User-Oriented High-Dimensional Linear Model Estimation

Taylor B. Arnold*

Abstract

Penalized M-estimators have garnered much attention in recent years for their use in estimating regression parameters in high dimensional linear models. A large amount of work as been done to produce efficient algorithms and code for solving the underlying optimization problems of these methods and extending results to related applied topics (e.g., graphical models, generalized linear models). Unfortunately, these optimization schemes have not been paired with recent theoretical innovations in order provide maximum utility to practitioners. Current R packages, for instance, tune models with inconsistent cross validation procedures rather than easy to compute choices based on analytical properties of the penalized estimator. The package hdlm rectifies this gap by using tuning parameters which guarantee asymptotic consistency as well calculating valid p values, standard errors, ANOVA tables, and producing useful graphical outputs. We pay particular attention to computational speed and memory issues which arise as a result of the resampling required to produce empirical p values and standard errors.

Key Words: penalized estimation, resampling methods, tuning parameters, two-stage methods

1. Overview

High dimensional model selection algorithms such as the lasso and orthonormal matching pursuit have garnered much attention in recent years. A large number of **R** packages for conducting high-dimensional regression exist but each concentrates on solving penalized minimization problems and bypass methods for producing regression tables. This can be frustrating for end-users as the lack of useful measurements of standard deviations and confidence intervals makes applied data analyses difficult.

Fortunately, this deficiency is not due to a lack of relevant theory but rather a lack of corresponding implementations. A two stage approach which pairs a generic high-dimensional model selection procedure and standard low-dimensional estimator was studied by [18] for the purpose of obtaining asymptotically valid high dimensional p values. This theory was further extended by [14] through the use of successive bootstraps and false discover rate techniques. Bayesian solutions where posterior distributions can be used for calculating measurements of standard errors, p values, and confidence intervals have also been suggested by, for instance, [8], [11], and [10]. With the exception of the work done by [8], where a basic MCMC sampling procedure is given, these theoretical frameworks have been presented without code.

The package **hdlm** provides a unified, optimized, and easy to use implementation of the two-stage frequentist approach as well as several hierarchical Bayesian models for producing regression tables. In addition, the returned objects are supported by a complete set of S3 methods; these mimic the methods provided for standard linear models and generalized linear models. This set includes methods for plotting, extracting coefficients, and summarizing the fitted model's output.

We have taken the philosophy that the package should be both easy to use out of the box as well as quickly adapted to as wide a range of options as possible. As the frequentist method requires a generic high-dimensional model selector, we have incorporated as

*Yale University, 24 Hillhouse Ave. New Haven, CT 06511

a default the highly optimized package **glmnet** [7]. The use of this popular point estimator, along with the fact that we have written our code in the same general syntax as the standard lm function, allows users to quickly obtain reasonably good output from a high dimensional regression using the functions found within package **hdlm**. Instead of boxing the function with several popular variants, we have designed a system which allows for the user to define custom functions to override defaults. In fact, our implementation of `hdglm` (for generalized linear models) is essentially a wrapper to the standard `hdlm` routine with linking functions passed in the required format.

The remainder of this article is organized as follows: Section 2 gives a brief overview of the theory behind frequentist high dimensional regression tables. Section 3 gives the high-level architecture of the package and a basic overview of options. Section 4 gives explicit examples of how various default options can be altered to incorporate new point estimation techniques. Section 5 illustrates the computational speed of the default algorithms. Finally, Section 6 closes with an outline of proposed future work.

2. Regression tables in high dimensions

2.1 Multi-stage methods and theory

Recent advancements in the theory of sparse high dimensional regression have largely concerned the convergence rates of point estimation procedures. The lack of research in the study of confidence intervals, standard errors, and other measurements of uncertainty stems in part from the fact that it is provenly hard to analyze model selection and inference which have been done either simultaneously or sequentially on the same dataset. Leeb [12], Pötscher [13], and Yang [20] have shown that there is in fact no generic procedure which uniformly estimates the conditional or unconditional distribution of post-model selection inference procedures. For a detailed discussion of what this means in data analysis, see the recent paper by [4].

A simple way to avoid these problems is to partition the observations, using part of the data for model selection and the other part for an independent (low-dimensional) parameter estimation on the selected model. More explicitly, the proposal by [18] to construct such an estimator first randomly splits the data observations into three groups: D_1, D_2, and D_3. Using the first subset, a series of methods are used to fit a number of sparse models:

$$b_\lambda := \phi_\lambda(D_1),\ \lambda \in \Lambda \tag{1}$$

Once these models have been fit, the following prediction error is calculated over the second subset of data:

$$e_\lambda := \sum_{i \in D_2} (y_i - x_i^\top b_\lambda)^2 \tag{2}$$

The goal is to determine which of the set of models fit in the first stage best predict the second set of data. This splitting methodology is a very simple version of cross validation. Using these estimates, an initial conservative guess of the support T is made:

$$\widehat{S}_n := \text{support}\left\{\arg\min_{\lambda \in \Lambda}(e_\lambda)\right\} \tag{3}$$

It is assumed, at this point, that there is a very high probability that $\widehat{S_n}$ contains the true model; in fact, this probability should tend to zero as the sample size limits to infinity. Under reasonable assumptions, this rate will fall off exponentially for popular model selection methods.

