rFSA: An R Package for Finding Best Subsets and Interactions

by Joshua Lambert, Liyu Gong, Corrine F. Elliott, Katherine Thompson and Arnold Stromberg

Abstract Herein we present the R package **rFSA**, which implements an algorithm for improved variable selection. The algorithm searches a data space for models of a user-specified form that are statistically optimal under a measure of model quality. Many iterations afford a set of *feasible solutions* (or candidate models) that the researcher can evaluate for relevance to his or her questions of interest. The algorithm can be used to formulate new or to improve upon existing models in bioinformatics, health care, and myriad other fields in which the volume of available data has outstripped researchers' practical and computational ability to explore larger subsets or higher-order interaction terms. The package accommodates linear and generalized linear models, as well as a variety of criterion functions such as Allen's PRESS and AIC. New modeling strategies and criterion functions can be adapted easily to work with **rFSA**.

Introduction

In recent years, novel statistical modeling techniques have become more computationally intensive in an effort to accommodate the massive datasets afforded by advances in fields such as data mining and genetic sequencing. The volume and complexity of available data continue to grow, overwhelming even the fastest modern computers (Goudey, 2015). Efforts to analyze such complex data usually involve some level of data reduction (*e.g.*, Principle Component Analysis, Partial Least Squares, Sufficient Dimension Reduction), which can yield convoluted statistical models whose parameters researchers and statisticians alike struggle to interpret. Furthermore, many of these strategies are limited to specific model forms and lack the flexibility to explore higher-order interactions. Herein we present an alternative to data reduction that accommodates a variety of modeling strategies, including large-subset selection and identification of higher-order interaction terms. The resulting models, coefficient estimates, and predictions remain easily interpretable and flexible to traditional modes of statistical inference.

The Feasible Solutions Algorithm (FSA) overcomes the problems described above by searching a reduced data space to produce *feasible solutions* (Hawkins, 1994; Miller, 1984). Feasible models (Hawkins, 1994) are optimal under a given criterion function in the sense that no single exchange of an explanatory variable contained in the model for a variable outside the model will yield an improvement to the criterion. Miller (1984) first introduced the idea of a sequential-replacement algorithm such as the FSA. According to Miller, this computationally "cheap" method boasts rapid convergence; flexible implementation; and improved results compared to forward or backward selection. Miller also noted that the replacement algorithm could give too many solutions if repeated. However, Hawkins (1994) applied a similar exchange algorithm to the problem of finding robust regression estimators and minimum-volume ellipsoid estimators in multivariate data, demonstrating that with a sufficient number of random starts, the algorithm could find the optimal solution with arbitrarily high probability. Upon testing the algorithm, Hawkins also exhibited its superior performance relative to exhaustive search.

Exhaustively searching for an optimal model containing interactions is particularly demanding, computationally speaking, and not always reasonable or even attainable. For this reason, many published analyses do not even attempt to explore interactions. In other cases, interaction terms are identified by a primary investigator on the basis of his or her prior knowledge of the field, and then screened on an individual level by a statistician or data scientist. Such a process is tedious and time-consuming, and usually results in interactions being ignored or overlooked due to the sheer number of possibilities. These factors unite to afford a widespread lack of consideration for interactions, thereby undermining the predictive power of models attempting to capture complex relationships (Foster and Stine, 2004). We address these limitations by implementing an FSA with the capacity to explore higher-order terms, combined with the accessibility and ease of use associated with an R package.

The R implementation of our Feasible Solution Algorithm, **rFSA 0.9.1**, is now accessible *via* GitHub and CRAN.

Feasible Solutions Algorithm

Statisticians are often faced with the problem of identifying an informative subset of m explanatory variables from a set of p predictor terms, which we will denote $\mathbf{X}^p$. Consider selecting $p^+ \geq 0$ explanatory variables, denoted $\mathbf{X}^{p^+}$, to compose a preliminary model. Let $g(\mathbf{Y}, \mathbf{X})$ be an objective (criterion) function, generally a measure of model quality such as R^2, AIC, or PRESS. We wish to identify the m additional variables, $\mathbf{X}^m$, that when added to the existing model serve to optimize the objective function $g(\mathbf{Y}, \mathbf{X}^{(p^+ + m)})$.

The Feasible Solution Algorithm implements the following strategy, where we initially consider only first-order predictor terms:

1. Randomly select m variables to compose $\mathbf{X}^m$, and compute the objective function $g(\mathbf{Y}, \mathbf{X}^{(p^+ + m)})$.
2. Under each possible exchange of one of the m variables in the working model for a variable not contained in the model, compute the new value of the objective function.
3. Make the single exchange of variables from Step 2 that most improves the objective function.
4. Continue making exchanges until no single additional exchange improves the objective function. The variables $\mathbf{X}^{p^+}, \mathbf{X}^{m^*}$ composing the final model constitute a single feasible solution.
5. Return to (1) to search for another feasible solution.

