signal is stronger. In general the bound decreases as the proportion of $\boldsymbol{\beta}^*$ that lines up with the top PC of $\mathbf{X}$ increases. Again, we expect this, since pcLasso should be most accurate when $\boldsymbol{\beta}^*$ lines up with the top PC of $\mathbf{X}$.

From Figure A.8, it is tempting to conclude that the best value of θ is always $\theta = 1/n$. This is not the case, as we recall that the lines in the figure are a mean over 100 simulation runs. In Figure A.9, we plot the value of the bound vs. $n\theta$ for each of 8 simulation runs, with $SNR = 1$ and $d_2 = 0.5$. We can see that the best value of θ depends very much on data at hand. We note further that the quantity we are plotting is merely an upper bound of $\left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\right\|_2$, not $\left\|\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}^*\right\|_2$ itself. In practice, we recommend selecting θ by cross-validation.

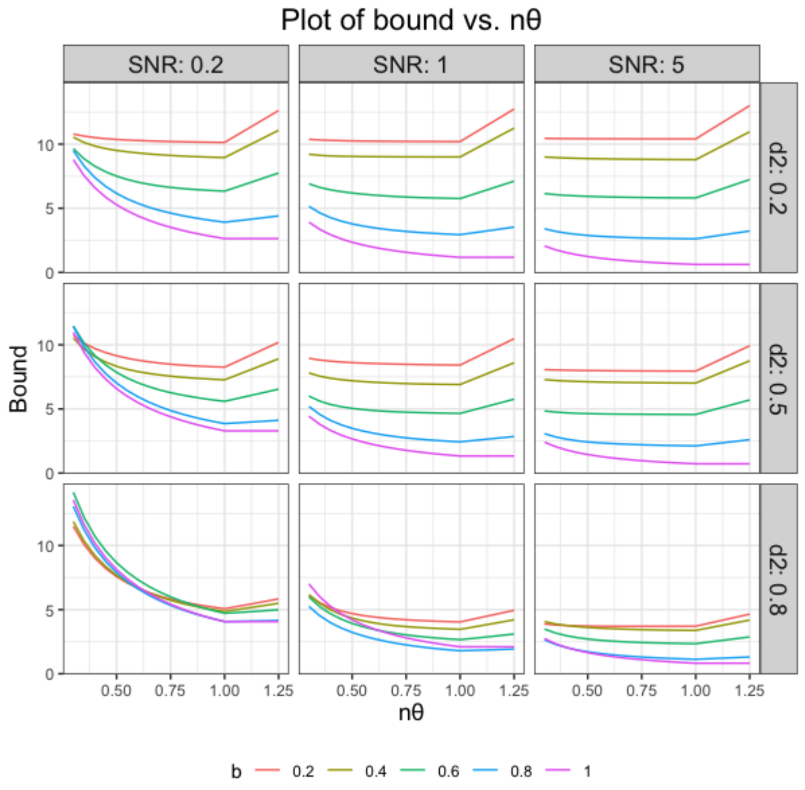

Figure A.8: *Value of the bound on the RHS of Theorem 5 for data generated according to Algorithm 10. Each line is the mean of 100 simulations. The SNR in the response increases as we go from left to right, and the relative strength of the second PC of* $\mathbf{X}$ *increase as we go from top to bottom. Within each panel, each line corresponds to one value of b, the proportion of the true coefficient vector* $\boldsymbol{\beta}^*$ *that aligns with the top PC of* $\mathbf{X}$*.*

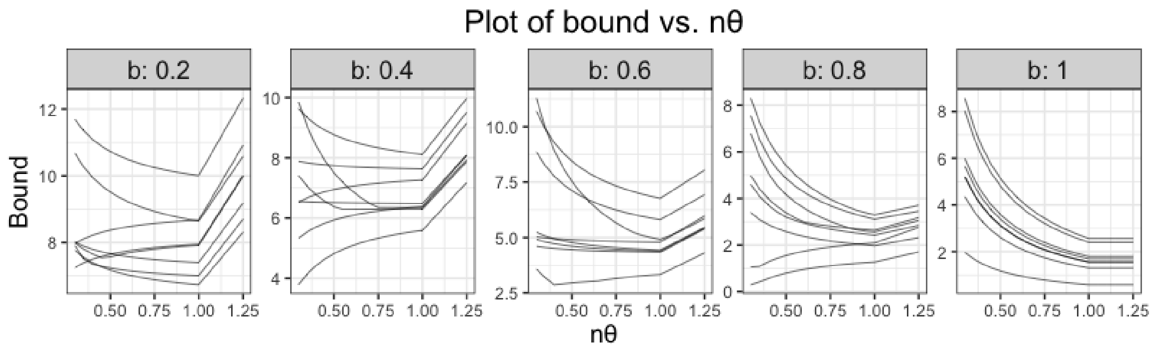

Figure A.9: *Value of the bound on the RHS of Theorem 5 for data generated according to Algorithm 10. Each line represents the bound for one simulation run.* $SNR = 1$ *and* $d_2 = 0.5$*. Each panel corresponds to one value of the parameter b.*

A.6 Strong rules for pcLasso

Recall that in computing the principal components lasso (pcLasso) solution, we typically hold θ fixed and compute the solution for a path of λ values. By casting the pcLasso optimization problem as a lasso problem with a different response and design matrix, we can derive strong rules for pcLasso from that for the lasso.

We first derive strong rules for pcLasso where predictors do not have preassigned groups, i.e. the solution to Equation (2.1). If we define

$$\widetilde{\mathbf{y}} = \begin{pmatrix} \mathbf{y} \\ \mathbf{0} \end{pmatrix}, \qquad \widetilde{\mathbf{X}} = \begin{pmatrix} \mathbf{X} \\ \sqrt{\theta}\mathbf{A}^{1/2} \end{pmatrix}, \tag{A.7}$$

where $\mathbf{A} = \mathbf{V}\mathbf{D}_{d_1^2 - d_j^2}\mathbf{V}^T$, then pcLasso is equivalent to the lasso with response vector $\widetilde{\mathbf{y}}$ and design matrix $\widetilde{\mathbf{X}}$. Fix θ, fix a path of λ values $\lambda_1 \geq \lambda_2 \geq \dots$, and let $\hat{\boldsymbol{\beta}}(\lambda_i)$ denote the pcLasso solution at $\lambda = \lambda_i$. Applying the lasso strong rules Tibshirani et al. (2012) to this set-up, the sequential strong rule for pcLasso for discarding predictor j at $\lambda = \lambda_i$ is

$$\begin{aligned} & |\mathbf{X}_j^T(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}(\lambda_{i-1})) - \theta(\mathbf{A}^{1/2})_j \mathbf{A}^{1/2}\hat{\boldsymbol{\beta}}(\lambda_{i-1})| < 2\lambda_i - \lambda_{i-1}, \\ \Leftrightarrow \quad & |\mathbf{X}_j^T(\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}(\lambda_{i-1})) - \theta(\mathbf{A}\hat{\boldsymbol{\beta}}(\lambda_{i-1}))_j| < 2\lambda_i - \lambda_{i-1}. \end{aligned} \tag{A.8}$$

Next we derive strong rules for pcLasso where predictors come in preassigned non-overlapping groups, i.e. the solution to (2.3). If we define $\mathbf{A}_k = \mathbf{V}_k\mathbf{D}_{d_{k1}^2 - d_{kj}^2}\mathbf{V}_k^T$ for $k = 1, \dots, K$, $\mathbf{A}$ as the block diagonal matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 & & 0 \\ & \ddots & \\ 0 & & \mathbf{A}_K \end{pmatrix},$$

and $\widetilde{\mathbf{y}}$ and $\widetilde{\mathbf{X}}$ as in Equation (A.7), then pcLasso is again equivalent to the lasso with $\widetilde{\mathbf{y}}$ and $\widetilde{\mathbf{X}}$. This results in the same sequential strong rule as in the single group case (Equation A.8), although the matrix $\mathbf{A}$ is defined differently.