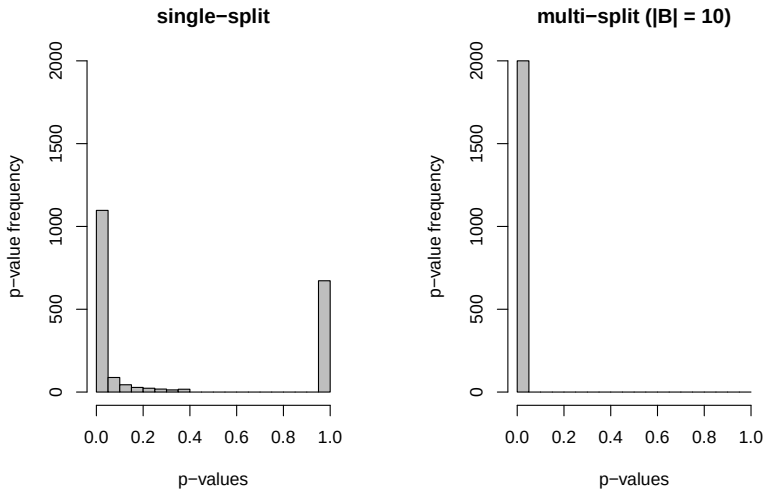

Figure 1: Distribution of p values in a simple simulation with $p = 50$, $n = 25$, and $\beta^t = (1, 0, \ldots, 0)$, over 2000 trials. The lasso is used, with 10-fold cross validation to select the tuning parameter. The left plot shows the p values resulting from the single-split method, whereas the right plot shows p values resulting from the multi-split method with 10 bootstrap runs and p values combined via the discussed FDR procedure.

Now, with the final set of data D_3, ordinary least squares regression is run on the variables contained in $\widehat{S_n}$. The final trimming is done by screening p values, and creating $\widehat{T_n}$ from those variables with a p value less than the desired level α. If the model selection in the first two steps is truly conservative, this should correctly control the component-wise probability of a Type-I Error.

In our implementation, the first stage of the method has been abstracted out of the basic design. We only split the data into two groups $D1 \cup D_2$ and D_3; the first set being used to select a model and the second set to fit a low dimensional regression. This modification allows for a wider range of model tuning techniques such as n-fold cross validation and information criteria based methods.

2.2 Bootstrapping multistage p-values

The p values of the multi-stage method of Wasserman and Roeder can be quite sensitive to the choice of the random splitting of data. Figure 1 shows the distribution of p values for one variable given different partitions of the dataset into the model selection step and the parameter estimation step. The random nature of the output makes it at best difficult to analyze and at worse a rather useless statistical method. Fortunately, a proposal by [14] provides a method for obtaining p values which bootstraps over many splits of the dataset and intelligently pastes the results together We refer to their method as the 'multi-split method', and the former by the 'single-split method'.

The major difficulty of the multi-step method is determining how to combine the p values from each run into a single set of p values. Consider just one column of the data matrix X and let $\{P_b\}_{b \in B}$ be the set of p values for this one variable across all bootstrap runs b in B. This set can be viewed as a multiple hypothesis testing problem, with the strange property that the null hypothesis in each test is the same: $H_0 : \beta_j \neq 0$. In multiple hypothesis testing there are essentially two forms of error which can be controlled. The family-wise error rate (FWER) concerns the chance of making any Type-II errors, whereas the false dis-

covery rate (FDR) concerns the proportion of Type-II errors to rejected hypotheses. While not stressed in the Meinshausen paper, for our setting where all of the hypothesis tests have the same null hypothesis, these two rates will be exactly the same. What differs is the power of the generic methods for controlling these two error rates, when applied to our specific situation.

The classic method for control the family-wise error rate is the Bonferroni correction. Here p values are 'adjusted' by multiplying by the total number of hypothesis tests. Null hypotheses are rejected if the corresponding adjusted value is less than the desired family-wise error rate. A modification by Sture Holm, which is uniformly more powerful and still valid for any pattern of independence or dependence amongst the hypothesis, is to sort the p values from smallest $P_{(1)}$ to largest $P_{(|B|)}$ and then adjust as:

$$\widetilde{P}_{(k)} := \min\left(1, \max_{j \leq k} P_{(j)} \cdot (|B| - j + 1)\right), \tag{4}$$

Where $|B|$ is the total number of hypothesis tests. The adjusted values are again compared to the maximum desired error rate and rejected accordingly.

The control of the false discovery rate is done in a similar fashion, as proven by Yosef Hochberg, Yoav Benjamini, and (in the case of dependent hypotheses) Daniel Yekutieli [1, 3]. Ordering again the p values, find the smallest k such that:

$$P_{(k)} \leq \frac{\alpha k}{|B| \cdot c(|B|)} \tag{5}$$

Where α is the desired maximum false discovery rate and $c(|B|)$ is one in the case of independent or positively correlated tests and $\sum_i^{|B|} i^{-1} \approx \log(|B|) + 0.57721$ in the case of any other dependence structure. All tests corresponding to p values less than this $P_{(k)}$ have their null hypotheses rejected at the given level. Notice that Equation 5 does not necessarily behave monotonically, and it is possible for instance to have $P_{(2)}$ not follow the above inequality even when $P_{(3)}$ does. In this case both null hypotheses are still rejected. It is possible to also write the FDR procedure in equivalent terms using adjusted p values. In our case, the hypothesis tests can certainly be negatively correlated and therefore we need to use the most conservative formulation.

