Software for Prediction and Estimation with Applications to High-Dimensional Genomic and Epidemiologic Data

by

Stephan Johannes Ritter

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Biostatistics

and the Designated Emphasis

in

Computational Science and Engineering

in the

Graduate Division

of the

University of California, Berkeley

Fall 2013

Abstract

Software for Prediction and Estimation with Applications to High-Dimensional Genomic and Epidemiologic Data

by

Stephan Johannes Ritter

Doctor of Philosophy in Biostatistics

Designated Emphasis in Computational Science and Engineering

University of California, Berkeley

Professor Alan E. Hubbard, Chair

Three add-on packages for the R statistical programming environment (R Core Team, 2013) are described, with simulations demonstrating performance gains and applications to real data. Chapter 1 describes the **relaxnet** package, which extends the **glmnet** package with relaxation (as in the relaxed lasso of Meinshausen, 2007). Chapter 2 describes the **widenet** package, which extends **relaxnet** with polynomial basis expansions. Chapter 3 describes the **multiPIM** package, which takes a causal inference approach to variable importance analysis. Section 3.7 describes an analysis of data from the PRospective Observational Multicenter Major Trauma Transfusion (PROMMTT) study (Rahbar *et al.*, 2012; Hubbard *et al.*, 2013), for which the **multiPIM** package is used in conjunction with the **relaxnet** and **widenet** packages to estimate variable importances.

Contents

Contents **i**

List of Figures **iv**

List of Tables **vi**

Acknowledgements **vii**

1 relaxnet: Extending the glmnet R Package with Relaxation **1**

1.1 Introduction . . . 1

1.1.1 Motivation . . . 1

1.1.2 Chapter overview . . . 1

1.2 Overview of penalized regression and some associated R packages . . . 2

1.2.1 The linear regression setting, OLS and ridge regression . . . 2

1.2.2 Subset selection . . . 3

1.2.3 The lasso . . . 3

1.2.4 The elastic net . . . 3

1.2.5 The relaxed lasso . . . 4

1.2.6 Extensions to generalized linear models and package **glmnet** . . . 6

1.2.7 Advantages of the **relaxnet** package . . . 6

1.3 The **relaxnet** R package . . . 7

1.3.1 Main function: `relaxnet` . . . 7

1.3.2 Functions for cross-validation: `cv.relaxnet` and `cv.alpha.relaxnet` 8

1.3.3 Options for parallel execution . . . 9

1.3.4 Predict, summary and print methods . . . 9

1.3.5 Example . . . 10

1.4 Simulations . . . 11

1.4.1 Simulation 1: Continuous outcome, uncorrelated predictors . . . 11

1.4.2 Simulation 2: Binary outcome, uncorrelated predictors . . . 16

1.4.3 Simulation 3: Continuous outcome, correlated predictors . . . 23

1.4.4 Using simulation 1 to compare running time of the `relaxnet` function with that of `relaxo` . . . 28

1.5 Cross-validation with leukemia data . . . 28
1.6 Discussion . . . 29

2 widenet: An R Package and Machine Learning Algorithm Combining Regularization with Polynomial Basis Expansions 32
2.1 Introduction . . . 32
2.1.1 Motivation . . . 32
2.1.2 Chapter overview . . . 33
2.2 Previous approaches . . . 33
2.2.1 The deletion/substitution/addition algorithm and the **DSA** R package 33
2.2.2 MARS and the **polspline** R package . . . 34
2.2.3 The **polywog** R package . . . 35
2.3 The **widenet** R package . . . 35
2.3.1 Main function: `widenet` . . . 35
2.3.2 Options for parallel execution . . . 36
2.3.3 Predict, summary and print methods . . . 36
2.3.4 Memory use . . . 37
2.3.5 Example . . . 37
2.4 Simulation . . . 39
2.5 Cross-validation with benzene data . . . 43
2.6 Discussion . . . 44

3 multiPIM: A Causal Inference Approach to Variable Importance Analysis 46
3.1 Introduction . . . 46
3.1.1 Motivation . . . 46
3.1.2 Chapter overview . . . 48
3.2 Statistical methodology . . . 48
3.2.1 Data structure and parameter of interest . . . 48
3.2.2 Estimators . . . 50
3.2.3 Properties of the estimators . . . 52
3.2.4 Super learner and recommendations for estimation of nuisance parameters . . . 52
3.2.5 Inference . . . 53
3.3 The **multiPIM** R Package . . . 54
3.3.1 Input and output . . . 54
3.3.2 Adjustment Set and the `adjust.for.other.As` argument . . . 55
3.3.3 Rebuilding of models used to estimate $Q(0, W)$ . . . 55
3.3.4 Implementation of super learner . . . 56
3.3.5 Default binary outcome candidates . . . 56
3.3.6 Default continuous outcome candidates . . . 57
3.3.7 Alternative regression methods and other user options . . . 58
3.3.8 `multiPIMboot` function . . . 59

3.3.9 Statistical recommendations and effects on computation time 60
3.3.10 Summary method and example 60
3.4 Simulation 63
3.4.1 Simulated Data 63
3.4.2 Estimators 63
3.4.3 Results 64
3.5 Analysis of data from the western collaborative group study 64
3.5.1 Study background 64
3.5.2 Description of data 65
3.5.3 Estimator used 67
3.5.4 Results 67
3.6 Study on water contact and schistosomiasis infection 70
3.6.1 Study background 70
3.6.2 Description of data 70
3.6.3 The analysis 71
3.7 PROMMTT trauma study 71
3.7.1 Study background 72
3.7.2 Description of data and the analysis 72
3.7.3 Results 74
3.8 Discussion 75

Bibliography **76**

A Help File for `relaxnet` Function **83**

B Help File for `widenet` Function **86**

C Help File for `multiPIM` Function **89**

List of Figures

1.1 Relaxed lasso example 5
1.2 Prediction performance for simulation 1, $p = 100$ 13
1.3 Prediction performance for simulation 1, $p = 1000$ 13
1.4 Median number of false positives for simulation 1, $p = 100$ 14
1.5 Median number of false positives for simulation 1, $p = 1000$ 14
1.6 Median number of true positives for simulation 1, $p = 100$ 15
1.7 Median number of true positives for simulation 1, $p = 1000$ 15
1.8 Probability histograms for simulation 2 17
1.9 Mean test set misclassification rate for simulation 2, $q = 10$ and $c = 2$ 19
1.10 Mean test set misclassification rate for simulation 2, $q = 10$ and $c = 8$ 19
1.11 Mean test set log likelihood for simulation 2, $q = 10$ and $c = 2$ 20
1.12 Mean test set log likelihood for simulation 2, $q = 10$ and $c = 8$ 20
1.13 Median number of false positives for simulation 2, $q = 10$ and $c = 2$ 21
1.14 Median number of false positives for simulation 2, $q = 10$ and $c = 8$ 21
1.15 Median number of true positives for simulation 2, $q = 10$ and $c = 2$ 22
1.16 Median number of true positives for simulation 2, $q = 10$ and $c = 8$ 22
1.17 Median number of false positives for simulation 2, $q = 100$ 23
1.18 Relative mean of test set mean squared error for simulation 3, $\rho = 0.25$ 25
1.19 Relative mean of test set mean squared error for simulation 3, $\rho = 0.95$ 26
1.20 Relative mean of test set mean squared error for simulation 3, $\rho = 0.99$ 26
1.21 Median of false positives for simulation 3, $\rho = 0.95$ 27
1.22 Median of true positives for simulation 3, $\rho = 0.95$ 27
1.23 Relaxnet timings 28

2.1 Running time of DSA is supracubic in maxsize 34
2.2 Prediction performance for widenet simulation 41
2.3 Relative prediction performance of widenet methods 41
2.4 Mean of true positives for widenet simulation 42
2.5 Mean of false positives for widenet simulation 42
2.6 ROC curves for benzene data cross-validation 45

3.1 Simulation results: TMLE estimator vs. G-computation estimator 64

3.2 Bootstrap cumulative standard deviations for WCGS exposure variables 68
3.3 Bootstrap distributions of parameter estimates for each WCGS exposure variable. 69

List of Tables

1.1 Results for leukemia data cross-validation 29
1.2 Set of genes found by relaxnet for leukemia data 30

2.1 Memory requirements for widenet 37
2.2 Timings and final model size for benzene data 44

3.1 Summary information for the five binary exposure variables used in the WCGS analysis. 66
3.2 Correlation matrix for all variables used in the WCGS analysis 66
3.3 WCGS multiPIMboot results table 67
3.4 Results for schisto analysis. The tool washing and rice planting variables remain significant after Bonferroni adjustment. 72
3.5 Descriptions of variables used for PROMMTT analysis 73
3.6 Candidates chosen by the super learner for PROMMTT analysis 73
3.7 Results for PROMMTT analysis 74

Acknowledgments

There are many people who have helped get me to the point of being able to complete this dissertation. I would like to start by thanking my primary advisor, Alan Hubbard, for his continuous support and encouragement throughout my graduate carreer. I would also like to thank Jack Colford, Sandrine Dudoit, Nick Jewel, Steve Selvin, Phil Spector and Mark van der Laan for serving on committees and for the teaching and guidance they have provided; and Sharon Norris and Burke Bundy for much appreciated support throughout this process.

I would like to thank the Group in Biostatistics, the Graduate Division and the National Institute of Environmental Health Sciences for their financial support.

I would also like to acknowledge the many people who have contributed to the open source software projects which have been vital to my work. For example, this dissertation was typeset using LaTeX(`http://www.latex-project.org`), Sweave (Leisch, 2002) and style files maintained by Paul Vojta in the Math department (`http://math.berkeley.edu/~vojta/ucbthesis`) and from the Journal of Statistical Software (`http://www.jstatsoft.org/style`).

To fellow students, friends and family: thank you for your company, encouragement, and insights. I would especially like to thank my parents, Maria and Hans Georg, my brother Thomas, and finally my wife Anya for their unconditional love and support.

Chapter 1

relaxnet: Extending the glmnet R Package with Relaxation

1.1 Introduction

1.1.1 Motivation

As stated by Zou and Hastie (2005), two important properties of a prediction algorithm are accuracy of its predictions and interpretability of the final model. A good way to keep a model easily interpretable is to restrict the predictions to linear combinations of the inputs. In a high dimensional context, such as genomic experiments where the inputs might be expression levels of up to thousands of genes, generating an interpretable model will need to involve parsimony, for example by restricting the predictions to be based on only a certain subset of the inputs. Methods of this type are the subject of this chapter, and the goal is to provide software which addresses the concerns of prediction accuracy and model interpretability, while at the same time providing computational efficiency and reasonable running times.

1.1.2 Chapter overview

Section 1.2 gives a brief review of penalized regression and associated R packages. Section 1.3 describes the components of the **relaxnet** package in detail. In Section 1.4, simulations using 3 different data generating models are described. Section 1.5 reports the results of a cross-validation analysis using a genomic data set with a binary leukemia sub-class outcome (Golub *et al.*, 1999). The chapter closes with a discussion in Section 1.6.

1.2 Overview of penalized regression and some associated R packages

1.2.1 The linear regression setting, OLS and ridge regression

In the standard multiple linear regression setting we have p predictors, $x_1, \ldots, x_p$, each of length n, and a response y (also length n) which is estimated as a linear combination of the x's, plus an intercept:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \cdots + \hat{\beta}_p x_p$$

If we let X be the n by $p+1$ matrix containing first a column of 1's followed by the x_j's, for $j = 1, \ldots, p$, and let $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)$, then the residual sum of squares is given by

$$\text{RSS}(\boldsymbol{\beta}) = (y - X\boldsymbol{\beta})^T (y - X\boldsymbol{\beta}).$$

The Ordinary Least Squares (OLS) estimate of $\boldsymbol{\beta}$ is the minimizer of the RSS:

$$\hat{\boldsymbol{\beta}}^{OLS} = (X^T X)^{-1} X^T y.$$

This estimator enjoys many nice statistical properties (see for example Freedman, 2005, ch. 3), however, in the case of predictors which are strongly correlated, the estimates may become unstable and the prediction accuracy may suffer. Hoerl and Kennard (1970) introduced L_2 penalized (ridge) regression as a way to overcome this problem. $\boldsymbol{\beta}$ is instead estimated as:

$$\hat{\boldsymbol{\beta}}^{ridge} = (X^T X + \lambda I)^{-1} X^T y$$

where $\lambda \geq 0$ is a tuning parameter and I is the p by p identity. This estimator sacrifices a little bias in exchange for lower variance, which can lead to improvements in prediction accuracy over the OLS estimator, especially for the case of highly correlated predictors. The general effect is to "shrink" the β_j's towards zero relative to the OLS estimates, with the degree of shrinkage depending on the value of λ. The ridge solution may also be expressed in the Lagrangian form as:

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\boldsymbol{\beta}} \, \text{RSS}(\boldsymbol{\beta}) + \lambda P(\boldsymbol{\beta}), \tag{1.1}$$

$$P^{ridge}(\boldsymbol{\beta}) = \sum_{j=1}^{p} \beta_j^2.$$

For ridge regression, $P(\boldsymbol{\beta})$ is the L_2 penalty. Note that the intercept has been left out of the penalty (it is "unpenalized").

Returning to the issues of parsimony and high-dimensional data, both OLS and ridge regression have disadvantages here. OLS can not even be applied successfully to data with

$p > n$ as the solution is not unique. Also, for the case of $p < n$, OLS will assign a nonzero value to each β_j, leading to a lack of parsimony and poor interpretability. While ridge regression will generate a solution when $p > n$, it also lacks a parsimony property and will also assign a nonzero value to each β_j, including in the $p > n$ case.

1.2.2 Subset selection

As a way of addressing the parsimony issue, many different "subset selection" methods were proposed. These select, by various criteria and algorithms, only a certain subset of the predictors to include in the final model (see Miller, 2002). However, due to the discrete nature of the selection, these methods suffer from instability and high variance, and many are computationally infeasible when p is even moderately high.

1.2.3 The lasso

A breakthrough was achieved when Tibshirani (1996) introduced the lasso, or least absolute shrinkage and selection operator. As the name suggests, this method both shrinks the coefficients, and also selects predictors by shrinking some coefficients all the way to zero, providing a nice balance between ridge regression and subset selection. The Lagrangian formulation of the lasso is the same as equation 1.1, only now the penalty is an L_1 norm:

$$P^{lasso}(\boldsymbol{\beta}) = \sum_{j=1}^{p} |\beta_j|.$$

As a convex problem, solving the lasso was tractable even for high-dimensional data, but had to be done separately for each value of the regularization parameter. This changed when Efron *et al.* (2004) realized that the lasso was a variation on a more general procedure they termed Least Angle Regression, or LARS, and that the entire lasso regularization path could be computed with the same computational complexity as an OLS fit. An R package, **lars** (Hastie and Efron, 2012), implementing the method was released on the Comprehensive R Archive Network (CRAN).

Lasso also has some disadvantages, as described in Zou and Hastie (2005). Unlike ridge regression, lasso selects at most n predictors when $p > n$. It saturates. Also, ridge regression often has better performance when correlations between predictors are high. In addition, lasso tends to arbitrarily select only one out of a set of highly correlated predictors, and leave the rest out of the model entirely.

1.2.4 The elastic net

Zou and Hastie (2005) proposed a second compromise to address these issues, this time between the lasso and ridge regression. Their "elastic net" mixes the L_1 and L_2 penalties:

$$P^{elnet}(\boldsymbol{\beta}) = (1-\alpha)\frac{1}{2}\sum_{j=1}^{p}\beta_j^2 + \alpha\sum_{j=1}^{p}|\beta_j|. \tag{1.2}$$

With $0 < \alpha < 1$.[1] $\alpha = 0$ corresponds to the ridge penalty and $\alpha = 1$ to the lasso. Another compromise between lasso and ridge regression would be to use the penalty

$$P(\boldsymbol{\beta}) = \sum_{j=1}^{p}|\beta_j|^q, \tag{1.3}$$

with $1 < q < 2$. This corresponds to a subclass of the bridge estimators (Frank and Friedman, 1993; Fu, 1998). This class of penalties is also convex, however, as pointed out by Zou and Hastie (2005), using this type of penalty does not result in any variable selection, as all predictors are kept in the model. The elastic net penalty, like the lasso, does result in variable selection as well as shrinkage. Unlike the lasso, it does not saturate and may select up to p predictors in the $p > n$ case, and it also exhibits the "grouping effect," meaning that the coefficients for highly correlated predictors will have similar profiles, i.e., they will shrink along similar paths. The lasso tends to choose one predictor from the group and set the rest to zero (Zou and Hastie, 2005).

Using an elastic net penalty instead of the lasso may lead to improvements in prediction accuracy as well as in selection of predictors, especially when the predictors are highly correlated. However, one of the advantages of the lasso for high-dimensional problems, namely the sparsity of the final model, is diminished by using the elastic net. For a given value of λ, the sparsity of the elastic net solution decreases as α decreases from 1 to 0, so the elastic net solutions tend to be less sparse (fewer coefficients equal to zero) than the corresponding lasso solution.

Many researchers have explored the question of the accuracy of model selection by the lasso, i.e., whether the predictors which truly contribute to the outcome are in fact selected by the procedure. This work is reviewed by Peter Bühlmann in his comments on Tibshirani (2011). One of his conclusions is that "two-stage procedures such as the adaptive lasso (Zou, 2006) or the relaxed lasso (Meinshausen, 2007) are very useful [for removing false positives]." The adaptive lasso is a modification of the lasso in which each coefficient may be penalized differently using adaptively chosen weights. This has also been extended to the elastic net penalized models (Zou and Zhang, 2009). Here I will focus on the relaxed lasso of Meinshausen (2007).

1.2.5 The relaxed lasso

Meinshausen (2007) proposed the relaxed lasso as a way to overcome the slow convergence rate of the lasso for sparse, high-dimensional data. The relaxed lasso also leads to sparser

[1]In the help files for the **glmnet** R package (Friedman *et al.*, 2010), α is called the "elastic net mixing parameter." Note that the formulation of the elastic net penalty in equation 1.2 is as given in Friedman *et al.* (2010), and is slightly different from that in the original paper.

models than standard lasso. The set of relaxed lasso solutions, indexed by the parameters λ and ϕ, is a super set of the set of lasso solutions:

$$\hat{\boldsymbol{\beta}}^{relaxed} = \text{argmin}_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \{\boldsymbol{\beta} \cdot \mathbf{1}_{\mathscr{M}_\lambda}\})^2 + \phi\lambda \sum_{j=1}^{p} |\beta_j|. \tag{1.4}$$

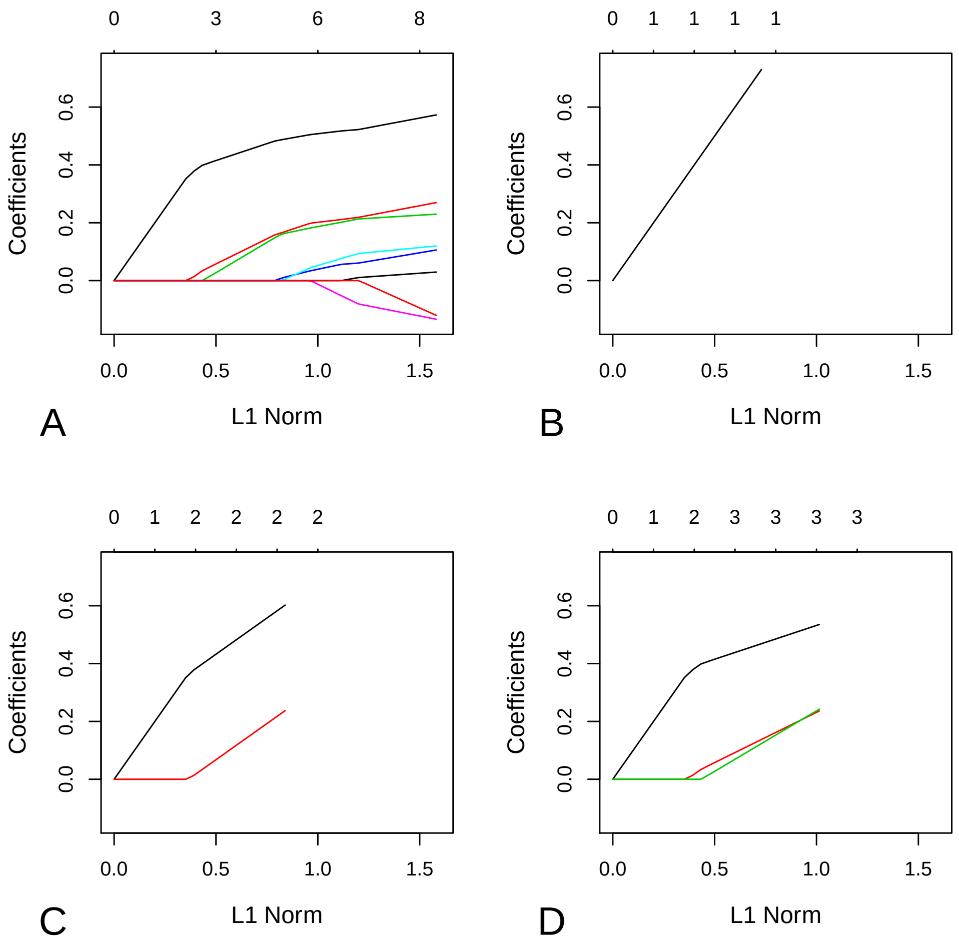

Figure 1.1: Relaxed lasso example. The data are from Stamey *et al.* (1989), made available through the **ElemStatLearn** R package (Hastie *et al.*, 2009; Halvorsen, 2012). A: Full lasso coefficient profile. B-D: Coefficient profiles for the first three "relaxed models." Notice that the coefficients for the variables present in the relaxed models reach higher values than they do for the full lasso model.

Here, y_i is the ith element of y and x_i is the ith row of X, for $i = 1, \ldots, n$; $\mathscr{M}_\lambda$ is the set of indices corresponding to predictors selected by the lasso at a certain λ value and $\mathbf{1}_{\mathscr{M}_\lambda}$ is the indicator function on that set, so that

$$\{\boldsymbol{\beta} \cdot \mathbf{1}_{\mathscr{M}_\lambda}\}_k = \begin{cases} 0, & k \notin \mathscr{M}_\lambda \\ \beta_k, & k \in \mathscr{M}_\lambda \end{cases}$$

(Meinshausen, 2007). One way to find the relaxed lasso solutions is to first find the lasso solutions for all λ, then run lasso again on each distinct subset of predictors from along the solution path. This allows the values of the coefficients for these subsets to "relax" back up from their lasso values, to the values of the OLS solution for just that subset. An example is given in figure 1.1. An R package implementing the relaxed lasso, **relaxo** (Meinshausen, 2012), is available on CRAN. This package depends on the **lars** package.

1.2.6 Extensions to generalized linear models and package glmnet

So far I have discussed penalization methods for linear models, but there has also been work to extend these methods to generalized linear models (Nelder and Wedderburn, 1972; McCullagh and Nelder, 1989). For example, Park and Hastie (2007) published a LARS-like solution path algorithm for L_1 penalized generalized linear models. They also released an associated R package, **glmpath** (Park and Hastie, 2013).

A major computational breakthrough was achieved with the release of the **glmnet** R package by Friedman *et al.* (2010). Inspired by recent work, reviewed in their paper, on applying cyclical coordinate descent methods to the lasso, they developed efficient algorithms for solving penalized regression problems, for linear as well as generalized linear models, and for lasso, ridge and elastic net penalties, along a grid of values of the regularization parameter λ. Solutions for previous λ values are used as warm starts for the following values, and in some cases computing an entire path can be faster than finding the solution at a single point (Friedman *et al.*, 2010). Timings in the paper show the method to be faster than competing R packages, especially for large data sets, for both linear and generalized linear models. The underlying Fortran code has two variants to handle either a dense or a sparse matrix of predictors. The function `cv.glmnet` is provided to perform v-fold cross-validation to select λ by one of two different rules: the min rule for selecting λ chooses that value with the minimum cross-validated error, while the 1se rule chooses the highest value of λ whose error is within one standard error of the minimum error. This leads to more parsimonious models since the amount of shrinkage is greater for higher λ values. However, in section 1.4 it will be shown that the prediction accuracy may be much worse when the 1se rule is applied vs. the min rule.

1.2.7 Advantages of the relaxnet package

The **relaxnet** package takes advantage of the existing **glmnet** package (Friedman *et al.*, 2010), in order to provide relaxed lasso functionality for generalized linear models. Applying relaxation to glmnet models offers several advantages over glmnet alone:

- Prediction error may be reduced when the data generating distribution is truly sparse, while the error will stay similar to that for the non-relaxed model otherwise

- The resulting model will usually be sparser, with a smaller false positive rate. This is true for a wide range of underlying true levels of sparsity of the data generating model (see Section 1.4)

- Applying relaxation to elastic net penalized models can be especially effective, since the prediction accuracy improvements of elastic net over pure lasso may be retained without sacrificing the sparsity of the final model (i.e. one keeps the false positive rate low). As shown in Section 1.4.3, for the case of a sparse data-generating model with small correlated groups of truly contributing predictors, combining relaxation with the elastic net penalty can result in significant additional improvements in prediction error on top of the improvement due to using just relaxed lasso

The fact that **relaxnet** is based on **glmnet** offers several advantages over the **relaxo** package, which is based on the **lars** package:

- As discussed above, **glmnet** is based on a very efficient algorithm coded in Fortran, while **lars** is based on an earlier algorithm and is coded entirely in R. **relaxnet** therefore has a considerable speed advantage over **relaxo**, especially for when the number of predictors is large (see Section 1.4.4)

- **relaxnet** works for generalized linear models as well as linear models. The idea of applying relaxation to L_1 penalized generalized linear models was discussed in Meinshausen (2007, section 2.2). So far only linear (`family = "gaussian"`) and and binary outcome logistic (`family = "binomial"`) models have been implemented for **relaxnet**

- **relaxnet** allows application of elastic net penalties in addition to the lasso penalty

1.3 The relaxnet R package

1.3.1 Main function: `relaxnet`

The arguments to the main function, `relaxnet`, are as follows:

```
relaxnet(x, y, family = c("gaussian", "binomial"),
         nlambda = 100,
         alpha = 1,
         relax = TRUE,
         relax.nlambda = 100,
         relax.max.vars = min(nrow(x), ncol(x)) * 0.8,
         lambda = NULL,
         relax.lambda.index = NULL,
         relax.lambda.list = NULL,
         ...)
```

The arguments `x`, `y`, `family`, `nlambda`, `alpha`, and `lambda` are as for the `glmnet` function, the main function from the **glmnet** package. `x` and `y` are the matrix of predictors and the outcome vector, `family` specifies whether a linear model or a logistic model will be fit, `nlambda` specifies the length of the sequence of λ values to use, i.e., the fineness of the grid of λ values. `alpha` is the elastic net mixing parameter, α (the default of 1 means that the lasso penalty is used by default), and `lambda` is used to override the default mechanism for choosing λ values.

`relaxnet` calls `glmnet` once, passing these arguments, in order to fit the main `glmnet` model. Then the matrix of coefficients, `beta`, of the resulting object is examined in order to determine all of the distinct subsets of the predictors on which to fit relaxed models. Then `glmnet` is run again on each of these subsets, but this time passing in `relaxnet`'s `relax.nlambda` argument instead of `nlambda`.

The `relax` argument can be set to `FALSE`, which turns off relaxation and causes `relaxnet` to behave similarly to `glmnet`. The `relax.lambda.index` and `relax.lambda.list` arguments, (as well as the `lambda` argument) are meant primarily for use by the cross-validation function, `cv.relaxnet`, in order to ensure that all the tuning parameters match up between the different cross-validation folds.

Due to the fact that the lasso solution is found along a discrete grid of λ values, `relaxnet` may not necessarily compute the entire set of relaxed lasso solutions. Relaxation will be applied to each subset from along the `glmnet` solution path, but this may not necessarily include all of the subsets from the complete lasso solution. In practice, this does not significantly effect the performance of `relaxnet` when comparing it with the `relaxo` function from the **relaxo** package (see Section 1.4.1). The `relax.max.vars` argument can be used to set a maximum number of variables above which relaxed models will not be fit. This can be useful to save computation time.

1.3.2 Functions for cross-validation: `cv.relaxnet` and `cv.alpha.relaxnet`

The `cv.relaxnet` function, which is based on **glmnet**'s `cv.glmnet` function, allows the user to perform v-fold cross-validation to select tuning parameters for a relaxnet model. It takes all the arguments that `relaxnet` takes, with the addition of the `nfolds` and `foldid` arguments. `nfolds` specifies the value of v (number of "folds") and `foldid` can optionally be used to specify exactly how the observations should be divided into folds, (as for the `cv.glmnet` function).

While the `cv.relaxnet` function optimizes over the values of lambda for both the main and the relaxed models, the `cv.alpha.relaxnet` function can be used to in addition optimize over a set of values for α. This function simply calls `cv.relaxnet` once for each value in it's `alpha` argument, which should be a vector of α values.

v-fold Cross-validation has been proven to be an effective method for selecting among candidate learners (Van Der Laan and Dudoit, 2003), and in order to encourage users to make aggressive use of cross-validation for selecting tuning parameters, we provide options for parallelizing the execution.