Miller (1984) illustrates the general procedure as follows: Suppose you have a dataset containing 26 predictor variables labeled A through Z, and imagine you wish to find the best subset of four predictor variables. Randomly select four initial predictors; suppose these are $ABCD$. Consider exchanging one of A, B, C, or D with each of the 22 remaining variables. Make the change that most improves the objective function; suppose we swap C for X, so that the working subset becomes $ABXD$. Next consider exchanging one of A, B, X, or D. (In point of fact, considering an exchange involving X is here redundant and unnecessary.) Repeat this process until no additional exchange of variables yields further improvement to the objective function.

If instead a user requests interaction terms of m^{th} order, our FSA randomly selects m effects to compose an initial interaction term in step (1). In step (2), the algorithm considers exchanging any one of the m variables involved in the interaction term. In accordance with the hierarchical paradigm, all associated lower-order interactions and main effects are exchanged as well, prior to calculating the associated objective function; no linear or lower-order terms are included in the model beyond those required to complete the hierarchy unless the user requests a variable be fixed in the model. The sequential-replacement procedure otherwise operates as described above, but this extension permits optimization with respect to the p-value of an interaction term as an alternative to standard model-building criteria.

Each iteration of the FSA yields a single feasible solution, which is the most-optimal model under a given criterion that can be reached from the (random) starting configuration by means of sequential replacement. Of course, depending on the initialized model, the algorithm may not converge to the *global* optimum with respect to the specified objective function. However, if a truly correct model exists and consists of explanatory variables present in the dataset, then under an appropriate objective function and with many iterations, **rFSA** is increasingly likely to discover it: A model containing correct explanatory variables should yield a value for the criterion superior to that of a model containing explanatory variables unrelated to the response. In our experience with **rFSA**, the models best supported by the data tend to manifest more often than inferior models, but it can be shown that with many random starts even relatively rare (locally or globally) optimal configurations may be identified (Hawkins, 1994). Thus, with multiple random starts, the FSA is likely to discover the optimal model, as well as additional models that would be overlooked if one considered only the 'best' model under a specified criterion, but which may contain additional information of interest from a clinical or scientific viewpoint.

Other algorithms for subset selection

Many existing methods attempt to identify the best subset of predictors that adequately explains a response variable. Common automatic variable-selection techniques include forward selection, backward elimination, and step-wise regression methods, while penalized regression techniques, such as LASSO, are commonly used for subset selection with many variables. Exhaustive search procedures simply examine all possible combinations of predictors under a viable model structure — but even on the most-powerful computers, this approach becomes impracticable for datasets containing many explanatory variables (Goudey, 2015). Algorithms implementing these procedures in the context of linear or generalized linear models, respectively, are currently available in the form of R packages **leaps** (Lumley and Miller, 2009), **glmulti** (Calcagno, 2013), and **glmnet** (Friedman, 2008).

Other algorithms for finding interactions

Computational methods for exploring potential interactions include Random Forest (Jiang, 2009) and Boosting (Lampa, 2014) methods. In the former approach, researchers examine branches of a regression tree to identify splits that contribute downstream to alternative classifications of the response variable. Bayesian methods also exist for identifying interactions, although they tend to specialize in exploration of large genetic datasets (Zhang and Liu, 2007). LASSO can be used to identify interactions by selecting informative variables from an initial pool containing all possible interactions as well as first-order terms (Bien et al., 2013), and LASSO for Hierarchical Interactions has been implemented in the R package **hierNet** (Bien and Tibshirani, 2014). Although all of these methods have been used previously for identifying interactions, the software for implementing them are limited with respect to the statistical methods they utilize and criterion functions they oblige. The statistical community would therefore benefit from a robust algorithm whose package can accommodate multiple statistical methods and criterion functions.

rFSA

Our R package **rFSA** implements the FSA described above for use in subset selection and identification of interaction terms. The primary function, `FSA`, accepts as two of its arguments any user-specified R functions for use in fitting models (*e.g.*, `lm`, `glm`) and calculating the model criterion (*e.g.*, `AIC`, `BIC`, `r.squared`), with only the restrictions that the criterion function must (1) accept as its argument the model object returned by the specified model-fitting function and (2) return a single numeric value. Additional arguments to `FSA` include the number of parameters to consider, whether to investigate interactions with or without quadratic terms, and whether to minimize or maximize the criterion function.