The proposal by Meinshausen et al. is a slight modification of FWER and FDR rates. For some $\gamma \in (0, 1)$ they define:

$$Q(\gamma) = \min\left\{1, q_\gamma\left\{P_{(j)}/\gamma;\, j = 1, \ldots, |B|\right\}\right\} \tag{6}$$

Where q_γ is the empirical γ-quantile function. For a fixed value of γ this value serves as a valid adjusted p value; as the power of this procedure depends greatly on the choice of the quantile, a great improvement can be given by adaptively searching over a range of quantiles. It has been shown that this gives valid p values with the addition of a extra constant. Specifically, by fixing a minimum value γ_{min} define the following adjusted p value:

$$Q' = \min\left\{1, (1 - \log \gamma_{min}) \min_{\gamma \in (\gamma_{min}, 1)} Q(\gamma)\right\}. \tag{7}$$

We refer to this as the QA (quantile adjusted) p value. The authors suggest setting γ_{min} equal to 0.05, which gives a multiple of just less then 4. An easy way to relate this procedure to the others is to consider a set of possible values of γ consisting of $\{j/|B|,\, j = 1, \ldots, |B|\}$. Then, a slightly less powerful version of the above Q' is given as:

$$Q'' = \min\left\{1, (1 + \log |B|) \min_j \frac{P_{(j)}}{j}\right\} \tag{8}$$

In this form, the QA proposal appears similar to that of the FDR rate. The former has a slightly higher constant multiple, but benefits from continuously moving between observed p values. A true difference between these two methods comes when the number of bootstraps is large, so that one may reasonably pick γ_{min} to be larger than $|B|^{-1}$. The authors suggest a minimum value of gamma around 0.05, so this becomes a true factor when using more than a few dozen bootstrap replicates. Given the similarities, however, we lump FDR and QA together in the following discussion.

In our setting we care only if we choose to reject at least one or none of the hypotheses, as the null hypotheses are all the same. Notice that there is not a uniformly better choice between FWER and FDR/QA methods in this case. Consider for instance testing m hypotheses and getting the first p value to be some small value q and the other $m-1$ tests give a value of 1. We will reject the first hypothesis using FWER at the α level if $q \leq \alpha/m$ and using FDR/QA if $q \leq \alpha/(m\,c(m))$. Obviously the FWER will have a higher power (the same power is given by the FDR method if we were able to assume the tests were independent) in this case. In contrast, consider having all m of the p values being equal to some value q. The FWER will again only reject if $q \leq \alpha/m$ whereas FDR/QA rejects as long as $q \leq \alpha/c(m)$. Given that the function $c(m)$ can be approximated by $log(m) + 0.577$, the FDR/QA test will have a higher power; this difference will be quite drastic when the number of hypothesis tests is large.

In general, the false discovery rate and quantile adjusted methods are more powerful when there is a large number of tests and a large number of relatively small p values. Conversely, the family-wise error rate is more powerful when there is a small number of tests or one very small p value but a large number of bigger values. The FWER is a good alternative for quick simulation runs when the number of bootstrap trials is significantly reduced. The adaptive QA proposal is suggested only when one desires to determine a truly stable p value by using a very high number of bootstrap samples; its limiting behavior with a fixed γ_{min} is a bit more reliable than the FDR procedure, however for smaller runs it has a tendency to be marginally less powerful.

Given that no proposal is uniformly preferable, the package **hdlm** allows for specifying a particular method. As a default we use the QA method with a fixed $\gamma = 0.5$; in other words, we take the median across all bootstrap runs.

2.3 Obtaining confidence intervals and point estimators

Typically, regression tables give either confidence intervals or standard errors in addition to p values. In ordinary linear least squares regression these quantities all give equivalent information, albeit with a different focus; in other situations such as robust or quantile regression where the distribution of estimated coefficients is not assumed to be normal this is no longer true. In the single-split variant of two-stage high dimensional regression, the second stage is an ordinary linear regression and therefore reporting either of both of these quantities is not terribly difficult. With the multi-split method, the particulars of how to combine different runs makes calculating either for the final regression table somewhat non-trivial. Here we will concentrate on a method for constructing asymptotically valid confidence intervals; standard errors are not calculated as they are not a particularly helpful quantity to estimate when using biased point estimators.

We wish to construct confidence intervals, individually for each variable, using a similar procedure as used to construct p values from Section 2.2. Benjamini and Yekutieli [2] have written about a multiple confidence interval generalization to their FDR multiple hypothesis procedure. Unfortunately, unlike in the hypothesis setting, we find that having all of the confidence intervals correspond to the same quantity changes the natural method

for combining confidence intervals and therefore require some new theory. The main difficulty lies in the fact that confidence intervals from alternate bootstrap runs may in fact be disjoint; considering that the distribution of our multistage method is multi-modal, such a situation in fact arises quite frequently. Additionally, the proposed generalization of FDR to confidence intervals can occasionally yield no solution; that is, none of the confidence intervals are chosen, leaving us with no estimated interval. Such a situation seems inadequate given that a trivial conservative confidence interval can be constructed by a simple convex union of confidence intervals across all bootstrap runs.

