Package hdlm: Regression Tables for High Dimensional Linear Model Estimation

Taylor B. Arnold
Yale University

Abstract

We present the R package **hdlm**, created to facilitate the study of high dimensional datasets. Our emphasis is on the production of regression tables and a class 'hdlm' for which new extensions can be easily written. We model our work on the functionality given for linear and generalized linear models from the functions `lm` and `glm` in the recommended package **stats**. Reasonable default options have been selected so that the package may be used immediately by anyone familiar with the low dimensional variants; however, a generic procedure for using alternative point estimators is also provided.

Two techniques are given for constructing high dimensional regression tables. The first uses the the two-stage approach of Wasserman and Roeder (2009), with the generalization proposed by Meinshausen, Meier, and Bühlmann (2009) to increase robustness, in order to calculate high-dimensional p values. We introduce and implement a novel method for generalizing these p value methods to confidence intervals. The second technique constructs regression tables using a hierarchical Bayesian approach solved via Gibbs Sampling MCMC. In this article, we focus on design choices made in the package, relevant computational issues, and approaches to changing the default options.

Keywords: penalized estimation, regression tables, Gibbs sampling, R.

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 Wasserman and Roeder (2009) for the purpose of obtaining asymptotically valid high dimensional p values. This theory was further extended by Meinshausen *et al.* (2009) 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, Hans (2009), Lee, Sha, Dougherty, Vannucci, and Mallick (2003), and Kyung, Gill, Ghosh, and Casella (2010). With the exception of the work done by Hans (2009), 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 a default the highly optimized package **glmnet** (Friedman, Hastie, and Tibshirani 2010). 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 outlines the various methods pertaining to the class 'hdlm', while Section 5 gives explicit examples of how various default options can be altered to incorporate new point estimation techniques. Section 6 gives an outline of the Bayesian alternative to frequentist regression tables. Section 7 illustrates the computational speed of the default algorithms. Finally, Section 8 gives a brief application to textual word-count data and Senate voting records and Section 9 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 (Leeb and Pötscher 2005), Pötscher (Leeb and Pötscher 2006, 2008; Leeb and Pöetscher 2003), and Yang (Yang 2005) 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 Berk, Brown, and Zhao (2010).

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 Wasserman and Roeder (2009) 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.

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 Meinshausen *et al.* (2009) 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.

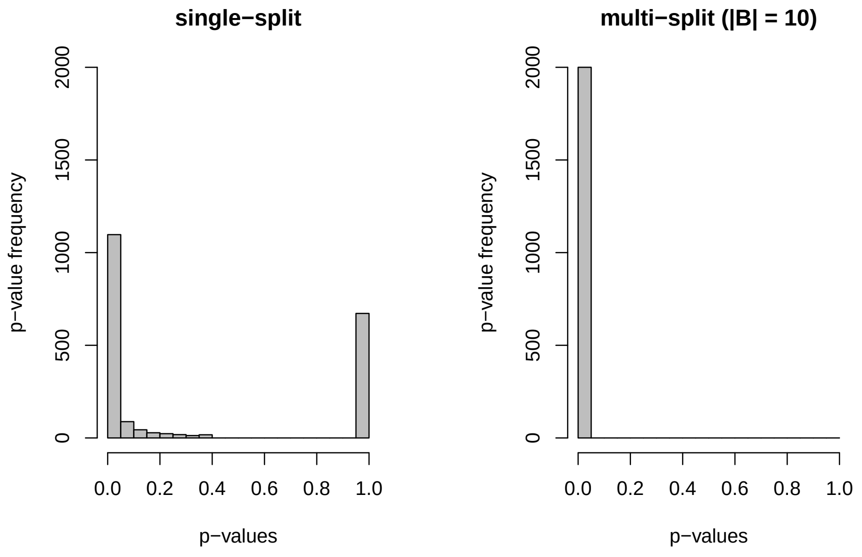

Figure 1: Distribution of p values in a simple simulation with $p = 50$, $n = 25$, and $\beta^t = (1, 0, \ldots, 0)$, over 2000 trials. The lasso is used, with 10-fold cross validation to select the tuning parameter. The left plot shows the p values resulting from the single-split method, whereas the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

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 discovery 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 (Benjamini and

Hochberg 1995; Benjamini and Yekutieli 2001). 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 the above equation 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 in the previous section. Benjamini and Yekutieli (Benjamini and Yekutieli 2005) 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.