Architecture

Within the R package **rFSA**, the function `FSA` is used to identify feasible models. A full list of the parameters belonging to `FSA` are described below, for reference in illustrating the architecture of the package:

Parameters:

`formula`	symbolic description of the functional model form to be fitted. All subsequent analysis will model the response variable designated by this parameter.
`data`	data frame containing the set of predictor variables of interest.
`fitfunc`	function to fit the model. Defaults to `lm`.
`fixvar`	vector of one or more variable names to be fixed in the model as main effects. Defaults to NULL.
`quad`	logical: whether to consider higher-order interactions containing two instances of the same variable. Defaults to FALSE.
`m`	number of predictors to compose a model. If `interactions` is TRUE then m is the order of interactions to consider.
`numrs`	number of random starts to perform.
`cores`	number of cores to use while running. Restricted to 1 for a Windows machine.
`interactions`	logical: whether to include interactions in model. Defaults to TRUE.
`criterion`	vector of criterion function(s) to maximize or minimize. Defaults to `AIC`.
`minmax`	vector of strings `"min"` or `"max"` specifying whether to minimize or maximize each criterion function. Defaults to `"min"`.
`checkfeas`	vector of variable names composing a model to test for feasibility. If multiple random starts specified, test of feasibility will occupy the final random start. Defaults to NULL.
`var4int`	variable to be fixed in the model as one component of the interaction term. Defaults to NULL.
`min.nonmissing`	minimum threshold for observations required to fit a model. Defaults to 1.
`return.models`	logical: whether to return fitted model objects. Defaults to FALSE.
`...`	additional arguments to be passed to the model-fitting function.

Assume we wish to discover the first-order linear model containing three predictors, chosen from a pool of eight predictors, with maximal R^2. To increase our probability of discovering the global optimum with respect to our chosen criterion, we choose to execute ten random starts. To decrease the time until completion, we run the algorithm on five cores. Let `dat.all` denote the data frame

containing the response variable, denoted `Y`, and all eight predictors. Then the code required to execute this search is provided below, acting on a simulated response variable and eight continuous predictors.

```
set.seed(40508)

# simulate explanatory variables, response, and uninformative variables
x1 <- rnorm(100)
x2 <- rnorm(100)
x3 <- rnorm(100)
dat.all <- data.frame(Y = 2 + 3*x1 - 4*x2 + 0.5*x3 + rnorm(100),
                      X1 = x1, X2 = x2, X3 = x3,
                      X4 = rnorm(100), X5 = rnorm(100), X6 = rnorm(100),
                      X7 = rnorm(100), X8 = rnorm(100))

# load package and execute rFSA
library(rFSA)
fsa.fit <- FSA(formula = 'Y~1', data = dat.all, fitfunc = lm, quad = FALSE, m = 3,
               numrs = 10, cores = 5, interactions = FALSE, criterion = r.squared,
               minmax = "max", return.models = FALSE)
print(fsa.fit)
```

rFSA first generates the ten random starts, each of which consists of three indices: one corresponding to each predictor in the initial model. For each random start, the algorithm generates every unique exchange of variables that can be achieved from the initial model. Corresponding models are fitted by calling the `fitfunc`, in this case `lm`, and the criterion function calculated for each of the resulting models. If the user specifies multiple cores, then the `mcapply` function from R package **parallel** is employed to fit multiple models simultaneously. Criterion values, here R^2 values, are stored in a hash table to prevent having to fit the same model multiple times across different random starts, thereby achieving an additional gain in computational efficiency as well as reducing the memory requirement. (If `return.models` is `TRUE`, then the model objects themselves are stored in a separate list.) **rFSA** makes the exchange that achieves the greatest improvement in model criterion, and then repeats the process of generating and making exchanges. The algorithm ceases iterating on a single random start when the best exchange yields no change in the model, or when the working model converges to a feasible solution discovered previously.

The `FSA` function is written as an `S3` object. Returned results assume class definition `FSA` and can be used in conjunction with standard `S3` functions `print`, `summary`, `predict`, `fitted`, and `plot`, described in greater detail below.