1.3.3 Options for parallel execution

Both the `cv.relaxnet` and the `cv.alpha.relaxnet` functions have a `multicore` argument which allows for the execution to be parallelized using the multicore functionality from R's **parallel** package (R Core Team, 2013). For `cv.relaxnet`, the parallelization is over cross-validation folds, while for `cv.alpha.relaxnet`, it is over the values of `alpha`. Internally, in order to allow reproducibility of results, the random number generator (RNG) type is set to "L'Ecuyer-CMRG" and the `mclapply` function is called with `mc.preschedule = TRUE`. The `mc.seed` argument is used to set the RNG seed prior to execution. For optimum load balancing, it is helpful for the relevant parameter (either the `nfolds` argument, for `cv.relaxnet`, or the length of the `alpha` argument, for `cv.alpha.relaxnet`) to be a multiple of the numer of cores to be used (as given by the `mc.cores` argument).

1.3.4 Predict, summary and print methods

Rounding out the **relaxnet** package are predict, summary and print methods. There is a separate predict methods for each of the three types of objects resulting from calls to the three functions mentioned in the previous two sections. As for `predict.glmnet`, what is returned depends on the `type` argument:

- `type = "link"` - returns the linear predictors for `family = "binomial"` models and the fitted values for `family = "gaussian"` models
- `type = "response"` - returns the predicted probabilities for `family = "binomial"` models and the fitted values for `family = "gaussian"` models
- `type = "coefficients"` - returns the values of the coefficients, including the intercept
- `type = "nonzero"` - returns a vector containing the names of the predictors selected by the model (i.e., those that had nonzero coefficients)
- `type = "class"` (`family = "binomial"` only) - returns the predicted class

The predict methods also take the argument `which.model`, with which the user can specify whether the predictions should come from the main `glmnet` model, or from one of the relaxed models. For `predict.cv.glmnet` and `predict.cv.alpha.glmnet`, the default is to use that model which "won" the cross-validation.

Finally, there are print and summary methods for class `"relaxnet"` objects which print a brief summary showing the number of variables in each relaxed model as well as the time taken to fit the model.

1.3.5 Example

In this section we run a small example in order to demonstrate the use and output of the `cv.relaxnet` function. First we load the **relaxnet** package and generate a 100 by 200 predictor matrix of standard normal variates:

```
R> library("relaxnet")
R> n <- 100
R> p <- 200
R> set.seed(23)
R> x <- matrix(rnorm(n * p), n, p)
```

The predictor matrix must have unique column names:

```
R> colnames(x) <- paste("x", 1:ncol(x), sep = "")
```

Next we generate an outcome which depends only on the first five predictors, plus a standard normal error term:

```
R> y <- rowSums(x[, 1:5]) + rnorm(nrow(x))
```

Now we run `cv.relaxnet`:

```
R> cv.result <- cv.relaxnet(x, y)
```

We can see which columns of `x` were chosen for the final model by calling `predict.cv.relaxnet` with `type = "nonzero"`:

```
R> predict(cv.result, type = "nonzero")[[1]]
```

```
[1]  1  2  3  4  5 73
```

We can also look at the same result for just the main `glmnet` model, without relaxation. First for the min rule...

```
R> drop(predict(cv.result$relaxnet.fit$main.glmnet.fit,
+            type = "nonzero",
+            s = cv.result$main.lambda.min))[[1]]
```

```
 [1]   1   2   3   4   5   9  10  15  19  33  51  54  59  61  66  67
[17]  73  96 116 126 136 145 147 153 161 163 167 169 179 192 195
```

...and the 1se rule:

```
R> predict(cv.result$relaxnet.fit$main.glmnet.fit,
+          type = "nonzero",
+          s = cv.result$main.lambda.1se)[[1]]

 [1]   1   2   3   4   5  15  73 116 126 136 167 179
```

We see that the `cv.relaxnet` model results in fewer false positives than `glmnet` alone. To see the actual values of the nonzero coefficients, we call predict with `type = "coefficients"`:

```
R> coefs <- drop(predict(cv.result, type = "coef"))
R> coefs[coefs != 0]

(Intercept)          x1          x2          x3          x4 
  0.1573491   0.8805052   0.8535547   1.1733494   1.1282360 
         x5         x73 
  0.8494066  -0.2547877 
```

1.4 Simulations

For each of the three types of data-generating models in the simulations described in this section, different parameters were varied in order to explore many different aspects of how **relaxnet** models behave and to compare them with other methods.

1.4.1 Simulation 1: Continuous outcome, uncorrelated predictors

The first simulated model used a continuous outcome and uncorrelated predictors. The purpose of this simulation was to demonstrate that **relaxnet** behaves similarly to **relaxo** when $\alpha = 1$, and to replicate the results of Meinshausen (2007) showing the improvements that can be achieved by applying relaxation when the model is truly sparse and high-dimensional. The n by p predictor matrix X consisted of independent standard normal variates, with sample size, n, held constant at 250 while the number of predictors, p, was varied, with 4 different values: 100, 200, 500 and 1000. The number of predictors actually contributing to the outcome, q, was held constant at 10, with the coefficients being randomly generated, only once, uniformly from the interval $(-1, 1)$. This resulted in the following coefficient values (rounded to two decimal spaces): 0.15, -0.55 -0.34, 0.42, 0.64, -0.15, 0.93, 0.96, 0.68 and 0.99. These are the first ten elements of the vector β, with all following elements equal to zero. The elements of the outcome vector were generated as

$$y_i \sim x_i^T \beta + \epsilon_i$$

with x_i being the vector made up of the elements of the ith row of X, and

$$\epsilon_i \sim N(0, \sigma^2),$$

for $i = 1, \dots, n$. The standard deviation of the error term, σ, was varied, taking on the following values: 0.25, 0.5, 1, 2, 4. Each condition was repeated 100 times with different simulated data sets. All evaluation of the prediction performance of the different methods was done using a single separately generated test set with $n = 10{,}000$.

The methods which were compared in this simulation were as follows:

- glmnetMin: this method implements a standard lasso estimator using 10-fold cross-validation with `cv.glmnet`'s min rule to select λ
- glmnet1se: like the previous method, but using the 1se rule to select λ (see Section 1.2.6 for an explanation of min rule vs. 1se rule)
- relaxnet: the relaxed lasso implemented using the `cv.relaxnet` function from the **relaxnet** package
- relaxo: relaxed lasso implemented using the `cvrelaxo` function from the **relaxo** package
- lmTrue: This is the "cheating" method, for comparison. Only the truly contributing predictors (the first ten) are entered into a standard linear model using R's `lm` function. The noise predictors are ignored

Note that the first four methods all use the lasso penalty ($\alpha = 1$) and 10-fold cross-validation to select tuning parameters. The benefits of combining relaxation with an elastic net penalty ($0 < \alpha < 1$) are explored in simulation 3 (Section 1.4.3).

Figures 1.2 and 1.3 (on page 13) show the prediction performance measures for all methods. The measure used is the mean over the 100 repetitions of the test set mean squared error, relative to the mean for the glmnetMin method. Figure 1.2 shows the results for $p = 100$, while Figure 1.3 shows the results for $p = 1000$. The results for $p = 200$ and $p = 500$ were similar and are not shown. As expected, the performance of the relaxnet and relaxo methods is quite similar, and they both outperform the glmnetMin method, especially at the higher levels of the signal to noise ratio (lower values of σ). At the lowest level of signal to noise ratio (highest σ value), these methods have similar performance to the glmnetMin method. This is in line with what was found by Meinshausen (2007). At the lowest values of σ, the performance for these methods approaches the performance of the reference method, lmTrue. The glmnet1se method is the clear loser with respect to the other methods included, consistently having in the range of 20-60% worse performance than the glmnetMin method. The main differences between $p = 100$ and $p = 1000$ are that the difference in relative performance between glmnetMin and the three better performing methods increases (as

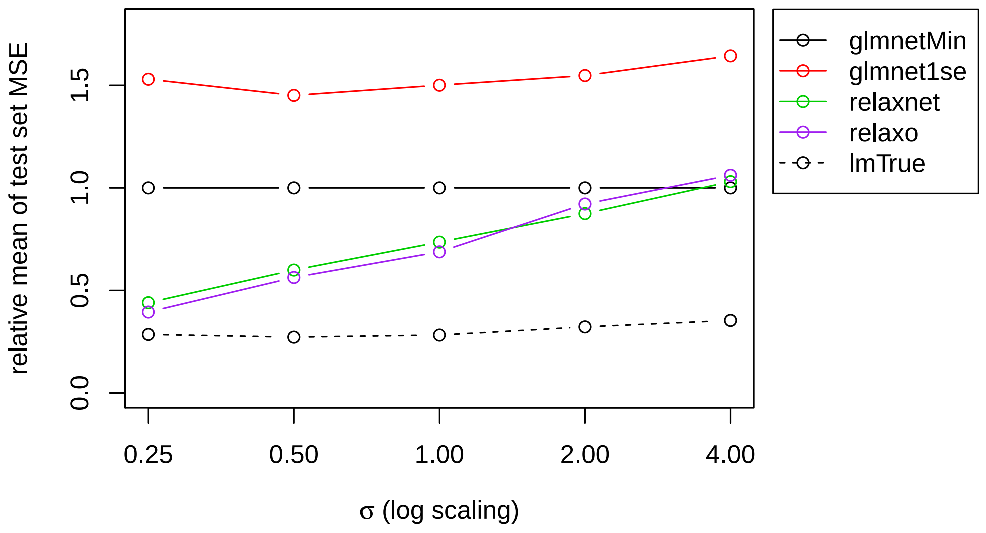

Figure 1.2: Prediction performance for simulation 1, $p = 100$. All values are plotted relative to the mean for the standard cross-validated lasso estimator (glmnetMin method).

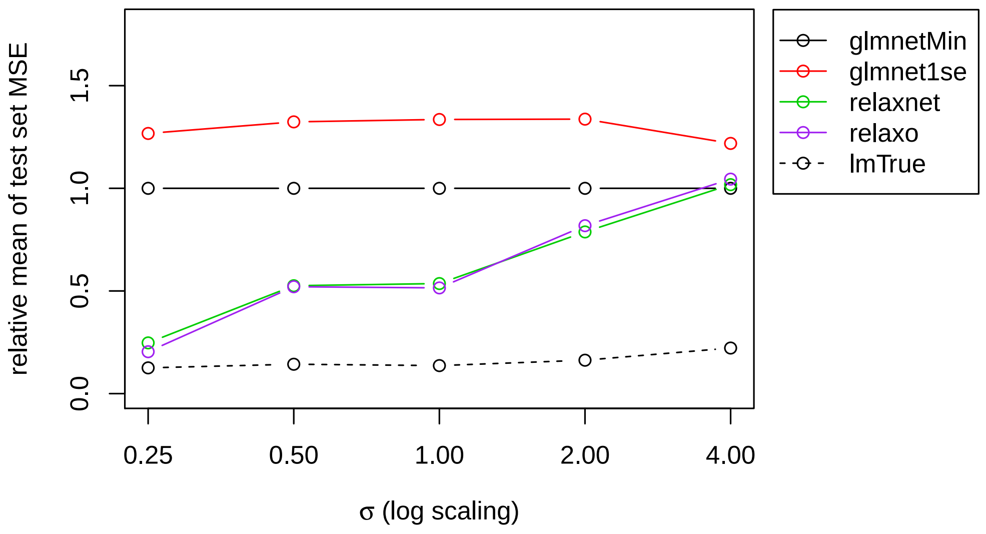

Figure 1.3: Prediction performance for simulation 1, $p = 1000$. All values are plotted relative to the mean for the standard cross-validated lasso estimator (glmnetMin method).

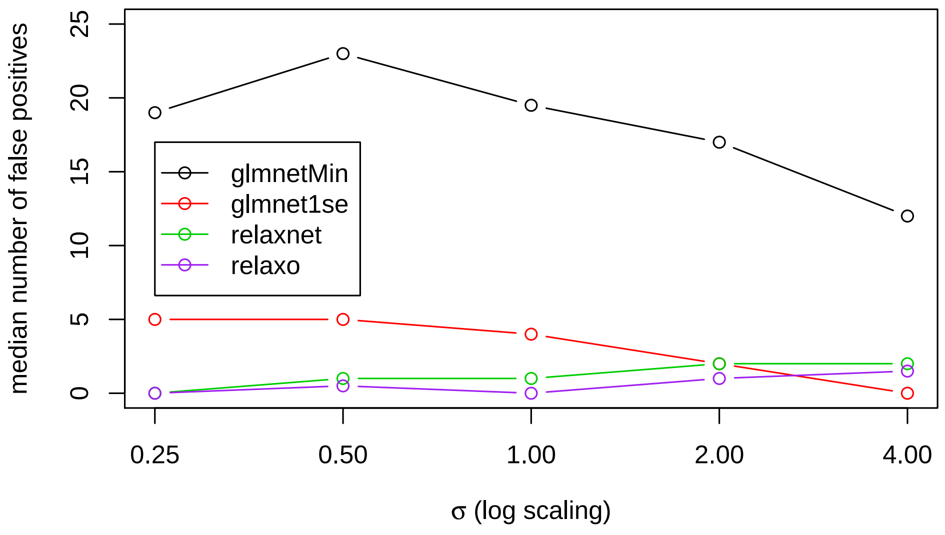

Figure 1.4: Median number of false positives for simulation 1, $p = 100$.

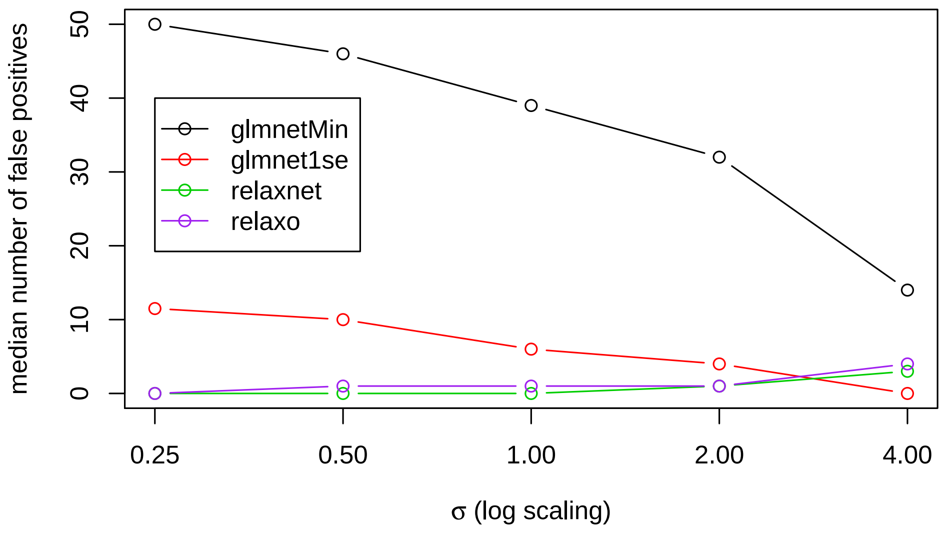

Figure 1.5: Median number of false positives for simulation 1, $p = 1000$.

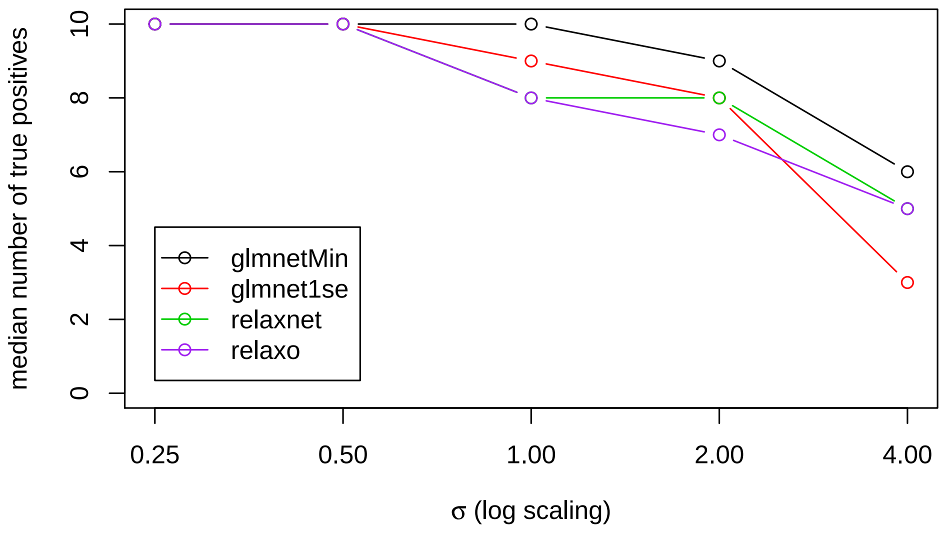

Figure 1.6: Median number of true positives for simulation 1, $p = 100$. There were 10 truly contributing predictors. The values for relaxnet are partially occluded by those for relaxo.

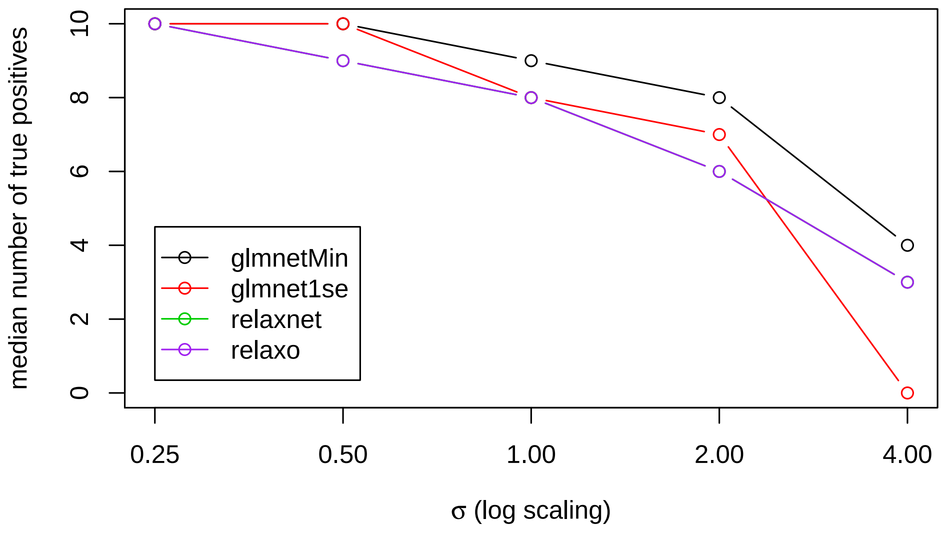

Figure 1.7: Median number of true positives for simulation 1, $p = 1000$. There were 10 truly contributing predictors. The values for relaxnet are fully occluded by those for relaxo.

expected due to the increase in the dimensionality) while the difference between glmnetMin and glmnet1se decreases.

Figures 1.4 and 1.5 (on page 14) show the median number of false positives over the 100 repetitions for the first four methods. False positives are predictors which do not truly contribute to the outcome, but which are assigned a nonzero coefficient in the final model for a given method. Figure 1.4 shows the results for $p = 100$, while Figure 1.5 shows the results for $p = 1000$. Again, as expected, the behavior of the relaxnet and relaxo methods is quite similar. They both result in much fewer false positives than the glmnetMin method. The glmnet1se method definitely improves on the glmnetMin method for this category. It is comforting that the trend for relaxo and relaxnet is a decreasing number of false positives as σ decreases, while for the glmnetMin and glmnet1se methods, the trend seems to be the opposite.

Figures 1.6 and 1.7 (on page 15) show the median number of true positives for the first four methods. True positives are predictors which *do* truly contribute to the outcome, and are included in the final model with a nonzero coefficient. Figure 1.6 shows the results for $p = 100$, while Figure 1.7 shows the results for $p = 1000$.
There is apparently a small price to be paid in terms of finding the truly contributing predictors when applying relaxation. In simulation 3 it will be shown that this effect can be somewhat counteracted by using the elastic net penalty instead of the lasso penalty. Also, now we see that the apparent improvement in number of false positives for the glmnet1se method over the relaxed lasso methods for the highest σ value is balanced by a corresponding decline in the number of true positives vs. the relaxed lasso methods at this σ value. For such low levels of signal to noise ratio, the glmnet1se method simply isn't able to find many of the correct predictors. For $p = 1000$, the median number of true positives for the glmnet1se method at $\sigma = 4$ was 0, implying that in more than half of the repetitions (actual number: 63 out of 100 repetitions), not a single truly contributing predictor was included in the final model.

1.4.2 Simulation 2: Binary outcome, uncorrelated predictors

The second simulation used a binary outcome with uncorrelated predictors. The purpose of this simulation was to compare the performance of **relaxnet** with other methods in the logistic regression setting, and to explore the effect of varying the level of sparsity of the true model. As for simulation 1, the n by p predictor matrix, X, was generated from independent standard normal variates. For this simulation, the sample size n was varied and took on the following values: 125, 250, 500, 1000, 2000, 4000. The number of predictors, p, was held constant at 1000. The number of predictors actually contributing to the outcome, q, was set at either 10, or 100, in order to compare an extremely sparse model to an only moderately sparse one. The coefficients, β, were again randomly generated, drawn uniformly from the interval $(-1, 1)$.

The elements of the outcome vector were generated as

$$y_i \sim \text{Bernoulli}(\text{expit}(\frac{c}{\sqrt{q}} x_i^T \beta)),$$

where expit refers to the inverse logit function:

$$\text{expit}(\alpha) = \text{logit}^{-1}(\alpha) = \frac{1}{1 + e^{-\alpha}}.$$

The parameter c serves as a substitute for the concept of a signal to noise ratio in the logistic regression context. c was varied, taking on the following values: 2, 4, 6, 8. The effect of including the factor $\frac{c}{\sqrt{q}}$ is to standardize the distribution of the probabilities from which the bernoulli variates (y) are generated across the two different values of q. This is shown in Figure 1.8. The number of repetitions was decreased as the sample size increased, starting with 400 repetitions for each condition for $n = 125$, and going down to 50 repetitions for each condition for the three highest sample sizes. Again, prediction performance was evaluated based on a separate test set with $n = 10,000$.

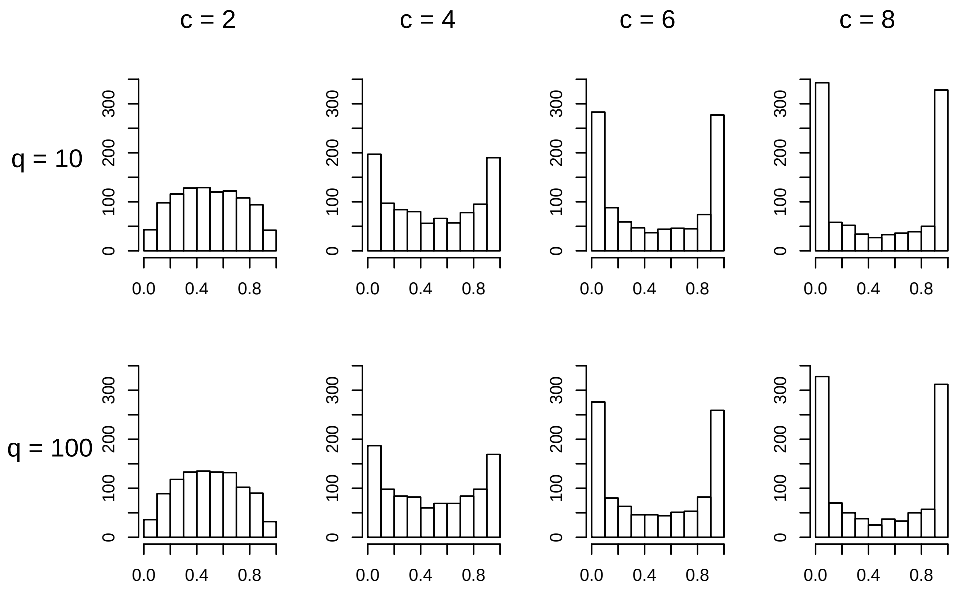

Figure 1.8: Histograms showing approximate distribution of y-generating probabilities for simulation 2, for the different values of the parameters c and q. The data set from which these plots were generated had $n = 1000$. The horizontal axes show probabilities, while the vertical axes show frequencies.

The methods applied for this simulation were:

- glmnetMin: L_1 (lasso) regularized logistic regression with 10-fold cross-validation. Uses the min rule to select λ

- glmnet1se: like the previous method, but uses the 1se rule to select λ
- relaxnet: L_1 (lasso) regularized logistic regression with relaxation and 10-fold cross-validation, implemented using the `cv.relaxnet` function from the **relaxnet** package
- randomForest: binary outcome random forests implemented using the `randomForest` function from package **randomForest** (Liaw and Wiener, 2002)
- glmTrue: Similar to lmTrue method from simulation 1, only this method uses logistic regression implemented using R's `glm` function. Only the truly contributing predictors are entered into the model and the noise predictors are ignored

Figures 1.9 and 1.10 show the mean test set misclassification rate for $q = 10$, with $c = 2$ and $c = 8$ respectively. Figures 1.11 and 1.12 show the mean test set log likelihood for the same parameter values. The three penalized methods all perform much better than random forest for this data generating distribution, and their performance approaches that of the glmTrue reference method for the higher sample sizes. In terms of misclassification performance, the glmnet1se method appears to be a slight disadvantage vs. the glmnetMin and relaxnet methods for the lower sample sizes when $c = 2$ (Figure 1.11). On the other hand, relaxnet and glmnet1se appear to have a slight advantage over glmnetMin in terms of misclassification for $c = 8$ (Figure 1.12), perhaps connected with having fewer false positives. The log likelihood results show a greater differentiation among the three penalized methods, with relaxnet the clear winner and glmnet1se the clear loser. Similarly to simulation 1, relaxnet shows greater improvement over the other methods for higher "signal" (higher values of c). The comparatively poor performance of randomForest may be due to the fact that the other methods are geared towards exactly this type of truly logistic data-generating distribution, as well as the fact that the performance of randomForest can be poor for very sparse data-generating models (Hastie *et al.*, 2009, Section 15.3.4).

Figures 1.13 and 1.14 show the median number of false positives for $q = 10$, with $c = 2$ and $c = 8$ respectively. In simulation 1, false positives increased for the glmnetMin method with decreasing standard deviation of the error term. Now we see that the false positives are increasing with increasing sample size, and with the signal parameter, c. False positives are also higher for the glmnet1se method with $c = 8$ than with $c = 2$. The relaxnet method consistently keeps false positives at a low level.

Figures 1.15 and 1.16 show the median number of true positives for the same parameter values. Again we see that there is a small price to be paid for keeping the false positives low in terms of the ability of the glmnet1se and the relaxnet methods to find the true positives.

In summary for the $q = 10$ condition, the gains in prediction accuracy resulting from applying relaxation are not as pronounced as they were in simulation 1 with a continuous outcome. However, there are clear gains in the false positive rate, with only very small losses in true positives.