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

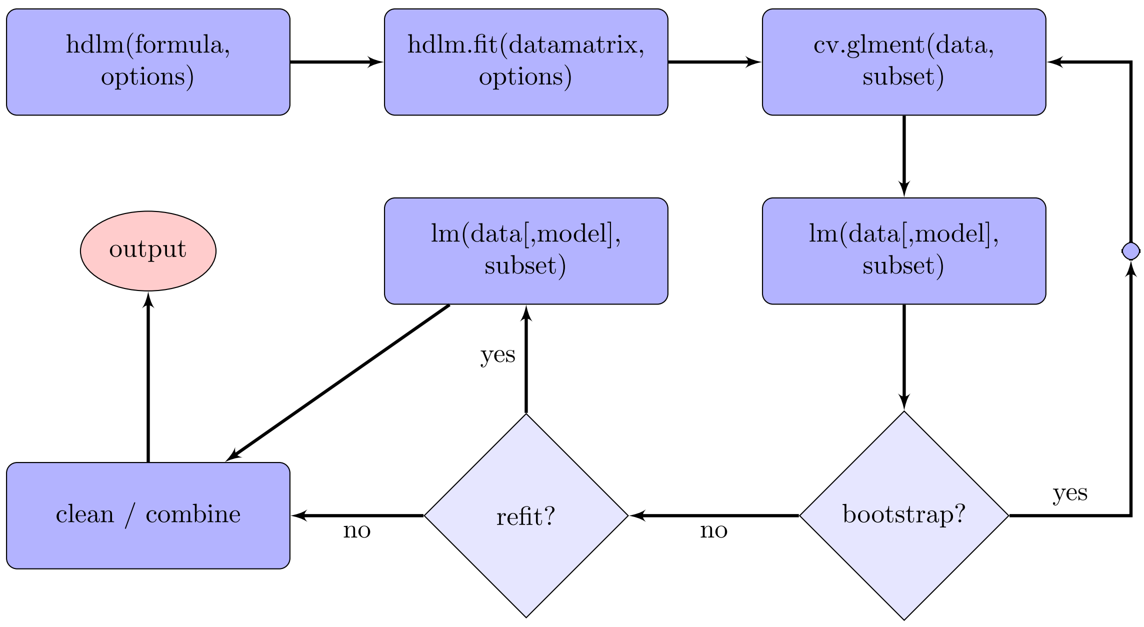

Figure 2: Primary decision tree of function `hdlm` under default options.

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 following sections; 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 5 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 5 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=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. Methods Provided for Class hdlm

4.1. Overview

The R language supports some basic elements of object oriented programming. Every object in R has a vector of classes to which it belongs. Mirroring these classes are generic methods which can be applied to objects and coded to behave differently depending on the class of the given object. For example, the method `summarize` will return a summary of columns when given a dataframe, but will return a regression table when given an object of class 'lm'.

add1.lm	alias.lm	**anova.lm**	**case.names.lm**
confint.lm	cooks.distance.lm	**deviance.lm**	dfbeta.lm
dfbetas.lm	drop1.lm	dummy.coef.lm	**effects.lm**
extractAIC.lm	**family.lm**	**formula.lm**	hatvalues.lm
influence.lm	kappa.lm	**labels.lm**	logLik.lm
model.frame.lm	**model.matrix.lm**	nobs.lm	**plot.lm**
predict.lm	**print.lm**	**proj.lm**	**qr.lm**
residuals.lm	rstandard.lm	rstudent.lm	**simulate.lm**
summary.lm	**variable.names.lm**	vcov.lm	

Table 1: Default methods for class *lm*. Bold methods have a corresponding method for class *hdlm*. Method anova.hdml does exist, but only produces a warning.

The use of this object oriented framework theoretically allows for the creation of an easy to read and quickly generalizable codebase. In practice, this extensibility is not utilized as extensively in R as it is in a language such as C++; despite this, method and classes still provide a clean interface to new data types and reduce the learning curve for users working with new packages.

Two competing paradigms for object oriented programming are currently used within R: an older S3 method and the newer S4. The latter offers far more extensibility, but suffers in also being quite a bit more difficult to implement. For the package **hdlm**, we have chosen to stick with the older S3 model; the primary reason for this choice was based on the fact that classes 'lm'/'glm' still use the older model. As we wanted to mimic these basic and familiar classes as much as possible, sticking to the original object oriented S3 methods seemed the more appropriate choice.

A final general point to make regarding classes is the notion of class inheritance. Objects may possess a hierarchical sequence of classes ranging from specific to general. When passed to a generic method, first a search is done for a method corresponding to the most specific class of the object. If this fails, then methods for less specific classes are then searched. This hierarchy of classes, known as class inheritance, allows for creating specific sub-classes without having to redefine every method. Given the very specific structure of the 'lm' class, we are not able to benefit from inheritance by having class 'hdlm' be a sub-class of 'lm'. However, we are able to benefit from the fact that (unlike with 'lm' and 'glm') the class 'hdglm', for generalized linear models, is able to inherit most of the methods for class 'hdlm'.