Appendix B

Appendix for Feature-Weighted Elastic Net

B.1 Alternative algorithm with $\boldsymbol{\theta}$ as a parameter

Assume that $\mathbf{y}$ and the columns of $\mathbf{X}$ are centered so that $\hat{\beta}_0 = 0$ and we can ignore the intercept term in the rest of the discussion. If we consider $\boldsymbol{\theta}$ as an argument of the objective function, then we wish to solve

$$\begin{aligned}(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\theta}}) &= \underset{\boldsymbol{\beta},\boldsymbol{\theta}}{\operatorname{argmin}}\ J_{\lambda,\alpha}(\boldsymbol{\beta}, \boldsymbol{\theta}) \\ &= \underset{\boldsymbol{\beta},\boldsymbol{\theta}}{\operatorname{argmin}}\ \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=1}^{p} w_j(\boldsymbol{\theta}) \left[\alpha|\beta_j| + \frac{1-\alpha}{2}\beta_j^2\right].\end{aligned}$$

J is not jointly convex $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$, so reaching a global minimum is a difficult task. Instead, we content ourselves with reaching a local minimum. A reasonable approach for doing so is to alternate between optimizing $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$: the steps are outlined in Algorithm 11.

Unfortunately, Algorithm 11 is slow due to repeated solving of the elastic net problem in Step 2(b)ii for each λ_i. The algorithm does not take advantage of the fact that once α and θ are fixed, the elastic net problem can be solved quickly for an entire path of λ values. We have also found that Algorithm 11 does not predict as well as Algorithm 2 in our simulations.

B.2 Proof of Theorem 10

For the moment, consider the more general penalty factor $w_j(\boldsymbol{\theta}) = \left[\sum_{\ell=1}^{p} f(\mathbf{z}_\ell^T \boldsymbol{\theta})\right] / \left[p f(\mathbf{z}_j^T \boldsymbol{\theta})\right]$, where f is some function with range $[0, +\infty)$. (Fwelnet makes the choice $f(x) = e^x$.)

Algorithm 11 *Minimizing the fwelnet objective function via alternating minimization*

```
1. Select a value of $\alpha \in [0,1]$ and a sequence of $\lambda$ values $\lambda_1 > \ldots > \lambda_m$.
2. For $i = 1, \ldots, m$:
    (a) Initialize $\boldsymbol{\beta}^{(0)}(\lambda_i)$ at the elastic net solution for $\lambda_i$. Initialize $\boldsymbol{\theta}^{(0)} = \mathbf{0}$.
    (b) For $k = 0, 1, \ldots$ until convergence:
        i. Fix $\boldsymbol{\beta} = \boldsymbol{\beta}^{(k)}$, update $\boldsymbol{\theta}^{(k+1)}$ via gradient descent. That is, set $\Delta\boldsymbol{\theta} = \left.\frac{\partial J_{\lambda_i,\alpha}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\beta}=\boldsymbol{\beta}^{(k)},\boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}}$ and update $\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \eta\Delta\boldsymbol{\theta}$, where $\eta$ is the step size computed via backtracking line search to ensure that $J_{\lambda_i,\alpha}\left(\boldsymbol{\beta}^{(k)}, \boldsymbol{\theta}^{(k+1)}\right) < J_{\lambda_i,\alpha}\left(\boldsymbol{\beta}^{(k)}, \boldsymbol{\theta}^{(k)}\right)$.
        ii. Fix $\boldsymbol{\theta} = \boldsymbol{\theta}^{(k+1)}$, update $\boldsymbol{\beta}^{(k+1)}$ by solving the elastic net with updated penalty factors $w_j(\boldsymbol{\theta}^{(k+1)})$.
```

First note that if feature j belongs to group k, then $\mathbf{z}_j^T\boldsymbol{\theta} = \theta_k$, and its penalty factor is

$$w_j(\boldsymbol{\theta}) = \frac{\sum_{\ell=1}^p f(\mathbf{z}_\ell^T\boldsymbol{\theta})}{pf(\mathbf{z}_j^T\boldsymbol{\theta})} = \frac{\sum_{\ell=1}^p f(\theta_\ell)}{pf(\theta_k)} = \frac{\sum_{\ell=1}^K p_\ell f(\theta_\ell)}{pf(\theta_k)},$$

where p_ℓ denotes the number of features in group ℓ. Letting $v_k = f(\theta_k)/\left[\sum_{\ell=1}^K p_\ell f(\theta_\ell)\right]$ for $k = 1, \ldots, K$, minimizing the fwelnet objective function (3.2) over $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ reduces to

$$\underset{\boldsymbol{\beta},\boldsymbol{\theta}}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{p}\sum_{k=1}^K \frac{1}{v_k}\left[\alpha\left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right].$$

For fixed $\boldsymbol{\beta}$, we can explicitly determine the v_k values which minimize the expression above. By the Cauchy-Schwarz inequality,

$$\begin{aligned}
&\frac{\lambda}{p}\sum_{k=1}^K \frac{1}{v_k}\left[\alpha\left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right] \\
&= \frac{\lambda}{p}\left(\sum_{k=1}^K \frac{1}{v_k}\left[\alpha\left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right]\right)\left(\sum_{k=1}^K p_k v_k\right) \\
&\geq \frac{\lambda}{p}\left(\sum_{k=1}^K \sqrt{p_k\left[\alpha\left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right]}\right)^2. \qquad \text{(B.1)}
\end{aligned}$$

Note that equality is attainable for (B.1): letting $a_k = \sqrt{\left[\alpha\left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right]/p_k}$, equality occurs when there is some $c \in \mathbb{R}$ such that

$$\begin{aligned} c \cdot \frac{1}{v_k}\left[\alpha \left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right] &= p_k v_k && \text{for all } k, \\ v_k &= \sqrt{c} a_k && \text{for all } k. \end{aligned}$$

Since $\sum_{k=1}^K p_k v_k = 1$, we have $\sqrt{c} = 1/\left(\sum_{k=1}^K p_k a_k\right)$, giving $v_k = a_k/\left(\sum_{k=1}^K p_k a_k\right)$ for all k. A solution for this is $f(\theta_k) = a_k$ for all k, which is feasible for f having range $[0, \infty)$. (Note that if f only has range $(0, \infty)$, the connection still holds if $\lim_{x \to -\infty} f(x) = 0$ or $\lim_{x\to+\infty} f(x) = 0$: the solution will just have $\theta = +\infty$ or $\theta = -\infty$.)

Thus, the fwelnet solution is

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{p}\left(\sum_{k=1}^K \sqrt{p_k\left[\alpha\left\|\boldsymbol{\beta}^{(k)}\right\|_1 + \frac{1-\alpha}{2}\left\|\boldsymbol{\beta}^{(k)}\right\|_2^2\right]}\right)^2. \tag{B.2}$$

When $\alpha = 0$, the penalty term is convex. Writing in constrained form, (B.2) becomes minimizing $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2/2$ subject to

$$\begin{aligned} \left(\sum_{k=1}^K \sqrt{p_k}\left\|\boldsymbol{\beta}^{(k)}\right\|_2\right)^2 &\leq C \text{ for some constant } C, \\ \sum_{k=1}^K \sqrt{p_k}\left\|\boldsymbol{\beta}^{(k)}\right\|_2 &\leq \sqrt{C}. \end{aligned}$$

Converting back to Lagrange form again, there is some $\lambda' \geq 0$ such that the fwelnet solution is

$$\underset{\boldsymbol{\beta}}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|_2^2 + \lambda' \sum_{k=1}^K \sqrt{p_k}\left\|\boldsymbol{\beta}^{(k)}\right\|_2.$$

B.3 Details on simulation study in Section 3.5

B.3.1 Setting 1: Noisy version of the true $\boldsymbol{\beta}$

1. Set $n = 100$, $p = 50$, $\boldsymbol{\beta} \in \mathbb{R}^{50}$ with $\beta_j = 2$ for $j = 1, \dots, 5$, $\beta_j = -1$ for $j = 6, \dots, 10$, and $\beta_j = 0$ otherwise.
2. Generate $x_{ij} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$ for $i = 1, \dots, n$ and $j = 1, \dots, p$.
3. For each $SNR_y \in \{0.5, 1, 2\}$ and $SNR_Z \in \{0.5, 2, 10\}$:
   (a) Compute $\sigma_y^2 = \left(\sum_{j=1}^p \beta_j^2\right)/SNR_y$.
   (b) Generate $y_i = \sum_{j=1}^p x_{ij}\beta_j + \varepsilon_i$, where $\varepsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_y^2)$ for $i = 1, \dots, n$.