Implementation

The core of the package is the FSA optimization procedure, which is depicted as a flowchart in Figure (1). As this figure illustrates, the procedure can be divided into several sub-procedures as follows:

- The algorithm first generates a number (specified by user, here denoted M) of random starts $P_1, P_2 \cdots P_M$. Each member P_i is a random combination of variables from the predictor set.
- For each random start P_i, **rFSA** then generates all possible variable exchanges $Q_{i1}, Q_{i2} \cdots Q_{iN}$. Similar to a random start, each Q_{ij} is also a combination of variables, but differs from its corresponding start position P_i by precisely one variable. As a result, we obtain $M \times N$ combinations.
- Next, for each combination Q_{ij}, **rFSA** fits the model using the function specified by the input argument `fitfunc` and calculates the corresponding criterion value using the function specified by `criterion`. Thus we obtain a criterion value C_{ij} for each combination Q_{ij}.
- The criterion values $C_{i1}, C_{i2} \cdots C_{iN}$ are used to identify the optimal exchange among $Q_{i1}, Q_{i2}, \cdots, Q_{iN}$ from the starting configuration P_i. The best swap combination, denoted P_i', is that with the smallest or largest criterion depending on the user-specified argument `minmax`.
- Finally, the algorithm decides whether to continue iterating by comparing the best swap P_i' and initial combination P_i. If they are equal, then the algorithm stops processing and stores P_i as a feasible solution. Otherwise, it updates P_i to P_i' and continues iterating until a feasible solution is found.

Note that the computationally most-intensive stage of the algorithm occurs when obtaining the model fit and criterion values for combinations $Q_{11} \cdots Q_{1N}, \cdots, Q_{M1} \cdots Q_{MN}$. Moreover, some of this work is redundant, as the same model fit may be required as many as M times. We therefore elect

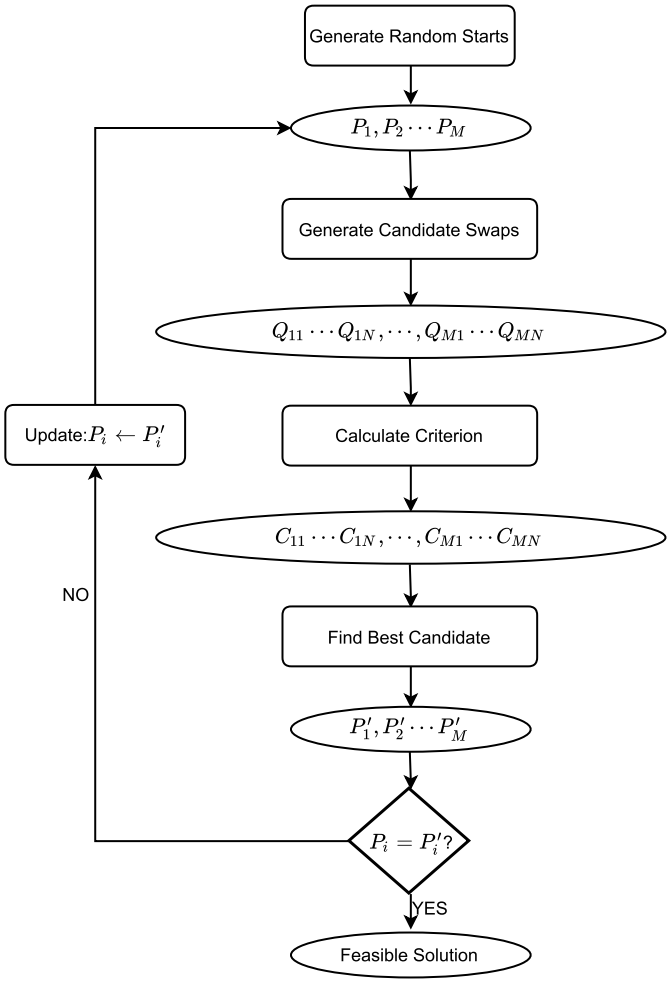

Figure 1: Overview of our implementation of the FSA optimization algorithm. A number of random starts are generated, followed by candidate variable exchanges and the criterion value associated with each new model. The best swap candidate is chosen according to the criterion values and then compared with the starting configuration. If they are equal, then they constitute a feasible solution. Otherwise, the starting configuration is updated by assuming the value of the best swap, and the procedure iterates until a feasible solution is found.

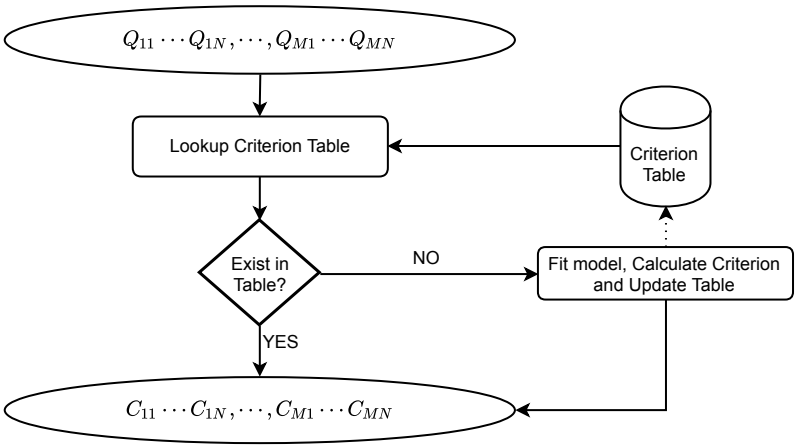

Figure 2: Illustration of storing and referencing criterion values using a global table. For each combination of variables, we first check for the model criterion in the table and return it if found. Otherwise, we fit the model, calculate the criterion, and add the calculated value to the table. To further speed execution, model fitting for different combinations of variables is processed in parallel using multiple cores.