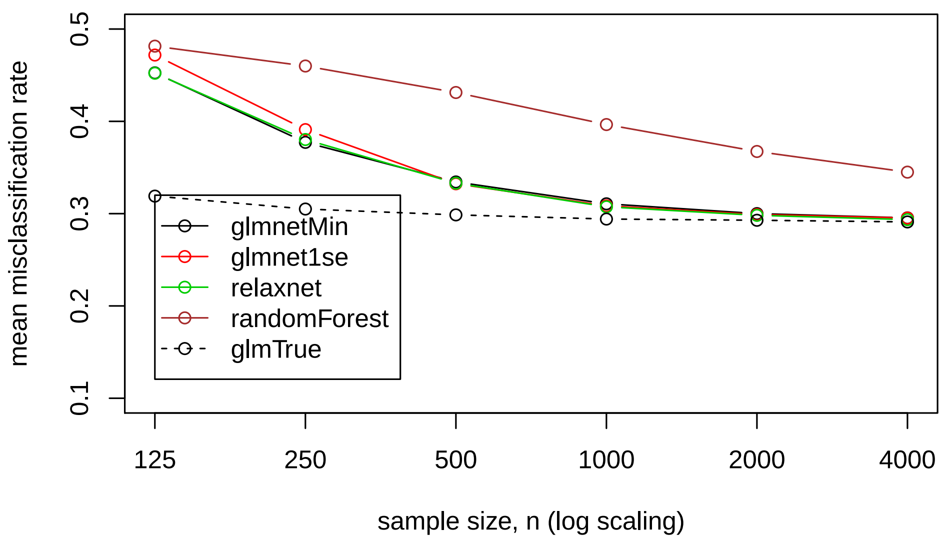

Figure 1.9: Mean test set misclassification rate for simulation 2, with $q = 10$ and $c = 2$

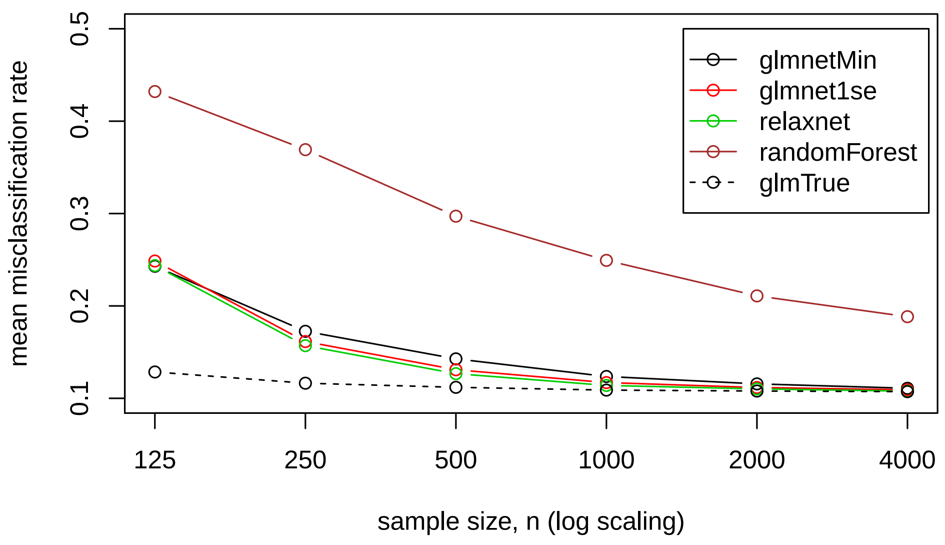

Figure 1.10: Mean test set misclassification rate for simulation 2, with $q = 10$ and $c = 8$

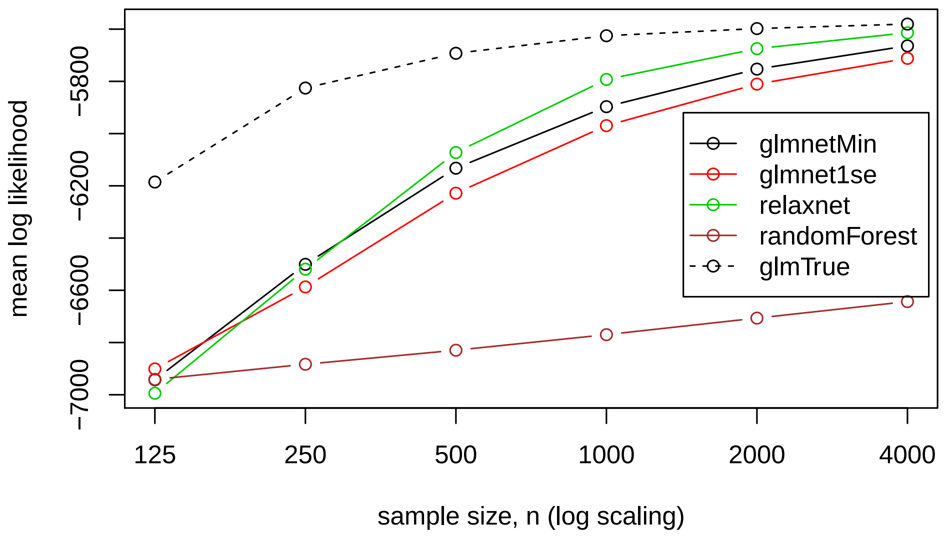

Figure 1.11: Mean test set log likelihood for simulation 2, with $q = 10$ and $c = 2$

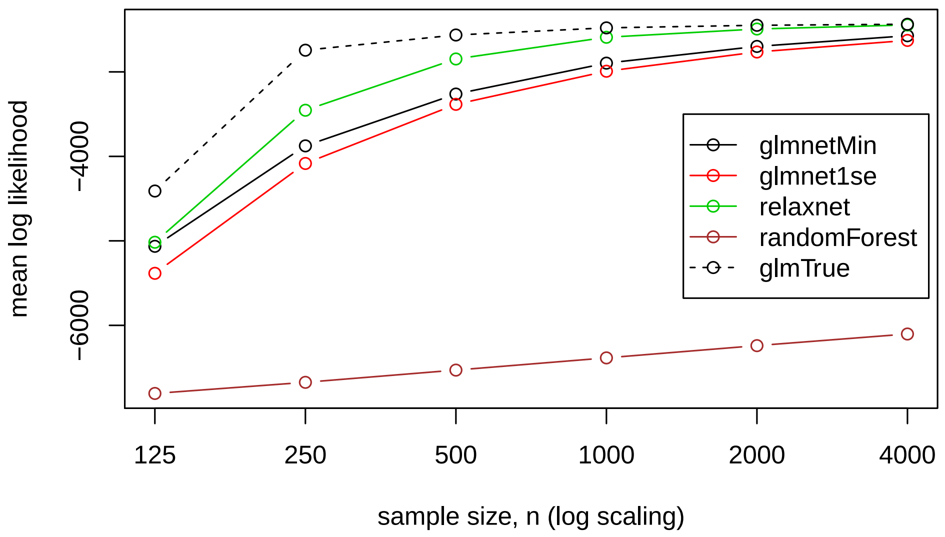

Figure 1.12: Mean test set log likelihood for simulation 2, with $q = 10$ and $c = 8$

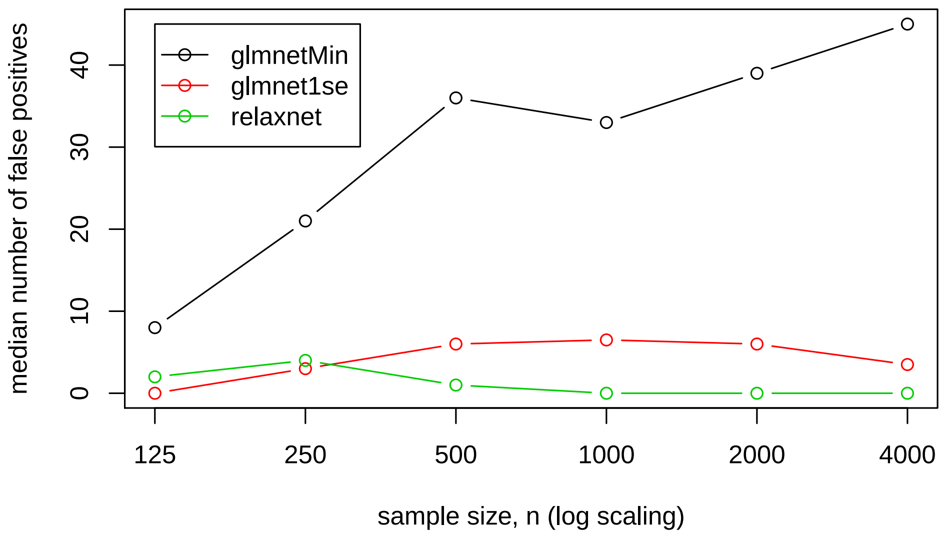

Figure 1.13: Median number of false positives for simulation 2, with $q = 10$ and $c = 2$

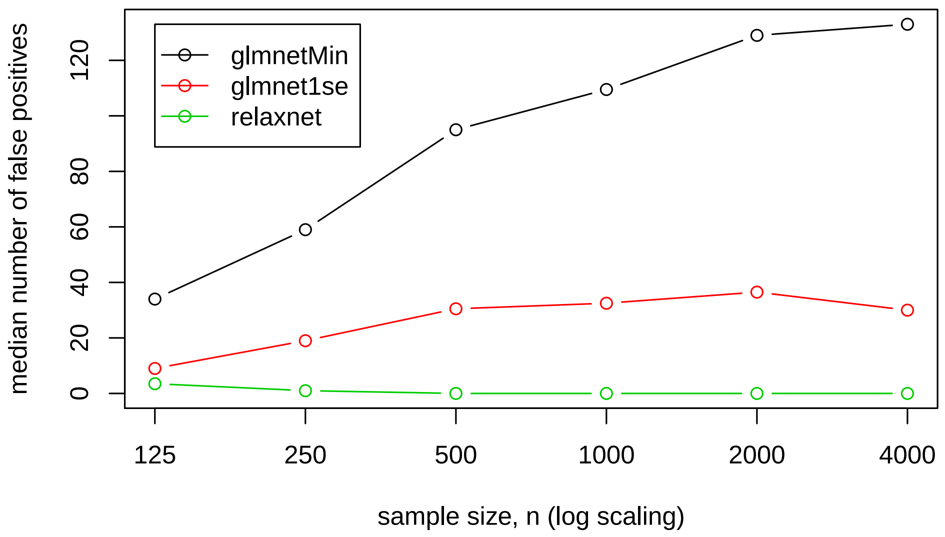

Figure 1.14: Median number of false positives for simulation 2, with $q = 10$ and $c = 8$

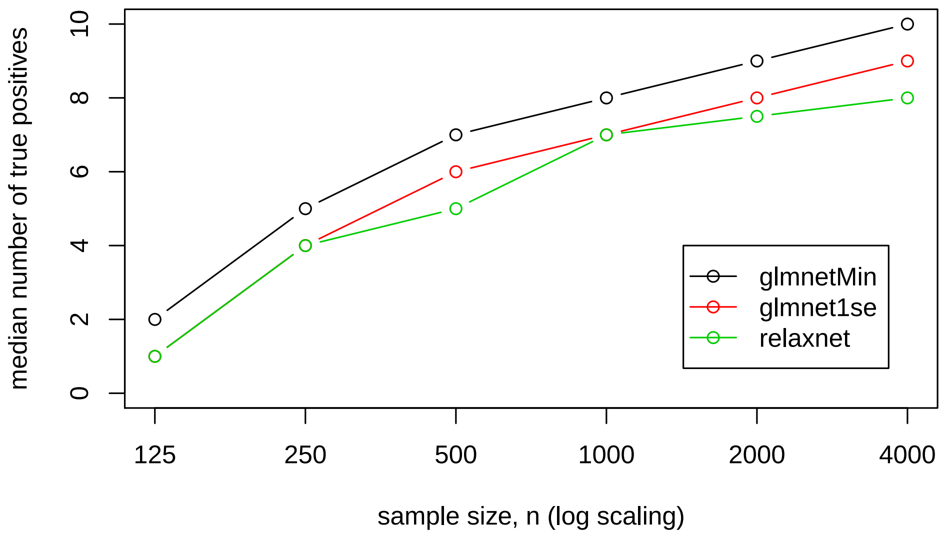

Figure 1.15: Median number of true positives for simulation 2, with $q = 10$ and $c = 2$

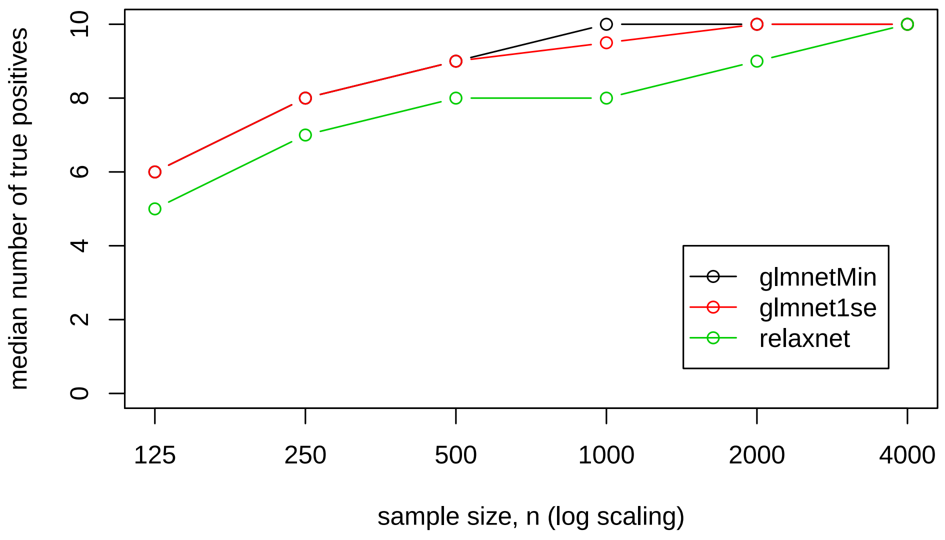

Figure 1.16: Median number of true positives for simulation 2, with $q = 10$ and $c = 8$

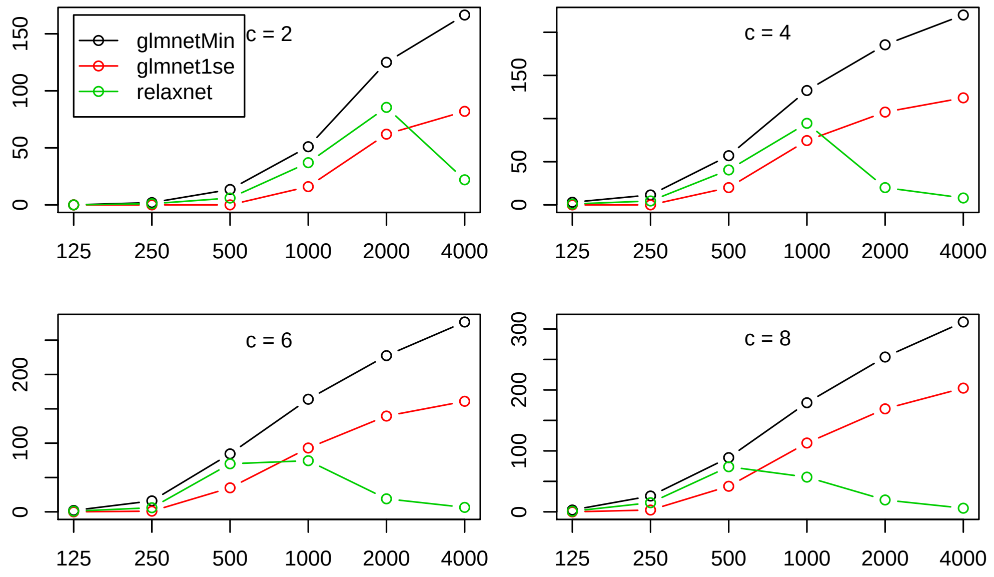

Figure 1.17: Median number of false positives for simulation 2, $q = 100$ (moderately sparse model). All four values of c are shown. The horizontal axes show sample size, n, with log scaling.

The prediction accuracy gains were even less pronounced for $q = 100$ (data not shown), however, there was an interesting trend in the false positives as c was varied, shown in Figure 1.17. The median number of false positives for relaxnet tends to be right in between those for glmnetMin and glmnet1se for lower sample sizes, with a downward trend starting at a certain point which decreases with increasing c. The low false positive levels at the higher sample sizes correspond to very low rates of the relaxnet cross-validation procedure choosing a solution from the main `glmnet` model (i.e. most often it was one of the relaxed models which resulted in the minimum cross-validated risk). So we see that even for only moderately sparse models, it is worth applying relaxation in order to reduce false positives, and that the reduction works especially well for higher sample sizes and higher levels of "signal" (when most y-generating probabilities are close to 0 or 1).

1.4.3 Simulation 3: Continuous outcome, correlated predictors

The third simulation was inspired by example 4 from Zou and Hastie (2005). As for simulation 1, the outcome was continuous. Only the first 15 predictors truly contributed to the outcome, and these were generated in three groups of 5 predictors each. Each of the three groups was generated independently as multivariate normal with within-group pairwise correlation ρ, which was varied with the following values: 0, 0.25, 0.5, 0.75, 0.9, 0.95, 0.98 and 0.99. The rest of the predictors were generated as independent standard normal variates, with a total of $p = 500$ predictors including the first 15. As for simulation 1, the sample size,

n, was held constant at 250, while the standard deviation of the error term was varied. The coefficient vector β consisted of the following values:

$$\beta_j = 1, j = 1, \dots, 15$$
$$\beta_j = 0, j = 16, \dots, 500.$$

The elements of the outcome were generated as

$$y_i \sim x_i^T \beta + \epsilon_i$$

with

$$\epsilon_i \sim N(0, \sigma^2).$$

for $i = 1, \dots, n$. σ, was varied, taking on the following values: 0.1, 0.25, 0.5, 1, 3, 6, 10, 15. Each condition was repeated 200 times, and the sample size for the test data set on which all performance was evaluated was 5000.

The methods which were compared were as follows:

- glmnetMinAlpha1, glmnet1seAlpha1, relaxnetAlpha1, lmTrue: These methods correspond to the glmnetMin, glmnet1se and lmTrue methods from simulation 1
- glmnetMinCValpha: elastic net with cross-validation on the value of α, the elastic net mixing parameter. No relaxation
- relaxnetCValpha: relaxnet with cross-validation on the value of α, using the `cv.alpha.relaxnet` function

The values of α used in the cross-validation for the final two methods were: 0.1, 0.3, 0.5, 0.7 and 0.9.

Figures 1.18, 1.19, and 1.20 show the relative mean of test set mean squared error for simulation 3, with $\rho = 0.25$, $\rho = 0.95$, and $\rho = 0.99$ respectively. Again, the values are relative to the means for the standard lasso estimator, glmnetMinAlpha1. We see that, once again, the 1se rule for choosing λ results in consistently worse performance than the min rule (red line vs. solid black line). There is clearly an improvement due to applying relaxation (relaxnetAlpha1 method, green line), and there is an additional improvement from applying an elastic net penalty in addition to relaxation (relaxnetCValpha method, blue line). The addition improvement seems to increase with increasing correlation. It is interesting to note that there does not seem to be much improvement for the elastic net method without relaxation (glmnetMinCValpha, orange line) vs. the standard lasso method, unless the correlation is at a very high level ($\rho = 0.99$, Figure 1.20). The difference between the elastic net penalty and the lasso penalty relaxed methods becomes apparent at much lower levels of correlation. A possible reason for this difference is the high dimensionality of the predictor matrix and the sparsity of the data generating distribution for this simulation. Due to this, the elastic net method without relaxation (glmnetMinCValpha) is at a disadvantage vs. the relaxed elastic net method (relaxnetCValpha). At high levels of correlation and high σ values, the relaxed methods even beat the lmTrue reference method, which is likely effected negatively by the high correlations.

Figures 1.21 and 1.22 show the median of false positives and true positives, respectively, for simulation 3 with $\rho = 0.95$. At this level of correlation, the relaxed methods do very well, with median 0 false positives across the board. It can be seen that the non-relaxed methods start to have many false positives at the higher σ values, with the elastic net method having a slight disadvantage vs. the lasso-penalized method for this category. In Figure 1.22, we see that the disadvantage of relaxnet with respect to finding the true positives is almost erased for the relaxnetCValpha method, which is very competitive with the glmnetMinCValpha method for this performance measure. This is likely a manifestation of the grouping effect for the elastic net penalty.

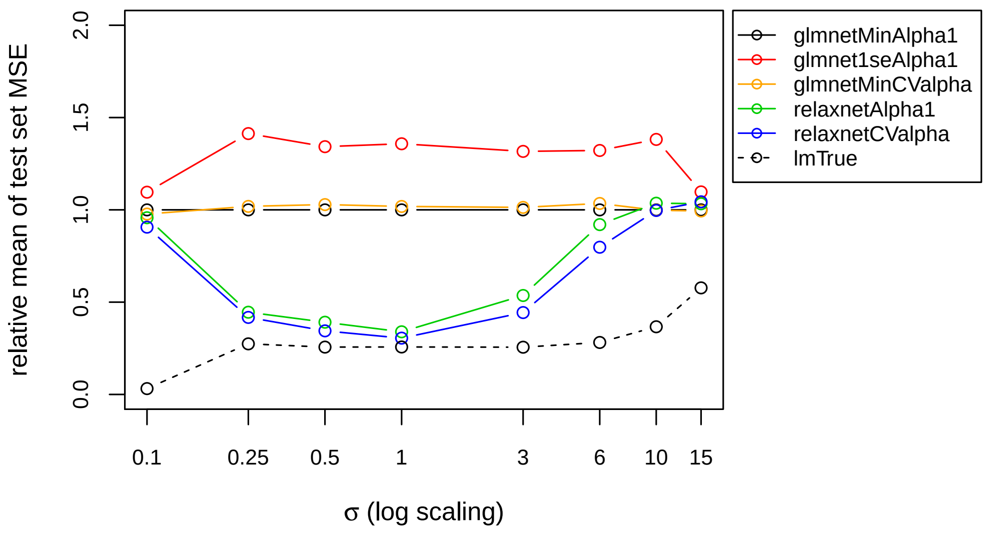

Figure 1.18: Relative mean of test set mean squared error for simulation 3, $\rho = 0.25$

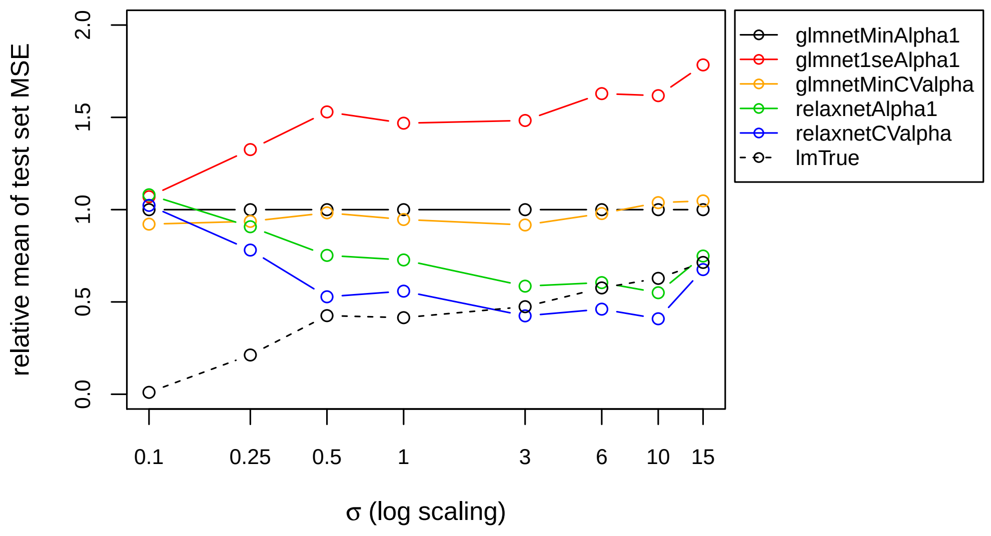

Figure 1.19: Relative mean of test set mean squared error for simulation 3, $\rho = 0.95$

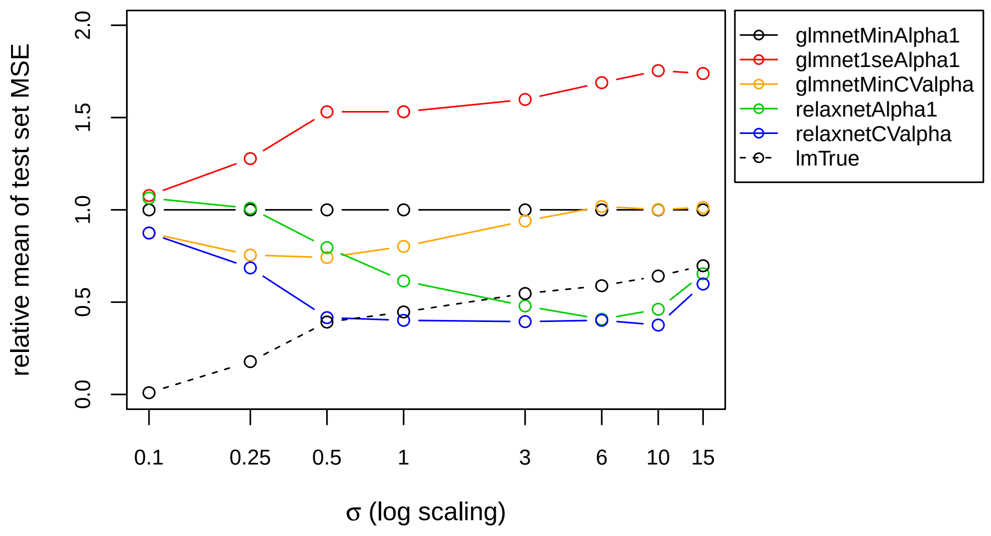

Figure 1.20: Relative mean of test set mean squared error for simulation 3, $\rho = 0.99$

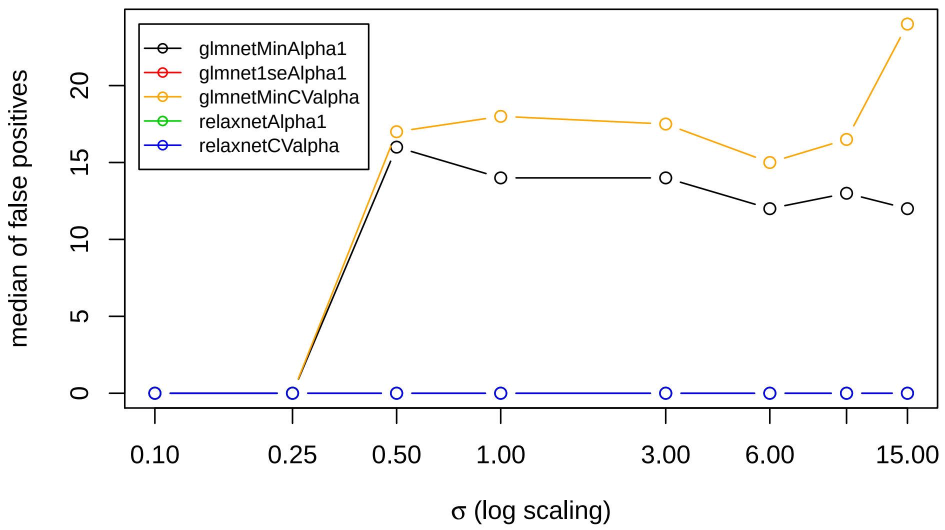

Figure 1.21: Median of false positives for simulation 3, $\rho = 0.95$. The values for the relaxnetAlpha1 method are occluded by those for the relaxnetCValpha method.

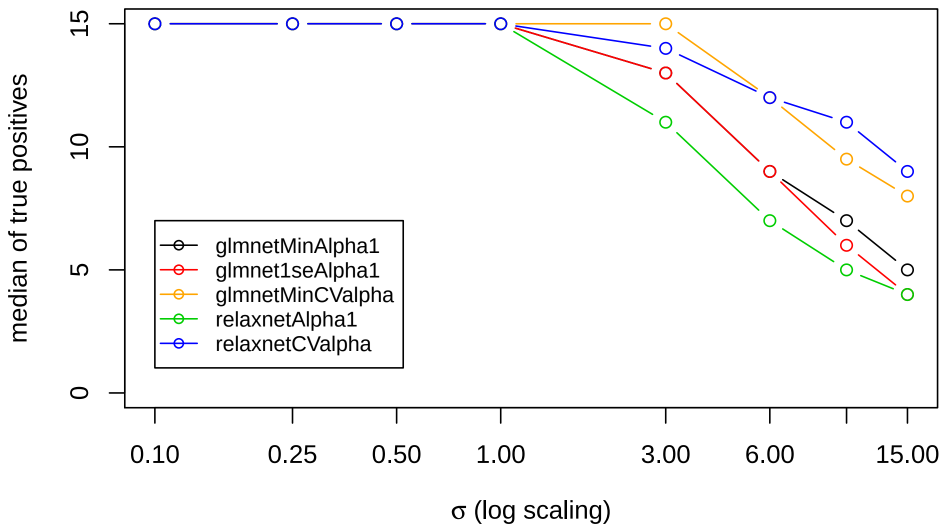

Figure 1.22: Median of true positives for simulation 3, $\rho = 0.95$

1.4.4 Using simulation 1 to compare running time of the `relaxnet` function with that of `relaxo`

In order to compare the running time of the `relaxnet` function with that of the `relaxo` function, the data generating model from simulation 1 was used, but the number of predictors, p, was varied, with 6 different values ranging between 500 and 20,000. The number of repetitions was decreased with increasing number of predictors, from 600 repetitions at $p = 500$, down to 100 repetitions at $p = 20,000$. The sample size, n, was held constant at 250. Figure 1.23 shows the relative timings (mean time for `relaxo` divided by mean time for `relaxnet`). The mean time for 20,000 predictors was 123 seconds for `relaxo`, while it was only 2.78 seconds for `relaxnet`, a speedup of 44.2X (only the base version of each function was run, there was no cross-validation).

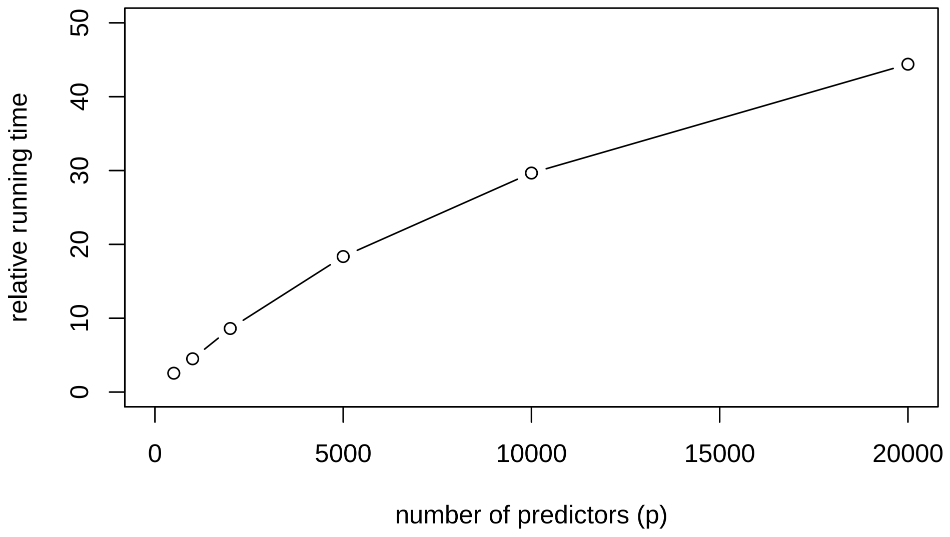

Figure 1.23: Relative timings: mean time for `relaxo` divided by mean time for `relaxnet`. The sample size, n, was held constant at 250.

1.5 Cross-validation with leukemia data

In order to compare the performance of relaxnet to other methods on a real data set, a cross-validation analysis was run using the microarray data from Golub *et al.* (1999). The data consisted of a total of 72 observations on 7129 genes, with a binary outcome indicating whether the leukemia type was acute lymphoblastic leukemia or acute myeloid leukemia. (In the paper the observations are split into a training set of size 38 and a validation set of size 34, but for the purposes of this analysis, the two sets were pooled.) The methods applied to this classification problem were the same methods as for simulation 2, except that 10 values of α were used in the cross-validation, instead of 5, for the glmnetCValpha and the relaxnetCValpha methods. The values were: 0.1, 0.2, ..., 0.9 and 1. An external level of cross-validation was applied, with $v = 12$ folds, and this was repeated with 10 separate splits of the data into 12 folds. Each method was also run once on the full data set of 72 observations. The results are shown in Table 1.1.