   (c) Compute $\sigma_Z^2 = \text{Var}(|\boldsymbol{\beta}|)/SNR_Z$.
   (d) Generate $z_j = |\beta_j| + \eta_j$, where $\eta_j \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_Z^2)$. Treat this as a column matrix to get $\mathbf{Z} \in \mathbb{R}^{p \times 1}$.

B.3.2 Setting 2: Grouped data setting

1. Set $n = 100$, $p = 150$.
2. For $j = 1, \dots, p$ and $k = 1, \dots 15$, set $z_{jk} = 1$ if $10(k-1) < j \leq 10k$, $z_{jk} = 0$ otherwise.
3. Generate $\boldsymbol{\beta} \in \mathbb{R}^{150}$ with $\beta_j = 3$ or $\beta_j = -3$ with equal probability for $j = 1, \dots, 10G$, $\beta_j = 0$ otherwise. $G = 1$ for the first scenario where the response depends on the first group only, and $G = 4$ for the second scenario where it depends on the first 4 groups.
4. Generate $x_{ij} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$ for $i = 1, \dots, n$ and $j = 1, \dots, p$.
5. For each $SNR_y \in \{0.5, 1, 2\}$:
   (a) Compute $\sigma_y^2 = \left(\sum_{j=1}^p \beta_j^2\right) / SNR_y$.
   (b) Generate $y_i = \sum_{j=1}^p x_{ij}\beta_j + \varepsilon_i$, where $\varepsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_y^2)$ for $i = 1, \dots, n$.

B.3.3 Setting 3: Noise variables

1. Set $n = 100$, $p = 100$, $\boldsymbol{\beta} \in \mathbb{R}^{100}$ with $\beta_j = 2$ for $j = 1, \dots, 10$, and $\beta_j = 0$ otherwise.
2. Generate $x_{ij} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$ for $i = 1, \dots, n$ and $j = 1, \dots, p$.
3. For each $SNR_y \in \{0.5, 1, 2\}$:
   (a) Compute $\sigma_y^2 = \left(\sum_{j=1}^p \beta_j^2\right) / SNR_y$.
   (b) Generate $y_i = \sum_{j=1}^p x_{ij}\beta_j + \varepsilon_i$, where $\varepsilon_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma_y^2)$ for $i = 1, \dots, n$.
   (c) Generate $z_{jk} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$ for $j = 1, \dots, p$ and $k = 1, \dots 10$. This gives a matrix $\mathbf{Z} \in \mathbb{R}^{p \times 10}$.

B.4 Details on simulation study in Section 3.7

1. Set $n = 150$, $p = 50$.
2. Generate $\boldsymbol{\beta}_1 \in \mathbb{R}^{50}$ with

$$\beta_{1,j} = \begin{cases} 5 \text{ or } -5 \text{ with equal probability} & \text{for } j = 1, \dots, 5, \\ 2 \text{ or } -2 \text{ with equal probability} & \text{for } j = 6, \dots, 10, \\ 0 & \text{otherwise.} \end{cases}$$

3. Generate $\boldsymbol{\beta}_2 \in \mathbb{R}^{50}$ with

$$\beta_{2,j} = \begin{cases} 5 \text{ or } -5 \text{ with equal probability} & \text{for } j = 1, \ldots, 5, \\ 2 \text{ or } -2 \text{ with equal probability} & \text{for } j = 11, \ldots, 15, \\ 0 & \text{otherwise.} \end{cases}$$

4. Generate $x_{ij} \overset{i.i.d.}{\sim} \mathcal{N}(0,1)$ for $i = 1, \ldots, n$ and $j = 1, \ldots, p$.

5. Generate response 1, $\mathbf{y}_1 \in \mathbb{R}^{150}$, in the following way:

   (a) Compute $\sigma_1^2 = \left(\sum_{j=1}^p \beta_{1,j}^2\right)/0.5$.

   (b) Generate $y_{1,i} = \sum_{j=1}^p x_{ij}\beta_{1,j} + \varepsilon_{1,i}$, where $\varepsilon_{1,i} \overset{i.i.d.}{\sim} \mathcal{N}\left(0, \sigma_1^2\right)$ for $i = 1, \ldots, n$.

6. Generate response 2, $\mathbf{y}_2 \in \mathbb{R}^{150}$, in the following way:

   (a) Compute $\sigma_2^2 = \left(\sum_{j=1}^p \beta_{2,j}^2\right)/1.5$.

   (b) Generate $y_{2,i} = \sum_{j=1}^p x_{ij}\beta_{2,j} + \varepsilon_{2,i}$, where $\varepsilon_{2,i} \overset{i.i.d.}{\sim} \mathcal{N}\left(0, \sigma_2^2\right)$ for $i = 1, \ldots, n$.

Appendix C

Appendix for Reluctant Generalized Additive Models

C.1 Full details of simulation study

In all the simulations that follow, the feature values X_{ij} are independent draws from the Unif$[-1, 1]$ distribution. The response is $y_i = \mu_i + \varepsilon_i = f(X_{i1}, \ldots, X_{ip}) + \varepsilon_i$, where f is a function that depends on the simulation and the ε_i's are independent $\mathcal{N}(0, \sigma^2)$ draws. The data generating process for the signal is such that the linear and non-linear components are orthogonal. σ^2 is set so that the data has the desired signal-to-noise ratio (SNR).

We compare the following methods across a range of settings:

1. The null model, i.e. mean of the training responses,
2. The lasso (Tibshirani, 1996),
3. Generalized additive model selection (GAMSEL) (Chouldechova and Hastie, 2015),
4. Reluctant generalized additive modeling (RGAM), where non-linear features are constructed for all p main effects, and
5. RGAM where non-linear features are only constructed for the main effects which are in the active set after Step 1 of Algorithm 5 (denoted RGAM_SEL).

For all methods, 5-fold cross-validation (CV) was performed to select the hyperparameter λ only: default values were used for all other hyperparameters. Each boxplot is the result of 30 simulation runs. The test error metric is mean-squared error where the target is the true signal value, i.e. $\mathbb{E}[(\hat{y}_{test} - \mu_{test})^2]$ instead of $\mathbb{E}[(\hat{y}_{test} - y_{test})^2]$. (With this test error metric, the oracle which knows the data generating model would have a test error of 0.) Test error is estimated using 5,000 test points.

For each simulation setting, the results are presented in a 3×3 panel. Each row of the panel corresponds to the one of three SNR values: 1, 2 or 5. In each row, the left-most plot presents test error as a fraction of the test error achieved by the null model (denoted by the dotted red line). The middle plot in each row presents the number of features each model selected, with the two numbers on top of the boxplots being the median number of linear components and non-linear components selected. The number of true features is indicated by the dotted red line, and the number of true linear and non-linear components is in the plot's subtitle. (Note that the number of non-zero features is not necessarily the sum of the number of non-zero linear and non-linear components: this is because a feature can have both a linear and non-linear component.) The right-most plot in each row presents the number of true features each method selected, with the total number of true features indicated by the dotted red line.