In order to address the issue with FDR confidence intervals, we propose an alternative procedure for aggregating confidence intervals across simulation runs. We do not suppose a particular process for the construction of confidence intervals, but rather allow for the use of any valid confidence interval construction procedure. Let $(L_{j,b}, U_{j,b})$ be the confidence interval for the coefficient β_j from the b-th bootstrap replicate. For a fixed $\alpha \in (0,1)$, $k \in \{1, 2, \ldots \lfloor (m+1)/2 \rfloor\}$ and co-ordinate $j \in \{1, 2, \ldots p\}$, define the confidence intervals:

$$CI_j(\alpha) := \left(L_{j,(k)}(\alpha),\, U_{j,(m-k+1)}(\alpha)\right), \tag{9}$$

Where $L_{j,(k)}$ and $U_{j,(m-k+1)}$ are the k-th and $(m-k+1)$-th order statistics, respectively. If the original confidence intervals are valid, then we have:

$$\mathsf{P}\left[\beta_j \in CI_j(\alpha)\right] \geq 1 - \alpha \cdot (2k-1), \tag{10}$$

Individually for each coefficient β_j. The details of the probability calculation can be found at `http://euler.stat.yale.edu/~tba3/thesis/ch2`, as well as in the package's supplemental documentation.

We note a few interesting properties of the proposed confidence intervals. Notice that if $k = 1$, the resulting confidence interval is simply the convex hull of the intersection of all bootstrapped confidence intervals. Consequently, the corresponding significance of the resulting interval CI_j is $1 - \alpha$; therefore, we see that when $k = 1$ our method collapses to the union bound. In contrast, consider for the moment setting $k = (m+1)/2$, The resulting confidence interval corresponds to taking the maximum lower bound together with the minimum upper bound; the resulting confidence interval should have a significance level of $1 - \alpha \cdot m$. In other words, when $k = m$ we are using a Bonferroni correction and (assuming all of the lower bounds are less than all of the upper bounds) taking the intersection of the resulting intervals. We see then that our method allows for choosing an interval somewhere between the extremes of the Bonferroni correction and the union bound.

The size of k is restricted due to the fact that for larger k it is possible that the resulting confidence interval is in fact an empty set. Regardless of the validity of the confidence intervals given in Equation 9, the resulting $CI_j(\alpha)$ is non-empty given that $m + 1 \geq 2 \cdot k$; that is, we have the right end-point of the interval no larger than the left end-point. We can see this from the following simple calculation:

$$L_{j,(k)}(\alpha) \leq L_{j,(m-k+1)}(\alpha) \leq U_{j,(m-k+1)}(\alpha) \tag{11}$$

Therefore the collapsed confidence intervals $CI_j(\alpha)$, which are given in the the form of $\left(L_{j,(k)}(\alpha),\, U_{j,(m-k+1)}(\alpha)\right)$, must necessarily be non-empty. Notice that this property does not hold uniformly for higher values of k; as an example take a set of mutually disjoint confidence intervals. Our method does not typically provide the smallest possible intervals; using $\widetilde{A}_h$ for a particular order of the indices would, for instance, typically give smaller intervals. However, our construction is a conservative choice, but should not be overly conservative, and has the nice property of always constructing non-empty confidence intervals.

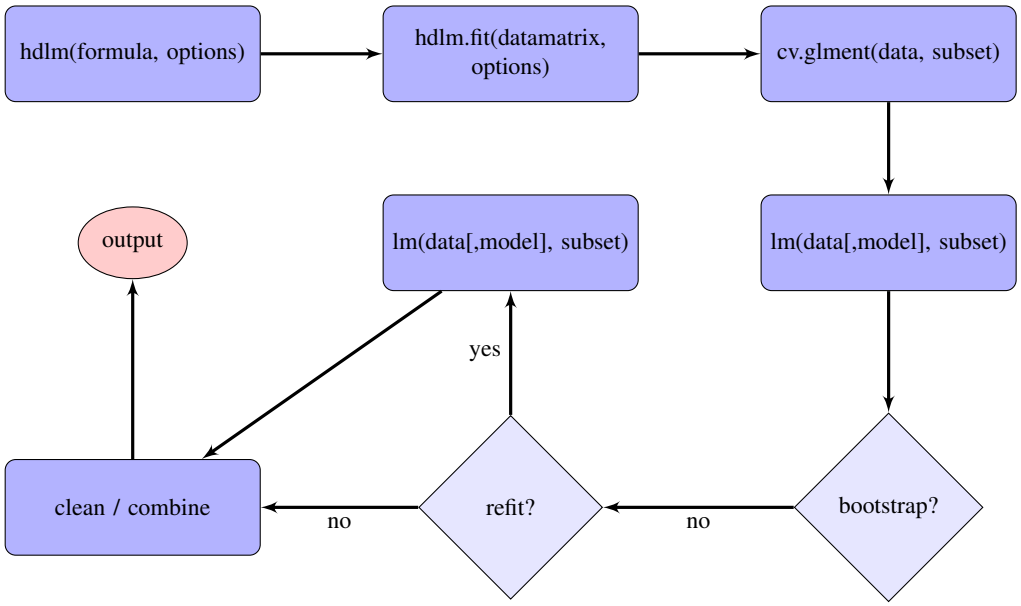

Figure 2: Primary decision tree of function `hdlm` under default options.