Clearly the best performing methods (those which gave the least number of misclassifications) are glmnetMinCValpha, relaxnetCValpha and randomForest. Among these three, relaxnetCValpha has the advantage of having resulted in the most simple model, with the classification being dependent on only 36 of the genes. In contrast, the glmnetCValpha method chose 452 genes. Here we again see the advantage of mixing relaxation with the elastic net penalty. The performance advantage of elastic net has been retained, without sacrificing the sparsity of the final model.

The identifiers and descriptions of the 36 genes found by relaxnet are listed in table 1.2. 21 of the 36 are part of the set of 50 which were found by Golub *et al.* (1999) to have the highest correlation with the outcome and formed the basis for their predictor.

	# misclassified for each data split											final
method	1	2	3	4	5	6	7	8	9	10	α_{min}	model size
glmnetMinAlpha1	3	3	5	4	5	4	4	3	5	3		26
glmnet1seAlpha1	8	8	9	8	8	6	5	6	7	9		18
glmnetMinCValpha	2	2	2	2	2	2	2	2	2	2	0.1	452
relaxnetAlpha1	5	4	8	6	5	5	4	7	5	6		9
relaxnetCValpha	2	2	2	2	2	3	2	2	2	2	0.2	36
randomForest	3	1	1	1	2	2	2	1	1	3		

Table 1.1: Results for leukemia data cross-validation. The main section shows the number of observations (out of 72) which were misclassified for each data split (this is the cross-validated error). α_{min} is the value of α chosen by the (internal) cross-validation procedure when the method was run on the full data set. Final model size is the number of predictors remaining in the final model (those whose coefficients were nonzero), also for the run on the full data set.

1.6 Discussion

In agreement with Meinshausen (2007), we have shown that the relaxed lasso can outperform the lasso with sparse, high dimensional data, and that it reduces the number of false positives. Cross-validating the lasso using the 1se rule for choosing λ is another way to reduce false positives, but as we have shown, this also has a large negative effect on prediction

identifier	Description
Y08612	nucleoporin 88kDa
M13690	serpin peptidase inhibitor, clade G (C1 inhibitor), member 1
U82759	Not Found
M31211	myosin, light chain 6B, alkali, smooth muscle and non-muscle
S50223	zinc finger protein 22
Y07604	NME/NM23 nucleoside diphosphate kinase 4
M28170	CD19 molecule
M23197	CD33 molecule
D49950	interleukin 18 (interferon-gamma-inducing factor)
J05243	spectrin, alpha, non-erythrocytic 1
X85116	stomatin
M84371	CD19 molecule
M65214	transcrptn. factor 3 (E2A immunoglbln. enhancer binding factors E12/E47)
M31523	transcrptn. factor 3 (E2A immunoglbln. enhancer binding factors E12/E47)
HG1612	Not Found
M16038	v-yes-1 Yamaguchi sarcoma viral related oncogene homolog
M22960	cathepsin A
U22376	v-myb myeloblastosis viral oncogene homolog (avian)
X95735	zyxin
U50136	leukotriene C4 synthase
M55150	fumarylacetoacetate hydrolase (fumarylacetoacetase)
L09209	amyloid beta (A4) precursor-like protein 2
M92287	cyclin D3
M33680	CD81 molecule
X59417	proteasome (prosome, macropain) subunit, alpha type, 6
M63138	cathepsin D
U05259	CD79a molecule, immunoglobulin-associated alpha
M27891	cystatin C
M19507	myeloperoxidase
M84526	complement factor D (adipsin)
X17042	serglycin
U46499	microsomal glutathione S-transferase 1
M96326	azurocidin 1
Y00787	interleukin 8
M11147	ferritin, light polypeptide
X14008	lysozyme

Table 1.2: The set of genes found by `relaxnet` for the leukemia data.

accuracy, and for the case of very low signal to noise ratios, the 1se rule performs worse than the relaxed lasso in terms of selecting the truly contributing predictors (true positives). We have shown that the **relaxnet** package has similar performance to the **relaxo** package when the lasso penalty ($\alpha = 1$) is applied, but is more computationally efficient. In addition, we have shown that it can be very effective to combine relaxation and elastic net penalties. When there are high correlations among predictors, the elastic net penalties result in better prediction accuracy. Applying relaxation counteracts the reduction of sparsity brought about by using an elastic net penalty instead of the lasso, while retaining the gains in prediction accuracy. The **relaxnet** package provides a multicore option, allowing the user to parallelize execution over cross-validation folds or parameter (α) values, thereby reducing running times.

One drawback of the **relaxnet** approach is that the predictions are restricted to linear combinations of the predictors. In real data settings, the model is rarely truly linear, and thus more flexible modeling approaches can be helpful. One such approach is described in Chapter 2. The approach described in this chapter is also not particularly useful if the goal is to obtain measures of variable importance for each predictor. Chapter 3 describes an approach to variable importance based on nonparametric, causal inference methodology (van der Laan and Rose, 2011).

Chapter 2

widenet: An R Package and Machine Learning Algorithm Combining Regularization with Polynomial Basis Expansions

2.1 Introduction

2.1.1 Motivation

This chapter describes the **widenet** R package, which extends the **glmnet** (Friedman *et al.*, 2010) and **relaxnet** (see Chapter 1) R packages with polynomial basis expansions. The main function, `widenet`, allows the user to select an order of basis expansion, and will run **relaxnet** in order to select a subset of basis functions following expansion of the basis to the given order. If more than one order is specified, cross-validation will be used to select the order in addition to the **relaxnet** tuning parameters. In order to accommodate high dimensional data, the option is provided to prescreen the predictors, and the screening takes place separately within cross-validation folds. In addition, a multicore option for parallelizing the computation over cross-validation folds is provided.

The polynomial basis expansions serve the purpose of increasing the size of the statistical model, in order to remain somewhat more nonparametric than purely linear methods such as least squares, lasso and **relaxnet**. Methods such as **widenet** and DSA, the deletion/substitution/addition algorithm (described in Section 2.2.1) provide a middle ground between the linear methods and more fully nonparametric methods such as random forests (Breiman, 2001; Liaw and Wiener, 2002). The nonparametric methods are fully geared towards prediction, and are not very useful from an explanation standpoint, when one would like to have some understanding of how the predictor variables are contributing to the outcome. Linear methods, on the other hand, give intuitive explanation, since the contribution of each predictor is given by a single number. However, linear methods may be very biased if the model is not truly linear. **widenet** avoids these extremes and settles somewhere in the middle, remaining flexible to nonlinearities in the data, but returning a relatively interpretable final model.

Due to the fact that polynomial basis expansion leads to a high-dimensional problem with potentially high correlations between columns of the predictor matrix, the **relaxnet** method is ideally suited for selecting the final subset of the basis functions. **relaxnet** combines elastic net penalties with relaxation in order to take advantage of improved performance in the case of correlated predictors, without sacrificing the sparsity of the final model. False positives (basis functions selected to be in the final model which do not truly contribute to the outcome) are kept at a low level (see Chapter 1).

2.1.2 Chapter overview

In Section 2.2 we review some previous approaches to regression with an expanded basis. In Section 2.3 we describe the components of the **widenet** package in more detail. In Section 2.4 we describe the results of a simulation in which **widenet** is compared to other methods. In Section 2.5 we report on a cross-validation data analysis using data from a study on gene expression effects of benzene exposure (McHale *et al.*, 2011). We close with a discussion in Section 2.6.

2.2 Previous approaches

2.2.1 The deletion/substitution/addition algorithm and the DSA R package

The deletion/substitution/addition algorithm (DSA; Sinisi and van der Laan, 2004) defines a method for doing a sequential search through subsets of candidate model elements using deletion, substitution and addition moves. Sinisi and van der Laan (2004) originally applied it to polynomial basis functions, and here I will focus on this application, but it has also been applied to partitioning by Molinaro *et al.* (2010), in a context similar to classification and regression trees (Breiman, 1984). Neugebauer and Bullard (2010) provide an R package, **DSA**, implementing the DSA with polynomial basis expansions.

The **DSA** package works by expanding the predictors into polynomial terms based on two parameters which define the maximum order of interaction (`maxorderint`) and the maximum sum of powers (`maxsumofpow`) of the expanded basis functions. The DSA algorithm is applied and does a sequential search through subsets of basis functions, fitting a new model at each step and evaluating the model fit in order to determine the next step. The search ends when the maximum model size (`maxsize`) has been reached. Finally, cross-validation is used in order to choose from among the models at each model size which had minimum prediction error.

DSA does not restrict candidate models to those which are "nested", i.e. it is allowed to have a model with high order terms, the elements of which are not included as lower order terms. For example, the term $x_1 : x_2$ may be in the model even though neither x_1 nor x_2 appears as a first order term. **DSA** will often find some linear combination of a small subset of high order terms which ends up having very good prediction performance.

Since it is basically a subset selection procedure, **DSA** suffers from high variance due to the discrete nature of the selection procedure as described in Section 1.2.2. An additional disadvantage is that the running time increases dramatically with increasing number of predictors and increasing values of the tuning parameters `maxorderint`, `maxsumofpow` and `maxsize`. In particular, Figure 2.1 shows that the running time of the `DSA` function is supracubic in the value of `maxsize`. This is a substantial disadvantage since it means that it may not be feasible for the user to allow for less sparse models which have a higher number of truly contributing terms. In addition, the fact that the bulk of the running time goes into the sequential search through subsets of basis functions means that the algorithm is not easy to parallelize.

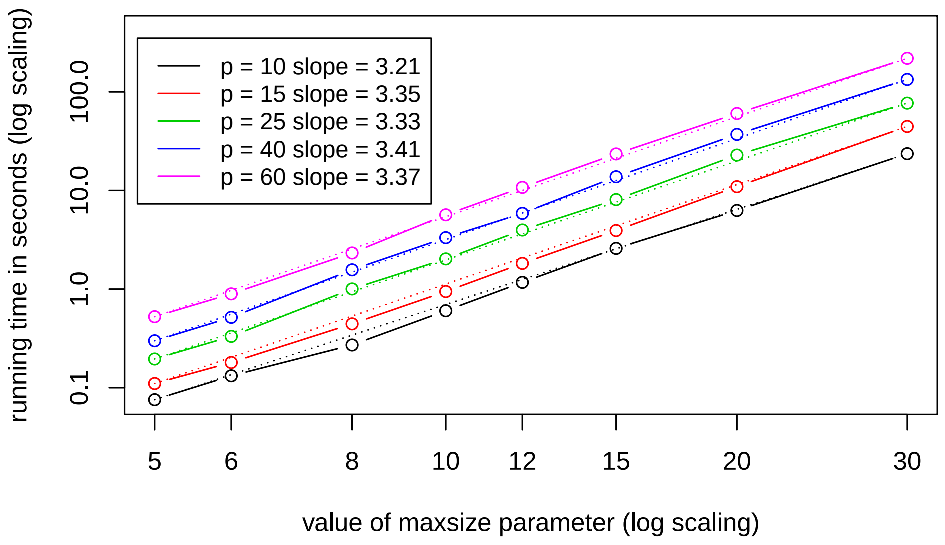

Figure 2.1: The running time of DSA is supracubic in maxsize. Plot shows mean running time over 10 repetitions on a log-log plot. Slopes are those of the dotted lines connecting the first and final data points at each value of p.

2.2.2 MARS and the polspline R package

Prior to the introduction of the DSA, the multivariate adaptive regression spline (MARS) procedure was developed by Friedman (1991). It is based on basis functions made up of linear splines. Later, Stone *et al.* (1997) and Kooperberg *et al.* (1997) extended this procedure with polynomials. Their algorithms are implemented by the functions `polyclass` and `polymars` from the **polspline** R package (Kooperberg, 2010). Unlike DSA, these functions require "nested" models, in that the lower-order terms on which higher order basis functions depend must be in the model before the higher order terms are allowed. This reduces the flexibility of the procedure.

2.2.3 The polywog R package

Kenkel and Signorino (2013) provide the **polywog** R package, which combines polynomial basis expansion with oracle model selection using either the adaptive lasso of Zou (2006) or the smoothly clipped absolute deviation penalty of Fan and Li (2001). This package provides functionality similar to the **widenet** package, however, there are certain disadvantages:

- It does not allow specification of elastic net penalties
- There is no option to prescreen variables separately within cross-validation folds
- The `polywog` function applies a QR factorization after the basis expansion in order to screen out colinear predictors. A side effect of this is that the final design matrix must be of full column rank, i.e. it may not have more columns than rows. Columns in excess of the number of rows are discarded

These last two disadvantages, in particular, make the **polywog** package unsuitable for higher-dimensional data. Considering that expanding the predictors out to the third order results in on the order of a cubing of the original number of predictors, the QR factorization greatly restricts the dimension of the data for which this method will be appropriate.

2.3 The widenet R package

2.3.1 Main function: `widenet`

The arguments to the main function of the **widenet** package are as follows:

```
widenet(x, y, family = c("gaussian", "binomial"),
        order = 1:3,
        alpha = 1,
        nfolds = 10,
        foldid,
        screen.method = c("none", "cor", "ttest"),
        screen.num.vars = 50,
        multicore = FALSE,
        mc.cores,
        mc.seed = 123,
        ...)
```

`x` is the matrix of predictors, `y` is the outcome vector, and `family` specifies whether the model should be linear or logistic. `order` specifies the order of basis expansion. If there is more than one element, cross-validation is used to select among the different orders. Similarly, alpha specifies the elastic net mixing parameter to be applied, and if this is a vector with more than one element, the value will be selected by cross-validation. `nfolds` and `foldid` determine how the data will be split up for v-fold cross-validation. With the `screen.method` argument, the user may specify a method to use for prescreening predictors, or that no screening should be done. `screen.method = "cor"` uses bivariate correlations with the outcome to screen predictors, while `screen.method = "ttest"` uses t-tests. If a screening method is selected, the screening will be done separately within cross-validation folds. `screen.num.vars` determines the number of predictor variables to be "screened in" or kept within each fold. The function returns an object of class `"widenet"`.

2.3.2 Options for parallel execution

The arguments `multicore`, `mc.cores` and `mc.seed` control the parallel execution. If `multicore = TRUE`, the execution will be parallelized over the cross-validation folds using the multicore functionality from R's **parallel** package (R Core Team, 2013). `mc.cores` specifies the number of processors/cores to use, and `mc.seed` allows the specification of a random seed for reproducibility.

2.3.3 Predict, summary and print methods

The `predict.widenet` method allows predictions to be made from class `"widenet"` objects. The method has arguments with which one may specify the values of `order` and `alpha` for which one would like the predictions to be made. The default values, however, are those values which resulted in the minimum cross-validated risk (i.e., those which "won" the cross-validation). The `summary.widenet` and `print.widenet` methods allow for the printing of a summary of the result, which shows the minimum cross-validated risk for each value of `order` and `lambda`.

	n	p = 50	p = 100	p = 200	p = 300	p = 500	p = 1000	p = 2000
order = 1	50	20 kB	40 kB	80 kB	120 kB	200 kB	400 kB	800 kB
	100	40 kB	80 kB	160 kB	240 kB	400 kB	800 kB	1.6 MB
	200	80 kB	160 kB	320 kB	480 kB	800 kB	1.6 MB	3.2 MB
	500	200 kB	400 kB	800 kB	1.2 MB	2 MB	4 MB	8 MB
	1000	400 kB	800 kB	1.6 MB	2.4 MB	4 MB	8 MB	16 MB
	5000	2 MB	4 MB	8 MB	12 MB	20 MB	40 MB	80 MB
	10000	4 MB	8 MB	16 MB	24 MB	40 MB	80 MB	160 MB
order = 2	50	530 kB	2.1 MB	8.1 MB	18 MB	50 MB	200 MB	800 MB
	100	1.1 MB	4.1 MB	16 MB	36 MB	100 MB	400 MB	1.6 GB
	200	2.1 MB	8.2 MB	32 MB	73 MB	200 MB	800 MB	3.2 GB
	500	5.3 MB	21 MB	81 MB	180 MB	503 MB	2 GB	8 GB
	1000	11 MB	41 MB	160 MB	360 MB	1 GB	4 GB	16 GB
	5000	53 MB	206 MB	812 MB	1.8 GB	5 GB	20 GB	80 GB
	10000	106 MB	412 MB	1.6 GB	3.6 GB	10 GB	40 GB	160 GB
order = 3	50	9.4 MB	71 MB	550 MB	1.8 GB	8.4 GB	67 GB	530 GB
	100	19 MB	140 MB	1.1 GB	3.7 GB	17 GB	130 GB	1.1 TB
	200	37 MB	280 MB	2.2 GB	7.3 GB	34 GB	270 GB	2.1 TB
	500	94 MB	710 MB	5.5 GB	18 GB	84 GB	670 GB	5.3 TB
	1000	190 MB	1.4 GB	11 GB	37 GB	170 GB	1.3 TB	11 TB
	5000	937 MB	7.1 GB	55 GB	180 GB	840 GB	6.7 TB	53 TB
	10000	1.9 GB	14 GB	110 GB	370 GB	1.7 TB	13 TB	110 TB

Table 2.1: Memory requirements for widenet by matrix size and order of basis expansion. Assumes a dense matrix of doubles.

2.3.4 Memory use

For large predictor matrices, the `widenet` function may require a substantial amount of memory. The maximum order of basis expansion is 3, but even at this level, the amount of memory required to store the design matrix is significantly greater than that for the unexpanded predictors. Table 2.1 shows memory requirements for matrices of different dimensions before and after basis expansion.

2.3.5 Example

In this section we give a small example demonstrating the use and output of the `widenet` function. First we load the **widenet** package and generate a 500 by 5 predictor matrix of independent standard normal variates:

```
R> library("widenet")
R> n <- 500
R> p <- 5
R> set.seed(23)
R> x <- matrix(rnorm(n*p), n, p)
```

The predictor matrix must have unique column names:

```
R> colnames(x) <- paste("x", 1:ncol(x), sep = "")
```

Next we generate an outcome based on truly contributing basis functions of order 1 and 2, plus a standard normal error term:

```
R> y <- x[, 1] + x[, 2] + x[, 3] * x[, 4] + x[, 5]^2 + rnorm(n)
```

Now we run widenet. The default is to run all three orders from 1 to 3. We will use three different α values which will be cross-validated against each other.

```
R> widenet.result <- widenet(x, y, family = "gaussian",
+                            alpha = c(0.1, 0.5, 0.9))
```

Now we display the results. We call `summary.widenet` and print the nonzero coefficients for the final chosen model:

```
R> summary(widenet.result)

The call was:

 widenet(x = x, y = y, family = "gaussian", alpha = c(0.1, 0.5,      0.9))

The minimum cross-validated risk by order and alpha value:

     order
alpha        1        2        3
  0.1 4.200071 1.038802 1.027527
  0.5 4.186725 1.053491 1.026082
  0.9 4.182633 1.049731 1.038028

The total time taken to fit this model was:

[1] 8.94

seconds.

R> coefs <- drop(predict(widenet.result, type = "coef"))
R> coefs[coefs != 0]

(Intercept)          x1          x2       x3:x4     I(x5^2)
 0.05942064  0.91541366  0.98333529  1.09731775  0.96212394
```

`widenet` has found all of the truly contributing basis functions. Further inspection reveals which part of the model gave the minimum cross-validated risk:

```
R> order.min <- widenet.result[["which.order.min"]]
R> order.min

[1] "3"

R> alpha.min <- widenet.result[["which.alpha.min"]]
R> alpha.min

[1] 0.5

R> order.min.index <- which(widenet.result$order == order.min)
R> order.min.index

[1] 3

R> alpha.min.index <- which(widenet.result$alpha == alpha.min)
R> alpha.min.index

[1] 2

R> min.cv.relaxnet.fit <-
+    widenet.result$cv.relaxnet.results[[order.min.index]][[alpha.min.index]]
R> min.cv.relaxnet.fit$which.model.min

[1] 3
```

The third relaxed model from the `cv.relaxnet` run with `order = 3` and `alpha = 0.5` gave the minimum cross-validated risk.

2.4 Simulation

In order to compare the performance and basis function selection behavior of **widenet** with other methods, a simulation was run in which the data generating model depended on a small number of terms of different orders. The predictor matrix X was given dimension $n = 250$ by $p = 100$ and generated from independent standard normal variates. The outcome was continuous, with the elements of the outcome vector being generated as

$$y_i \sim x_{i1} + x_{i2} + x_{i1}x_{i2}^2/2 + x_{i1}^3/4 + \epsilon_i$$

where x_{ij} is the element of X from the ith row and the jth column, and

$$\epsilon_i \sim N(0, \sigma^2).$$

The standard deviation of the error term, σ, was varied, taking on the following values: 0.1, 0.25, 0.5, 1, 3, 6, 10. The truly contributing basis functions were x_1, x_2, $x_1 : x_2^2$, and x_1^3. Each condition was repeated 200 times with a separately generated data set, and all evaluation of prediction performance was done using a test set of size $n = 5000$.

The methods which were compared in this simulation were as follows:

- widenetCValpha: this method used the `widenet` function, with the default value for order, in other words the order of either 1, 2, or 3 was chosen by cross-validation. In addition, the α value of either 0.1, 0.3, 0.5, 0.7, or 0.9 was also chosen by cross-validation. `screen.method` was set to `"cor"` and `screen.num.vars` was set to 20

- widentAlpha$x$: this represents six separate methods, for six different values of α. The `widenet` function was used, with cross validation on three levels of the order of basis expansion, but instead of cross-validating to select α, it was kept constant at one of the following values: 0.1, 0.3, 0.5, 0.7, 0.9 or 1. Screening was used with the same parameter values as for the widenetCValpha method

- DSA: this method used the `DSA` function from the **DSA** package, with no pre-screening, `maxorderint = 3`, `maxsumofpow = 3`, and `maxsize = 5`

- DSAscreen: this also used the `DSA` function with the same parameters, but for this method, 20 of the predictors were screened in (using the bivariate correlation with the outcome) prior to running `DSA`. Note that the `DSA` function does not allow screening separately within cross-validation folds

- randomForest: the `randomForest` function from the package **randomForest** (Liaw and Wiener, 2002) was used with the default options

- lmTrue: this method used `R`'s built-in `lm` function to run a linear model using the actual basis functions which truly contribute to the outcome, for reference

The results of the simulation are shown in Figures 2.2 through 2.5. Figure 2.2 shows the performance (mean of test set mean squared error) for a subset of the methods. Figure 2.3 compares all of the widenet methods by showing their performance relative to the widenetAlpha1 method. Figures 2.4 and 2.5 show the mean number of true and false positives, respectively, for the DSA methods and a subset of the widenet methods. Unlike the DSA methods and the randomForest method, the performance of the widenetCValpha method approaches that of the lmTrue reference method at the lower values of σ. The DSA methods and the randomForest method both do poorly compared to the widenetCValpha method. For randomForest, this is again likely to be due to the sparsity of the model (see Section 1.4.2). Both of the DSA methods also do poorly in terms of true positives, i.e. they are not able to consistently find all the true basis functions, as seen in Figure 2.4. This accounts for the poor performance of the DSA methods. The performance of most of the

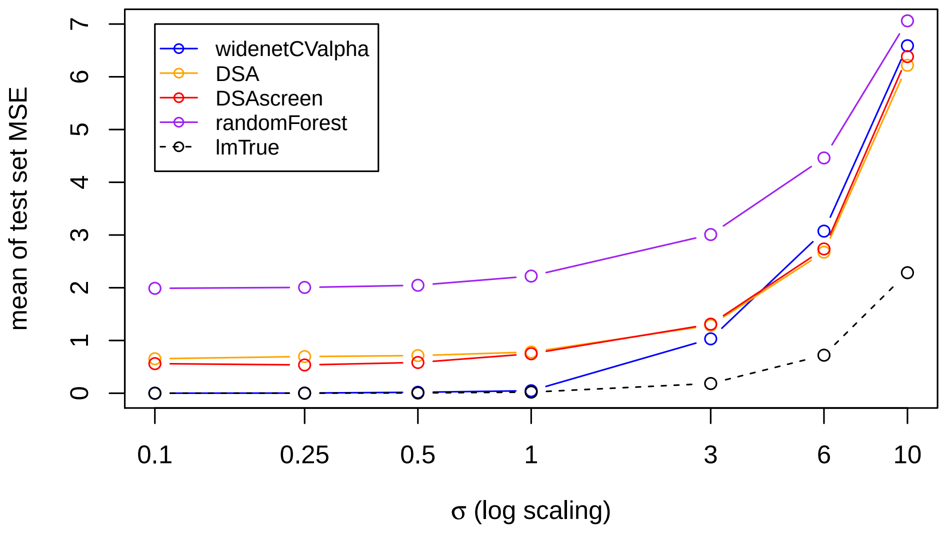

Figure 2.2: Prediction performance for the widenet simulation. Values are means of test set mean squared error over the 200 repetitions. The performance of the other widenet methods was similar to widenetCValpha and is shown in the next figure.

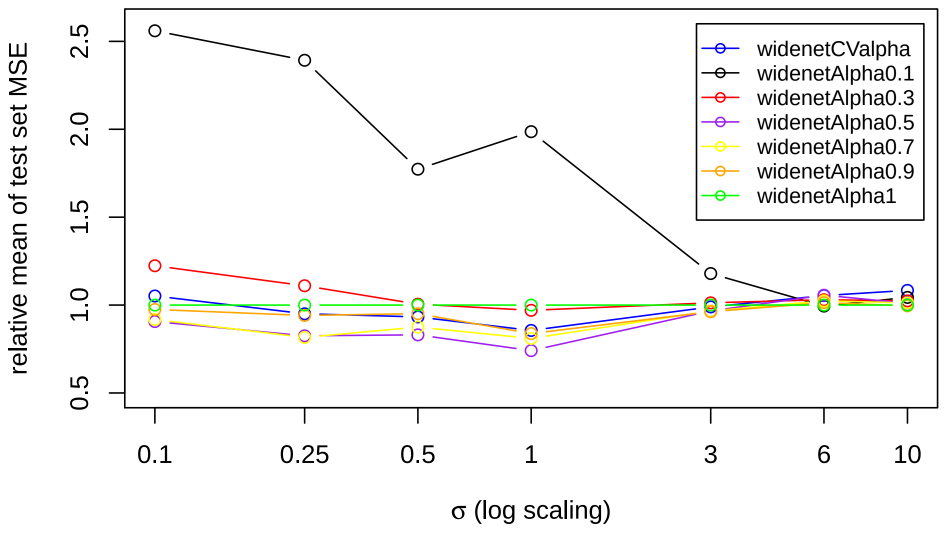

Figure 2.3: Relative prediction performance for the widenet methods. All values are plotted relative to the mean for the widenetAlpha1 method.

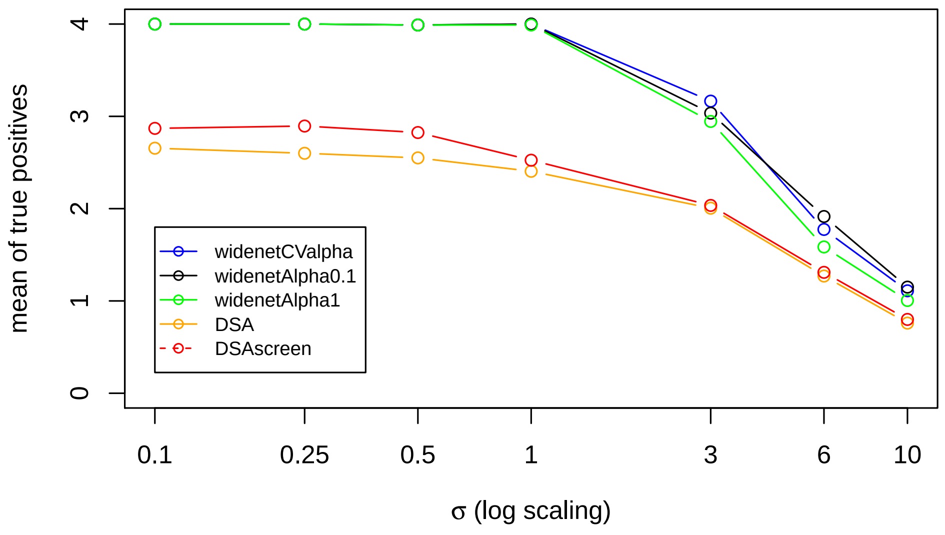

Figure 2.4: Mean of true positives for widenet simulation. There were four truly contributing basis functions.