Note: The simulations that follow are not presented in the same order as in the main text.

C.1.1 Small n and p, fully linear signal

$n = 100$, $p = 200$, $\boldsymbol{\mu} = \sum_{j=1}^{10} \mathbf{X}_j$.

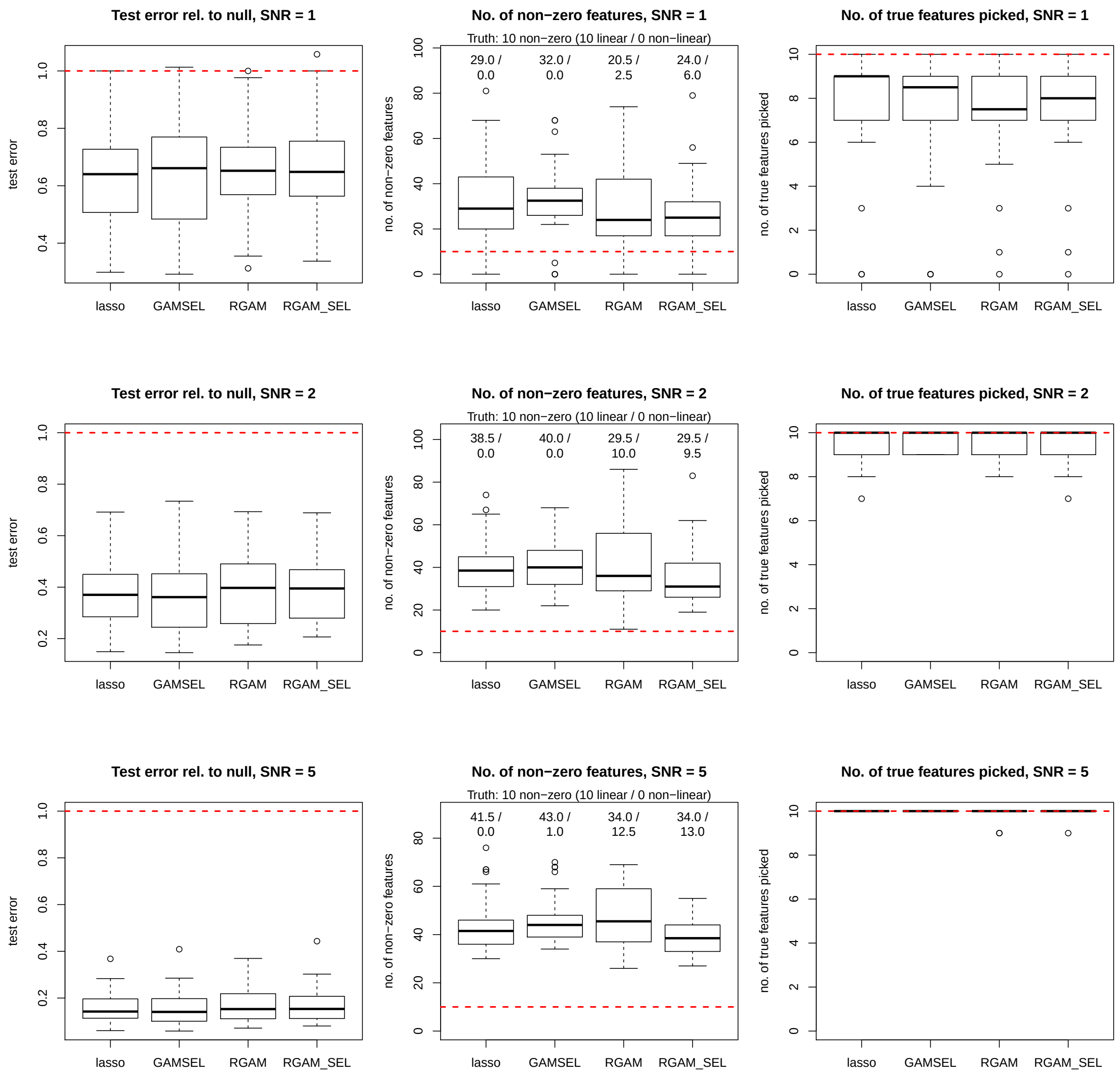

Figure C.1: Results for simulation setting 1: Small n and p, fully linear signal.

C.1.2 Small n and p, hierarchical non-linear signal

$n = 100$, $p = 200$, $\boldsymbol{\mu} = \sum_{j=1}^{5}[\mathbf{X}_j + 2(3\mathbf{X}_j^2 - 1)/3]$.

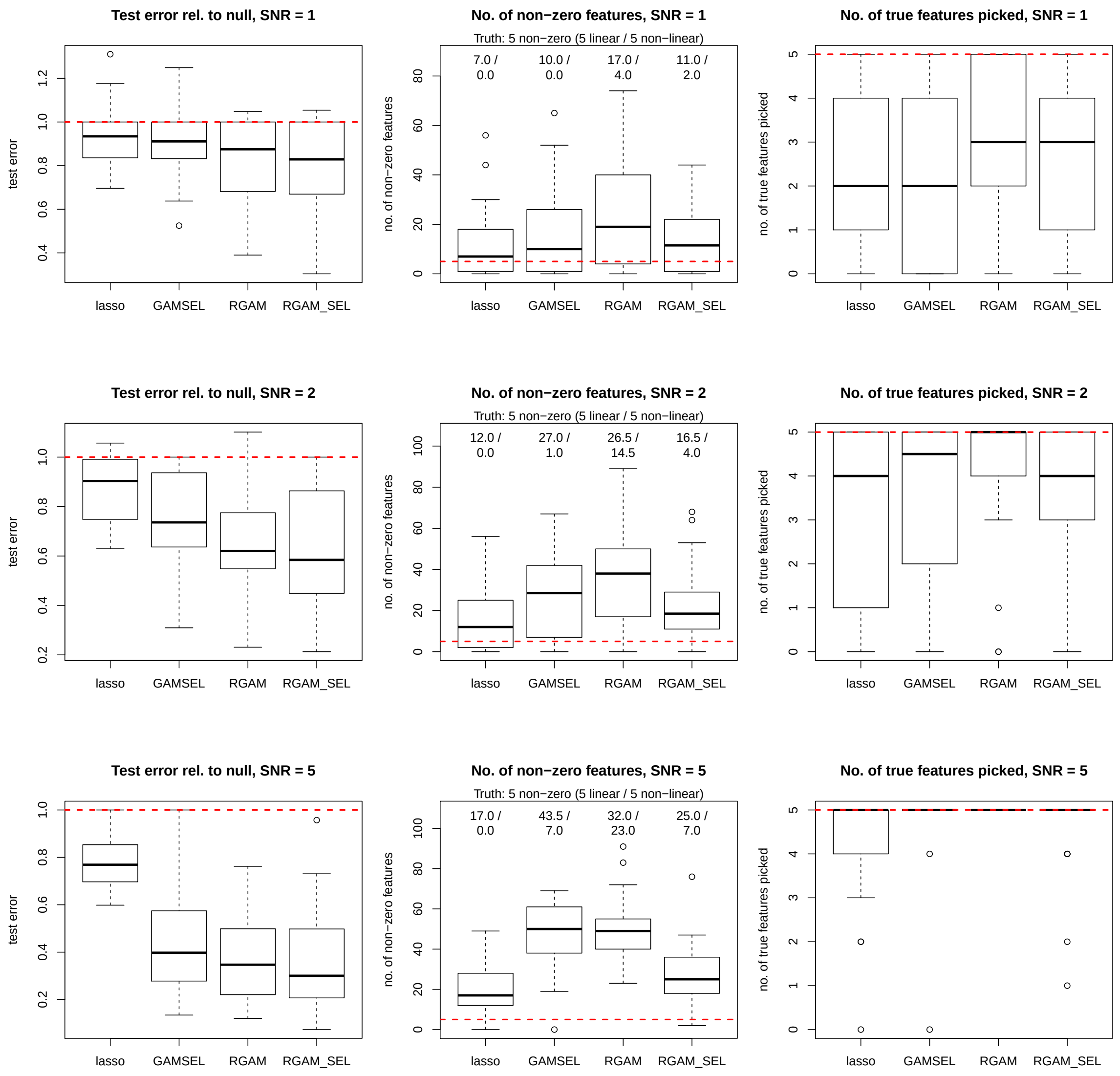

Figure C.2: Results for simulation setting 2: Small n and p, hierarchical non-linear signal.