A list of all methods produced in the default build of R for class lm is given in Table 1. Methods which have been written for 'hdlm' are denoted in bold. The majority of these are fairly straightforward re-writes of the code for 'lm'. Those methods which we have not re-created generally either do not have a good analogue to high-dimensional models, or cannot be calculated efficiently without simply re-running the function. The two instances where we have made substantive changes are in regards to printing and plotting, which we extrapolate on in the next two subsections.

4.2. Printing

In printing output from a run of the function `hdlm`, we again felt it was most helpful to mimic the original lm output. Therefore, printing the raw output object gives only a terse overview of the results. As some models might have a very large number of variables we do not print out the vector of coefficients but rather give a few basic facts regarding the final model:

```
R> library('hdlm')
R> set.seed(1)
R> x <- matrix(rnorm(100*40),ncol=100)
R> y <- x[,1] + x[,2] * 0.5 + rnorm(40, sd=0.1)
R> out <- hdlm(y ~ x, , bootstrap = 2, pval.method="fdr")
R> out

Call:
hdlm(formula = y ~ x, bootstrap = 2, pval.method = "fdr")

Parameters:
 Observations = 40, Variables = 100
```

Obviously, much more information is packed into the output, but this simple return gives a good summary of what model was given and the respective arguments to the function `hdlm`. In order to get a more complete summary of the data, the summary method can be applied to the output. Given that, again, there will likely be a large number of variables, by default only those which have p values less than 1 are listed:

```
R> summary(out)

Call:
hdlm(formula = y ~ x, bootstrap = 2, pval.method = "fdr")

Residuals:
     Min       1Q   Median       3Q      Max
-0.26905 -0.05898  0.07646  0.18221  0.55557

Coefficients:
            Estimate Lower Bd Upper Bd  P-value
(Intercept)   0.0446  -0.4777   0.3761 0.635525
x1            0.8761   0.5921   1.0530  < 2e-16 ***
x2            0.4109  -0.1353   0.7855 0.000368 ***
x8            0.0786  -0.1096   0.2219 0.375558
x41           0.0406  -0.1327   0.1327 0.366392
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Estimated sigma: 0.1827
```

Notice that we have a complete regression table, description of the residuals, and even a summary of the number of variables which were selected for (any) model in the bootstrap simulation. The major differences from the lm summary table is the absence of an F-test and use of confidence intervals rather than standard errors. The lack of an F-test is primarily due to the fact that it doesn't seem particularly useful, at least in its present form, in the context of high dimensional models. The use of confidence intervals is preferred since standard errors are a somewhat deviant characteristic of a biased estimator.

In some settings it may be useful to have a larger subset of coefficients listed in the regression table. This may occur when there is a low signal to noise ratio (so no variables have p values not equal to 1), highly correlated variables (so, again, no variables have p values not equal to 1), or when p is actually reasonably small so that a larger table is not too cumbersome. In this case, the argument `level = 2` can be passed to the summary command to list any variable with a non-zero coefficient:

```
R> summary(out, level=2)

Call:
hdlm(formula = y ~ x, bootstrap = 2, pval.method = "fdr")

Residuals:
     Min       1Q   Median       3Q      Max
-0.26905 -0.05898  0.07646  0.18221  0.55557

Coefficients:
             Estimate  Lower Bd  Upper Bd  P-value
(Intercept)  0.044587 -0.477748  0.376084 0.635525
x1           0.876111  0.592095  1.053008  < 2e-16 ***
x2           0.410858 -0.135276  0.785486 0.000368 ***
x8           0.078629 -0.109636  0.221881 0.375558
x13         -0.046929 -0.316233  0.128516 1.000000
x20          0.007678 -0.223574  0.107575 1.000000
x27          0.000143 -0.151706  0.132671 1.000000
x33          0.013654 -0.258737  0.107575 1.000000
x36          0.026691 -0.132671  0.132671 1.000000
x38         -0.033098 -0.375813  0.132671 1.000000
x39          0.020809 -0.149253  0.132671 1.000000
x41          0.040635 -0.132671  0.132671 0.366392
x43         -0.036838 -0.300074  0.132671 1.000000
x51          0.059159 -0.358453  0.132671 1.000000
x52          0.018939 -0.145277  0.168282 1.000000
x53         -0.059619 -0.431859  0.132671 1.000000
x63         -0.018264 -0.498011  0.111566 1.000000
x64          0.004717 -0.123237  0.107575 1.000000
x67          0.025398 -0.168189  0.107575 1.000000
x70         -0.002825 -0.399767  0.107575 1.000000
x73         -0.017463 -0.186703  0.116849 1.000000
x75          0.005221 -0.132671  0.132671 1.000000
x76         -0.021678 -0.319059  0.107575 1.000000
x78          0.015344 -0.298045  0.107575 1.000000
```

```
x80         -0.031903 -0.286707  0.107575 1.000000
x86          0.019427 -0.130999  0.107575 1.000000
x87         -0.010134 -0.132671  0.087307 1.000000
x96          0.025557 -0.211184  0.107575 1.000000
x91          0.000000 -0.135727  0.132671 0.639236
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Estimated sigma: 0.1827
```