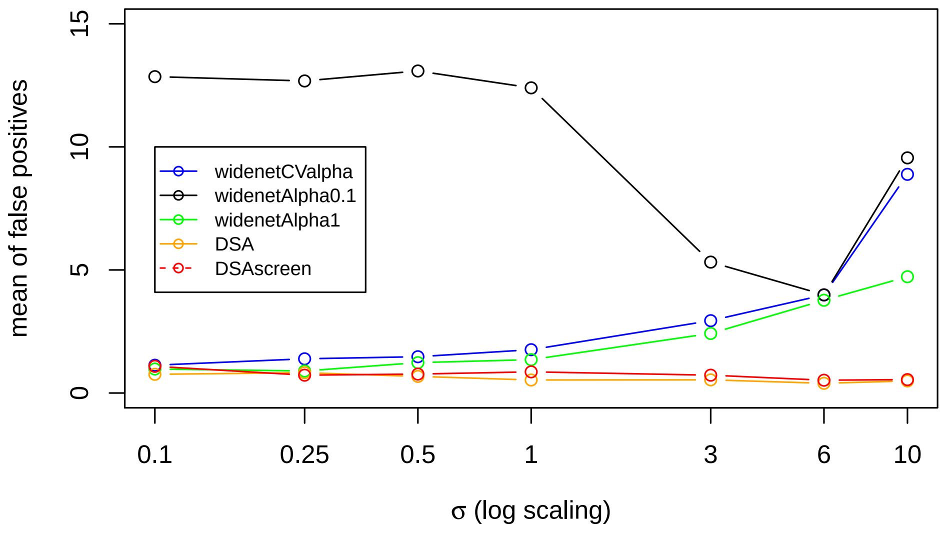

Figure 2.5: Mean of false positives for widenet simulation. Each additional basis function that had a nonzero coefficient in the final model but did not truly contribute to the outcome was counted as a false positive.

widenetAlpha$x$ methods was similar to that for the widenetCValpha method, with the exception of the widenetAlpha0.1 method, which had much worse performance at the lower levels of σ. Looking at Figure 2.5, we see that this method also had a much higher number of false positives at these σ values, which may account for its poor performance relative to the other widenet methods. The level of correlation among the truly contributing basis functions (in the range of 0.5-0.8, pairwise, for the terms x_1, $x_1 : x_2^2$, and x_1^3) was apparently high enough to cause problems for the two DSA methods in finding all of the true basis functions, but not high enough to create a significant performance advantage for elastic net penalties over the lasso penalty within the widenet methods.

2.5 Cross-validation with benzene data

In this section we compare the performance of **widenet** with that of **DSA** and **randomForest** using data from a study of occupational benzene exposure in factory workers (Lan *et al.*, 2004; Vermeulen *et al.*, 2004; McHale *et al.*, 2011). The subjects worked in three clothes-manufacturing factories near Tianjin, China. For this analysis, a cross-validation was performed using data from 59 workers with low levels of benzene exposure (<1ppm), along with 42 workers from the control, unexposed group. The binary outcome consisted of the class label, exposed or unexposed. Each subject had had the expression levels of 22,177 genes assessed by microarray analysis (McHale *et al.*, 2011), and the expression measures made up the predictor matrix. The following methods were compared:

- widenetOrder2Screen250: The `widenet` function was used, with `order` set to both order 1 and order 2; cross-validation on five α values: 0.1, 0.3, 0.5, 0.7 and 0.9; `screen.method = "ttest"` and `screen.num.vars = 250`
- widenetOrder3Screen50: Like the previous method, only with `order` set to do all three orders, 1, 2 and 3, and `screen.num.vars` set to 50.
- DSAorder2Screen100: First 100 genes were screened in using t-tests, then DSA was run using `maxorderint = 2`, `maxsumofpow = 2` and `maxsize = 6`
- DSAorder3Screen40: Like the previous method, only 40 genes were screened instead of 100, and `maxorderint` and `maxsumofpow` were set to 3
- randomForest: This method used the `randomForest` function from the **randomForest** package, with the default options

An outside level of 10-fold cross-validation was performed, and the resulting cross-validated predicted probabilities were used to draw ROC curves, shown in Figure 2.6. We see that although the randomForest method had the highest area under the curve (AUC) overall, the widenet methods were not far behind, and both of the widenet methods did better than the two DSA methods. It seems that for both widenet and DSA, there was a slight advantage of expanding the basis all the way to order 3, instead of just order 2, even though the order 3 methods had fewer genes to work with.

Finally, each method was also run on the entire data set with the timings being recorded and the final model examined for model size. The results are shown in Table 2.2. Note that the widenet methods were run using multicore processing on 4 cores. Despite the fact that each widenet method had to be repeated for each of the 5 separate α values, the timings for the widenet method were not excessively longer than those for the other methods. It must also be considered that, for the DSA methods, the `maxsize` argument was set to only 6, meaning that these methods would not have been able to find solutions containing more than 6 basis functions. As shown in figure 2.1, setting the `maxsize` argument at a higher level may have significantly increased the running time of the DSA methods.

method	running time (s)	final model size
widenetOrder2Screen250	113.9	7
widenetOrder3Screen50	172.1	22
DSAorder2Screen100	40.1	1
DSAorder3Screen40	30.9	3
randomForest	26.6	

Table 2.2: Timings and final model size for run on the full benzene data set. Model size gives the number of basis functions and does not include the intercept.

2.6 Discussion

In this chapter we have introduced the **widenet** R package. Like the **DSA** package, **widenet** uses polynomial basis expansions in order to keep the statistical model larger than for purely linear methods. However, **widenet** has several advantages over **DSA**:

- As shown in Sections 2.4 and 2.5, the prediction accuracy of **widenet** is very competitive with that of **DSA**
- With **widenet** it is not necessary to specify a `maxsize` argument. The algorithm is adaptive to the level of sparsity of the true model. The running time of **DSA** is highly dependent on the value of the `maxsize` argument
- While most of the running time for **DSA** goes into a sequential search through basis functions, **widenet** is based mainly on cross-validation and is therefore more easily parallelizable than **DSA**

- **widenet** is based on **glmnet** and **relaxnet**, which are well suited to high-dimensional problems with $p >> n$. In addition, **widenet** allows for screening the predictors separately within cross-validation folds. Cross-validating after applying screening has been shown to lead to over-fitting (Cawley and Talbot, 2010)

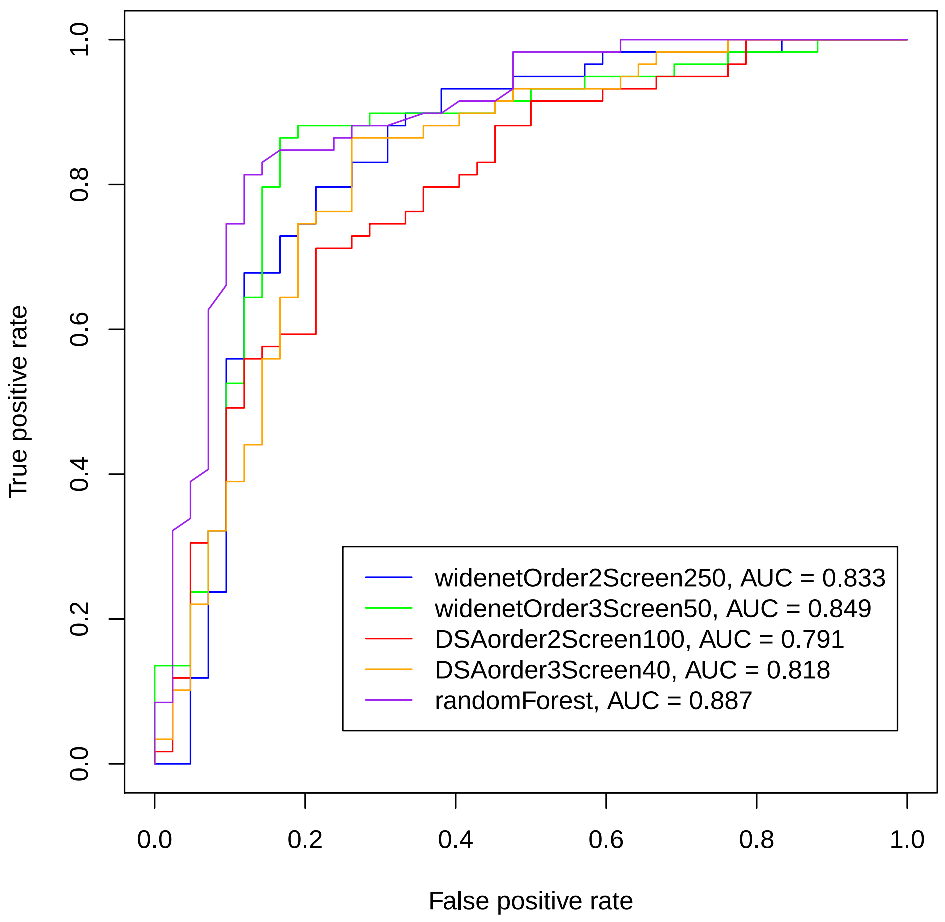

Figure 2.6: ROC curves for benzene data cross-validation. The area under the curve (AUC) for each method is shown in the legend.

Non-parametric methods such as **randomForest** are hard to beat when it comes to prediction accuracy. Nevertheless, we have shown a case where the model is very sparse and **widenet** outperforms **randomForest** (simulation in Section 2.4). Also, the final model returned by **widenet** is much more interpretable than that for **randomForest**.

Chapter 3

multiPIM: A Causal Inference Approach to Variable Importance Analysis

Note: A portion of this material has appeared previously in my master's thesis (Ritter, 2011), and it has been submitted as an article that is currently being reviewed by the *Journal of Statistical Software* (Ritter *et al.*, submitted). This material is being included here with permission from the graduate division and from the coauthors.

3.1 Introduction

3.1.1 Motivation

In most observational epidemiological studies, the investigators are interested in determining the independent association between one or more exposure variables and an outcome of interest, which requires adjustment for potentially confounding variables. Typically, and especially when the study is one of the first to be done in a certain field, there is little or no *a priori* knowledge of which variables are "important." Thus, a wide net is cast, many variables are measured, and a multivariate regression model is built in an effort to adjust for confounding. The supposed effects of each exposure variable are read off from the estimated coefficients of this model, and statistical inference (confidence intervals or p values) is based on the standard error associated with each coefficient.

However, the interpretation of the model coefficients as the causal effects, and the inference obtained, are only valid if the model used in the analysis 1) closely approximates the true model and 2) has been specified *a priori*, without any feedback from the data. Unfortunately, these two conditions are somewhat mutually exclusive. Since there is rarely any real, *a priori*, knowledge of the true form of the relationships between the variables, the only way that one can hope to specify the model accurately is through a feedback cycle of multiple candidate models which are tested against the data and modified accordingly. Usually only a very small space of possible models can be searched by this *ad hoc* method. Thus, there is not much chance that the final model selected will closely approximate the true model. An incorrectly specified model will result in biased estimates of the adjusted associations. Failure to account for using feedback from the data in the analysis will typically result in artificially low standard errors. The likely result is a misleading, narrow confidence interval (or low p-value) around a biased estimate.

Even more problematic is the fact that the parameter used may depend on the model selected, and thus on the data. For example, for one realization of the data, the informal model selection may result in no interaction terms with the variable of interest, and thus only one measure of association would be reported. However, for another realization of the data, the procedure may choose a multiplicative interaction term in the final model, and in this case the researcher would report two measures of the adjusted association, for the two different levels of the presumed effect modifier. In this scenario, the concept of a sampling distribution of an estimator, which forms the basis for inference, breaks down.

van der Laan (2006) has proposed a solution to the problems outlined above. He takes a causal inference approach, and suggests that the first step is to decide on a real-valued parameter of interest (the variable importance measure). Then an efficient and robust estimator of this parameter can be derived using estimating equation methodology (van der Laan and Robins, 2003) or targeted maximum likelihood (van der Laan and Rubin, 2006; van der Laan and Rose, 2011). This estimator is then applied to each variable in turn, instead of estimating a single, global regression model.

The whole data analysis can be specified *a priori*, and one can still optimize the model selection by making aggressive use of modern machine learning algorithms to estimate the nuisance parameters of the model. These algorithms will data-adaptively search a large model space, and thereby, hopefully, come up with a reasonable approximation to the true data-generating distribution. By turning this entire multi-step procedure into an *a priori* specified black box, one can harness the power of aggressive computing to obtain consistent estimates with honest inference.

Other approaches to variable importance analysis are provided by R packages such as **randomForest** (Liaw and Wiener, 2002) and **caret** (Kuhn *et al.*, 2011; Kuhn, 2008). While these approaches also make use of modern machine learning methods, most of them are based on assessing changes in the risk function, while the approach of van der Laan (2006) is a type of adjusted mean inspired by causal inference (though the general targeted maximum likelihood approach can also be adapted to estimation of the risk function itself). This has several additional advantages that are not shared by these other approaches. For example, the choice of the parameter of interest can be tailored to the specific problem, and since the method is not tied to a specific learning algorithm, one can combine several arbitrary algorithms in a super learner in order to estimate nuisance parameters (see Section 3.2).

3.1.2 Chapter overview

In this chapter, we will introduce the package **multiPIM**, written for the R statistical computing environment (R Core Team, 2013), and available for download from the Comprehensive R Archive Network (CRAN, `http://cran.r-project.org/package=multiPIM`)[1]. **multiPIM** performs variable importance analysis by fitting multiple Population Intervention Models (PIMs, Hubbard and van der Laan, 2008) to a data set with possibly many exposures (or treatments) of interest and possibly many outcomes of interest. In Section 3.2, we summarize the statistical properties of PIMs, describing in particular the procedure for their estimation using data-adaptive machine learning algorithms. In Section 3.3, we go into more detail about **multiPIM** and its functions. Section 3.4 describes a simulation which demonstrates the benefit of using a double robust estimator. In Section 3.5 and Section 3.6, we report on reanalyses of data from the Western Collaborative Group Study (WCGS, Rosenman *et al.*, 1966, 1975) and from a study of water contact and schistosomiasis infection (Spear *et al.*, 2004; Sudat *et al.*, 2010). Section 3.6 (schistosomiasis example) includes the code for running the analysis. In Section 3.7 we report on a variable importance analysis for the PRospective Observational Multicenter Major Trauma Transfusion (PROMMTT) study (Rahbar *et al.*, 2012; Hubbard *et al.*, 2013). We close with a discussion in Section 3.8.

3.2 Statistical methodology

The approach for estimating PIMs was presented by Hubbard and van der Laan (2008), and is also described in Young (2007) and Young *et al.* (2009). We summarize the main points here.

3.2.1 Data structure and parameter of interest

Let us assume that we have an observed data structure

$$O = (Y, A, W) \sim P_0,$$

where Y is an outcome (which may be either binary or continuous), A is a binary exposure or treatment variable, W may be a vector of possible confounders of the effect of A on Y, and P_0 is the data-generating distribution. The parameter of interest is

$$\begin{aligned}\psi(P_0) = \psi &= E_W[E(Y|A=0,W) - E(Y|W)] \\ &= E_W[E(Y|A=0,W)] - E(Y).\end{aligned}$$

This parameter is a type of attributable risk; it compares the overall mean of the outcome to the mean for a target group (defined by $A = 0$), averaged over strata of W. The hypothetical full data is given by

[1]As of this writing, the current version is 1.3-1

$$X = (Y_0, Y, W) \sim P_X,$$

where Y_0 is the counterfactual outcome for $A = 0$, i.e., the outcome as it would be under universal application of treatment $A = 0$. The causal analogue to $\psi(P_0)$ is

$$\begin{aligned}\psi(P_X) &= E_W[E(Y_0|W) - E(Y|W)] \\ &= E[Y_0 - Y].\end{aligned}$$

There are certain assumptions under which $\psi(P_0) = \psi(P_X)$. The assumptions are:

1. Time ordering assumption: W preceded A and A preceded Y in time. More generally, the data-generating distribution conforms to a specific nonparametric structural equation model (Pearl, 2000).

2. Consistency assumption: The observed data structure, O, is a missing data structure on X (van der Laan and Robins, 2003).

3. There is no unmeasured confounding, or, equivalently, A is conditionally independent of Y_0, given W (van der Laan and Robins, 2003).

4. The experimental treatment assignment (ETA) assumption or positivity assumption: the probability that A is zero, given W, is bounded away from zero, or $Pr(A = 0|W) > 0$. This is a relaxed version of the ETA assumption, which for certain other parameters requires a positive probability of having *each* of the treatment levels over the distribution of W in the target population (van der Laan and Robins, 2003; Messer *et al.*, 2010).

If these four assumptions hold, and the relevant models are estimated consistently, ψ can be thought of as an actual causal effect of A on Y. More specifically, it can be thought of as measuring the hypothetical effect of an intervention in which everyone in the population is made to be like the members of the target group. For example, if the target group corresponds to "unexposed", then ψ would be the effect, on the overall mean of the outcome, of eliminating the exposure entirely. However, even if some of these assumptions do not hold, ψ may still be an interesting and worthwhile parameter to pursue. In this case, it can still be thought of as a type of variable importance (van der Laan, 2006), and thus as a good way of quantifying the (W-adjusted) level of association between A and Y.

The term attributable risk has had several slightly different meanings in the epidemiological literature, and some authors prefer the term attributable fraction (e.g., Greenland and Drescher 1993). For the case of a binary outcome, our parameter, ψ, is like a causal (W-adjusted) and sign-reversed version of what Gordis (2004) calls the "attributable risk in the total population":

$$\begin{pmatrix}\text{Incidence in}\\ \text{total population}\end{pmatrix} - \begin{pmatrix}\text{Incidence in}\\ \text{nonexposed group}\\ \text{(background risk)}\end{pmatrix} \tag{3.1}$$

(Gordis, 2004, Formula 12.3). In Section 3.5 below we will calculate a "naive attributable risk," (naive since it is bivariate only and does not account for confounding) as the difference between the mean of the binary coronary heart disease outcome for an unexposed group and the overall mean of the outcome.

The goal of this work is to create an automated algorithm for estimating ψ for data sets with potentially many outcomes (Y's) and many exposures (A's) of interest, as well as potentially high dimensional W.

3.2.2 Estimators

Two general classes of estimators are available for estimating ψ: plug-in maximum likelihood-type estimators and estimating equation estimators. In the **multiPIM** package, we have implemented two estimators from each class. The estimator to be used is specified by supplying the `estimator` argument to the `multiPIM` or `multiPIMboot` functions.

Estimating equation approaches

The estimating equation estimators available in the **multiPIM** package are the inverse-probability-of-censoring-weighted (IPCW) estimator, and its doubly-robust extension (DR-IPCW). These estimators are derived in Hubbard and van der Laan (2008), and the derivations are based on the approach described in van der Laan and Robins (2003).

Let $O_1, O_2, \ldots, O_n$ be independent and identically distributed observations of O, with $O_i = (Y_i, A_i, W_i)$, $i \in \{1, 2, \ldots, n\}$. An IPCW estimator is given by

$$\hat{\psi}_n^{IPCW} = \frac{1}{n}\sum_{i=1}^{n}\left[\left(\frac{I(A_i = 0)}{g_n(0|W_i)} - 1\right) Y_i\right], \tag{3.2}$$

and the corresponding DR-IPCW estimator is given by

$$\hat{\psi}_n^{DR-IPCW} = \frac{1}{n}\sum_{i=1}^{n}\left[\left(\frac{I(A_i = 0)}{g_n(0|W_i)} - 1\right) Y_i - \left(\frac{I(A_i = 0)}{g_n(0|W_i)} - 1\right) Q_n(0, W_i)\right]. \tag{3.3}$$

$g_n(0|W)$ and $Q_n(0, W)$ are estimates of the nuisance parameters, $g(0|W)$ and $Q(0, W)$, respectively. $g(a|W)$ is the probability of having treatment level $A = a$ given covariates W (also known as the treatment mechanism or propensity score), and thus $g(0|W)$ is the probability of being in the target treatment group (the group defined by $A = 0$) given observed covariates W, or $P(A = 0|W)$. Similarly, $Q(a, W)$ is the mean value of the outcome, Y, given treatment level $A = a$ and covariates W, and thus $Q(0, W)$ is the mean value of Y, given treatment level $A = 0$, and given the observed level of covariates W, or $E[Y|A = 0, W]$.

These nuisance parameters usually need to be estimated from the data. Since A is binary, $g(0|W)$ can be estimated using some form of regression with which one can predict the class probabilities for a binary outcome. The estimate, $g_n(0|W)$, is taken as the predicted probability of being in the class given by $A = 0$, for a subject with covariates W. $Q(0, W)$ can be estimated by regressing Y on A and W. The estimate, $Q_n(0, W)$, can be found by using this regression model to predict on a new data set for which every element of A is set to zero, but W stays the same. The type of regression which should be used for building this model depends on whether Y is a binary or continuous variable.

Plug-in estimators

multiPIM also makes available two plug-in estimators: the graphical computation (G-computation) estimator (Robins, 1986, 2000; van der Laan and Rubin, 2006), and the targeted maximum likelihood estimator (TMLE, van der Laan and Rubin, 2006; van der Laan and Rose, 2011).

The G-computation estimator is given by

$$\begin{aligned}\hat{\psi}^{G-COMP} &= \hat{E}[Y_0] - \hat{E}[Y] \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[Q_n^0(0, W_i)\right] - \bar{Y}.\end{aligned}$$

It is referred to as `"G-COMP"` in the package (e.g., `estimator = "G-COMP"`). Greenland and Drescher (1993) proposed an estimator that would encompass this parameter in the context of parametric logistic regression.

For the case of continuous Y, the TMLE is given by

$$\begin{aligned}\hat{\psi}^{TMLE} &= \hat{E}[Y_0] - \hat{E}[Y] \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[Q_n^1(0, W_i)\right] - \bar{Y} \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[Q_n^0(0, W_i) + \hat{\epsilon}Z(0, W)\right] - \bar{Y}.\end{aligned}$$

Here, $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$, $Q_n^0(0, W)$ is an initial estimate of $Q(0, W)$, and $Q_n^1(0, W)$ is an updated estimate that has targeted bias reduction for ψ. The updating is done by fitting a linear regression: Y is regressed on a "clever covariate" with no intercept and with $Q_n^0(A, W)$ (the fitted values from the model used for $Q(0, W)$) as offset (Rose and van der Laan, 2011a).

The clever covariate is given by

$$Z(A, W) = \frac{I(A = 0)}{g_n(A|W)}$$

and $\hat{\epsilon}$ is the estimated coefficient for this covariate.

For the case of binary Y, the updating is done by logistic regression, the offset is $\text{logit}(Q_n^0(A, W))$, and

$$\hat{E}[Y_0] = \frac{1}{n}\sum_{i=1}^{n} \text{expit}\Big(\,\text{logit}\big(Q_n^0(0, W_i)\big) + \hat{\epsilon}Z(0, W)\Big)\,.$$

where expit refers to the inverse logit function:

$$\text{expit}(\alpha) = \text{logit}^{-1}(\alpha) = \frac{1}{1 + e^{-\alpha}}.$$

An article about another R package for targeted maximum likelihood estimation, **tmle**, has recently appeared (Gruber and van der Laan, 2012). This package is also available on CRAN, and implements a TMLE for the marginal additive effect of a binary point treatment. It also calculates estimates of the risk ratio and odds ratio for binary outcomes, and can be used to estimate controlled direct effects and the parameter of a marginal structural model.

3.2.3 Properties of the estimators

The IPCW estimator will be consistent if $g(0|W)$ is estimated consistently, and the G-Computation estimator will be consistent if $Q(0, W)$ is estimated consistently. The DR-IPCW estimator and the TMLE are both based on the efficient influence curve for the semiparametric model, and as a result, they have the double robustness property (meaning that that they will be consistent if either $g(0|W)$ or $Q(0, W)$ is estimated consistently), and they are asymptotically locally efficient (Hubbard and van der Laan, 2008; van der Laan and Robins, 2003; van der Laan and Rose, 2011). As a plug-in estimator, the TMLE has the advantage that the parameter estimate is guaranteed to fall in the natural range, assuming that an appropriate estimate of $Q(0, W)$ is used. For additional advantages of TMLE over other estimators, see Rose and van der Laan (2011b).

3.2.4 Super learner and recommendations for estimation of nuisance parameters

Since the consistency of the estimators is dependent upon consistent estimation of the nuisance parameters, it is worthwhile to devote some statistical and computational effort to this estimation. Thus, the **multiPIM** package makes use of the super learner (Sinisi *et al.*, 2007; van der Laan *et al.*, 2007; Polley *et al.*, 2011; see also Breiman, 1996). The super learner is a data-adaptive meta learner which can be used to do prediction of binary or continuous outcomes. The form of the super learner implemented in the **multiPIM** package uses v-fold cross-validation to select the best from among a set of "candidate learners" (Sinisi *et al.*, 2007). This form is also known as the "discrete" super learner. In the more recent super learner algorithm, the final predictor is a weighted combination of the learners in the "library" (Polley *et al.*, 2011). The candidate learners may normally be any arbitrary regression method or machine learning algorithm which maps the values of a set of predictors into an outcome value. In the **multiPIM** package, we have implemented a small set of candidates. Most of these rely on separate R packages, which are also available on CRAN. The user can select from among these candidates in building a super learner to estimate $g(0|W)$ or $Q(0, W)$. This will be described in greater detail in Section 3.3.

Theoretical results have shown that, asymptotically, the super learner performs as well as the so-called oracle selector, which always chooses the best-performing candidate (the level of performance of a candidate is measured by a specific loss function; Sinisi *et al.*, 2007; van der Laan *et al.*, 2007). Since it is usually the case that the data-generating distributions are unknown, combining many different candidates using a super learner (and thereby searching a very large model space) should reduce the bias of $g_n(0|W)$ and $Q_n(0, W)$. Thus we recommend using the super learner for estimation of nuisance parameters. However, there is an exception to this rule when using the TMLE. As a consequence of the bias reduction step (the updating of $Q_n(0, W)$ with the clever covariate), using a very agressive algorithm to estimate $g(0|W)$ may cause empirical ETA violations, which could result in biased parameter estimates (Petersen *et al.*, 2011). Thus super learning is not recommended for estimating $g(0|W)$ when using the TMLE. Since TMLE is the default estimator in the **multiPIM** package, we have set the default method for estimating $g(0|W)$ to be main terms logistic regression (see Section 3.3).

A more elegant solution to this problem is to use a sequence of increasingly non-parametric estimates of $g(0|W)$. This is the collaborative targeted maximum likelihood estimator (van der Laan and Gruber, 2010; Gruber and van der Laan, 2011). However, this estimator has not yet been implemented for the population intervention parameter we are using here.

3.2.5 Inference

One can show that the four estimators described above are asymptotically linear, with asymptotically normal sampling distributions under very general conditions (van der Laan and Robins, 2003; Zheng and van der Laan, 2011). Two methods of estimating the variance are available in the **multiPIM** package. The "plug-in" estimates are based on the influence curve (van der Laan and Robins, 2003; this method is not available for the G-Computation estimator). Specifically, if $IC(O; \hat{P})$ is the plug-in estimate of the influence curve (where estimates of the relevant parts of P_0 are represented by $\hat{P}$), then the plug-in standard error is

$$\hat{\sigma}^{plug-in} = \sqrt{\frac{\hat{Var}(IC(O;\hat{P}))}{n}}.$$

For example, note from Equation 3.2 and Equation 3.3 in Section 3.2.2, that the IPCW and DR-IPCW estimators are both expressed as means over a certain vector. To get the plug-in standard error for each of these estimators, take the standard deviation over this vector instead of the mean, and divide by $\sqrt{n}$.

The preferred method for estimating the variance is to use the bootstrap (Efron, 1979). The bootstrap method is more robust, however it of course requires much more computation time. We note that the plug-in (influence curve-based) estimates of the variance for the IPCW estimator in particular tend to be overly conservative (van der Laan *et al.*, 2003).

3.3 The multiPIM R Package

The **multiPIM** package consists of two main functions, two methods, and four character vectors. The `multiPIM` function provides the main variable importance analysis functionality. The `multiPIMboot` function can be used to bootstrap the `multiPIM` function and get bootstrap standard errors for the parameter estimates. There is a summary method for the class `"multiPIM"` objects which are returned by the functions `multiPIM` and `multiPIMboot`, and a print method for the summary objects. Finally, the elements of the four character vectors (`all.bin.cands`, `default.bin.cands`, `all.cont.cands` and `default.cont.cands`) represent the regression methods/machine learning algorithms which are available for estimating the nuisance parameters $g(0|W)$ and $Q(0,W)$ (see Section 3.3.5 and Section 3.3.6), and are meant to be passed in as arguments to `multiPIM` and `multiPIMboot`.

3.3.1 Input and output

The arguments to the `multiPIM` function are as follows:

```
multiPIM(Y, A, W = NULL,
         estimator = c("TMLE", "DR-IPCW", "IPCW", "G-COMP"),
         g.method = "main.terms.logistic", g.sl.cands = NULL,
         g.num.folds = NULL, g.num.splits = NULL,
         Q.method = "sl", Q.sl.cands = "default",
         Q.num.folds = 5, Q.num.splits = 1,
         Q.type = NULL,
         adjust.for.other.As = TRUE,
         truncate = 0.05,
         return.final.models = TRUE,
         na.action,
         check.input = TRUE,
         verbose = FALSE,
         extra.cands = NULL,
         standardize = TRUE,
         ...)
```

The main input to the `multiPIM` function is in the form of three data frames: `W`, `A` and `Y`. Each of these data frames may contain multiple columns. The data frame `A` should contain binary (0/1) exposure variables, and `Y` should contain outcome variables. `W` is optional. If supplied, it should contain covariate variables which the user wishes to include in the adjustment set, but for which he/she is not interested in estimating a variable importance. The function calculates one estimate of ψ for each exposure-outcome pair. That is, if $A^{(j)}$ is the *jth* of J exposure variables (i.e., the *jth* of J columns of `A`) and if $Y^{(k)}$ is the *kth* of K outcome variables (i.e., the *kth* of K columns of `Y`), then JK estimates, $\hat{\psi}_{j,k}$, of ψ will be calculated, one for each pair $(A^{(j)}, Y^{(k)})$ such that $j \in \{1, 2, \ldots, J\}$ and $k \in \{1, 2, \ldots, K\}$.