Given a confidence interval $CI_j(\alpha)$ as in Equation 9, we use the following definition of a point estimator for β:

$$\widehat{\beta} := \text{median}\left\{\beta_{j,b} \,:\, \beta_{j,b} \in CI_j(\alpha)\right\} \tag{12}$$

This assures that the point estimator lies within the confidence interval. Use of the median rather than mean or other measure comes from the special status of coefficient estimates which are set exactly the zero. If a majority of the time, a coefficient is not included in the final model it makes sense to set it equal to exactly zero. On the other hand, if a coefficient is usually estimated to be around 1 but not included in about 20% of the model selection steps, a point estimator around 1 is better than one around 0.8. The median assures that both of these properties hold. We avoid giving a detailed convergence result regarding either point estimator, as convergence depends in a complex way on the underling data matrix as well as on the chosen model selection and confidence interval procedures. Obviously, since we are using a sample median, if the single-split procedure yields asymptotically consistent point estimators and the number of bootstrap trials is fixed, our resulting point estimator be consistent as well.

We note that a possible alternative point estimator would be to use the average of $L_{(\lfloor (m+1)/2 \rfloor)}$, and $U_{(\lfloor (m+1)/2 \rfloor)+1}$. Such a method is not likely to have the property of setting additional coefficients to zero, but does have the nice property that the point estimator does not depend on the confidence level α, while still having the point estimator always contained in the given confidence interval.

3. Basic design and usage

The high-level design of the package with default options is given in Figure 2. The top level function `hdlm` creates a model matrix, standardizes the variables, and parses various options. The real estimation work is done inside of `hdlm.fit`. Here, part of the dataset is analyzed by a cross validated version of the elastic net procedure. The model corresponding to the best fit is then used in conjunction with the standard ordinary least squares

function `lm`. When bootstrapping is turned on, this sub-routine is looped over for each required bootstrap run. Finally, the output is sent back up through the hierarchy of functions, cleaned, and returned to the user. When desired, the final model can also be 'refit' internally; basically, in this case the final model is used to run a reduced ordinary least squares fit on the whole dataset. Such a refitting should be done carefully, as issues of multiple hypothesis testing and the validity of the p values may come into question. Nonetheless, such a refitting procedure does seem to often produce lower mean square errors in wide range of simulation studies. When the refit option is turned on, the output is simply a standard linear regression object with two extra attributes signifying the output of the model selection steps.

Arguments available to pass to the top-level function `hdlm` come in a few different groups. Looking at the function call:

```
hdlm(formula, data, subset, bootstrap = 10, siglevel = 0.05,
  alpha = 0.5, M = NULL, N = NULL, model = TRUE, x = FALSE,
  y = FALSE, scale=TRUE, pval.method=c('median', 'fdr', 'holm',
  'QA'), ..., FUNCVFIT = NULL, FUNLM = NULL, bayes=FALSE,
  bayesIters=NULL, bayesTune = c(1,1), refit=FALSE)
```

We see some parallels between arguments for `hdlm` and those for standard `lm`. The options `formula`, `data`, `subset`, and logical arguments `model`, `x`, and `y` all intentionally behave in the same manor as for the basic linear model function. These specify the basic linear model and data at play, as well as which elements of the input should be returned in the output.

The other options exist to deal with the particular options for high dimensional regression. Many options are covered more carefully in the online documentation; we give a brief description of each here:

- `bootstrap`: number of bootstrap trails to conduct when bayes=FALSE. Default is 10.
- `siglevel`: significance level to use for confidence bounds. Default is 0.05.
- `alpha`: elastic net mixing parameter sent to default FUNCVFIT, can be any value in (0,1]. When alpha = 1, this is the lasso penalty and when alpha = 0 (not supported) this is the ridge penalty. See the package glmnet for more details.
- `M`: maximum model size sent to the second stage low-dimensional regression. When more than M variables are chosen in the first stage, the model is trimmed by succesively taking larger sized coefficients until only M remain. If NULL, M is taken to be 90% of the number of samples in the second stage. If M = 0, the model is fit with all of the data once, and the estimated parameters are returned as is.
- `N`: Number of observations to include in the first stage regression. Default is (# samples / 2), so that the data is split evenly amongst the two stages; will be set when N=NULL.
- `scale`: Logical; should the variables in the data matrix be scaled. Default is TRUE.
- `pval.method`: one of 'median', 'fdr', 'holm', or 'QA'. Signifies the method used to combine p-values when bootstrap is greater than 1. For details and relative strengths of the three methods, see the package vignette.

- `...`: additional arguments to be passed to the low level regression fitting functions (see below).