C.1.3 Small n and p, purely non-linear signal

$n = 100$, $p = 200$, $\boldsymbol{\mu} = \sum_{j=1}^{5} 2(5\mathbf{X}_j^3 - 3\mathbf{X}_j)$.

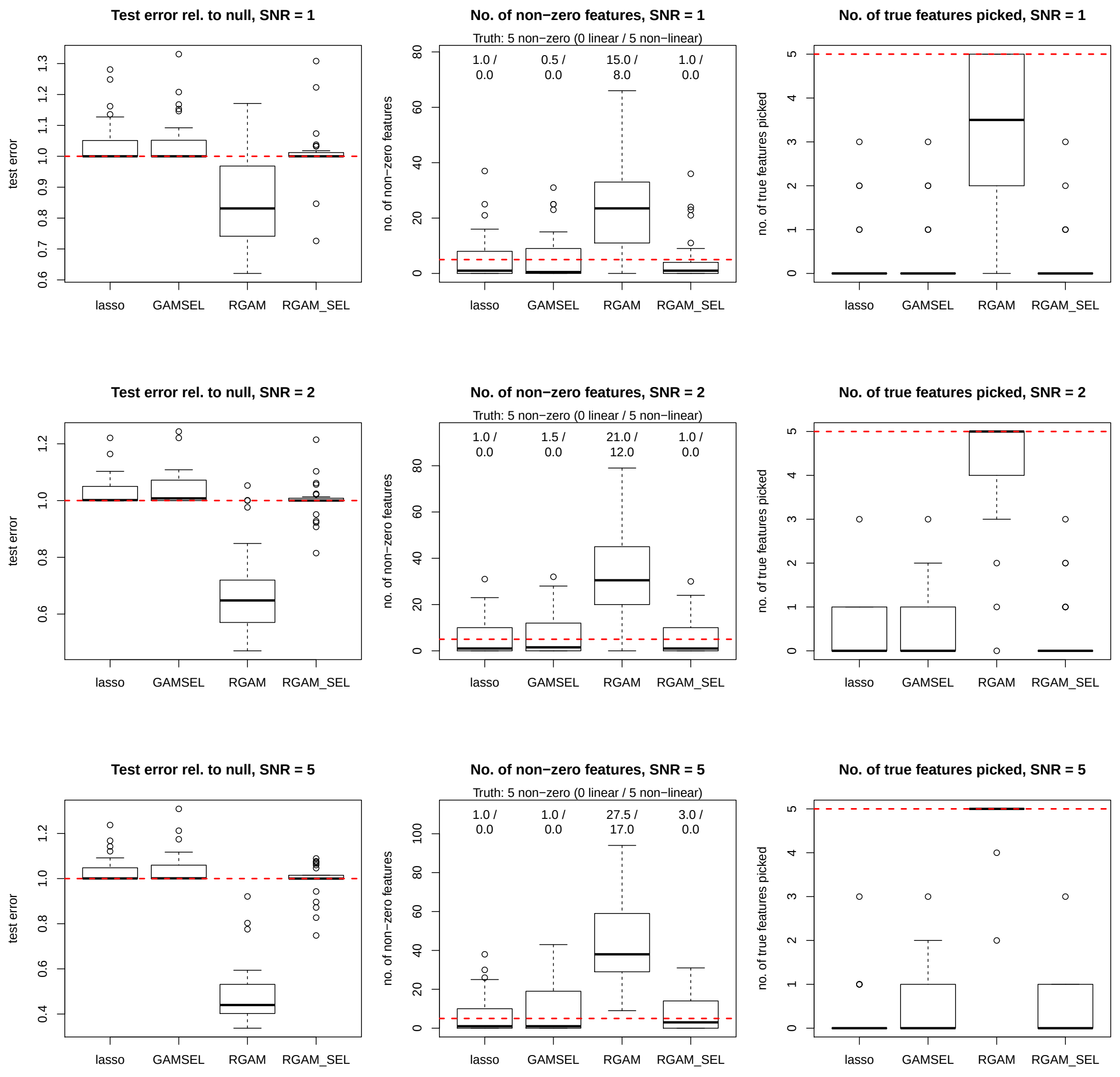

Figure C.3: Results for simulation setting 3: Small n and p, purely non-linear signal.

C.1.4 Small n and p, linear and non-hierarchical non-linear signal

$n = 100$, $p = 200$, $\boldsymbol{\mu} = \sum_{j=1}^{5} \mathbf{X}_j + \sum_{j=6}^{10} 2(3\mathbf{X}_j^2 - 1)/3$.