3.3.2 Adjustment Set and the `adjust.for.other.As` argument

With the `adjust.for.other.As` argument, the user can control which variables are kept in the adjustment set in calculating the estimate, $\hat{\psi}_{j,k}$, for each pair, $(A^{(j)}, Y^{(k)})$. If `adjust.for.other.As` is `TRUE`, the other columns of the data frame `A`, i.e., all $A^{(j^*)}$ such that $j^* \neq j$, will be included in the adjustment set. That is, in the notation of Section 3.2, they will be included as members of W, and thus will be included, along with the columns of the data frame `W`, as possible covariates to select from in building the models from which $g(0|W)$ and $Q(0, W)$ are estimated. If `adjust.for.other.As` is set to `FALSE`, the other columns of `A` will not be included in the adjustment set. In this case, the data frame `W` must be supplied. If `W` is supplied, the variables it contains will be included in the adjustment set for all models, no matter the value of `adjust.for.other.As`.

3.3.3 Rebuilding of models used to estimate $Q(0, W)$

When **multiPIM** is run, it builds only one model per exposure variable (column of `A`), from which $g(0|W)$ is estimated. The estimate of $g(0|W)$ based on each model is then used in the calculation of each of the parameter estimates which involves the corresponding exposure variable. However, this is not the case for the models from which $Q(0, W)$ is estimated. The model for a specific outcome variable is rebuilt each time the effect of a new exposure variable is being calculated. One property of some of the machine learning algorithms used as candidates for the super learner is that they may drop certain covariate variables from the model. In order to ensure the smoothness of the sampling distribution of the estimator, the relevant exposure variable is forced back in to any model for $Q(0, W)$ from which it has been dropped (see Section 3.3.5 and Section 3.3.6 and see also Zheng and van der Laan, 2011). Thus, one such model must be built per exposure-outcome pair.

3.3.4 Implementation of super learner

As mentioned in Section 3.2.4, the preferred method for estimating the nuisance parameters is to use a super learner with many candidates (with the exception of estimating $g(0|W)$ when using the TMLE). The implementation of the super learner in the **multiPIM** package uses v-fold cross-validation to select the best from among a set of candidate learners. All exposure variables in `A` must be binary, and thus only binary outcome regression methods are implemented for use in building a super learner to estimate $g(0|W)$. However, the outcome variables in `Y` may be binary or continuous, and thus some continuous outcome regression methods/machine learning algorithms are implemented as well, in order to be used in a super learner for estimating $Q(0, W)$. The performance of candidates in a binary outcome super learner is evaluated using the negative log likelihood loss function. The performance of candidates in a continuous outcome super learner is evaluated using the mean squared error loss function. The following two sections describe the default candidate algorithms which have been implemented.

3.3.5 Default binary outcome candidates

The default binary outcome super learner candidates are given by the vector `default.bin.cands`:

```
default.bin.cands <- c("polyclass", "penalized.bin", "main.terms.logistic")
```

The point of making this vector, and the vectors `all.bin.cands`, `default.cont.cands`, and `all.cont.cands` available to the user is so that they (or subsets of their elements) may be passed to the `multiPIM` and `multiPIMboot` functions as the arguments `g.method`, `Q.method`, `g.sl.cands` or `Q.sl.cands`. This is the mechanism whereby the user may specify which regression methods should be used in building models to estimate $g(0|W)$ and $Q(0, W)$.

Polyclass candidate

This super learner candidate uses the function `polyclass` from the R package **polspline** (Kooperberg *et al.*, 1997; Stone *et al.*, 1997; Kooperberg, 2010). `polyclass` fits linear splines and their tensor products using a model selection process guided by the Akaike information criterion (Kooperberg *et al.*, 1997).

Since there is a possibility with this algorithm that certain variables may be dropped from the model, the implementation is slightly different for the case when this candidate is being used to estimate $Q(0, W)$ vs. when it is being used to estimate $g(0|W)$. In order to make sure that the relevant exposure variable stays in the model when estimating $Q(0, W)$ (see Section 3.3.3), the object returned by `polyclass` is inspected to see if the relevant variable is a member of the basis functions selected. If it is not, a second, logistic regression model is fit using the predictions from the `polyclass` model as a covariate, along with the relevant exposure variable which was dropped from the `polyclass` model. In order to stay somewhat flexible, an interaction term between the exposure variable and the predictions from `polyclass` is also included in this logistic model. Thus, this secondary logistic model has the form

$$\text{logit}\left(Q_k^*(A^{(j)}, W^*)\right) = \beta_0 + \beta_1 A^{(j)} + \beta_2 \,\text{logit}\left(\hat{Q}_k^0(A^{(j)}, W^*)\right) + \beta_3 A^{(j)} \,\text{logit}\left(\hat{Q}_k^0(A^{(j)}, W^*)\right),$$

where W^* depends on the value of the `adjust.for.other.As` argument, $\hat{Q}_k^0(A^{(j)}, W^*)$ are the predictions from a `polyclass` model for which the outcome was $Y^{(k)}$, and the β's are regression coefficients. Note that this model contains the original `polyclass` model as a submodel (just set $\beta_2 = 1$ and $\beta_0, \beta_1, \beta_3 = 0$).

After both the polyclass and the logistic model have been fit, prediction on a new data set is done by first getting the `polyclass` model's predictions on this new data, and then predicting on these new predictions using the logistic model.

Penalized candidate

The second binary outcome candidate, named `"penalized.bin"` to distinguish it from the continuous outcome version, is based on the `penalized` function from the `R` package **penalized** (Goeman, 2010, 2011). This function performs regressions with either L1 (lasso) or L2 (ridge) penalties, or with a combination of the two. The implementation of this candidate in the `multiPIM` function uses only an L1 penalty. The value of the penalty is chosen using the `profL1` function, also from package **penalized**. With `profL1`, cross-validation is carried out on ten possible values of the L1 penalty. The values range from zero to the minimum value which would cause all coefficients to shrink to zero.

When estimating $g(0|W)$, all columns of `W` (and, depending on `adjust.for.other.As`, possibly all other columns of `A` besides the one actually being modeled) are put into the `penalized` model as penalized main terms. However, when estimating $Q(0, W)$, in order to prevent shrinking of its coefficient to zero, the relevant exposure variable is added to the model as an unpenalized covariate.

Main terms logistic candidate

The final binary outcome candidate uses the function `glm` from the package **stats** (`R` Core Team, 2013) to build a main terms logistic regression model. Since `glm` does not drop any covariates, it is not necessary to change the implementation for when $Q(0, W)$ is being estimated vs. when $g(0|W)$ is being estimated.

3.3.6 Default continuous outcome candidates

The default continuous outcome super learner candidates are given by the vector `default.cont.cands`:

```
default.cont.cands <- c("polymars", "lars", "main.terms.linear")
```

Since the exposure variables are always binary, continuous outcome candidates are always used for estimating $Q(0, W)$, and never for $g(0|W)$. Thus, they must always allow for the forcing of variables into the model.

Polymars candidate

The polymars candidate uses the `polymars` function, from package **polspline**. This function is similar to the `polyclass` function, but it can be used for modeling a continuous outcome instead of a categorical outcome. Another difference between the two functions is that the `polymars` function has a mechanism for forcing specific variables into the model. This mechanism is used in the polymars candidate, and thus no extra regression model needs to be built in order to force the exposure variable back into the model in case it is dropped.

Lars candidate

This candidate is based on the function `lars` from the package **lars** (Efron *et al.*, 2004; Hastie and Efron, 2011). `lars` performs least angle regression, a variant of the lasso. For the `lars` candidate, the function `cv.lars` is used to cross-validate on a grid of possible points on the solution path. If the final `lars` model has a coefficient of 0 for the relevant exposure variable, a secondary linear regression model is fit using the exposure variable and the predictions from the `lars` model as covariates, similarly to the logistic regression model described above for the `polyclass` candidate (see Section 3.3.5).

Main terms linear candidate

The `"main.terms.linear"` candidate uses the function `lm` from the package **stats** to fit a main-terms-only linear regression model. Again, since no covariates are dropped by this method, no secondary forcing model is necessary.

3.3.7 Alternative regression methods and other user options

In addition to the default candidates mentioned above, there are several optional candidates which can be added to super learners. For both binary and continuous outcomes, there is a candidate based on the `rpart` function from package **rpart** (Therneau *et al.*, 2010; Breiman, 1984). There is also a continuous outcome version of the penalized candidate. The user of the `multiPIM` function may also use any of the super learner candidates mentioned as a stand-alone regression method for estimating $g(0|W)$ or $Q(0, W)$. This can be done by specifying the `g.method` or `Q.method` arguments as the the name of the desired candidate. The user may also supply one or more arbitrary, self-implemented, regression methods, either to add to a super learner as candidates, or to use as stand-alone regression methods. This makes it possible to use the more recent version of the super learner, which has been implemented in the CRAN package **SuperLearner** (Polley and van der Laan, 2011). Another recommended learner which can be added as a user-supplied candidate/method is the Deletion/Substitution/Addition algorithm (the **DSA** package, Neugebauer and Bullard, 2010; Sinisi and van der Laan, 2004). The mechanism for specifying user-supplied candidates/methods is via the `extra.cands` argument and is fully documented in the Candidates help file:

```
> ?Candidates
```

The user may also choose:

- How many "folds" and splits to use for the v-fold cross-validation in the super learner (the defaults are 5 and 1, respectively)
- Whether and at which value to truncate (from below) $g_n(0|W)$ in order to avoid instability of the estimator (the default is to truncate at 0.05)

3.3.8 `multiPIMboot` function

The `multiPIMboot` function can be used instead of the `multiPIM` function when the user wishes to use bootstrapping to calculate standard errors. `multiPIMboot` will run `multiPIM` once on the original data set, then sample with replacement from the rows of the data and rerun `multiPIM` the desired number of times on resampled data sets. All arguments to `multiPIM` are also available for `multiPIMboot`, and these have the same defaults. Additional arguments are:

- `times`, for specifying the number of bootstrap replicates
- `id`, to identify clusters and perform a clustered bootstrap
- `multicore`, `mc.num.jobs`, and `mc.seed` for running the bootstrap on multiple processor cores from a single R session. This requires the **parallel** package, which is distributed with R as of version 2.14.0. Previous versions of **multiPIM** relied on the **multicore** and **rlecuyer** packages for this functionality (Urbanek, 2011; Sevcikova and Rossini, 2009).

Based on our experience with two different quad-core systems, the factor of speedup when `multicore = TRUE` is close to the number of (physical) cores used, which is not surprising since bootstrapping is embarrassingly parallel. Note that the single run of `multiPIM` on the original data is done first in serial, before **multicore**'s `parallel` function sends multiple bootstrap jobs to the cores. Thus the CPU usage will not hit 100% of all cores until the first run is complete.

In order to improve reproducibility of parallel runs when `multicore = TRUE`, the random number generator type is automatically set using

```
> RNGkind("L'Ecuyer-CMRG")
```

This causes R to use an implementation of the generator described by L'Ecuyer *et al.* (2002), which allows using different and reproducible streams within each parallel thread of execution.

3.3.9 Statistical recommendations and effects on computation time

Most of the computation time spent when running the `multiPIM` function goes into fitting the models from which $g(0|W)$ and $Q(0, W)$ are estimated. Also, assuming the same options are used, the `multiPIMboot` function will take much longer to run than the `multiPIM` function, since it just repeatedly calls the `multiPIM` function. Thus, three very effective ways of reducing the computation time are to:

1. Use plug-in standard errors by running `multiPIM` instead of `multiPIMboot`
2. Use the IPCW estimator, which requires estimating only $g(0|W)$, and not $Q(0, W)$ (unlike the two double robust estimators, TMLE and DR-IPCW)
3. Do not use super learning, but instead use methods which require very little computation time, such as main terms linear and main terms logistic regression

However, for maximal robustness and accuracy of inference, it is recommended to use multiPIMboot with the default arguments (bootstrap standard errors, TMLE estimator, super learning to estimate $Q(0, W)$). The multicore functionality has been added as a way to reduce the time required to run this full bootstrap analysis.

Using the bootstrap instead of plug-in standard errors will have the greatest effect on running time. If bootstrap is just not feasible, it is recommended to run `multiPIM` with the TMLE estimator. Additional fine tuning of the running time can be done by using the `summary` method (see next section) to see which super learner candidates are using the most computation time. The computation time of the super learner depends on which candidates are included and on the number of splits and "folds" used for cross-validation. Using a single split and 5 folds should be adequate in most cases, but it may be safer to use a higher number of folds, such as 10, especially if the data set has only a few hundred observations. Increasing the number of splits beyond one may also improve the accuracy of the cross-validation candidate selection mechanism (Molinaro *et al.*, 2005).

3.3.10 Summary method and example

Both the `multiPIM` and the `multiPIMboot` functions return objects of class `"multiPIM"`. We have written a summary method which can be called on these objects in order to generate numerical summaries of the statistical results (as well as a breakdown of where the computation time was spent). The method uses the parameter estimates and standard errors to calculate test statistics and unadjusted as well as Bonferroni-adjusted p values, and allows for easy and customizable printing of tables showing these results. We demonstrate this by running an example which has been adapted from the help file for the `multiPIM` function:

```
R> library("multiPIM")
R> num.columns <- 3
R> num.obs <- 250
R> set.seed(23)
```

We generate a data frame containing random binary data to use as exposure variables:

```
R> A <- as.data.frame(matrix(rbinom(num.columns*num.obs, 1, .5),
+                            nrow = num.obs, ncol = num.columns))
```

Next we generate outcomes based on the exposures, by starting with random noise and adding multiples of the exposure variables:

```
R> Y <- as.data.frame(matrix(rnorm(num.columns*num.obs),
+                            nrow = num.obs, ncol = num.columns))
R> for(i in 1:num.columns)
+    Y[, i] <- Y[, i] + i * A[, i]
```

Next we make sure that the two data frames have unique names, then run `multiPIM` on them and run `summary` on the resulting object:

```
R> names(A) <- paste("A", 1:num.columns, sep = "")
R> names(Y) <- paste("Y", 1:num.columns, sep = "")
R> result <- multiPIM(Y, A)
R> summary(result)

The call was:

multiPIM(Y = Y, A = A)

Results for the exposure "A1" vs the outcomes listed on the left:

outcome param.estimate stand.error test.stat     p.val p.val.bon.adj
     Y1       -0.57198     0.08182    6.9911 2.727e-12     2.455e-11
     Y2        0.02413     0.06810    0.3544 7.231e-01     1.000e+00
     Y3       -0.03655     0.06482    0.5639 5.728e-01     1.000e+00
```

```
Results for the exposure "A2" vs the outcomes listed on the left:

outcome param.estimate stand.error test.stat     p.val p.val.bon.adj
     Y1      -0.008016     0.06724    0.1192 9.051e-01     1.000e+00
     Y2      -0.935624     0.08708   10.7444 6.298e-27     5.668e-26
     Y3      -0.029830     0.05764    0.5176 6.048e-01     1.000e+00

Results for the exposure "A3" vs the outcomes listed on the left:

outcome param.estimate stand.error test.stat     p.val p.val.bon.adj
     Y1        0.04203     0.07485    0.5615 5.744e-01     1.000e+00
     Y2        0.11055     0.06871    1.6090 1.076e-01     9.685e-01
     Y3       -1.57871     0.11371   13.8832 8.012e-44     7.211e-43


Total time for main multiPIM run:

2.002508 seconds

Breakdown by g vs. Q modeling:

                        method seconds %.of.total
g modeling main.terms.logistic 0.01794     0.8958
Q modeling                  sl 1.94518    97.1374

Total time for Q model super learner cross validation (x-val):

1.804895 seconds

Breakdown by candidate:

                  seconds %.of.Q.x-val.time
polymars           0.3634            20.132
lars               1.2915            71.553
main.terms.linear  0.1358             7.522
```

Notice that for the pairs `A[, i]` vs. `Y[, i]`, `i` $=$ 1 to 3, the adjusted p values get progressively lower, since `Y[, i]` is `i * A[, i]` plus noise. However, off-diagonal p values are higher since there is no dependence of `Y[, i]` on `A[,  j]` when `i` $\neq$ `j`. There is a corresponding trend in the actual parameter estimates, which get progressively more negative for the diagonal (`A[, i]` vs. `Y[, i]`, `i` $=$ 1 to 3) exposure-outcome pairs.

The breakdown of the computation time shows that most of the time goes into the super learning for the Q model, most of this Q modeling time is being spent on the lars candidate.

3.4 Simulation

In order to demonstrate the benefit of using a double robust estimator, we compared the TMLE to the G-Computation estimator in a simulation. The complete R script which runs the simulation can be found in the supplements to this paper.

3.4.1 Simulated Data

The data consisted of four covariate variables $W = (W^{(1)}, W^{(2)}, W^{(3)}, W^{(4)})$, a single exposure variable A and a single outcome variable Y. W was generated as multivariate normal with mean vector $\mu = (0, 0, 0, 0)$ and covariance matrix

$$\Sigma = \begin{pmatrix} 1 & 0.2 & 0.2 & 0.2 \\ 0.2 & 1 & 0.2 & 0.2 \\ 0.2 & 0.2 & 1 & 0.2 \\ 0.2 & 0.2 & 0.2 & 1 \end{pmatrix}$$

A was generated from a logistic model based on W, with

$$A_i \sim \text{Bernoulli}(\text{expit}(\beta W_i^T)),$$

where $\beta = (0.2, 0.2, 0.2, 0.2)$ and $W_i = (W_i^{(1)}, W_i^{(2)}, W_i^{(3)}, W_i^{(4)})$. We made the data generating model for Y more complex:

$$Y_i = W_i^{(1)} W_i^{(2)} + W_i^{(3)} W_i^{(4)} + A_i \left[(W_i^{(1)})^2 + (W_i^{(2)})^2 + (W_i^{(3)})^2 + (W_i^{(4)})^2 \right] + \text{error},$$

with the errors i.i.d. Normal$(\mu = 0, \sigma^2 = 4)$, and with $(W_i^{(j)})^2$ equal to the square of $W_i^{(j)}$, for $j = 1, 2, 3, 4$.

Data sets were generated with sample sizes, n, evenly spaced on the log scale between 100 and 250,000, with one data set per value of n.

3.4.2 Estimators

To get estimates of ψ, we ran the `multiPIM` function twice on each data set, once with `estimator = "TMLE"` and once with `estimator = "G-COMP"`. Nuisance parameters were estimated using main terms logistic regression for $g(0|W)$ (TMLE only) and main terms linear regression for $Q(0, W)$. Thus the model for $g(0|W)$ was correctly specified, while the model for $Q(0, W)$ was misspecified.

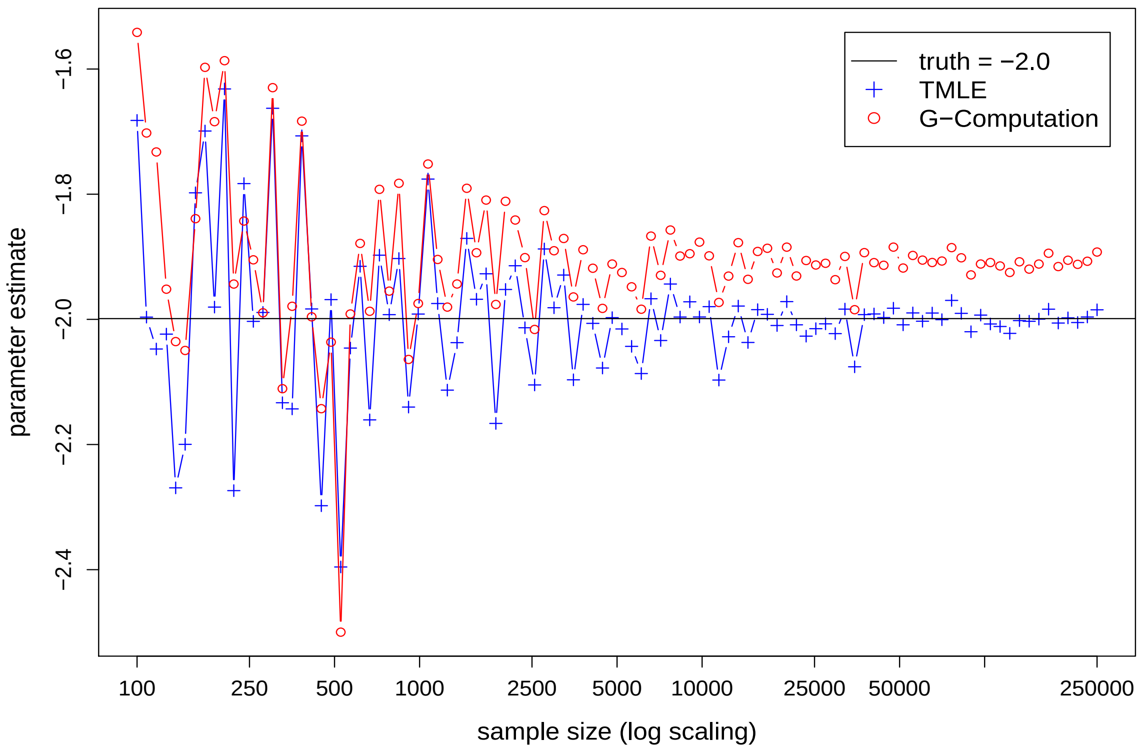

Figure 3.1: Simulation results. The TMLE estimates are centered around the true parameter value, while the G-Computation estimates are slightly biased.

3.4.3 Results

The results are shown in Figure 3.1. It is clear that there is some bias towards zero in the G-Computation estimates. This is due to the misspecified model for $Q(0, W)$. The TMLE estimator uses the same biased estimates of $Q(0, W)$ as the G-Computation estimator, but thanks to the double robustness and the fact that $g(0|W)$ is being estimated using the correct model, the TMLE estimates stay centered around the true parameter value. The variability of the two estimators appears to be similar throughout the range of sample sizes.

3.5 Analysis of data from the western collaborative group study

3.5.1 Study background

The Western Collaborative Group Study (WCGS) is a prospective study on coronary heart disease (CHD) which began in 1960 (Rosenman *et al.*, 1966). The subjects were males, aged 39-59 at the start of follow-up, who were employed in several different corporations in

California. The main goal of the study was to assess a possible effect of behavior type on incidence of CHD. After psychological testing, subjects were classified as having either of two behavior types, A or B. Type A is characterized by, among other qualities, “excessive drive, aggressiveness, and ambition”, while type B is characterized by having fewer of these qualities or having them in lesser extents (Rosenman *et al.*, 1966).

3.5.2 Description of data

The data set has been included as part of the **multiPIM** package and can be loaded with

```
R> data("wcgs")
```

The data analyzed represent a follow-up experience of 8.5 years (through 1969, Rosenman *et al.*, 1975). The original subject enrollment count at the start of the study was 3524. However, subjects were later excluded for the following reasons: not being in the correct age range at intake, having CHD already manifest at intake, working for one specific corporation which pulled itself out of the study, or being lost to follow up for various reasons (Rosenman *et al.*, 1975). This left 3154 subjects whose data was available for analysis. Since 12 of these subjects had missing values for one of the variables to be used in the analysis, these 12 subjects were also removed from the analysis for a final $n = 3142$. Of these final 3142 subjects, 257 (8.2%) had a CHD event.

Variables present in this data set include baseline covariates such as height and weight, the type A/B behavior pattern, total cholesterol levels, systolic and diastolic blood pressure, and smoking history (number of cigarettes smoked per day). Also present is the binary outcome variable indicating whether a CHD event occurred within the follow-up period.

The `multiPIM` function requires the exposure variables to be binary. Thus, they were dichotomized as follows:

- **typeAB**: This is just the behavior pattern variable with type A coded as 1 (exposed), and type B coded as 0 (unexposed). Thus the target group is type B behavior, in the sense that the resulting parameter estimate, $\hat{\psi}$, can be thought of as an attributable risk which compares the level of the outcome in the entire population to the level for those with type B behavior.

- **chol**: The total cholesterol, dichotomized with a cutoff point of 240. The target group is therefore those with total cholesterol less than 240.

- **cigs**: This was coded as 1 for smokers (anyone who smokes 1 or more cigarettes per day) and 0 for non-smokers. Thus, the target group is non-smokers.

- **highBP**: Instead of having two highly correlated cholesterol variables, this single variable was coded as 1 (or exposed) if *either* diastolic blood pressure was greater than 90, or systolic blood pressure was greater than 140, and coded as 0 otherwise. Thus, the target group consists of individuals for whom neither measure is elevated.

- **bmi**: First the body mass index (BMI) was calculated by dividing weight in kg by the square of height in m^2. Then this variable was dichotomized using a cutoff point of BMI = 25. Anything greater than 25 was coded as 1 and anything less than 25 as 0. Thus, the target group is those with BMI less than 25.

These five variables were passed to the `multiPIMboot` function as the data frame `A`. Summary statistics for these exposure variables are given in Table 3.1.

	unexposed group	proportion exposed	prop. unexp. with CHD	prop. exp. with CHD	naive attr. risk
typeAB	type B	0.504	0.051	0.112	-0.0311
chol	< 240	0.343	0.057	0.129	-0.0247
cigs	< 1/day	0.476	0.060	0.106	-0.0223
highBP	D<90 & S<140	0.212	0.070	0.126	-0.0119
bmi	< 25	0.411	0.073	0.095	-0.0089

Table 3.1: Summary information for the five binary exposure variables used in the analysis. The first column states the characteristic that defines the unexposed group; D: diastolic blood pressure; S: systolic blood pressure; prop. unexp./exp. with CHD: proportion of unexposed/exposed with CHD; naive attr. risk: naive attributable risk – this is just the value in the 3rd column minus the overall disease rate of 0.082 (8.2%)

	age	typeAB	chol	cigs	highBP	bmi	chd
age	1.00	0.09	0.10	0.00	0.12	0.02	0.12
typeAB		1.00	0.02	0.06	0.07	0.03	0.11
chol			1.00	0.08	0.10	0.04	0.12
cigs				1.00	-0.02	-0.10	0.09
highBP					1.00	0.17	0.08
bmi						1.00	0.04
chd							1.00

Table 3.2: Correlation matrix for all variables used in the analysis.

Also present in the data set was a variable giving the subjects' age in years. This variable was included in the analysis by being passed to `multiPIMboot` as the single-column data frame `W`. As stated above, ages ranged from 39-59 at the start of follow-up. Table 3.2 shows the correlation matrix for all variables used in the analysis.

3.5.3 Estimator used

The `multiPIMboot` function was run using the default estimator (TMLE) and the default methods for estimating nuisance parameters: main terms logistic regression for $g(0|W)$, super learning using the default binary outcome candidates (see Section 3.3.5) for the initial estimate of $Q(0, W)$. The `adjust.for.other.As` argument (see Section 3.3.2) was also kept at its default value of `TRUE`. The call to `multiPIMboot` was made as follows (for complete script see the supplements):

```
R> boot.result <- multiPIMboot(Y, A, W, times = 2000, multicore = TRUE,
+                              mc.num.jobs = 8, verbose = TRUE)
```

The elapsed time for running this job on a quad core iMac was about 4.6 hours.

3.5.4 Results

Table 3.3 shows the results of running the `multiPIMboot` function on the WCGS data. Three of the five exposure variables were found to have a significant effect on the CHD outcome after Bonferroni adjustment of p values. These were: A or B behavior type, cholesterol level, and cigarette smoking status. Since the outcome is binary, the parameter estimates (first column of Table 3.3), are on the scale of a proportion. For example, if one accepts the validity of the causal assumptions enumerated in Section 3.2.1, then the parameter estimate for the variable typeAB implies that if type A behavior were eliminated from the population and everyone was made to have behavior type B, the incidence of CHD would be reduced by about 2.7 percentage points (recall that 8.2% of the sample had an incident CHD event).

	$\hat{\psi}^{TMLE}$	$\hat{\sigma}^{plug-in}$	$\hat{\sigma}^{boot}$	T	p	$p^{Bonferroni}$
typeAB	-0.0271	0.0051	0.0051	5.33	9.66E-08	4.83E-07
chol	-0.0201	0.0042	0.0042	4.78	1.76E-06	8.78E-06
cigs	-0.0210	0.0048	0.0049	4.28	1.90E-05	9.50E-05
highBP	-0.0074	0.0031	0.0030	2.43	0.015	0.076
bmi	-0.0055	0.0044	0.0044	1.25	0.210	1.0

Table 3.3: Results of running the `multiPIMboot` function on the WCGS data. $\hat{\psi}^{TMLE}$: parameter estimates; $\hat{\sigma}^{plug-in}$: plug-in standard errors (from the influence curve); $\hat{\sigma}^{boot}$: bootstrap standard errors; T: test statistic (calculated using $\hat{\sigma}^{boot}$); p: unadjusted, two-sided p value from comparison of T with the standard normal distribution function; $p^{Bonferroni}$: Bonferroni-adjusted p value.

Comparison of Table 3.3 with Table 3.1 shows that the adjusted parameter estimates (first column of Table 3.3) are all less than the naive estimates (final column of Table 3.1).

Some of the apparent effect is being “adjusted out.” Also, the standard errors decrease with decreasing proportion exposed. For example, the lowest bootstrap standard error, 0.0030, is for the variable highBP, for which the proportion exposed was the lowest of all five exposure variables (0.212). Since the target group is the unexposed, low proportions of exposed subjects correspond to high proportions of subjects in the target group, and since the groups being compared are the entire sample vs. the target group, it makes sense that high proportions in the target group correspond with low standard errors.