When this is desired, it can be helpful to reduce the number of bootstrap trials; a large number of trials will often result in almost all variables having coefficients which are just slightly different from zero. In the case that all variables need to be shown, the option `level = 3` can be used. For space reasons we won't print this out, but the following are the first few lines of such an output:

```
R> output <- capture.output(summary(out, level=3))
R> cat(output[1:20], sep='\n')

Call:
hdlm(formula = y ~ x, bootstrap = 2, pval.method = "fdr")

Residuals:
     Min       1Q   Median       3Q      Max
-0.26905 -0.05898  0.07646  0.18221  0.55557

Coefficients:
             Estimate  Lower Bd  Upper Bd  P-value
(Intercept)  0.044587 -0.477748  0.376084 0.635525
x1           0.876111  0.592095  1.053008  < 2e-16 ***
x2           0.410858 -0.135276  0.785486 0.000368 ***
x3           0.000000  0.000000  0.000000 1.000000
x4           0.000000  0.000000  0.000000 1.000000
x5           0.000000  0.000000  0.000000 1.000000
x6           0.000000  0.000000  0.000000 1.000000
x7           0.000000  0.000000  0.000000 1.000000
x8           0.078629 -0.109636  0.221881 0.375558
x9           0.000000  0.000000  0.000000 1.000000
```

Notice that the extra rows are generally not too interesting since their coefficients and bounds are all zero, whereas the p values are all equal to 1. Nonetheless, this output can be useful when it makes sense to display all of the variables in order or the number of variables is fairly small.

4.3. Plotting

Creating a graphical output for hdlm objects requires a substantial bit of modification to the corresponding code for standard linear models. The primary purpose of the plotting routine is to assess the fit and possible violations of the model assumptions. In standard linear models, specific measurements such as leverage and Cook's distance can be used to provide insights into the model's fit. For high dimensional models, these unfortunately do not exist and more straightforward plots are needed.

An example of the graphical from plotting an hdlm object is shown in Figure 4.3. Our first two plots are the standard fitted versus residual plot and the qqnorm residual plot. The first can be used similarly to in standard regression for assessing heteroskedasticity, non-linearity, and outliers. The qqnorm plot additionally gives us a sense of whether the residuals are approximately normal in essentially the same way it would in a traditional model. Our third plot shows the distribution of coefficients, normalized by the relative sizes of their confidence intervals. The purpose is to give an assessment of the sparsity condition of high dimensional models. The final plot produced gives a relationship between the confidence intervals and the p values of the hdlm framework. We implement this by assuming the confidence intervals come from a standard ordinary least squares procedure with normal errors to reverse engineer standard errors. These standard errors are then used to determine new p values as they would be in a traditional linear model, which are plotted against the reported p values. Ideally, these should be fairly close; in practice what we really want is to have the reported p value be the conservative of the two.

Care has been taken to produce graphics which are useful in most situations including various generalized linear models. We use only confidence intervals, p values, coefficients, and residuals/fitted values as other measurements such as observed values may exhibit clustering in such models. We also restricted ourselves to easy to calculate quantities which did not require re-loading the original dataset. Other interesting plots involving principle components or raw correlations could often be helpful, but unfortunately may be prohibitively slow for larger datasets.

5. General Techniques for Extension

5.1. 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 (Bickel, Ritov, and Tsybakov 2009; Candes and Tao 2007) 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 (Koenker 2011):

```
R> dselector <- function(x,y){
+   n <- nrow(x)
```