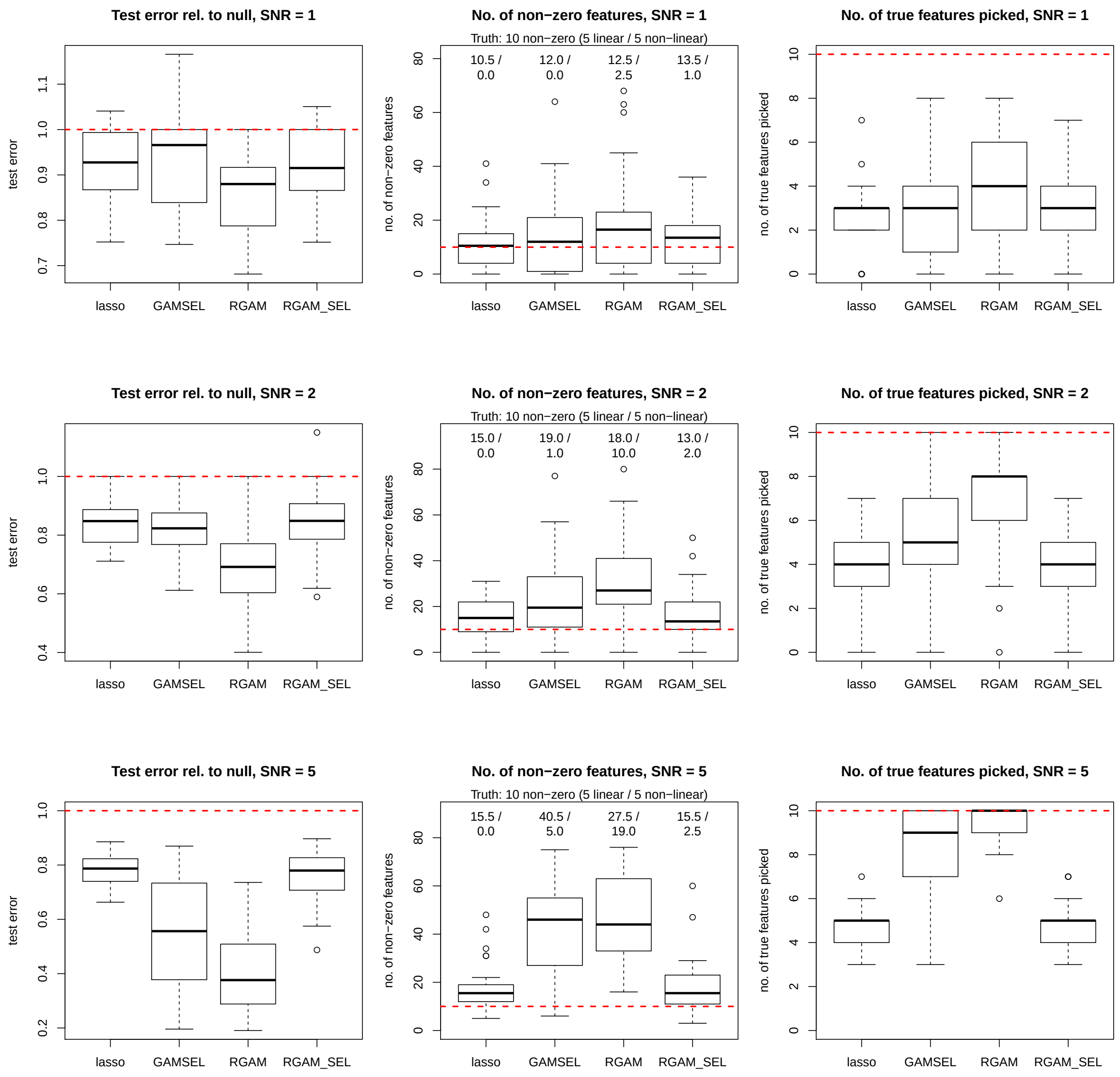

Figure C.4: Results for simulation setting 4: Small n and p, linear and non-hierarchical non-linear signal.

C.1.5 Small n and p, hierarchical and non-hierarchical non-linear signal

$n = 100$, $p = 200$, $\boldsymbol{\mu} = \sum_{j=1}^{5}[\mathbf{X}_j + 3(5\mathbf{X}_j^3 - 3\mathbf{X}_j)/4] + \sum_{j=6}^{8} 0.85(3\mathbf{X}_j^2 - 1)$.

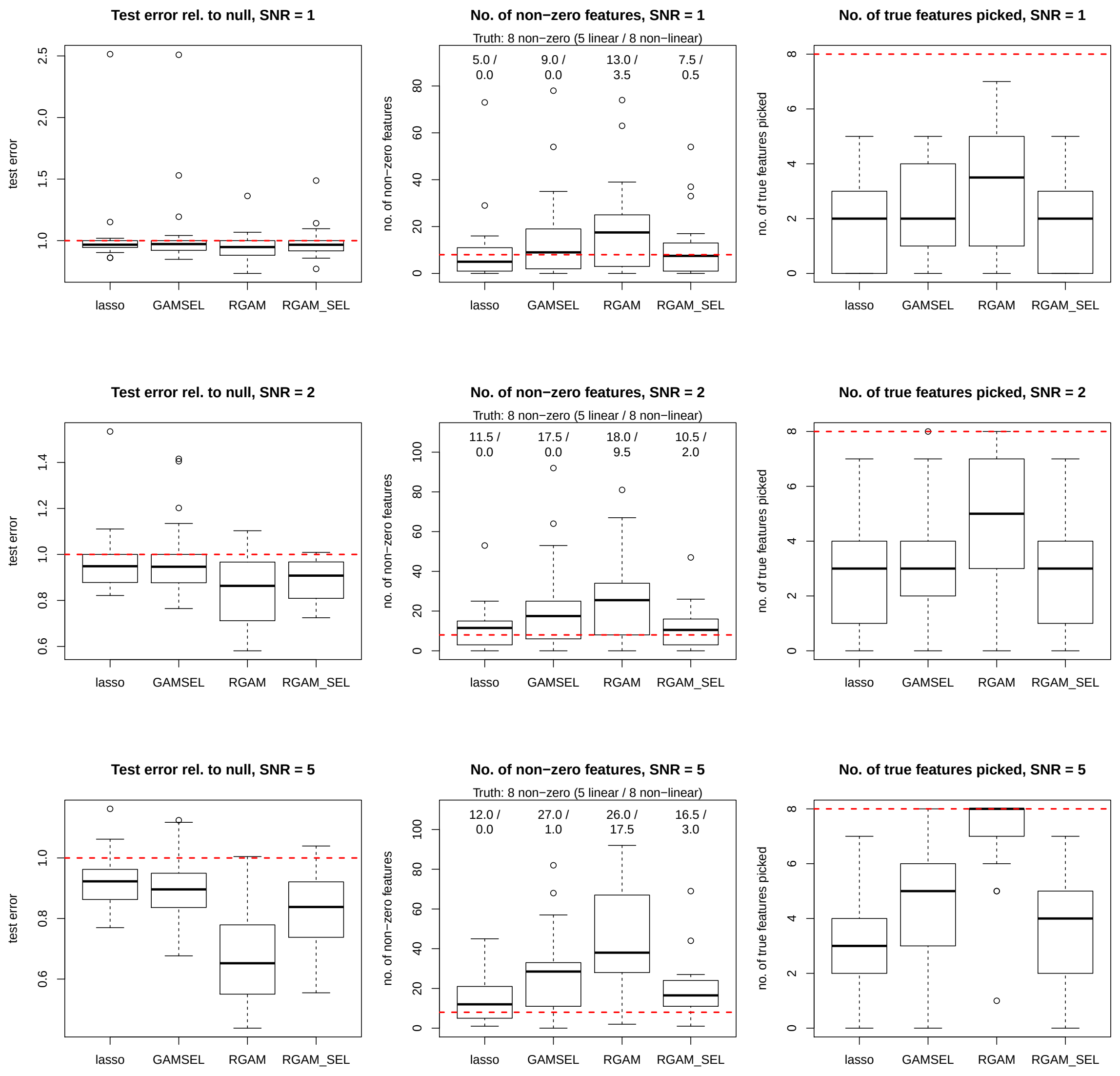

Figure C.5: Results for simulation setting 5: Small n and p, hierarchical and non-hierarchical non-linear signal.

C.1.6 Large n and p, hierarchical and non-hierarchical non-linear signal

$n = 1,000$, $p = 500$, $\boldsymbol{\mu} = \sum_{j=1}^{20} \mathbf{X}_j + \sum_{j=1}^{20} 3(5\mathbf{X}_j^3 - 3\mathbf{X}_j)/4 + \sum_{j=21}^{28}(3\mathbf{X}_j^2 - 1)$.