It is also interesting to note that the maximum off-diagonal value in the correlation matrix (Table 3.2), which was 0.17, corresponds to the correlation between the variables bmi and highBP. These are the two variables for which the results were insignificant after Bonferroni adjustment. It is possible that the effect measurements for each of these two variables may have been diluted since the other of the two was also included in the adjustment set. Also, it seems promising that the sum of the (adjusted) parameter estimates (0.0811) is approximately equal to the overall disease rate of 8.2%.

Figure 3.2 and Figure 3.3 provide some evidence for the validity of the bootstrap variance estimation procedure. The cumulative standard errors plotted in Figure 3.2 all appear to converge well before the 2000th bootstrap replicate, and the histograms in Figure 3.3 indicate that there were no glaring irregularities in the bootstrap distributions of the parameter estimates.

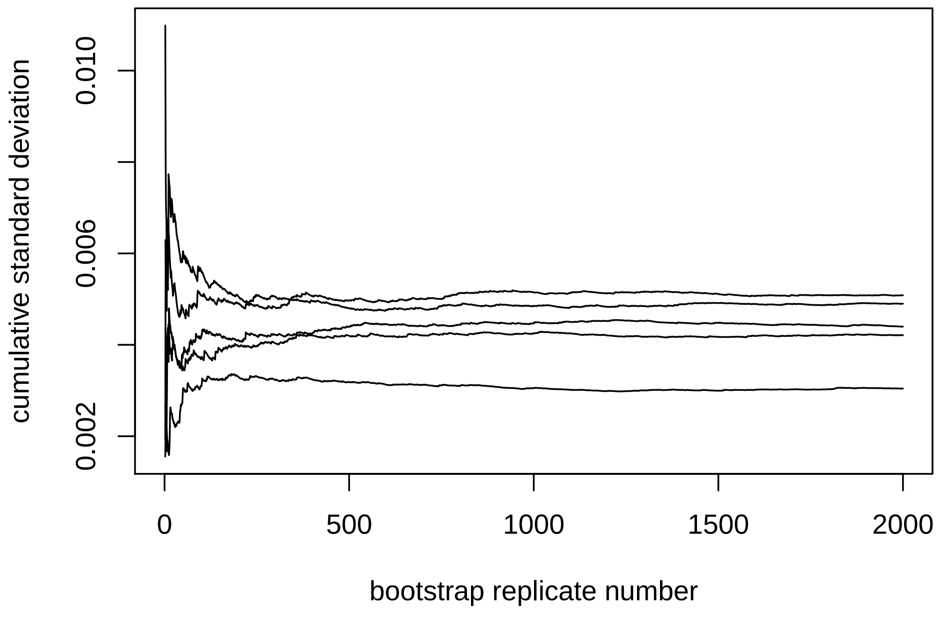

Figure 3.2: Cumulative standard deviations of the bootstrap parameter estimates for all five exposure variables.

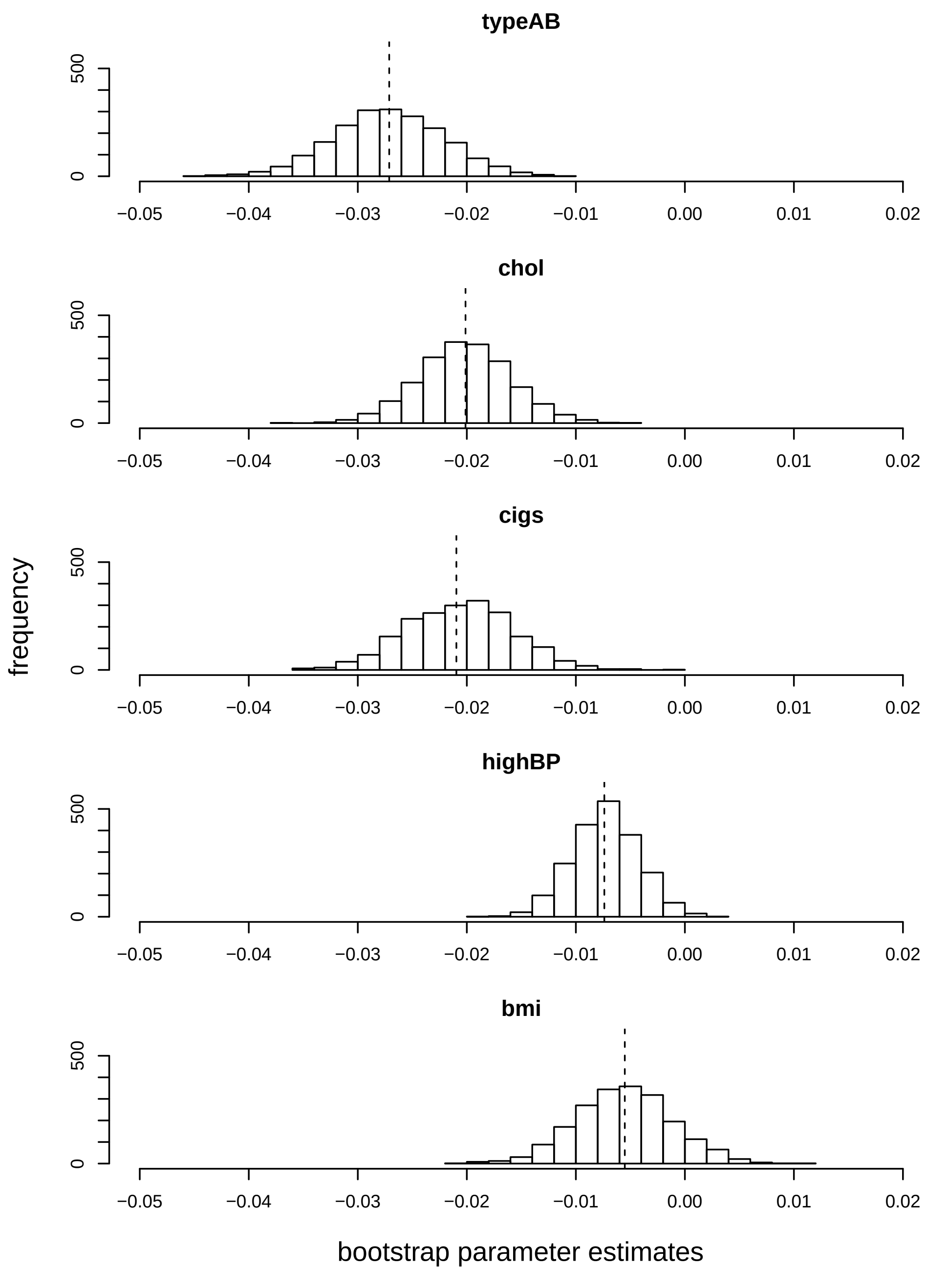

Figure 3.3: Histograms showing bootstrap distributions of parameter estimates for each exposure variable. Dashed vertical lines show actual parameter estimates. There were 2000 bootstrap replicates.

3.6 Study on water contact and schistosomiasis infection

In this section we provide more example code, using data which was collected as part of the study by Spear *et al.* (2004), and revisited by Sudat *et al.* (2010). This data has also been included as part of the **multiPIM** package and can be loaded with

```
R> data("schisto")
```

3.6.1 Study background

The study was conducted in a rural area in Sichuan Province, China, to which schistosomiasis is endemic. In November, villagers from 20 villages surrounding Qionghai Lake were interviewed about their activities over the past 7 months that brought them into contact with water. At the end of the infection season, stool sampling and analysis were carried out to determine which of the villagers had become infected.

3.6.2 Description of data

The `schisto` data frame contains the following columns:

- Outcome variable
  - **stoolpos**: 1 indicates infected, 0 indicates uninfected
- Exposure variables: these record the amount of water exposure of each subject with respect to doing the activities listed, during the 7 month period from April through October
  - **laundwc**: washing clothes or vegetables
  - **toolwc**: washing agricultural tools
  - **bathewc**: washing hands and feet
  - **swimwc**: playing or swimming
  - **ditchwc**: irrigation ditch cleaning and water diverting
  - **ricepwc**: planting rice
  - **fishwc**: fishing
- Covariates
  - **village**: a label for the village
  - **age.category**: 5 different age categories (< 18, 18-29, 30-39, 40-49, 50+)

We prepare the `W`, `A` and `Y` data frames:

```
R> W <- data.frame(lapply(schisto[, 9:10], as.factor))
R> A <- schisto[, 2:8]
```

`Y` will have only one column, so we need to use `drop = FALSE` to prevent it from being turned into a vector:

```
R> Y <- schisto[, 1, drop = FALSE]
```

3.6.3 The analysis

If one wishes to keep the information contained in continuous exposure variables when using them in the adjustment set, multiPIM can be called inside a for loop, with the other columns of `A` included as part of `W`.

To cut down on running time, we use the IPCW estimator and the rpart candidate as a stand-alone for `g.method`. This analysis is similar to that performed by Sudat *et al.* (2010) and gives similar results. The results are shown in Table 3.4.

```
R> set.seed(23)
R> num.types <- ncol(A)
R> multiPIM.results <- vector("list", length = num.types)
R> for(i in 1:num.types) {
+    A.current <- A[, i, drop = FALSE]
+    A.current[A.current > 0] <- 1
+    W.current <- cbind(W, A[, -i])
+    multiPIM.results[[i]] <- multiPIM(Y, A.current, W.current,
+                                      estimator = "IPCW",
+                                      g.method = "rpart.bin")
+  }
R> results.tab <-
+    t(sapply(multiPIM.results,
+             function(x) summary(x, bf.multiplier = 7)$sum[1,1,]))
R> rownames(results.tab) <- names(A)
R> library("xtable")
R> print(xtable(results.tab), floating = FALSE)
```

3.7 PROMMTT trauma study

In this section we report on an analysis of data from the PRospective Observational Multi-center Major Trauma Transfusion (PROMMTT) study (Rahbar *et al.*, 2012; Hubbard *et al.*, 2013).

	param.estimate	stand.error	test.stat	p.val	p.val.bon.adj
laundwc	-0.01	0.01	0.77	0.44	1.00
toolwc	-0.03	0.01	2.81	0.00	0.03
bathewc	0.03	0.04	0.70	0.48	1.00
swimwc	-0.01	0.02	0.75	0.45	1.00
ditchwc	-0.02	0.02	0.85	0.40	1.00
ricepwc	-0.09	0.03	2.75	0.01	0.04
fishwc	0.00	0.00	0.31	0.75	1.00

Table 3.4: Results for schisto analysis. The tool washing and rice planting variables remain significant after Bonferroni adjustment.

3.7.1 Study background

Rahbar *et al.* (2012) describe the study in detail, and we will summarize here. During the period from July 2009 to October 2010, all trauma patients meeting the criteria for highest level trauma activation at 10 major US trauma centers were enrolled in the study. Pregnant patients, prisoners, and anyone younger than age 16 was excluded. There were several other criteria for exclusion, including death within the first 30 minutes after admission.

3.7.2 Description of data and the analysis

The variables used in the analysis are described in Table 3.5. The outcome variable was death, and this was considered separately for 4 distinct time intervals following admission: 30 to 90 minutes, 90 to 180 minutes, 180 to 360 minutes, or > 360 minutes. Each interval was analyzed separately, with only those patients who were still alive at the beginning of the interval being included in the analysis for that interval. As described in Hubbard *et al.* (2013), data was missing for certain variables and this was accounted for using missingness indicators.

For each variable listed as an exposure variable (A) in Table 3.5, `multiPIM` was run with the TMLE estimator, adjusting for all variables listed as W in Table 3.5. The default method (`main.terms.logistic`) was used for `g.method`, but for `Q.method`, super learning was used with several candidates. The candidates were: the binary outcome `rpart` candidate (`"rpart.bin"`), and three candidates which used the `widenet` function from the **widenet** package (see Chapter 2), with one `widenet` candidate for each value of the order of basis expansion, from 1 to 3. Each widenet candidate was run using just the single value of the order, and they were added to the `multiPIM` function using the `extra.cands` mechanism. Certain continuous exposure variables were dichotomized by taking the lower quartile as the "unexposed" group and the upper three quartiles as the "exposed." The variables which were binary vs. continuous are shown in Table 3.7.

Variables	Description	Variable Type
ageveriel	age	W
anticoag	history of anti-coagulant	W
aspirin	history of aspirin	W
bdresed	initial ed base deficit results	A
ctubeed	chest tube	A
fastresed	fast results	A
fibrresed	initial fibrinogen results (mg/dl)	A
gcsed	gcs total intial	A
glued	initia glucose	A
hctresed	hematrocrit	A
hgbresed	initial ed hemoglobin results (g/dl)	A
hred1	heart rate (hr) (initial ed)	A
inrresed	initial inr results	A
intubed	intubation	A
iss	injury severity score	A,W
ndecomed	needle decompression	A
penetrating	penetrating mechanism of injury	W
plasmasum	cumulative plasma infusions	A
pltresed	initial platelet results	A
pltsum	cumulative platelet infusions	A
racemra	race	W
RBCsum	cumulative rbc infusions	A
rfviiaed	recombinant factor viia (rfviia)	A
sbped1	systolic blood pressure	A
sexel	sex	W
sked	initial serum potassium	A
snaed	initial serum sodium	A
toured	tourniquet	A
traxred	traction-extremity traction/external fixure	A

Table 3.5: Descriptions of variables used in the PROMMTT analysis. W: covariates, A: exposure variables. Source: Hubbard *et al.* (2013).

candidate	# of times chosen by super learner
rpart.bin	11
widenetOrder1	59
widenetOrder2	20
widenetOrder3	2

Table 3.6: Number of times each candidate was chosen by the super learner for the PROMMTT analysis.

		30-90 minutes		90-180 minutes		180-360 minutes		> 360 minutes	
	binary?	estimate	*p*-value	estimate	*p*-value	estimate	*p*-value	estimate	*p*-value
bdresed		0.024	7.09E-03	-1.24E-03	0.854	0.023	0.038	0.031	0.155
ctubeed	yes	-7.22E-03	0.102	-0.011	0.052	-7.13E-04	0.799	-0.010	0.237
fastresed	yes	7.50E-04	0.821	-2.36E-03	0.597	-4.25E-03	0.300	0.012	0.151
fibrresed		-0.010	0.259	0.049	0.081	2.92E-03	0.749	0.023	0.604
gcsed		0.022	0.063	5.97E-03	0.370	0.011	0.061	0.070	5.73E-05
glued		-0.011	0.102	-8.30E-03	0.329	-0.012	0.089	-0.049	4.29E-03
hctresed		0.023	0.061	0.015	0.186	0.011	0.338	-8.27E-03	0.652
hgbresed		0.022	0.045	0.010	0.301	6.45E-03	0.484	-0.016	0.370
hred1		-2.46E-03	0.744	6.52E-03	0.500	-5.66E-04	0.946	0.015	0.484
inrresed		-0.012	0.052	-0.018	9.27E-04	-0.016	2.33E-03	-0.055	4.63E-04
intubed	yes	-5.23E-04	0.865	-6.80E-03	0.069	-0.010	6.61E-03	-8.13E-03	0.191
iss		-0.014	1.09E-03	-0.021	4.00E-05	-0.009	0.283	-0.088	5.63E-08
ndecomed	yes	-1.72E-03	0.220	6.63E-04	0.155	-9.36E-04	0.374	-9.28E-04	0.530
plasmasum		-6.89E-03	0.063	-0.020	2.31E-04	-0.019	1.85E-05	-0.037	0.024
pltresed		0.044	5.60E-05	0.014	0.104	0.012	0.115	0.029	0.145
pltsum		4.54E-04	0.788	-3.66E-03	0.335	-8.34E-03	0.049	-0.020	8.73E-03
rbcsum		-0.018	1.87E-05	-0.020	1.59E-03	-0.013	0.017	-0.030	0.025
rfviiaed	yes	1.02E-04	0.120	3.28E-04	3.89E-01	7.05E-05	0.297	-3.26E-04	0.845
sbped1		0.019	0.033	-4.99E-03	0.487	4.45E-04	0.952	-7.99E-03	0.639
sked		3.82E-04	0.954	-6.24E-03	0.356	3.92E-03	0.623	-0.020	0.181
snaed		-5.77E-03	0.191	9.00E-03	0.308	-2.52E-03	0.728	-3.08E-03	0.842
toured	yes	-4.36E-04	0.678	6.62E-04	0.014	-1.46E-03	0.322	1.66E-03	0.259
traxred	yes	-1.11E-03	0.444	1.42E-04	0.896	-1.53E-04	0.892	1.72E-03	0.521

Table 3.7: Results for variable importance analysis of PROMMTT data. The first column shows which of the exposure variables were binary and therefore did not need to be dichotomized. The following columns give the parameter estimates and p-values for the analyses of the four different time intervals. Note that the p-values are unadjusted.

3.7.3 Results

Table 3.6 shows the number of times each of the candidates was selected by the super learner. The resulting parameter estimates and p-values are shown in Table 3.7. Notable among these results is the parameter estimate for pltresed variable, or initial platelet levels. This was one of the continuous variables which was dichotomized, and it appears that there is a tendancy towards incrasing probability of death during the first time interval for higher levels of this variable. Also notable is the cummulative red blood cell infusions, rbcsum, which showed a significantly decreased probability of death during the first two time intervals.

3.8 Discussion

The issue of how data can be used to select models and how this should be reflected in the final inference is extremely important in epidemiology today. Ioannidis (2005), in discussing "why most published research findings are false", writes about the detrimental effects of "flexibility in designs, definitions, outcomes and analytical modes" (page 0698). Leaving the analyst to make arbitrary choices in the model selection process can lead to bias and distorted inference. However, while it is desirable to reduce the flexibility of the analyst to make arbitrary decisions based on the data, it is helpful to make use of automated machine learning methods which are flexible in that they search a large space of possible models.

One common complaint against the complex models which often result from machine learning methods is that they lack interpretability. The solution is the use of low dimensional parameters of interest that have meaningful public health interpretation, and are not tied directly to the model used for $E[Y|A, W]$. By divorcing ourselves from the need to return directly interpretable parameters as coefficients of a regression model, we open the door to powerful data-adaptive procedures that give hope of consistently estimating the parameter of interest. The researchers can choose the parameter of the data-generating distribution that best addresses the scientific question.

multiPIM accomplishes what has been a contradictory goal: it uses flexible modeling, but still returns asymptotically normal measures of variable importance with robust inference. An additional benefit, which depends on the validity of the causal assumptions and on how the target levels for the exposure variables are chosen, is that the returned parameter estimates have a relevant public health interpretation (they quantify the effect of a hypothetical intervention). Also, since they are on the scale of the outcome, these estimates are directly comparable across different exposure variables, which will typically be on different scales. Due to the machine learning approach, no arbitrary a priori specification of the relevant models is required, and a meaningful ranking of relative importance is provided, with unbiased inference.

Though this is an important first step, there are several ways in which **multiPIM** could be improved, such as 1) by targeting the model selection of the treatment mechanism using a collaborative targeted maximum likelihood estimation approach, 2) by estimating the attributable risk as a smooth function of $A = a$, so that one will not need to dichotomize at a target level, and 3) by providing better integration with the **SuperLearner** and **caret** packages, in order to benefit as much as possible from R's latest machine learning capabilities. However, **multiPIM** represents a new way for researchers to harness the power of machine learning, avoid the bias of arbitrary parametric models, and still get an interpretable measure of relative variable importance, without the typical pitfalls (highlighted in Ioannidis, 2005), that plague current exploratory approaches.

Bibliography

Breiman L (1984). *Classification and Regression Trees.* Wadsworth International Group, Belmont, CA. ISBN 0534980538.

Breiman L (1996). "Stacked Regressions." *Machine Learning*, **24**(1), 49–64.

Breiman L (2001). "Random forests." *Machine learning*, **45**(1), 5–32. ISSN 0885-6125.

Cawley GC, Talbot NL (2010). "On Over-Fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation." *The Journal of Machine Learning Research*, **99**, 2079–2107.

Efron B (1979). "Bootstrap Methods: Another Look at the Jackknife." *The Annals of Statistics*, **7**(1), 1–26.

Efron B, Hastie T, Johnstone I, Tibshirani R (2004). "Least Angle Regression." *The Annals of Statistics*, **32**(2), pp. 407–451.

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

Frank lE, Friedman JH (1993). "A Statistical View of Some Chemometrics Regression Tools." *Technometrics*, **35**(2), 109–135.

Freedman DA (2005). *Statistical Models: Theory and Practice.* Cambridge University Press.

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

Friedman JH (1991). "Multivariate Adaptive Regression Splines." *The annals of statistics*, pp. 1–67.

Fu WJ (1998). "Penalized Regressions: The Bridge versus the Lasso." *Journal of computational and graphical statistics*, **7**(3), 397–416.

Goeman JJ (2010). "L1 Penalized Estimation in the Cox Proportional Hazards Model." *Biometrical Journal*, **52**(1), 70–84.

Goeman JJ (2011). **penalized***: L1 (Lasso) and L2 (Ridge) Penalized Estimation in GLMs and in the Cox Model.* R package version 0.9-37, URL `http://www.msbi.nl/goeman`.

Golub TR, Slonim DK, Tamayo P, Huard C, Gaasenbeek M, Mesirov JP, Coller H, Loh ML, Downing JR, Caligiuri MA, *et al.* (1999). "Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring." *Science*, **286**(5439), 531–537.

Gordis L (2004). *Epidemiology.* 3rd edition. Elsevier.

Greenland S, Drescher K (1993). "Maximum Likelihood Estimation of the Attributable Fraction from Logistic Models." *Biometrics*, **49**(3), 865–72.

Gruber S, van der Laan MJ (2011). "C-TMLE of an Additive Point Treatment Effect." In van der Laan and Rose (2011), chapter 19.

Gruber S, van der Laan MJ (2012). "**tmle**: An R Package for Targeted Maximum Likelihood Estimation." *Journal of Statistical Software*, **51**(13). URL `http://www.jstatsoft.org/v51/i13`.

Halvorsen K (2012). **ElemStatLearn***: Data sets, functions and examples from the book: "The Elements of Statistical Learning, Data Mining, Inference, and Prediction" by Trevor Hastie, Robert Tibshirani and Jerome Friedman.* R package version 2012.04-0, URL `http://CRAN.R-project.org/package=ElemStatLearn`.

Hastie T, Efron B (2011). **lars***: Least Angle Regression, Lasso and Forward Stagewise.* R package version 0.9-8, URL `CRAN.R-project.org/package=lars`.

Hastie T, Efron B (2012). **lars***: Least Angle Regression, Lasso and Forward Stagewise.* R package version 1.1, URL `http://CRAN.R-project.org/package=lars`.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd edition. Springer-Verlag, New York. ISBN 9780387848570.

Hoerl AE, Kennard RW (1970). "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics*, **12**(1), 55–67.

Hubbard A, Munoz ID, Decker A, Holcomb JB, Schreiber MA, Bulger EM, Brasel KJ, Fox EE, del Junco DJ, Wade CE, *et al.* (2013). "Time-Dependent Prediction and Evaluation of Variable Importance Using Superlearning in High-Dimensional Clinical Data." *Journal of Trauma-Injury, Infection, and Critical Care*, **75**(1), S53–S60.

Hubbard AE, van der Laan MJ (2008). "Population Intervention Models in Causal Inference." *Biometrika*, **95**(1), 35–47.

Ioannidis JPA (2005). "Why Most Published Research Findings Are False." *PLoS Medicine*, **2**(8), e124. doi:10.1371/journal.pmed.0020124.

Kenkel B, Signorino CS (2013). *polywog: Bootstrapped Basis Regression with Oracle Model Selection*. R package version 0.3-0, URL `http://CRAN.R-project.org/package=polywog`.

Kooperberg C (2010). **polspline***: Polynomial Spline Routines*. R package version 1.1.5, URL `http://CRAN.R-project.org/package=polspline`.

Kooperberg C, Bose S, Stone CJ (1997). "Polychotomous Regression." *Journal of the American Statistical Association*, **92**(437), 117–127.

Kuhn M (2008). "Building Predictive Models in R Using the **caret** Package." *Journal of Statistical Software*, **28**(5), 1–26.

Kuhn M, Wing J, Weston S, Williams A, Keefer C, Engelhardt A (2011). **caret***: Classification and Regression Training*. R package version 5.07-001, URL `http://CRAN.R-project.org/package=caret`.

Lan Q, Zhang L, Li G, Vermeulen R, Weinberg RS, Dosemeci M, Rappaport SM, Shen M, Alter BP, Wu Y, *et al.* (2004). "Hematotoxicity in Workers Exposed to Low Levels of Benzene." *Science*, **306**(5702), 1774–1776.

L'Ecuyer P, Simard R, Chen EJ, Kelton WD (2002). "An Object-Oriented Random-Number Package with Many Long Streams and Substreams." *Operations Research*, **50**(6), 1073–1075. doi:10.1287/opre.50.6.1073.358.

Leisch F (2002). "Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis." In W Härdle, B Rönz (eds.), *Compstat 2002 — Proceedings in Computational Statistics*, pp. 575–580. Physica Verlag, Heidelberg. URL `http://www.stat.uni-muenchen.de/~leisch/Sweave`.

Liaw A, Wiener M (2002). "Classification and Regression by **randomForest**." *R News*, **2**(3), 18–22. URL `http://CRAN.R-project.org/doc/Rnews/`.

McCullagh P, Nelder JA (1989). *Generalized Linear Models*, volume 37 of *Monographs on Statistics and Applied Probability*. 2nd edition. Chapman and Hall/CRC.

McHale CM, Zhang L, Lan Q, Vermeulen R, Li G, Hubbard AE, Porter KE, Thomas R, Portier CJ, Shen M, *et al.* (2011). "Global Gene Expression Profiling of a Population Exposed to a Range of Benzene Levels." *Environmental Health Perspectives*, **119**(5), 628.

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, URL `http://CRAN.R-project.org/package=relaxo`.

Messer LC, Oakes JM, Mason S (2010). "Effects of Socioeconomic and Racial Residential Segregation on Preterm Birth: A Cautionary Tale of Structural Confounding." *American Journal of Epidemiology*, **171**(6), 664–73. doi:10.1093/aje/kwp435.

Miller AJ (2002). *Subset Selection in Regression*, volume 95 of *Monographs on Statistics and Applied Probability*. 2nd edition. Chapman & Hall/CRC.

Molinaro AM, Lostritto K, van der Laan M (2010). "partDSA: Deletion/Substitution/Addition Algorithm for Partitioning the Covariate Space in Prediction." *Bioinformatics*, **26**(10), 1357–1363.

Molinaro AM, Simon R, Pfeiffer RM (2005). "Prediction Error Estimation: A Comparison of Resampling Methods." *Bioinformatics*, **21**(15), 3301–3307.

Nelder JA, Wedderburn RW (1972). "Generalized Linear Models." *Journal of the Royal Statistical Society: Series A*, pp. 370–384.

Neugebauer R, Bullard J (2010). **DSA***: Deletion/Substitution/Addition Algorithm.* R package version 3.1.4, URL `http://www.stat.berkeley.edu/~laan/Software/`.

Park MY, Hastie T (2007). "L1-Regularization Path Algorithm for Generalized Linear Models." *Journal of the Royal Statistical Society: Series B*, **69**(4), 659–677.

Park MY, Hastie T (2013). *glmpath: L1 Regularization Path for Generalized Linear Models and Cox Proportional Hazards Model.* R package version 0.97, URL `http://CRAN.R-project.org/package=glmpath`.

Pearl J (2000). *Causality: Models, Reasoning, and Inference.* Cambridge University Press, Cambridge, U.K. ISBN 9780521895606.

Petersen ML, Porter KE, Gruber S, Wang Y, van der Laan MJ (2011). "Positivity." In van der Laan and Rose (2011), chapter 10.

Polley EC, Rose S, van der Laan MJ (2011). "Super Learning." In van der Laan and Rose (2011), chapter 3.

Polley EC, van der Laan MJ (2011). **SuperLearner***: Super Learner Prediction.* R package version 2.0-4, URL `http://CRAN.R-project.org/package=SuperLearner`.

Rahbar MH, Fox EE, del Junco DJ, Cotton BA, Podbielski JM, Matijevic N, Cohen MJ, Schreiber MA, Zhang J, Mirhaji P, *et al.* (2012). "Coordination and Management of Multicenter Clinical Studies in Trauma: Experience from the PRospective Observational Multicenter Major Trauma Transfusion (PROMMTT) Study." *Resuscitation*, **83**(4), 459–464.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org/`.

Ritter SJ (2011). *Variable Importance Analysis with Population Intervention Models*. Master's thesis, University of California, Berkeley.

Ritter SJ, Jewell NP, Hubbard AE (submitted). "R Package **multiPIM**: A Causal Inference Approach to Variable Importance Analysis." Submitted to the *Journal of Statistical Software*.

Robins JM (1986). "A New Approach to Causal Inference in Mortality Studies with a Sustained Exposure Period–Application to Control of the Healthy Worker Survivor Effect." *Mathematical Modelling*, **7**(9-12), 1393–1512.

Robins JM (2000). "Marginal Structural Models versus Structural Nested Models as Tools for Causal Inference." In ME Halloran, D Berry (eds.), *Statistical Models in Epidemiology, the Environment, and Clinical Trials*, pp. 95–113. Springer-Verlag, New York. ISBN 0387989242.

Rose S, van der Laan MJ (2011a). "Understanding TMLE." In van der Laan and Rose (2011), chapter 5.

Rose S, van der Laan MJ (2011b). "Why TMLE?" In van der Laan and Rose (2011), chapter 6.