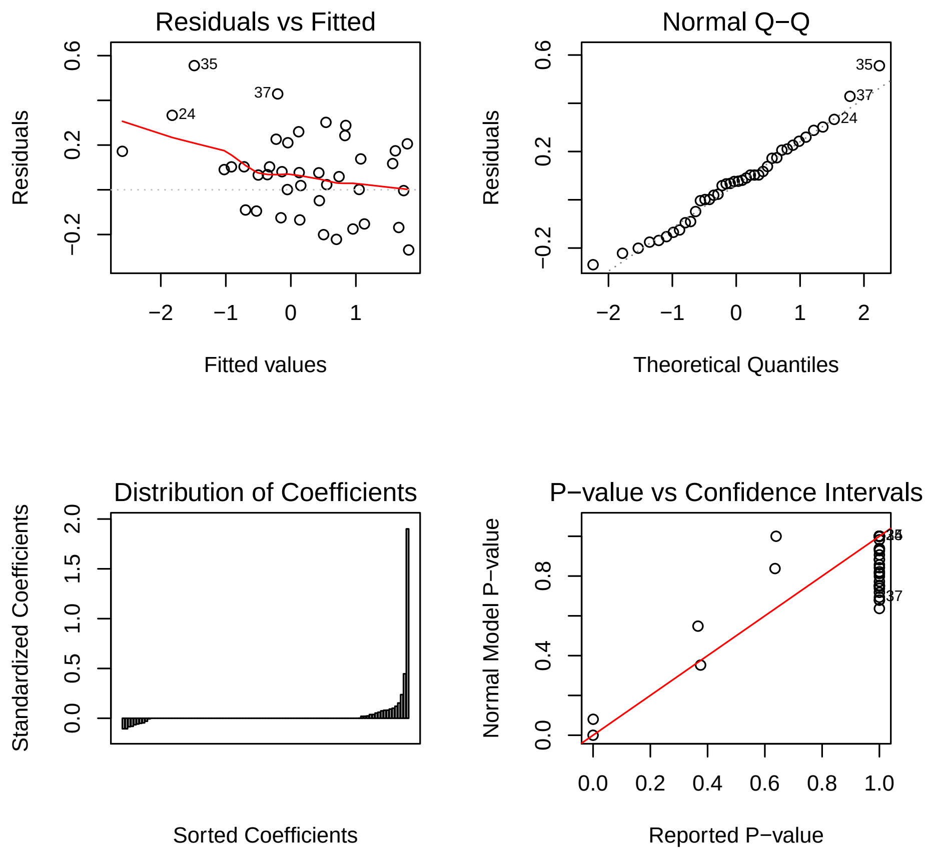

Figure 3: Graphical output of hdlm class.

```
+   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.

5.2. 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.

5.3. 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 Friedman *et al.* (2010). 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:

```
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.

6. Bayes Variants

The two-stage method of creating regression tables is quite accurate in a variety of settings. We have heuristically observed good performance in the linear model case as long as the sample size is no less than around 50 and, in the logistic regression case, as long as the dimension of the ambient space is of the same order of magnitude as the number of observations. Both of these statements depend of course on having reasonable signal to noise ratios. Situations outside of this range occur frequently, particularly in large gene based biological studies where one might have tens of thousands of variables, only a few dozen observations, and a categorical dependent variable. In this situation, we have found a hierarchical Bayesian approach can often out-perform the frequentist procedure. The Bayes approach benefits from not needing to split the dataset in order to produce regression tables (increasing the effective sample size), but suffers from sensitivity to the choice of hyper-parameters as well potentially being computationally intensive. Therefore we include it as an option in the package **hdlm** for both Gaussian and logistic datasets, but keep the two-stage method as the default.

For the standard linear model, we use the following hierarchical set-up as proposed (amongst various other options) in Hans (2009) and Hans (2010):

$$\begin{aligned}
Y_i|\beta, \sigma^2, \tau, \phi &\sim \mathcal{N}(X_i^\top \beta, \sigma^2),\ i = 1, \ldots, n && (14)\\
\beta|\sigma^2, \tau, \phi &\sim \prod_{j=1}^{p} \left((1-\phi)\delta_0(\beta_j) + \phi \left(\frac{\tau}{2\sigma}\right) \exp(-\tau\sigma^{-1}|\beta_j|) \right) && (15)\\
\sigma^2 &\sim \mathrm{IG}(a, b) && (16)\\
\tau &\sim \mathrm{IG}(s, r) && (17)\\
\phi &\sim \mathrm{beta}(g, h) && (18)\\
a, b, s, r, g, h &\geq 0 && (19)
\end{aligned}$$

We use the parameters $a = b = s = r = 1$, and as a default have $g = h = 1$ as well. The latter can be changed by the user by calling the option `bayesTune`, in order to increase of decrease the prior on the sparsity of the model. The Gibbs sampler for this set-up is available in the R package **blasso**, though this was neither available on CRAN nor did it display the sampled posterior data in a regression output. In our package, we have used the C++ Gibbs sampler from Hans (2009), packaged in our own customized R wrapper. As many of the resulting posterior samples of each β_j will likely be exactly equal to zero given the point pass at zero, we use the proportion of samples for which this is true as the Bayesian p value. The posterior mean and quantiles are used in the standard way as point estimators and confidence intervals.