- `FUNCVFIT`: Used to pass an alternative model selection function. Must accept data matrix as its first element and response vector as second element. Return should be a vector of length p (the number of regressors), which indicates which variables are included in the final model. Zero terms are considered to be out of the model; typically all non-zero terms are treated as in the model, though if the model size is too large (see 'M' above), it will be trimmed relative to the absolute size of each non-zero term. Therefore, it is advised to return the model vector in a relative scale rather than an absolute one. The default, used when NULL, is the elastic net function from package glmnet, with the mixing parameter alpha from above. See Section 4 for more details and examples.

- `FUNLM`: Used to pass alternative second stage, low-dimensional function. Must accept as its first argument a formula object. The return class must have a summary method and the summary method in turn must have a coef method. The coef.summary should return a matrix where the first column contains the coefficients and the second column contains standard errors. Intercepts should be handled according to the passed formula. As an example, stats::lm works by default; stats::lm is additionally the default when FUNLM is set to NULL. See Section 4 for more details and examples.

- `bayes`: logical. Should Bayesian method be used in place of the two stage method.

- `bayesIters`: number of iterations to conduct in the Gibbs sampler when bayes is set to TRUE. A total of (bayesIters * 0.1) burn-in steps are included as well. The default is 1000.

- `bayesTune`: numerical vector tuning parameter for the Bayes estimator. Defines a Beta(bayesTune[1], bayesTune[2]) prior on the proportion of variables included in the true support.

- `refit`: Either a logical or number in (0,1]. When not equal to false, the final model will be refit from the entire dataset using FUNLM. When a numeric, the model is selected by only including variables with p-values less than refit. When set to TRUE, any variable corresponding to a non-zero p-value is included.

Special care has been taken to deal sensibly with sparse design matrices. We use the package Matrix to offer support for sparse matrix representations. This is particularly useful when dealing with data which has factors with a large number of levels; this occurs frequently in word count and network data.

4. General techniques for extension

4.1 Overriding elastic net

As mentioned, the default behavior for `hdlm` is to use the elastic net procedure of package **glmnet**. Our package has been written in such a way as to allow for using other model selection routines. Rather than incorporating a bulky set of predefined options, instead we have a simple technique for using any desired function by making sure the new function outputs values in the same format as `glmnet`.

As an example, consider the following penalized model selector know as the Dantzig selector [5, 6] given by:

$$\widehat{\beta} = \arg\min_{b} ||X^{\top}(Y - X\beta)||_{\infty} + \lambda||\beta||_{1}. \tag{13}$$

The following code produces the Dantzig selector for a particular tuning parameter λ via the **quantreg** package [9]:

```
R> dselector <- function(x,y){
+   n <- nrow(x)
+   p <- ncol(x)
+   lambda <- sqrt(log(ncol(x)+1) * nrow(x)) * sd(as.numeric(y))
+
+   A <- t(x) %*% x
+   R <- rbind(A, -A)
+   a <- c(as.matrix(t(x) %*% y))
+   r <- c(a-lambda, -a-lambda)
+   beta <- quantreg::rq.fit.fnc(diag(p), rep(0,p),
+         R=R, r=r)$coefficients
+   return(round(beta, 6))
+ }
```

This solution of the Dantzig selector was originally proposed by Roger Koenker, with code found at `http://www.econ.uiuc.edu/~roger/research/sparse/sfn.html`. In some cases, we may wish to tune the model via some form of cross-validation. For instance, the default elastic net procedure does this via 10-fold prediction loss. With the Dantzig selector, to keep our example simple, we simply use the analytical value $\lambda = \sigma \cdot \sqrt{n \log(p)}$. Notice that the return of the function is a vector of length p, where non-included variables are set to zero. The actual values are generally not used, unless too many variables are given to run low-dimensional routines in the second stage.

Having defined the Dantzig selector, it is easy to run `hdlm` with the new model selector in place of the standard elastic net:

```
R> hdlm(y ~ x, FITCVFUN = dselector)
```

While it would have been possible to include appropriate `FITCVFUN` arguments into the package **hdlm**, we have not done so for two reasons. First of all, it would require having an exceptionally large number of linked packages; while these could be loaded as needed it seemed to unnecessarily complicate what we feel is a currently streamlined package. Secondly, this would require making sure any necessary packages do not change their output or arguments over time. As we may wish to use cross validation routines, many of which are internal functions of packages rather than front-facing commands, this is likely to be the case going forward for at least a few options. As an alternative, we give basic transformations as supplementary material available separately in the the package documentation. This option allows for users to more easily adapt code as packages change over time.

4.2 Replacing ordinary least squares

Just as with the model selection routine, the function `hdlm` allows for replacement of the low dimensional fitting algorithm, the default of which is ordinary least squares through the use of `lm`. The method must be able to provide a point estimator and p values for the

hypothesis tests $\beta_i = 0$ for any i. Possible alternatives include Bayesian regression, ridge regression, and MM-regression. Here we demonstrate the procedure for generalization by implementing the quantile regression as an alternative to least squares.

The function supplied to the FUNLM argument requires two things:

1. To return an object which has a corresponding summary method.

2. For the summary method to produce an object with a coef method which produces a matrix with a first column of coefficients and a second column of standard errors.

Such conditions may seem slightly arbitrary, but coincide with the behavior of the lm function.