to store criterion values upon calculation, thereby reducing the computational burden and improving execution efficiency. A detailed schematic of this implementation is provided in Figure (2).

The FSA function implements the procedure depicted in Figure (2) to maintain a global table of criterion values for easy reference. Prior to calculating a model criterion, **rFSA** first checks whether the model has been fitted previously and, if so, uses the stored criterion value rather than refitting the model. If the model's criterion is not represented in the existing table, then **rFSA** fits the model, calculates the criterion value, and stores that value in the table. Specifically, we adopt following techniques to optimize our implementation of the algorithm:

- By maintaining a look-up table, we avoid redundant computations to fit the same model multiple times.
- We implement the look-up table as a hash table, which achieves constant-time insertion and querying.
- We use the R package **hashmap** (Russell, 2017) to implement the criterion table. **hashmap** is written in C++, yielding further improvement to the execution speed of **rFSA**.
- We fit models in parallel on multiple computation cores using the `mclapply` function of the **parallel** package. Because model fitting is independent across different sets of variables, the task is appropriate for multi-thread or multi-process parallelization.

Usage

In this section, we consider some nuances related to the arguments of the function FSA, and end with details of the results returned by the same function.

The `formula` parameter allows users to specify an initial fit. Because initial models are randomly generated, it is sufficient to specify a null model containing solely the desired response variable and an intercept term, *e.g.*, `Y ~ 1`. Variables to be fixed in the model should be provided in the form of a vector of character strings representing the variable names, provided to FSA as the argument `fixvar`.

The `data` structure should contain only the target response variable and predictor variables of interest, with each variable occupying a distinct column of the data structure. Variables known to be superfluous should be omitted from the data structure prior to executing FSA. Furthermore, categorical variables should be denoted in quotes or stored as factors to distinguish from quantitative variables, and should assume at least two levels to avoid complications due to invariance.

Models will be fitted using the user-specified `fitfunc`, to which the `formula`, `data`, and any additional arguments will be passed. The objective function `criterion` may be any user-specified R function that accepts as its argument the model object returned by `fitfunc`, provided the criterion function returns a single numeric value (or NA) that can be minimized or maximized. If users intend to write their own criterion functions, we recommend that they protect against errors by enshrouding the functions in `tryCatch` statements, available in base R. As an alternative, users may specify one of the criterion functions built into **rFSA**, which are detailed below. In either case, the specified criterion function will be maximized or minimized in accordance with the value of `minmax`.

The `mclapply` function used to support parallel processing in **rFSA** accommodates the use of multiple cores for Unix environments only. For this reason, **rFSA** automatically instantiates the `cores` parameter to 1 for Windows users, regardless of the user-specified value. For Unix machines with multiple cores, we recommend a maximum threshold for `cores` of one fewer than the number of cores available on the machine. Users can execute `parallel::detectCores()` to determine the number of cores on their computers.

To request subset selection without interactions, `interaction` must be set to FALSE. (In this case, the `quad` parameter will be ignored.) In this scenario, `m` denotes the number of predictors to compose each resulting subset. In the case of `interactions = TRUE`, the meaning of `m` changes to represent the desired interaction order. (During model fitting, all lower-order terms associated with the interactions are also included). In such an instance, the parameter `var4int` can be used to fix a variable as one member of the interaction term, thereby restricting the FSA function to identify m-way interactions containing the specified predictor along with any other $m - 1$ variables. The `quad` parameter specifies whether higher-order interactions should be permitted to contain the same variable twice. Regardless of these parameters, the FSA function yields feasible solutions equal in number to `numrs`. However, these models may not be unique if a feasible solution is accessible from multiple random starting positions. In order to provide cleaner results, **rFSA** produces a table summarizing these results (see Function Outputs, below).

Finally, the FSA function accommodates testing for the feasibility of a model, where the desired predictors may be provided as the argument checkfeas. If the model is feasible, then the resulting feasible solution will contain the same variables as checkfeas; otherwise, the FSA function returns the feasible model accessible from the initial model specified in checkfeas. If the user specifies multiple random starts, then the test of feasibility occupies the final random start.