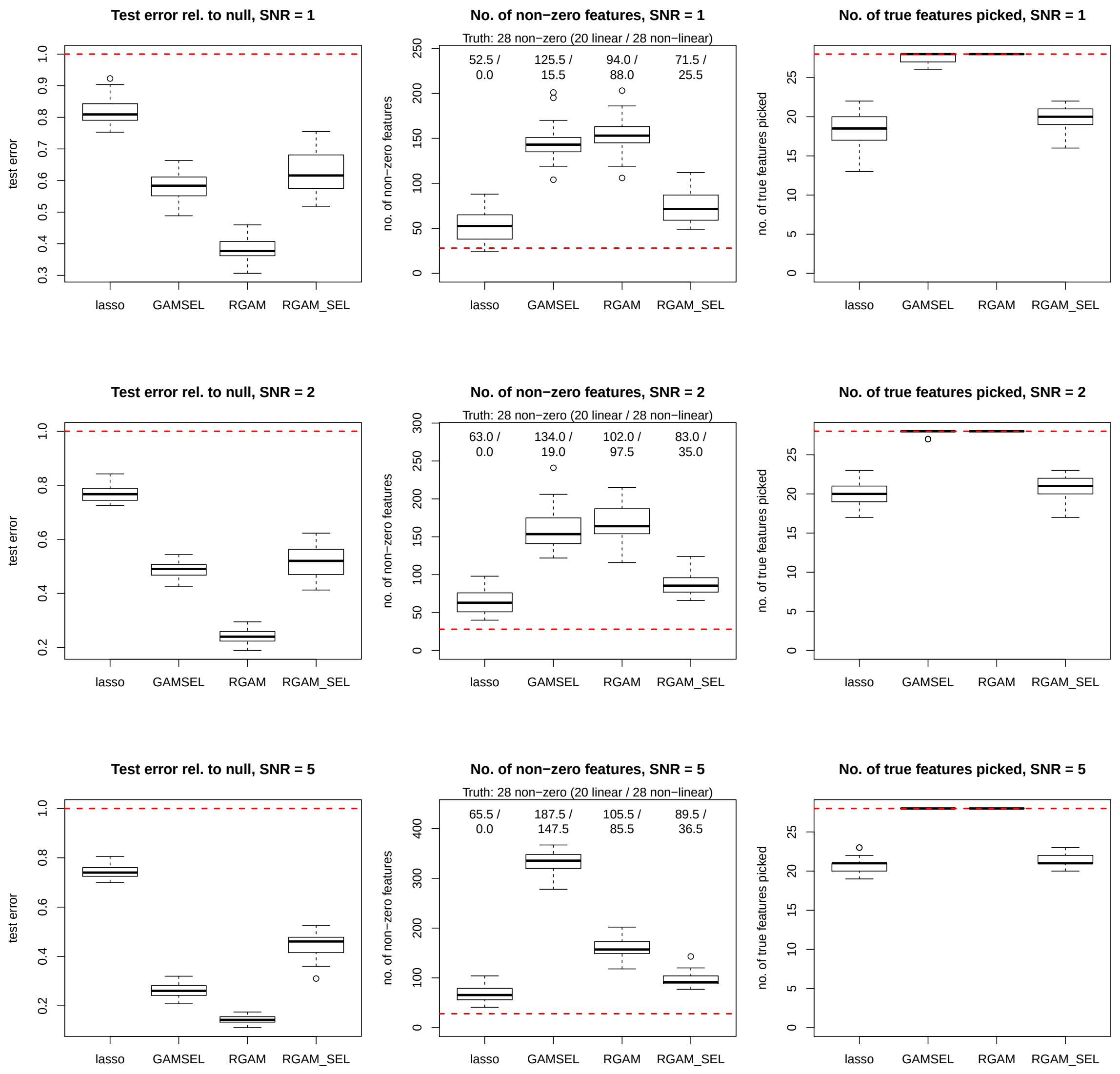

Figure C.6: Results for simulation setting 6: Large n and p, hierarchical and non-hierarchical non-linear signal.

Bibliography

Andersen, P. K. and Gill, R. D. (1982). Cox's Regression Model for Counting Processes: A Large Sample Study. *Annals of Statistics*, 10(4):1100–1120.

Baik, J. and Silverstein, J. W. (2006). Eigenvalues of Large Sample Covariance Matrices of Spiked Population Models. *Journal of Multivariate Analysis*, 97(6):1382–1408.

Bates, S., Hastie, T., and Tibshirani, R. (2021). Cross-Validation: What Does It Estimate and How Well Does It Do It? *ArXiv e-Prints*, pages 1–38.

Bergersen, L. C., Glad, I. K., and Lyng, H. (2011). Weighted Lasso with Data Integration. *Statistical Applications in Genetics and Molecular Biology*, 10(1).

Bergstra, J. and Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305.

Boulesteix, A.-L., De Bin, R., Jiang, X., and Fuchs, M. (2017). IPF-LASSO: Integrative L1-Penalized Regression with Penalty Factors for Prediction Based on Multi-Omics Data. *Computational and Mathematical Methods in Medicine*, 2017:1–14.

Breheny, P. and Huang, J. (2011). Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, 5(1):232–253.

Breheny, P. and Huang, J. (2015). Group Descent Algorithms for Nonconvex Penalized Linear and Logistic Regression Models with Grouped Predictors. *Statistics and Computing*, 25(2):173–187.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

Breslow, N. E. (1972). Contribution to the Discussion of the Paper by D. R. Cox. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34:216–217.

Cai, T. T., Ma, Z., and Wu, Y. (2013). Sparse PCA: Optimal Rates and Adaptive Estimation. *Annals of Statistics*, 41(6):3074–3110.

Chouldechova, A. and Hastie, T. (2015). Generalized Additive Model Selection. *arXiv preprint arXiv:1506.03850*, pages 1–24.

Cox, D. R. (1972). Regression Models and Life-Tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–220.

Efron, B. (1986). How Biased is the Apparent Error Rate of a Prediction Rule? *Journal of the American Statistical Association*, 81(394):461–470.

Efron, B. (2012). *Large-Scale Inference: Empirical Bayes Methods for Estimation, Testing, and Prediction*, volume 1. Cambridge University Press.

Erez, O., Romero, R., Maymon, E., Chaemsaithong, P., Done, B., Pacora, P., Panaitescu, B., Chaiworapongsa, T., Hassan, S. S., and Tarca, A. L. (2017). The Prediction of Late-Onset Preeclampsia: Results from a Longitudinal Proteomics Study. *PLoS ONE*, 12(7):e0181468.

Fan, J. and Li, R. (2001). Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. *Journal of the American Statistical Association*, 96(456):1348–1360.

Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, 33(1):1–24.

Goeman, J. J., Meijer, R. J., and Chaturvedi, N. (2018). *Penalized: L1 (Lasso and Fused Lasso) and L2 (Ridge) Penalized Estimation in GLMs and in the Cox Model.* R package version 0.9-51.

Gorst-Rasmussen, A. and Scheike, T. H. (2012). Coordinate descent methods for the penalized semiparametric additive hazards model. *Journal of Statistical Software*, 47(9):1–17.

Hastie, T. and Efron, B. (2013). *lars: Least Angle Regression, Lasso and Forward Stagewise.* R package version 1.2.

Hastie, T. and Tibshirani, R. (1986). Generalized Additive Models. *Statistical Science*, pages 297–310.

Hastie, T., Tibshirani, R., and Tibshirani, R. (2020). Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons. *Statistical Science*, 35(4):579–592.

Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations.* CRC press.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67.

Jabeen, M., Yakoob, M. Y., Imdad, A., and Bhutta, Z. A. (2011). Impact of Interventions to Prevent and Manage Preeclampsia and Eclampsia on Stillbirths. *BMC Public Health*, 11(S3):S6.

Johnstone, I. M. and Lu, A. Y. (2009). On Consistency and Sparsity for Principal Components Analysis in High Dimensions. *Journal of the American Statistical Association*, 104(486):682–693.

Jolliffe, I. T., Trendafilov, N. T., and Uddin, M. (2003). A Modified Principal Component Technique Based on the LASSO. *Journal of Computational and Graphical Statistics*, 12(3):531–547.

Li, C. and Li, H. (2008). Network-Constrained Regularization and Variable Selection for Analysis of Genomic Data. *Bioinformatics*, 24(9):1175–1182.

Lin, Y. and Zhang, H. H. (2006). Component Selection and Smoothing in Multivariate Nonparametric Regression. *Annals of Statistics*, 34(5):2272–2297.

Lokhorst, J., Venables, B., Turlach, B., and Maechler, M. (2020). *lasso2: L1 Constrained Estimation aka 'Lasso'*. R package version 1.2-21.1.

Lou, Y., Bien, J., Caruana, R., and Gehrke, J. (2016). Sparse Partially Linear Additive Models. *Journal of Computational and Graphical Statistics*, 25(4):1126–1140.

Mackey, L. (2009). Deflation Methods for Sparse PCA. *Advances in Neural Information Processing Systems*, pages 1017–1024.

McCullagh, P. and Nelder, J. A. (1983). *Generalized Linear Models*. Springer US.