Rosenman RH, Brand RJ, Jenkins CD, Friedman M, Straus R, Wurm M (1975). "Coronary Heart Disease in the Western Collaborative Group Study. Final Follow-up Experience of 8 1/2 Years." *JAMA*, **233**(8), 872–7.

Rosenman RH, Friedman M, Straus R, Wurm M, Jenkins CD, Messinger HB (1966). "Coronary Heart Disease in the Western Collaborative Group Study. A Follow-up Experience of Two Years." *JAMA*, **195**(2), 86–92.

Sevcikova H, Rossini T (2009). **rlecuyer***: R Interface to RNG with Multiple Streams*. R package version 0.3-1, URL `http://CRAN.R-project.org/package=rlecuyer`.

Sinisi SE, Polley EC, Petersen ML, Rhee SY, van der Laan MJ (2007). "Super Learning: An Application to the Prediction of HIV-1 Drug Resistance." *Statistical Applications in Genetics and Molecular Biology*, **6**(1), Article 7. doi:10.2202/1544-6115.1240.

Sinisi SE, van der Laan MJ (2004). "Deletion/Substitution/Addition Algorithm in Learning with Applications in Genomics." *Statistical Applications in Genetics and Molecular Biology*, **3**(1), Article 18. doi:10.2202/1544-6115.1069.

Spear RC, Seto E, Liang S, Birkner M, Hubbard A, Qiu D, Yang C, Zhong B, Xu F, Gu X, Davis GM (2004). "Factors Influencing the Transmission of *Schistosoma Japonicum* in the Mountains of Sichuan Province of China." *The American Journal of Tropical Medicine and Hygiene*, **70**(1), 48–56.

Stamey T, Kabalin J, McNeal J, Johnstone I, Freiha F, Redwine E, Yang N (1989). "Prostate Specific Antigen in the Diagnosis and Treatment of Adenocarcinoma of the Prostate. II. Radical Prostatectomy Treated Patients." *The Journal of urology*, **141**(5), 1076.

Stone C, Hansen M, Kooperberg C, Truong Y (1997). "Polynomial Splines and their Tensor Products in Extended Linear Modeling." *The Annals of Statistics*, **25**(4), 1371–1425.

Sudat SEK, Carlton EJ, Seto EYW, Spear RC, Hubbard AE (2010). "Using Variable Importance Measures from Causal Inference to Rank Risk Factors of Schistosomiasis Infection in a Rural Setting in China." *Epidemiologic Perspectives & Innovations*, **7**(1), 3.

Therneau TM, Atkinson B, Ripley BD (2010). **rpart***: Recursive Partitioning.* R package version 3.1-46, URL `http://CRAN.R-project.org/package=rpart`.

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

Tibshirani R (2011). "Regression Shrinkage and Selection via the Lasso: A Retrospective." *Journal of the Royal Statistical Society: Series B*, **73**(3), 273–282.

Urbanek S (2011). **multicore***: Parallel Processing of R Code on Machines with Multiple Cores or CPUs.* R package version 0.1-7, URL `http://CRAN.R-project.org/package=multicore`.

van der Laan MJ (2006). "Statistical Inference for Variable Importance." *The International Journal of Biostatistics*, **2**(1), Article 2. doi:10.2202/1557-4679.1008.

Van Der Laan MJ, Dudoit S (2003). "Unified Cross-Validation Methodology for Selection Among Estimators and a General Cross-Validated Adaptive Epsilon-Net Estimator: Finite Sample Oracle Inequalities and Examples."

van der Laan MJ, Gill RD, Robins JM (2003). "Locally Efficient Estimation in Censored Data Models: Theory and Examples." *U.C. Berkeley Division of Biostatistics Working Paper Series*, (Working Paper 85). URL `http://biostats.bepress.com/ucbbiostat/paper85/`.

van der Laan MJ, Gruber S (2010). "Collaborative Double Robust Targeted Maximum Likelihood Estimation." *The International Journal of Biostatistics*, **6**(1), Article 17. doi:10.2202/1557-4679.1181.

van der Laan MJ, Polley EC, Hubbard AE (2007). "Super Learner." *Statistical Applications in Genetics and Molecular Biology*, **6**(1), Article 25. doi:10.2202/1544-6115.1309.

van der Laan MJ, Robins JM (2003). *Unified Methods for Censored Longitudinal Data and Causality*. Springer-Verlag, New York. ISBN 0387955569.

van der Laan MJ, Rose S (2011). *Targeted Learning: Causal Inference for Observational and Experimental Data*. Springer-Verlag, New York. ISBN 9781441997814.

van der Laan MJ, Rubin D (2006). "Targeted Maximum Likelihood Learning." *The International Journal of Biostatistics*, **2**(2), Article 11. doi:10.2202/1557-4679.1043.

Vermeulen R, Li G, Lan Q, Dosemeci M, Rappaport SM, Bohong X, Smith MT, Zhang L, Hayes RB, Linet M, *et al.* (2004). "Detailed Exposure Assessment for a Molecular Epidemiology Study of Benzene in two Shoe Factories in China." *Annals of Occupational Hygiene*, **48**(2), 105–116.

Young JG (2007). *Statistical Methods for Complicated Current Status and High-Dimensional Data Structures with Applications in Environmental Epidemiology*. Ph.D. thesis, University of California, Berkeley.

Young JG, Hubbard AE, Eskenazi B, Jewell NP (2009). "A Machine-Learning Algorithm for Estimating and Ranking the Impact of Environmental Risk Factors in Exploratory Epidemiological Studies." *U.C. Berkeley Division of Biostatistics Working Paper Series*, (Working Paper 250). URL `http://www.bepress.com/ucbbiostat/paper250`.

Zheng W, van der Laan MJ (2011). "Cross-Validated Targeted Minimum-Loss-Based Estimation." In van der Laan and Rose (2011), chapter 27.

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

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

Zou H, Zhang HH (2009). "On the Adaptive Elastic-Net with a Diverging Number of Parameters." *Annals of Statistics*, **37**(4), 1733.

Appendix A

Help File for `relaxnet` Function

`relaxnet`	*Relaxation (as in Relaxed Lasso, Meinshausen 2007) applied to glmnet Models*

Description

Runs glmnet once on the full x matrix, then again on each distinct subset of columns from along the solution path. The penalty may be lasso (`alpha` $= 1$) or elastic net ($0 <$ `alpha` < 1). The outcome (`y`) may be continuous or binary.

Usage

```
relaxnet(x, y, family = c("gaussian", "binomial"),
         nlambda = 100,
         alpha = 1,
         relax = TRUE,
         relax.nlambda = 100,
         relax.max.vars = min(nrow(x), ncol(x)) * 0.8,
         lambda = NULL,
         relax.lambda.index = NULL,
         relax.lambda.list = NULL,
         ...)
```

Arguments

`x`	Input matrix, of dimension nobs x nvars; each row is an observation vector. Can be in sparse matrix format (inherit from class `"sparseMatrix"` as in package `Matrix`). Must have unique colnames.

`y`	response variable. Quantitative for `family="gaussian"`. For `family="binomial"` should be either a factor with two levels, or a two-column matrix of counts or proportions.
`family`	Response type (see above).
`nlambda`	The number of `lambda` values - default is 100. Determines how fine the grid of lambda values should be.
`alpha`	Elastic net mixing parameter (see `glmnet`).
`relax`	Should the model be relaxed. If FALSE, only the main glmnet model is run and no relaxed models are.
`relax.nlambda`	Like nlambda but for secondary (relaxed) models.
`relax.max.vars`	Maximum number of variables for relaxed models. No relaxation will be done for subsets along the regularization path with number of variables greater than relax.max.vars. If `ncol(x)` $>$ `nrow(x)` and `alpha` < 1, it may make sense to use a value $>$ `nrow(x)`, but this may lead to increased computation time.
`lambda`	See (see `glmnet`). Optional and meant primarily for use by `cv.relaxnet`.
`relax.lambda.index`	Vector which indexes the lambda argument and specifyies the values at which a relaxed model should be fit. Optional and meant primarily for use by `cv.relaxnet`. Ignored if lambda argument is `NULL`.
`relax.lambda.list`	List of lambda values to use for the relaxed models. Optional and meant primarily for use by `cv.relaxnet`. Ignored if lambda argument is `NULL`.
`...`	Further aruments passed to glmnet. Use with caution as this has not yet been tested. For example, setting `standardize = FALSE` will probably work correctly, but setting an offset probably won't.

Value

Object of class code"relaxnet" with the following components:

`call`	A copy of the call which produced this object
`main.glmnet.fit`	The object resulting from running `glmnet` on the entire `x` matrix.
`relax`	The value of the relax argument. If this is `FALSE`, then several of the other elements of this result will be set to `NA`.

`relax.glmnet.fits`
: A list containing the secondary `glmnet` fits gotten by running `glmnet` on the distinct subsets of the columns of `x` resulting along the solution path of lambda values.

`relax.num.vars`
: Vector giving the number of variables in each ”relaxed” model.

`relax.lambda.index`
: This vector indexes result$main.glmnet.fit$lambda and gives the lambda values at which the relax.glmnet.fits were obtained.

`total.time` Total time in seconds to produce this result.

`main.fit.time`
: Time in seconds to produce the main glmnet fit.

`relax.keep` In certain cases some of the relaxed models are removed after fitting. `relax.fit.times` records times for these removed models as well. This logical vector shows which of the models whose timings are given in `relax.fit.times` were actually kept and have results given in `relax.glmnet.fits` above. Hopefully this will not be necessary in later versions.

`relax.fit.times`
: Vector of times in seconds to produce secondary ”relaxed” models.

Author(s)

Stephan Ritter, with design contributions from Alan Hubbard.

Much of the code (and some help file content) is adapted from the **glmnet** package, whose authors are Jerome Friedman, Trevor Hastie and Rob Tibshirani.

See Also

`glmnet`, `cv.relaxnet`, `predict.relaxnet`

Appendix B

Help File for `widenet` Function

`widenet` *Extends the relaxnet Package with Polynomial Basis Expansions*

Description

Expands the basis according to the `order` argument, then runs relaxnet in order to select a subset of the basis functions. Multiple values of `order` and `alpha` (the elastic net tuning parameter) may be specified, leading to selection of a specific value by cross-validation.

Usage

```
widenet(x, y, family = c("gaussian", "binomial"),
        order = 1:3,
        alpha = 1,
        nfolds = 10,
        foldid,
        screen.method = c("none", "cor", "ttest"),
        screen.num.vars = 50,
        multicore = FALSE,
        mc.cores,
        mc.seed = 123,
        ...)
```

Arguments

`x`	Input matrix, each row is an observation vector. Sparse matrices are not yet supported for the `widenet` function. Must have unique colnames.

`y`	Response variable. Quantitative for `family="gaussian"`. For `family="binomial"` should be either a factor with two levels, or a two-column matrix of counts or proportions.
`family`	Response type (see above).
`order`	The order of basis expansion. Elements must be in the set `c(1, 2, 3)`. If there is more than one element, cross-validation is used to chose the order with best cross-validated performance.
`alpha`	The elastic net mixing parameter, see `glmnet`. If there is more than one element, cross-validation is used to chose the value with best cross-validated performance.
`nfolds`	Number of folds - default is 10. Although `nfolds` can be as large as the sample size (leave-one-out CV), it is not recommended for large datasets. Smallest value allowable is `nfolds=3`.
`foldid`	An optional vector of values between 1 and `nfold` identifying what fold each observation is in. If supplied, `nfolds` can be missing.
`screen.method`	The method to use to screen variables before basis expansion is applied. Default is no screening. `"cor"` = correlation, i.e. bivariate correlation with the outcome. ttest is meant for binary outcomes (`family = "binomial"`). The screening methods are adapted from the **SuperLearner** package, the author of which is Eric Polley.
`screen.num.vars`	The number of variables (columns of `x` to screen in when using screening.
`multicore`	Should execution be parallelized over cv folds (for `cv.relaxnet`) or over alpha values (for `cv.alpha.relaxnet`) using multicore functionality from R's parallel package?
`mc.cores`	Number of cores/cpus to be used for multicore processing. Parallelization is over cross-validation folds.
`mc.seed`	Integer value with which to seed the RNG when using parallel processing (internally, `RNGkind` will be called to set the RNG to `"L'Ecuyer-CMRG"`). Will be ignored if `multicore` is `FALSE`. If `mulicore` is `FALSE`, one should be able to get reprodicible results by setting the seed normally (with `set.seed`) prior to running.
`...`	Further arguments passed to `relaxnet` or `cv.relaxnet`, which should also be passed on to `glmnet`. Use with caution as this has not been tested.

Details

The `type.measure` argument has not yet been implemented. For type = gaussian models, mean squared error is used, and for type = binomial, binomial deviance is used.

Value

Returns and object of class `"widenet"` with the following elements:

`call`	A copy of the call which generated this object
`order`	The value of the `order` argument
`alpha`	The value of the `alpha` argument
`screen.method`	The value of the `screen.method` argument
`screened.in.index`	A vector which indexes the columns of `x`, indicating those variables which were screened in for the run on the full data
`colsBinary`	A vector of length `ncol(x)` representing which of the columns of `x` contained binary data. These columns will be represented by a 2. The other columns will have a 3.
`cv.relaxnet.results`	A list of lists containing `"cv.relaxnet"` objects, one for each combination of values of alpha and order.
`min.cvm.mat`	A matrix containing the minimum cross-validated risk for each combination of values of alpha and order
`which.order.min`	The order which ”won” the cross-validation, i.e. resulted in minimum cross-validated risk.
`which.alpha.min`	The alpha value which ”won” the cross-validation.
`total.time`	Total time in seconds to produce this result.

Author(s)

Stephan Ritter, with design contributions from Alan Hubbard.

Much of the code (and some help file content) is adapted from the **glmnet** package, whose authors are Jerome Friedman, Trevor Hastie and Rob Tibshirani.

See Also

`predict.widenet`, `relaxnet`, `cv.relaxnet`

Appendix C

Help File for `multiPIM` Function

`multiPIM`	*Estimate Variable Importances for Multiple Exposures and Outcomes*

Description

The parameter of interest is a type of causal attributable risk. One effect measure (and a corresponding plug-in standard error) will be calculated for each exposure-outcome pair. The default is to use a Targeted Maximum Likelihood Estimator (TMLE). The other available estimators are Inverse Probability of Censoring Weighted (IPCW), Double-Robust IPCW (DR-IPCW), and Graphical Computation (G-COMP) estimators. PIM stands for Population Intervention Model.

Usage

```
multiPIM(Y, A, W = NULL,
         estimator = c("TMLE", "DR-IPCW", "IPCW", "G-COMP"),
         g.method = "main.terms.logistic", g.sl.cands = NULL,
         g.num.folds = NULL, g.num.splits = NULL,
         Q.method = "sl", Q.sl.cands = "default",
         Q.num.folds = 5, Q.num.splits = 1,
         Q.type = NULL,
         adjust.for.other.As = TRUE,
         truncate = 0.05,
         return.final.models = TRUE,
         na.action,
         check.input = TRUE,
         verbose = FALSE,
         extra.cands = NULL,
         standardize = TRUE,
         ...)
```

Arguments

`Y`	a data frame of outcomes containing only numeric (integer or double) values. See details for the default method of determining, based on the values in `Y`, which regression types to allow for modelling Q. Must have unique names.
`A`	a data frame containing binary exposure variables. *Binary* means that all values must be either 0 (indicating unexposed, or part of target group) or 1 (indicating exposed or not part of target group). Must have unique names.
`W`	an optional data frame containing possible confounders of the effects of the variables in `A` on the variables in `Y`. No effect measures will be calculated for these variables. May contain numeric (integer or double), or factor values. Must be left as `NULL` if not required. See details.
`estimator`	the estimator to be used. The default is `"TMLE"`, for the targeted maximum likelihood estimator. Alternatively, one may specify `"DR-IPCW"`, for the Double-Robust Inverse Probability of Censoring-Weighted estimator, or `"IPCW"`, for the regular IPCW estimator, or `"G-COMP"` for the Graphical Computation estimator. If the regular IPCW estimator is selected, all arguments which begin with the letter Q are ignored, since only g (the regression of each exposure on possible confounders) needs to be modeled in this case. Similarly, if the G-COMP estimator is selected, all arguments which begin with the letter g, as well as the truncate argument, will be ignored, since only Q needs to be modeled in this case. Note: an additional characteristic of the G-COMP estimator is that there are no plug-in standard errors available. If you want to use G-COMP and you need standard errors, the `multiPIMboot` function is available and will provide bootstrap standard errors.
`g.method`	a length one character vector indicating the regression method to use in modelling g. The default value, `"main.terms.logistic"`, is meant to be used with the default TMLE estimator. If a different estimator is used, it is recommended to use super learning by specifying `"sl"`. In this case, the arguments `g.sl.cands`, `g.num.folds` and `g.num.splits` must also be specified. Other possible values for the `g.method` argument are: one of the elements of the vector `all.bin.cands`, or, if `extra.cands` is

supplied, one of the names of the `extra.cands` list of functions. Ignored if `estimator` is `"G-COMP"`.

`g.sl.cands`	character vector of length ≥ 2 indicating the candidate algorithms that the super learner fits for g should use. The possible values may be taken from the vector `all.bin.cands`, or from the names of the `extra.cands` list of functions, if it is supplied. Ignored if `estimator` is `"G-COMP"`. or if `g.method` is not `"sl"`. NOTE: The TMLE estimator is recommended, but if one is using either of the IPCW estimators, a reasonable choice is to specify `g.method = "sl"` and `g.sl.cands = default.bin.cands`.
`g.num.folds`	the number of folds to use in cross-validating the super learner fit for g (i.e. the v for v-fold cross-validation). Ignored if `estimator` is `"G-COMP"`, or if `g.method` is not `"sl"`.
`g.num.splits`	the number of times to randomly split the data into `g.num.folds` folds in cross-validating the super learner fit for g. Cross-validation results will be averaged over all splits. Ignored if `estimator` is `"G-COMP"`, or if `g.method` is not `"sl"`.
`Q.method`	character vector of length 1. The regression method to use in modelling Q. See details to find out which values are allowed. The default value, `"sl"`, indicates that super learning should be used for modelling Q. Ignored if `estimator` is `"IPCW"`.
`Q.sl.cands`	either of the length 1 character values `"default"` or `"all"` or a character vector of length ≥ 2 containing elements of either `all.bin.cands` or of `all.cont.cands`, or of the names of the `extra.cands` list of functions, if it is supplied. See details. Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.num.folds`	the number of folds to use in cross-validating the super learner fit for Q (i.e. the v for v-fold cross-validation). Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.num.splits`	the number of times to randomly split the data into `Q.num.folds` folds in cross-validating the super learner fit for Q. Ignored if `estimator` is `"IPCW"` or if `Q.method` is not `"sl"`.
`Q.type`	either `NULL` or a length 1 character vector (which must be either `"binary.outcome"` or `"continuous.outcome"`). This provides a way to override the default mechanism for deciding which candidates will be allowed for modeling Q (see details). Ignored if `estimator` is `"IPCW"`.
`adjust.for.other.As`	a single logical value indicating whether the other columns of `A` should be included (for `TRUE`) or not (for `FALSE`) in the g and Q models used to

	calculate the effect of each column of `A` on each column of `Y`. See details. Ignored if `A` has only one column.
`truncate`	either `FALSE`, or a single number greater than 0 and less than 0.5 at which the values of g(0, W) should be truncated in order to avoid instability of the estimator. Ignored if `estimator` is `"G-COMP"`.
`return.final.models`	single logical value indicating whether final g and Q models should be returned by the function (in the slots `g.final.models` and `Q.final.models`). Default is `TRUE`. If memory is a concern, you will probably want to set this to FALSE.
`na.action`	currently ignored. If any of `Y`, `A` or (a non-null) `W` has missing values, `multiPIM` will throw an error.
`check.input`	a single logical value indicating whether all of the input to the function should be subjected to strict error checking. `FALSE` is not recommended.
`verbose`	a single logical value indicating whether messages about the progress of the evaluation should be printed out. Some of the candidate algorithms may print messages even when `verbose` is set to `FALSE`.
`extra.cands`	a named list of functions. This argument provides a way for the user to specify his or her own functions to use either as stand-alone regression methods, or as candidates for a super learner. See details.
`standardize`	should all predictor variables be standardized before certain regression methods are run. Passed to all candidates, but only used by some (at this point, lars, penalized.bin and penalized.cont)
`...`	currently ignored.

Details

The parameter of interest is a type of attributable risk. This means that it is a measure (adjusted for known confounders) of the *difference* between the mean value of `Y` for the units in the target (or unexposed) group and the *overall* mean value of `Y`. Units which are in the target (or unexposed) group with respect to one of the variables in `A` are characterized as such by having the value 0 in the respective column of `A`. Members of the the non-target (or exposed) group should have a 1 in that column of `A`. *Assuming all causal assumptions hold* (see the paper), each parameter estimate can be thought of as estimating the hypothetical effect on the respective outcome of totally eliminating the respective exposure from the population (i.e. setting everyone to 0 for that exposure). For example, in the case of a binary outcome, a parameter estimate for exposure x and outcome y of -0.03 could be interpreted as follows: the effect of an intervention in which the entire population was set to exposure x = 0 would be to reduce the level of outcome y by 3 percentage points.

If `check.input` is `TRUE` (which is the default and is highly recommended), all of the arguments will be checked to make sure they have permissible values. Many of the arguments, especially those for which a single logical value (`TRUE` or `FALSE`) or a single character value (such as, for example, `"all"`) is expected, are checked using the `identical` function, which means that if any of these arguments has any extraneous attributes (such as names), this may cause `multiPIM` to throw an error.

On the other hand, the arguments `Y` and `A` (and `W` if it is non-null) *must* have valid names attributes. `multiPIM` will throw an error if there is any overlap between the names of the columns of these data frames, or if any of the names cannot be used in a `formula` (for example, because it begins with a number and not a letter).

By default, the regression methods which will be allowed for fitting models for Q will be determined from the contents of `Y` as follows: if all values in Y are either 0 or 1 (i.e. all outcomes are binary), then "logistic"-type regression methods will be used (and only these methods will be allowed in the arguments `Q.method` and `Q.sl.cands`); however, if there are any values in `Y` which are not equal to 0 or 1 then it will be assumed that all outcomes are continuous, "linear"-type regression will be used, and the values allowed for `Q.method` and `Q.sl.cands` will change accordingly. This behavior can be overriden by specifying `Q.type` as either `"binary.outcome"` (for logistic-type regression), or as `"continuous.outcome"` (for linear-type regression). If `Q.type` is specified, `Y` will not be checked for binaryness.

The values allowed for `Q.method` (which should have length 1) are: either `"sl"` if one would like to use super learning, or one of the elements of the vector `all.bin.cands` (for the binary outcome case), or of `all.cont.cands` (for the continuous outcome case), if one would like to use only a particular regression method for all modelling of Q. If `Q.method` is given as `"sl"`, then the candidates used by the super learner will be determined from the value of `Q.sl.cands`. If the value of `Q.sl.cands` is `"default"`, then the candidates listed in either `default.bin.cands` or `default.cont.cands` will be used. If the value of `Q.sl.cands` is `"all"`, then the candidates listed in either `all.bin.cands` or `all.cont.cands` will be used. The function will automatically choose the candidates which correspond to the correct outcome type (binary or continuous). Alternatively, one may specify `Q.sl.cands` explicitly as a vector of names of the candidates to be used.

If `A` has more than one column, the `adjust.for.other.As` argument can be used to specify whether the other columns of `A` should possibly be included in the g and Q models which will be used in calculating the effect of a certain column of `A` on each column of `Y`.

With the argument `extra.cands`, one may supply alternative R functions to be used as stand-alone regression methods, or as super learner candidates, within the `multiPIM` function. `extra.cands` should be given as a named list of functions. See Candidates for the form (e.g. arguments) that the functions in this list should have. In order to supply your own stand alone regression method for g or Q, simply specify `g.method` or

`Q.method` as the name of the function you want to use (i.e. the corresponding element of the names attribute of `extra.cands`). To add candidates to a super learner, simply use the corresponding names of your functions (from the names attribute of `extra.cands`) when you supply the `g.sl.cands` or `Q.sl.cands` arguments. Note that you may mix and match between your own extra candidates and the built-in candidates given in the `all.bin.cands` and `all.cont.cands` vectors. Note also that extra candidates must be explicitly specified as `g.method`, `Q.method`, or as elements of `g.sl.cands` or `Q.sl.cands` – Specifying `Q.sl.cands` as `"all"` will not cause any extra candidates to be used.

Value

Returns an object of class `"multiPIM"` with the following elements:

`param.estimates`	a matrix of dimensions `ncol(A)` by `ncol(Y)` with `rownames` equal to `names(A)` and `colnames` equal to `names(Y)`, with each element being the estimated causal attributable risk for the exposure given by its row name vs. the outcome given by its column name.
`plug.in.stand.errs`	a matrix with the same dimensions as `param.estimates` containing the corresponding plug-in standard errors of the parameter estimates. These are obtained from the influence curve. Note: plug-in standard errors are not available for `estimator = "G-COMP"`. This field will be set to `NA` in this case.
`call`	a copy of the call to `multiPIM` which generated this object.
`num.exposures`	this will be set to `ncol(A)`.
`num.outcomes`	this will be set to `ncol(Y)`.
`W.names`	the names attribute of the `W` data frame, if one was supplied. If no `W` was supplied, this will be `NA`.
`estimator`	the estimator used.
`g.method`	the method used for modelling g.
`g.sl.cands`	in case super learning was used for g, the candidates used in the super learner. Will be `NA` if `g.method` was not `"sl"`.
`g.winning.cands`	if super learning was used for g, this will be a named character vector with `ncol(A)` elements. The ith element will be the name of the candidate which ”won” the cross validation in the g model for the ith column of `A`.

`g.cv.risk.array`
: array with dim attribute `c(ncol(A), g.num.splits, length(g.sl.cands))` containing cross-validated risks from super learner modeling for g for each exposure-split-candidate triple. Has informative dimnames attribute. Note: the values are technically not risks, but log likelihoods (i.e. winning candidate is the one for which this is a *max*, not a min).

`g.final.models`
: a list of length `nrow(A)` containing the objects returned by the candidate functions used in the final g models (see Candidates).

`g.num.folds`
: the number of folds used for cross validation in the super learner for g. Will be `NA` if `g.method` was not `"sl"`.

`g.num.splits`
: the number of splits used for cross validation in the super learner for g. Will be `NA` if `g.method` was not `"sl"`.

`Q.method`
: the method used for modeling Q. Will be `NA` if `double.robust` was `FALSE`.

`Q.sl.cands`
: in case super learning was used for Q, the candidates used in the super learner. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.

`Q.winning.cands`
: if super learning was used for Q, this will be a named character vector with `ncol(Y)` elements. The ith element is the name of the candidate which ”won” the cross validation in the super learner for the Q model for the ith column of `Y`.

`Q.cv.risk.array`
: array with dim attribute `c(ncol(A), ncol(Y), Q.num.splits, length(Q.sl.cands))` containing cross-validated risks from super learner modeling for Q. Has informative dimnames attribute. Note: the values will be log likelihoods when `Q.type` is `"binary.outcome"` (see note above for `g.cv.risk.array`), and they will be mean squared errors when `Q.type` is `"continuous.outcome"`.

`Q.final.models`
: a list of length `ncol(A)`, each element of which is another list of length `ncol(Y)` containing the objects returned by the candidate functions used for the Q models. I.e. `Q.final.models[[i]][[j]]` contains the Q model information for exposure i and outcome j.

`Q.num.folds`
: the number of folds used for cross validation in the super learner for Q. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.

`Q.num.splits`
: the number of splits used for cross validation in the super learner for Q. Will be `NA` if `double.robust` was `FALSE` or if `Q.method` was not `"sl"`.

`Q.type`	either `"continuous.outcome"` or `"binary.outcome"`, depending on the contents of `Y` or on the value of the `Q.type` argument, if supplied.
`adjust.for.other.As`	logical value indicating whether the other columns of `A` were included in models used to calculate the effect of each column of `A` on each column of `Y`. Will be set to `NA` when `A` has only one column.
`truncate`	the value of the `truncate` argument. Will be set to NA if estimator was `"G-COMP"`.
`truncation.occured`	logical value indicating whether it was necessary to trunctate. `FALSE` when `truncate` is `FALSE`. Will be set to NA if estimator was `"G-COMP"`.
`standardize`	the value of the `standardize` argument.
`boot.param.array`	this slot will be `NULL` for objects returned by the `multiPIM` function. See `multiPIMboot` for details on what this slot is actually used for.
`main.time`	total time (in seconds) taken to generate this multiPIM result.
`g.time`	time in seconds taken for running g models.
`Q.time`	time in seconds taken for running Q models.
`g.sl.time`	if g.method is ”sl”, time in seconds taken for running cross-validation of g models.
`Q.sl.time`	if Q.method is ”sl”, time in seconds taken for running cross-validation of Q models.
`g.sl.cand.times`	if g.method is ”sl”, named vector containing time taken, with each element corresponding to a super learner candidate for g.
`Q.sl.cand.times`	if Q.method is ”sl”, named vector containing time taken, with each element corresponding to a super learner candidate for Q.

Author(s)

Stephan Ritter, with design contributions from Alan Hubbard and Nicholas Jewell.

See Also

`multiPIMboot` for running `multiPIM` with automatic bootstrapping to get standard errors.

`summary.multiPIM` for printing summaries of the results.

`Candidates` to see which candidates are currently available, and for information on writing user-defined super learner candidates and regression methods.