For logistic data, we use the following hierarchical set-up as suggested by Lee *et al.* (2003):

$$Y_i|Z_i = \begin{cases} 0, & Z_i < 0 \\ 1, & Z_i \geq 0 \end{cases} \tag{20}$$

$$Z_i|\beta,\gamma \sim \mathrm{MVN}(X_i^\top \beta, 1),\ i = 1, \ldots, n \tag{21}$$

$$\beta|\gamma \sim \mathrm{MVN}(0, c(X_\gamma^\top X_\gamma)) \tag{22}$$

$$\gamma_j \sim \mathrm{Bernoulli}(\pi_j),\ j = 1, \ldots p \tag{23}$$

$$\pi_1, \ldots \pi_p \in (0,1),\ \ c > 0 \tag{24}$$

By default, we set each π_j to 0.01 and $c = 100$. The value for π_j can be changed directly within the function `hdglm` via the 'bayesTune' option, and the value of c is as suggested by the original authors. Our implementation follows the original paper's proposed Gibbs sampler, the code for which was not found to be publicly available. By sampling from the conditional distributions, we can approximate the posterior distribution of both γ as well as β. One minus the estimated mean values of γ_1 (which are either 0 or 1 in a given run) is used as the Bayesian p value and the mean and quantiles of the sampled posterior of β are used as the reported point estimators and confidence intervals.

We note that both the linear and logistic regression Bayesian variants differ substantially from the Bayesian lasso of Park and Casella (2008) and many of its natural variants, in that both of our posterior distributions have a point mass at zero. This concentration at zero assists in creating sensible p values, as well as more closely mimicking the frequentist lasso.

7. 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 2 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 2.

Actual run times for various sample and model sizes for default options are given in Table 3. 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 Weston (2011), 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

	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 2: 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.

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 4, 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.

8. Example Applications

As a proof-of-concept of how the **hdlm** package can be utilized for data analysis, we demonstrate an application to a linguistics text corpus as well as an application to voting data. A wealth of other examples exist in biology and finance, however we have chosen these two examples as their results require less specialized knowledge to interpret. We have chosen examples which illustrate possible issues with the default options as well as examples of how to address these issues.

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 3: 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 4: Comparison of run times (in seconds) for various number of used cores and number of bootstrap runs.

8.1. Newsgroup Records

The text corpus data consists of the relatively well known 20 newsgroups dataset available, amongst other places at `http://people.csail.mit.edu/jrennie/20Newsgroups/`. The full dataset consists of approximately 20,000 newsgroup documents broken into 20 different newsgroup categories. We use the data which is typically treated as the 'testing' set in the machine learning literature and those in the related categories 'talk.politics.guns' and 'talk.politics.mideast', in order to have a high dimensional model where the number of variables is approximately equal to the number of observations. It has been used extensively as a testing dataset for text mining and machine learning; see for instance the study by Joachims (1996).

We wish word occurrence data to predict the length (in words) of a newsgroup record. The data was cleaned in order to remove quoted messages and header information; this left only 523 records. After removing a standard set of English stopwords (e.g., 'and', 'the', 'because'), we determined the 526 most frequently used words, and then calculated which records used each word. A logarithm transform was taken on the total word count response data to reduce heavy tails.

When running the model using the default options, we received the following error:

```
R> set.seed(1)
R> out <- hdlm(Log_Length ~ ., data=wordDataset, bootstrap=5)

Error in { :
  task 1 failed - "FUNLM not reporting p-values; possible overfit model
See help pages for more information"
```

The issue is that the data matrix itself is sparse, with a lot of zeros and a few ones, and it is not unlikely that the model matrix selected for the second stage unpenalized model will have two or more columns with all zeros. This results in a singular matrix and the algorithm terminates. In order to rectify this we need to control for the number of selected variables.

We re-ran the model using the lasso model selector (given by `alpha = 1`), only allowing the 20 most influential variables into the second stage regression by passing an option to M, and using the Holm-Bonferroni method to compute p values. With these options selection, the algorithm yields the following regression table:

```
R> set.seed(1)
R> out <- hdlm(Log_Length ~ ., data=wordDataset, bootstrap = 5, alpha=1,
+     M=20, pval.method = "holm")
R> summary(out)

Call:
hdlm(formula = Log_Length ~ ., data = wordDataset, bootstrap = 5,
    alpha = 1, M = 20, pval.method = "holm")

Residuals:
    Min      1Q  Median      3Q     Max
-3.0677 -0.7078 -0.2359  0.2440  1.6920

Coefficients:
            Estimate Lower Bd Upper Bd  P-value
(Intercept)   4.8700   4.7215   5.0186  < 2e-16 ***
de            0.3469  -0.1382   1.4734 0.079542 .
exists        0.2287  -0.1486   1.1380 0.019868 *
opinions      0.2770  -0.1275   0.6815 0.091168 .
single        0.3619  -0.1752   1.1055  < 2e-16 ***
student       0.6252   0.0454   1.1187 0.000135 ***
tell          0.5775   0.1127   1.0462 0.003877 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Estimated sigma: 0.7785
```

Here we see that three of the selected variables have highly significant p values; however the variable ‘single’ also includes zero in the 95% confidence interval. The reason for this is that, as discussed in Section 2.3, in order to have confidence intervals which are always defined, confidence intervals must necessarily be more conservative than the optimal p value method. While here we have used the Holm-Bonferroni method for p values, this behavior can be seen using any of the related p value methods. To guarantee that both measurements of uncertainty are directly comparable is needed, one can either set `bootstrap = 1` or `bayes = TRUE`.