Criterion Functions

Criterion functions built into **rFSA** include R^2 (r.squared) and adjusted R^2 (adj.r.squared), Allen's PRESS statistic (apress), interaction p-values (int.p.val), Akaike's Information Criterion (AIC) and the Bayesian Information Criterion (BIC), penalized Quasi-likelihood under the Independence model Criterion (QICu.geeglm), and Bhattacharyya Distance (bdist). These functions may not be appropriate for all functions provided to fitfunc as, for example, r.squared does not accommodate generalized linear models.

Most of the above metrics are well known and thus do not require introduction, but we summarize briefly two of the less-common statistics. The Bhattacharyya Distance (Bhattacharyya, 1946) is a distance measure that quantifies the disparity between two distributions. The Bhattacharyya Distance is particularly useful for a binary response when the user's interest lies in exploring two-way interactions between continuous explanatory variables (Janse, 2017). This is a common problem in large genetic datasets, in which context bdist is doubly useful because it converges faster than the other criterion functions. The Quasi-likelihood under the Independence model Criterion (QIC) (Pan, 2001) is a goodness-of-fit statistic for generalized estimating equations, analogous to AIC. The function built into **rFSA** accepts an object of class geeglm, fitted with **geepack** (Højsgaard et al., 2006; Yan and Fine, 2004; Yan, 2002), and returns the penalized QIC, denoted QICu, which is preferred for subset selection.

Function Outputs

A successful run of the FSA function returns a list with seven elements, described below:

Returned Values:

$originalfit	model object representing the fit of the user-specified original model.
$call	list of FSA function parameters and their actualized values.
$solutions	list of initial and final (*i.e.*, feasible) predictors contained in each model; criterion function values for feasible models; and number of exchanges made. If return.models=TRUE then solutions will contain an object of all fitted models called checked.model.
$table	data frame of unique feasible solutions; corresponding criterion values; and the number of random starts that converged to each solution.
$efficiency	character string contrasting the total number of fitted models and criterion values calculated using **rFSA** against those required for exhaustive search on the same dataset.
$nfits	integer representing the total number of fitted models.

Given an FSA object as its argument, the print command will display a table containing the original user-specified model and all feasible solutions that the algorithm found over numrs random starts. The table also contains criterion values for each model, and the number of random starts that converged to each feasible solution. Generally speaking, the original fit is not a feasible solution, and thus the number of random starts for that model will be listed as NA. The summary command acting on an object of class FSA will display a list containing as its elements the summary output of each fitted model; the format therefore depends on the chosen fitfunc. The predict and fitted functions operate in a similar fashion to summary, returning a list of predicted or fitted values, respectively, for each model. Finally, the plot function generates a Q-Q plot and a plot of residuals against fitted values for each model, displayed in a compact manner.

Availability

Currently, **rFSA version 0.9.1** is available to download from the Comprehensive R Archive Network at https://cran.r-project.org/web/packages/rFSA. To install the newest beta version of **rFSA**, first install the **devtools** package in R, and then run devtools::install_github("joshuawlambert/rFSA").

Shiny App

An easy-to-use Shiny Application is also available to facilitate the basic functions of the package. We believe this application will serve an important role for researchers unfamiliar with R, who would

nonetheless benefit from the ability to describe and/or make predictions from large datasets. Our Shiny Application allows users to upload their own data and specify parameter values *via* radio buttons and drop-down boxes, as well as to visualize fundamental elements of the resulting models. The application currently subsists on a server hosted by the University of Kentucky, accessible from https://shiny.as.uky.edu/mcfsa/. **Important:** Users *should not* upload sensitive data to our FSA Shiny Application. To accommodate protected data, a future version of **rFSA** will permit users to run local instances of the Shiny Application.

Comparison to other packages

Several R packages exist already for finding best subsets and exploring pairwise interactions. Two of the most-popular packages in current use are the **leaps** (Lumley and Miller, 2009) package and **glmulti** (Calcagno, 2013). These packages use criterion functions such as R^2, adjusted R^2, Mallow's C_p, residual sum of squares, and AIC/BIC to identify the best subset of predictor variables to describe a given response variable. Although these packages are useful, they offer a limited selection of statistical models and criterion functions. In this section, we describe these and other limitations on existing packages, and explain how **rFSA** seeks to overcome them.