Meier, L., van De Geer, S., and Bühlmann, P. (2009). High-Dimensional Additive Modeling. *Annals of Statistics*, 37(6B):3779–3821.

Meinshausen, N. (2007). Relaxed Lasso. *Computational Statistics & Data Analysis*, 52(1):374–393.

Meinshausen, N. (2012). *relaxo: Relaxed Lasso*. R package version 0.1-2.

Mollaysa, A., Strasser, P., and Kalousis, A. (2017). Regularising Non-Linear Models Using Feature Side-Information. *Proceedings of the 34th International Conference on Machine Learning*, pages 2508–2517.

Nakao, K., Mehta, K. R., Fridlyand, J., Moore, D. H., Jain, A. N., Lafuente, A., Wiencke, J. W., Terdiman, J. P., and Waldman, F. M. (2004). High-Resolution Analysis of DNA Copy Number Alterations in Colorectal Cancer by Array-Based Comparative Genomic Hybridization. *Carcinogenesis*, 25(8):1345–1357.

Nelder, J. A. and Wedderburn, R. W. M. (1972). Generalized Linear Models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384.

Obozinski, G., Jacob, L., and Vert, J.-P. (2009). Group Lasso with Overlaps: the Latent Group Lasso Approach. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 433–440.

Obozinski, G., Taskar, B., and Jordan, M. I. (2010). Joint Covariate Selection and Joint Subspace Selection for Multiple Classification Problems. *Statistics and Computing*, 20(2):231–252.

Park, M. Y. and Hastie, T. (2018). *glmpath: L1 Regularization Path for Generalized Linear Models and Cox Proportional Hazards Model.* R package version 0.98.

Petersen, A. and Witten, D. (2019). Data-Adaptive Additive Modeling. *Statistics in Medicine*, 38(4):583–600.

Petersen, A., Witten, D., and Simon, N. (2016). Fused Lasso Additive Model. *Journal of Computational and Graphical Statistics*, 25(4):1005–1025.

Prados, J. (2019). *bmrm: Bundle Methods for Regularized Risk Minimization Package.* R package version 4.1.

Ravikumar, P., Liu, H., Lafferty, J., and Wasserman, L. (2007). Sparse Additive Models. In *Advances in Neural Information Processing Systems*, pages 1201–1208.

Sadhanala, V. and Tibshirani, R. J. (2017). Additive Models with Trend Filtering. *arXiv preprint arXiv:1702.05037.*

Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2011). Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent. *Journal of Statistical Software*, 39(5):1–13.

Simon, N., Friedman, J., Hastie, T., and Tibshirani, R. (2013). A Sparse-Group Lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245.

Simon, N. and Tibshirani, R. (2012). Standardization and the Group Lasso Penalty. *Statistica Sinica*, 22(3):983–1001.

Singh, D., Febbo, P. G., Ross, K., Jackson, D. G., Manola, J., Ladd, C., Tamayo, P., Renshaw, A. A., D'Amico, A. V., Richie, J. P., Lander, E. S., Loda, M., Kantoff, P. W., Golub, T. R., and Sellers, W. R. (2002). Gene Expression Correlates of Clinical Prostate Cancer Behavior. *Cancer Cell*, 1(2):203–209.

Slawski, M., zu Castell, W., and Tutz, G. (2010). Feature Selection Guided by Structural Information. *Annals of Applied Statistics*, 4(2):1056–1080.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959.

Subramanian, A., Tamayo, P., Mootha, V. K., Mukherjee, S., Ebert, B. L., Gillette, M. A., Paulovich, A., Pomeroy, S. L., Golub, T. R., Lander, E. S., and Mesirov, J. P. (2005). Gene Set Enrichment Analysis: A Knowledge-Based Approach for Interpreting Genome-Wide Expression Profiles. *Proceedings of the National Academy of Sciences of the United States of America*, 102(43):15545–15550.

Tai, F. and Pan, W. (2007). Incorporating Prior Knowledge of Predictors into Penalized Classifiers with Multiple Penalty Terms. *Bioinformatics*, 23(14):1775–1782.

Tay, J. K. and Tibshirani, R. (2020). Reluctant Generalised Additive Modelling. *International Statistical Review*, S1:S205–S224.

Therneau, T. M. (2020). *A Package for Survival Analysis in R*. R package version 3.2-7.

Therneau, T. M. and Grambsch, P. M. (2000). *Modeling Survival Data: Extending the Cox Model*. Springer.

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J., and Tibshirani, R. J. (2012). Strong Rules for Discarding Predictors in Lasso-Type Problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 74(2):245–266.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., and Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108.

van de Wiel, M. A., Lien, T. G., Verlaat, W., van Wieringen, W. N., and Wilting, S. M. (2016). Better Prediction by Use of Co-Data: Adaptive Group-Regularized Ridge Regression. *Statistics in Medicine*, 35(3):368–381.

van der Kooij, A. J. (2007). *Prediction Accuracy and Stability of Regression with Optimal Scaling Transformations*. Ph.d. thesis, Leiden University.

Velten, B. and Huber, W. (2018). Adaptive Penalization in High-Dimensional Regression and Classification with External Covariates using Variational Bayes. *arXiv preprint arXiv:1811.02962*.

Witten, D. M., Tibshirani, R., and Hastie, T. (2009). A Penalized Matrix Decomposition, with Applications to Sparse Principal Components and Canonical Correlation Analysis. *Biostatistics*, 10(3):515–534.

Yan, X. and Bien, J. (2017). Hierarchical Sparse Modeling: A Choice of Two Group Lasso Formulations. *Statistical Science*, 32(4):531–560.

Yata, K. and Aoshima, M. (2012). Effective PCA for High-Dimension, Low-Sample-Size Data with Noise Reduction via Geometric Representations. *Journal of Multivariate Analysis*, 105(1):193–215.

Yu, G., Bien, J., and Tibshirani, R. (2019). Reluctant Interaction Modeling. *arXiv preprint arXiv:1907.08414*, pages 1–32.

Yuan, M. and Lin, Y. (2006). Model Selection and Estimation in Regression with Grouped Variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

Zeisler, H., Llurba, E., Chantraine, F., Vatish, M., Staff, A. C., Sennström, M., Olovsson, M., Brennecke, S. P., Stepan, H., Allegranza, D., Dilba, P., Schoedl, M., Hund, M., and Verlohren, S. (2016). Predictive Value of the sFlt-1:PlGF Ratio in Women with Suspected Preeclampsia. *New England Journal of Medicine*, 374(1):13–22.

Zeng, Y. and Breheny, P. (2016). Overlapping Group Logistic Regression with Applications to Genetic Pathway Selection. *Cancer Informatics*, 15:CIN.S40043.

Zeng, Y. and Breheny, P. (2017). The biglasso package: A memory- and computation-efficient solver for lasso model fitting with big data in r. *ArXiv e-Prints*.

Zhang, C.-H. (2010). Nearly Unbiased Variable Selection under Minimax Concave Penalty. *The Annals of Statistics*, 38(2):894–942.

Zou, H. (2005). *Some Perspectives on Sparse Statistical Modeling*. PhD thesis, Department. of Statistics, Stanford.

Zou, H. (2006). The Adaptive Lasso and its Oracle Properties. *Journal of the American Statistical Association*, 101(476):1418–1429.

Zou, H. and Hastie, T. (2005). Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.

Zou, H. and Hastie, T. (2020). *elasticnet: Elastic-Net for Sparse Estimation and Sparse PCA*. R package version 1.3.

Zou, H., Hastie, T., and Tibshirani, R. (2006). Sparse Principal Component Analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286.

Zou, H., Hastie, T., and Tibshirani, R. (2007). On the "Degrees of Freedom" of the Lasso. *Annals of Statistics*, 35(5):2173–2192.