The basic quantile regression procedure has been already coded in the package **quantreg**. What remains for us is to turn the default output of **quantreg** into the form specified above. The sole difficulty lies in the fact that the summary method for **quantreg** does not produce p values by default, but does so only via an option to the summary command. In order to address this, we define a wrapper function for the the `rq` function which does nothing different save using a different class for the output:

```
R> library("quantreg")
R> rq2 <- function(formula) {
+   out <- rq(formula)
+   class(out) <- "rq_new"
+   return(out)
+ }
```

We now define a new summary method for this new class which uses different default parameters to force a calculation of standard errors:

```
R> summary.rq_new <- function(out) {
+   class(out) <- 'rq'
+   val <- summary.rq(out, se='nid')
+   return(val)
+ }
```

Now, we can run `hdlm` with quantile regression with the following:

```
R> out <- hdlm(y ~ x, FUNLM=rq2)
```

Similar tricks can be used to have `hdlm` work with a wide range of low dimensional estimators; essentially anything that is able to report standard errors can be adapted without having to change any of the code in the **hdlm** package.

4.3 Generalized models

As generalized linear models are often encountered in high dimensional data, particularly binomial responses in social and laboratory sciences, we have included a pre-built function for dealing with such responses. We have done this easily by using the default `hdlm` function with different arguments for the high and low dimensional estimators. The high dimensional estimator uses the generalized elastic net as described by [7]. The low dimensional estimators uses the `glm` function found in the standard R package stats.

While wrapped with additional checks and options, the basic method of extension can be achieved by the following code for the binomial model:

	self.time	self.pct	total.time	total.pct
.Fortran	10.41	54	10.41	54
matrix	1.56	8.07	1.80	9.35
glmnet	1.18	6.11	14.71	76.25
as.double	0.69	3.59	0.69	3.59
cbind2	0.52	2.68	0.52	2.68
FUN	0.40	2.07	2.31	11.97
standardGeneric	0.33	1.71	1.17	6.06
double	0.30	1.58	0.30	1.58
t.default	0.27	1.42	0.27	1.42
sort	0.27	1.38	1.50	7.76
seq.default	0.25	1.30	0.25	1.32
nrow	0.22	1.13	0.29	1.49
sort.int	0.21	1.10	1.19	6.16
list	0.18	0.94	0.18	0.94
deparse	0.17	0.86	0.57	2.94
.deparseOpts	0.15	0.78	0.23	1.20
match.arg	0.13	0.67	0.94	4.85
eval	0.11	0.57	0.92	4.76
pnorm	0.09	0.49	0.11	0.58

Table 1: Profile output of `hdlm` with data generated from 250 observations, $10,000$ variables, 10 bootstraps, β with a support of size 1, and a signal to noise ratio of about 2. Only processes with the highest self time are listed.

```
R> LMFUN <- function(x,y) return(glm(y ~ x,
+       family=binomial(link=logit)))
R> FUNCVFIT <- function(x,y) return(cv.glmnet(x, y,
+       family='binomial'))
R> out <- hdlm(y ~ x, LMFUN = LMFUN,
+       FUNCVFIT = FUNCVFIT)
```

The resulting object could be displayed, plotted, and manipulated as with a generic hdlm output. Our actual implementation differs slightly; while the main regression table is the same as this simple example, ours correctly calculates the residuals and allows for calling the particular Bayes solution built specifically for binomial data.

5. Computational performance

Given that the methods described here are meant to be used with large, high-dimensional datasets, it is important to verify that there are no critical performance issues when using such data.

Table 1 gives a run profile of the `hdlm` function for a typical dataset. We see that over half of the time the function is processing Fortran code. Since there is a high degree of efficiency in compiled Fortran, this indicates that it would be hard to speed up our function by more than a factor of 2 by just changing the code's syntax. We would need to have fundamentally different algorithms underlying our method. As the most efficient algorithms for fitting high dimensional linear models have been used, therefore it seems that our code is about as fast as we could expect it to be, at least in the regime profiled in Table 1.

Sample Size	10	10	100	100	1000	1000	1000
Model Size	100	500	1000	5000	1000	5000	25000
hdlm time (bootstrap=10)	0.57	0.76	4.01	12.59	33.91	97.80	472.17
glmnet time	0.06	0.09	0.31	1.05	3.08	9.05	45.79

Table 2: Comparison of run times (in seconds) between `hdlm` and `cv.glmnet`. Notice that the latter calls the former 10 times. A β with a support of size 1, and a signal to noise ratio of about 2 was used throughout.

Number of Bootstrap Runs:	2	20	100	250
1 core	1.43	9.07	44.20	82.75
2 cores	1.21	5.93	23.11	42.29
4 cores	1.15	3.91	20.63	28.05

Table 3: Comparison of run times (in seconds) for various number of used cores and number of bootstrap runs.

Actual run times for various sample and model sizes for default options are given in Table 2. As seen in the profiling table, the computation time of `hdlm` is dominated by calls to `cv.glmnet`. For model sizes less than about 500, the code runs fast enough to consider results coming in 'real time'. Models approaching 25 thousand variables take long enough that a user would likely set it up while performing another task, as it may take upwards of 10 minutes. While this may seem like a long time, given that (outside of simulation runs) such large datasets typically take a very long time to clean and acquire, a 10 minutes wait for an analysis of the data is generally not terribly inhibitive.