The **leaps** package leverages exhaustive search, forward or backward selection, or a sequential-replacement algorithm to find the best subset of predictors to compose a model. This sequential-replacement algorithm is a variation on the FSA introduced by Miller (1984) and described above, and therefore serves as a relatively efficient option for analyzing large datasets. Although the **leaps** package is flexible and robust, at present it does not accommodate interaction terms, model forms other than first-order linear, or the use of alternative criterion functions to the aforementioned five. The **bestglm** (McLeod and Xu, 2017) package seeks to extend best-subset model selection to generalized linear models but is not natively capable of looking for higher-order interactions, using external criterion functions, or accommodating other statistical methods. It also makes no special consideration for large problems, rendering it unsuitable for datasets with more than 100 predictors. **glmulti** improves on the aforementioned packages in that it is capable of incorporating pairwise interaction terms into exhaustive search or a genetic algorithm, but it does not support other statistical methods, higher-order interactions, or flexible criterion functions.

Timing comparisons

We now present the results of a simulation conducted to compare the performance of **rFSA** against that of of exhaustive search with **leaps**. We include neither **glmulti** nor **bestglm** in the comparison because neither package was able to accommodate the high-dimensional datasets of interest ($p > 100$).

Simulations were conducted on twenty-five datasets of $N = 250$ observations for various numbers of predictors, again denoted by p. In each dataset, a continuous response was generated randomly from a standard normal distribution. Half of the predictors were generated randomly from a standard normal distribution, and the other half, from a Bernoulli distribution with $P(X = 1) = 0.50$.

The execution speed of each package was measured individually for each dataset. Simulations were conducted in R version 3.4.1 on a Linux machine with Intel(R) Xeon E5-2698 v4 @ 2.20GHz with 128.00 GB of memory. A single core was allocated for each of `regsubsets` and the `FSA` function. Code follows for calling the relevant function in each of packages **leaps** (version 3.0) and **rFSA** (version 0.9.1) to search for best subsets of size three:

```
leaps::regsubsets(x = ..., y = ..., nbest = 1, nvmax = 3, really.big = T,
                  method = "exhaustive")
rFSA::FSA(X1 $\sim$ 1, data = ..., interactions = F, m = 3, numrs = 1, criterion = AIC,
          minmax = "min")
```

Figure 3 compares the run-time in $\log(seconds)$ for these commands over twenty-five simulations and $p = (150, 250, 350, 450, 650, 750, 850)$ when searching for best subsets of size three. Exhaustive search with **leaps** exhibits a superior run-time on average for $p \leq 350$, while **rFSA** is more efficient when $p > 350$. **rFSA** was approximately 13 minutes faster for $p = 750$. The growth in execution time for **rFSA** slows relative to that of **leaps** as the number of variables increases: Although the overhead is higher for `FSA`, the costs associated with analyzing incrementally larger volumes of data are lower.

Figure 4 depicts run-times for twenty-five simulations of $p = (50, 100, 150, 200, 250, 300, 350)$ when searching for best subsets of size five. In this scenario, exhaustive search with **leaps** evinces a faster