8.2. Voting Data

Our second example dataset concerns voting records in the United States Senate during the 111th United States Congress (January 3, 2009 - January 3, 2011). Voting records were directly scraped from the website `http://www.senate.gov/`. We encoded votes of ‘Yea’ and ‘Guilty’ as a 1, and other votes as a 0. Our regression model of interest is a logistic regression, where we attempt to predict the voting record of the California Democrat Junior Senator Barbara Boxer from the voting records of other Senators. Our dataset consists of 598 votes and 96 variables; the latter was slightly reduced to include only those senators in office for the entire session. While this is not high-dimensional in the sense of having more variables that observations, it is a very easy to see that it behaves similarly since (given the tendency of Senators to voting along party lines) the data matrix is not of full rank.

Our initial attempt at modeling the dataset uses the default settings of function `hdglm` with the binomial family selected. The output is at first fairly uninteresting:

```
R> set.seed(1)
R> out <- hdglm(Boxer_D_CA ~ ., data=votingRecord, bootstrap=5,
+             family='binomial')
R> summary(out)

Call:
hdglm(formula = Boxer_D_CA ~ ., data = votingRecord, family = "binomial",
    bootstrap = 5)

Residuals:
   Min     1Q Median     3Q    Max
-49.93  41.91 113.87 116.48 182.85

Coefficients:
                   Estimate  Lower Bd  Upper Bd P-value
(Intercept)       -4.04e+01 -2.38e+05  2.38e+05       1
Baucus__D_MT       2.55e+01 -2.46e+05  2.46e+05       1
Brown__D_OH        3.04e+01 -5.41e+05  5.42e+05       1
Burr__R_NC        -3.12e+01 -1.77e+05  1.77e+05       1
Carper__D_DE       4.06e+00 -1.38e+05  1.38e+05       1
Chambliss__R_GA   -4.68e+00 -2.12e+05  2.12e+05       1
Dorgan__D_ND       1.25e+01 -3.40e+05  3.40e+05       1
Ensign__R_NV       3.02e+01 -8.67e+04  8.68e+04       1
Enzi__R_WY         1.15e+01 -1.07e+05  1.07e+05       1
Feinstein__D_CA    3.24e+01 -2.32e+05  2.32e+05       1
Grassley__R_IA    -5.64e+00 -8.28e+04  8.28e+04       1
Hagan__D_NC       -2.45e+01 -2.31e+05  2.31e+05       1
Harkin__D_IA       2.15e-01 -1.12e+05  1.12e+05       1
Klobuchar__D_MN    4.96e+01 -1.18e+05  1.19e+05       1
LeMieux__R_FL      3.19e+01 -1.61e+05  1.61e+05       1
McCaskill__D_MO   -1.72e+00 -2.23e+05  2.23e+05       1
Menendez__D_NJ     7.99e+00 -1.60e+05  1.60e+05       1
Merkley__D_OR      1.58e+01 -2.27e+05  2.27e+05       1
Murray__D_WA       1.71e+01 -3.14e+05  3.15e+05       1
Reid__D_NV         1.50e+00 -1.19e+05  1.19e+05       1
Roberts__R_KS      1.08e+01 -2.03e+05  2.03e+05       1
Rockefeller__D_WV -3.05e+00 -1.12e+05  1.12e+05       1
Specter__D_PA      6.96e-01 -2.76e+05  2.76e+05       1
```

```
Stabenow__D_MI    -1.62e+01 -2.49e+05  2.49e+05       1
Wyden__D_OR        2.99e+00 -8.28e+04  8.28e+04       1
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

We see that most variables have non-zero coefficients, but no p values are below 0.8. This likely indicates, for a given partition of the model, many variables are selected by cross-validation; however, the second low-dimensional step rarely finds any of these relationships to be significant. Our method actually becomes useful here, since we are able to distinguish this setting from one where the selected model has a high degree of statistical significance.

As discussed in Section 6, when given both discrete data as well as discrete responses a Bayesian solution often yields more interpretable results. Turning on the Bayes solution routine we get the following output:

```
R> set.seed(1)
R> out <- hdglm(Boxer_D_CA ~ ., data=votingRecord,
+              family='binomial', bayes=TRUE, bayesTune=0.01)
R> summary(out)

Call:
hdglm(formula = Boxer_D_CA ~ ., data = votingRecord, family = "binomial",
    bayes = TRUE, bayesTune = 0.01)

Residuals:
   Min     1Q Median     3Q    Max
    -1      0      0      0      1

Coefficients:
                Estimate Lower Bd Upper Bd P-value
(Intercept)       -2.004   -2.559   -1.432  <2e-16 ***
Feinstein__D_CA    1.732    0.928    2.698   0.003 **
Klobuchar__D_MN    0.825    0.000    1.709   0.288
Merkley__D_OR      1.311    0.772    1.903   0.001 **
Specter__D_PA      0.854    0.000    1.610   0.172
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Here we see that two variables have a fairly low p values of 0.003 and 0.001: the senior Democratic Senator from California, Dianne Feinstein, and Jeff Merkley from Oregon, another West Coast Democrat. It seems natural that these two Senators, which have both common geographical and ideological concerns, would be the best predictor of Barbara Boxer's voting patterns.

A natural extension of this analysis would be to study the entire graphical structure of all Senators' voting records. This was done, for instance, as an example by Banerjee, El Ghaoui, and d'Aspremont (2008). Such an application would ideal require a generalization of our package to graphical models, which we briefly discuss in the following section.

9. 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, van de Geer, Bühlmann, and Zhou (2011) 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 Meinshausen and Buhlmann (2006) 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

Banerjee O, El Ghaoui L, d'Aspremont A (2008). "Model selection through sparse maximum likelihood estimation for multivariate gaussian or binary data." *The Journal of Machine Learning Research*, **9**, 485–516.

Benjamini Y, Hochberg Y (1995). "Controlling the false discovery rate: a practical and powerful approach to multiple testing." *Journal of the Royal Statistical Society B*, pp. 289–300.

Benjamini Y, Yekutieli D (2001). "The Control of the False Discovery Rate in Multiple Testing under Dependency." *The Annals of Statistics*, **29**(4), pp. 1165–1188.

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

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

Bickel P, Ritov Y, Tsybakov A (2009). "Simultaneous analysis of Lasso and Dantzig selector." *The Annals of Statistics*, **37**(4), 1705–1732.

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

Friedman J, Hastie T, Tibshirani R (2010). "Regularization paths for generalized linear models via coordinate descent." *Journal of statistical software*, **33**(1), 1.

Hans C (2009). *Brief Technical Report to Accompany the R Package blasso Bayesian Lasso Regression*. URL http://www.stat.osu.edu/~hans/software/blasso/.

Hans C (2010). "Model uncertainty and variable selection in Bayesian lasso regression." *Statistics and Computing*, **20**(2), 221–229.

Joachims T (1996). "A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization." *Technical report*, DTIC Document.

Koenker R (2011). *quantreg: Quantile Regression*. R package version 4.71, URL http://CRAN.R-project.org/package=quantreg.

Kyung M, Gill J, Ghosh M, Casella G (2010). "Penalized regression, standard errors, and bayesian lassos." *Bayesian Analysis*, **5**(2), 369–412.

Lee K, Sha N, Dougherty E, Vannucci M, Mallick B (2003). "Gene selection: a Bayesian variable selection approach." *Bioinformatics*, **19**(1), 90–97.

Leeb H, Pöetscher B (2003). "The finite-sample distribution of post-model-selection estimators and uniform versus nonuniform approximations." *Econometric Theory*, **19**(1), 100–142.

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

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

Leeb H, Pötscher B (2008). "Can One Estimate the Unconditional Distribution of Post-Model-Selection Estimators?" *Econometric Theory*, **24**(02), 338–376.

Meinshausen N, Buhlmann P (2006). "High-Dimensional Graphs and Variable Selection with the Lasso." *The Annals of Statistics*, **34,3**.

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

Park T, Casella G (2008). "The bayesian lasso." *Journal of the American Statistical Association*, **103**(482), 681–686. ISSN 0162-1459.

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

Wasserman L, Roeder K (2009). "High dimensional variable selection." *The Annals of Statistics*, **37**(5A), 2178.

Weston S (2011). *foreach: Foreach looping construct for R.* R package version 1.3.2, URL `http://CRAN.R-project.org/package=foreach`.

Yang Y (2005). "Can the strengths of AIC and BIC be shared? A conflict between model identification and regression estimation." *Biometrika*, **92**(4), 937–950.

Affiliation:

Taylor B. Arnold
Department of Statistics
Yale University
24 Hillhouse Avenue
E-mail: `taylor.arnold@yale.edu`