Using **foreach** by [19], we are able to provide a parallel execution for both the frequentist and Bayesian hdlm implementation. This package allows the user to declare a number of parallel backends (e.g., MPI, or multicore), so that the most appropriate method for a given workstation or cluster may be used. The frequentist `hdlm` method can be implemented as an embarrassingly parallel algorithm, where different cores compute different bootstrap runs. The Bayesian implementation on the other hand, is executed in parallel by conducting independent MCMC runs and pasting all of the runs together before calculating p values, point estimators, and confidence intervals. The computational performance of the parallelized code is given in Table 3, which uses the parallel backend provided by **doMC**. These runs were performed on a single four core server, though given the lack of communication between threads, similar performance is expected when running in other architectures. We see, unsurprisingly, that as the number of bootstrap runs increases, the advantage of a parallel execution increases. With a high bootstrap count, the advantage appears to (and should) approach near perfect parallelization.

6. Further discussion

We have presented the basic functionality and design choices of the package **hdlm**. Our hope is that it will be be a useful tool for data analysis and as well as for testing new model selection algorithms. As previously discussed, additional linking functions which allow for a wide variety of options and model selectors are included as additional R code in the package documentation available via CRAN.

While the primary purpose of the package is to construct regression tables, we have found two related uses which work quite nicely. By setting the option `M = 0`, the function

returns a simple point estimator using the inputted model selection routine allowing the package to serve as a convenient wrapper function when applying a variety of algorithms to a single dataset. On a separate point, by setting the option `refit = TRUE` all variables with a p value below a given cut-off are used to refit a model using the whole set of observations. While the resulting p values and standard errors cannot be trusted, the predicted point estimator often outperforms the unfit version in terms of both parameter estimation and prediction. See, for instance, [17] for more details.

A planned future extensions of **hdlm** is to extend the current methods for sparse linear models to methods for learning the structure of sparse graphical models. As a large amount of high dimensional data analysis is presented in a network form, such as social network data, gene interactions, and citation networks, this a generalization would be extremely useful. This extension is a not a simple task to do well; particularly when considering the increased computational cost. A simple, node based implementation as suggested by [15] for instance would require running the equivalent of hdlm separately on each variable. Additionally, since no low-dimensional template currently exists, even the format of a high dimensional graphical model selector must be created from scratch.

References

[1] Y. Benjamini and Y. Hochberg. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society B*, pages 289–300, 1995.

[2] Y. Benjamini and D. Yekutieli. False discovery rate-adjusted multiple confidence intervals for selected parameters. *Journal of the American Statistical Association*, 100(469):71–81, 2005.

[3] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *The Annals of Statistics*, 29(4):pp. 1165–1188, 2001.

[4] R. Berk, L. Brown, and L. Zhao. Statistical inference after model selection. *Journal of Quantitative Criminology*, 26(2):217–236, 2010.

[5] P.J. Bickel, Y. Ritov, and A.B. Tsybakov. Simultaneous analysis of lasso and dantzig selector. *The Annals of Statistics*, 37(4):1705–1732, 2009.

[6] E. Candes and T. Tao. The dantzig selector: Statistical estimation when p is much larger than n. *The Annals of Statistics*, 35(6):2313–2351, 2007.

[7] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010.

[8] C. Hans. *Brief Technical Report to Accompany the R Package **blasso** Bayesian Lasso Regression*, 2009.

[9] Roger Koenker. ***quantreg**: Quantile Regression*, 2011. R package version 4.71.

[10] M. Kyung, J. Gill, M. Ghosh, and G. Casella. Penalized Regression, Standard Errors, and Bayesian Lassos. *Bayesian Analysis*, 5(2):369–412, 2010.

[11] K.E. Lee, N. Sha, E.R. Dougherty, M. Vannucci, and B.K. Mallick. Gene selection: A bayesian variable selection approach. *Bioinformatics*, 19(1):90–97, 2003.

[12] H. Leeb and B.M. Pötscher. Model selection and inference: Facts and fiction. *Econometric Theory*, 21(1):21–59, 2005.

[13] H. Leeb and B.M. Pötscher. Can one estimate the conditional distribution of post-model-selection estimators? *The Annals of Statistics*, 34(5):2554–2591, 2006.

[14] N. Meinshausen, L. Meier, and P. Bühlmann. P-values for high-dimensional regression. *Journal of the American Statistical Association*, 104(488):1671–1681, 2009.

[15] Nicolai Meinshausen and Peter Buhlmann. High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics*, 34,3, 2006.

[16] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

[17] S. van de Geer, P. Bühlmann, and S. Zhou. The adaptive and the thresholded lasso for potentially misspecified models (and a lower bound for the lasso). *Electronic Journal of Statistics*, 5:688–749, 2011.

[18] L. Wasserman and K. Roeder. High dimensional variable selection. *The Annals of Statistics*, 37(5A):2178, 2009.

[19] Steve Weston. ***foreach****: Foreach Looping Construct for R*, 2011. R package version 1.3.2.

[20] Y. Yang. Can the strengths of aic and bic be shared? a conflict between model identification and regression estimation. *Biometrika*, 92(4):937–950, 2005.