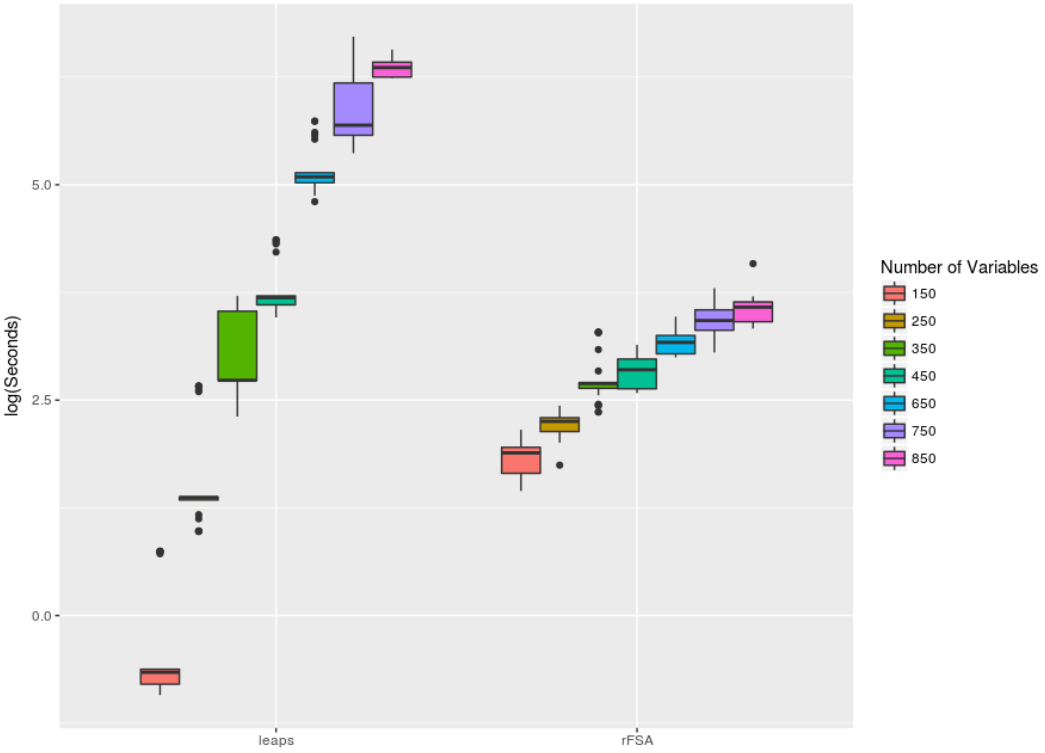

Figure 3: Timing comparison of exhaustive search with **leaps** *versus* **rFSA** with one random start. Each package was assigned to identify best subsets of size three from datasets of size $n = 250$ and various p.

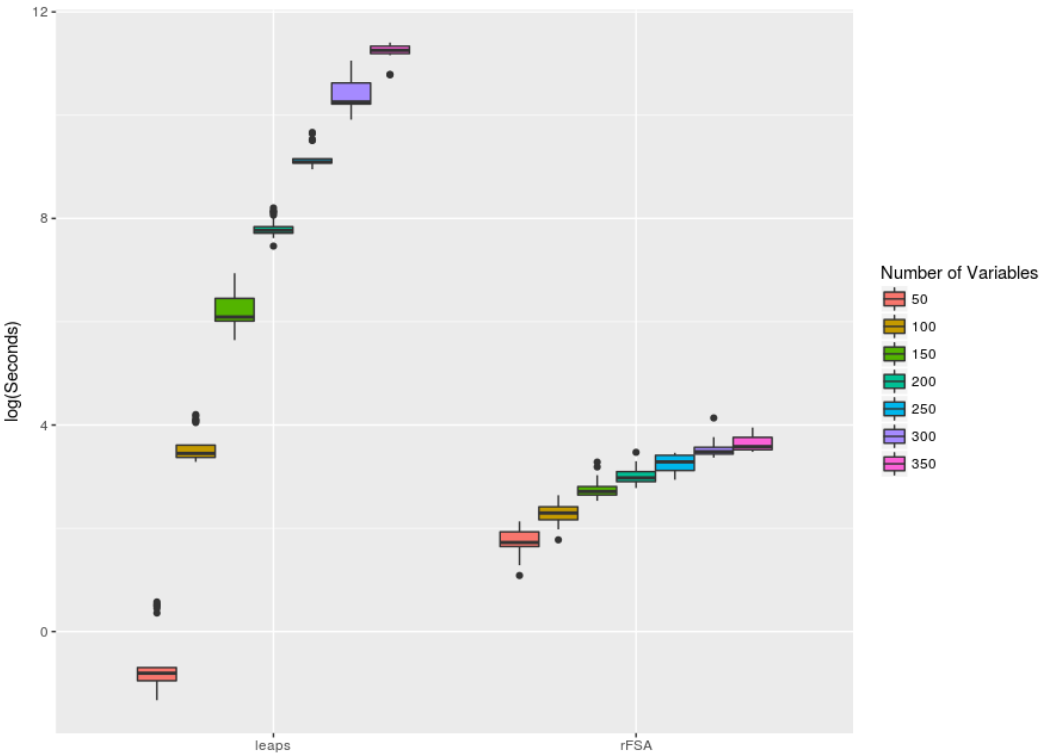

Figure 4: Timing comparison of exhaustive search with **leaps** *versus* **rFSA** with one random start. Each package was assigned to identify best subsets of size five from datasets of size $N = 250$ and various p.

time for $p = 50$, while **rFSA** exhibits a superior run-time for all other datasets ($p > 50$). The largest difference in time between `leaps::regsubsets` and `rFSA::FSA` for best subsets of size five and a fixed p was 89,815 seconds (or 24.9 hours) for $p = 350$. Figures 3 and 4 both illustrate that the growth in run-time of our implemented FSA algorithm is much slower than **leaps**, which confirms the better time complexity of **rFSA**. As a result, our implemented algorithm is able to analyze datasets with large p at a much faster rate than the conventional alternative package.

Across the 175 simulations for best subsets of size three, **rFSA** identified the optimal solution in 158 of the simulations. When searching for best subsets of size five, **rFSA** discovered the optimal solution in 144 simulations and afforded gains in run-time of up to 24 hours. Although **rFSA** does not implement an exhaustive search, execution with many random starts often requires fewer computations while yet producing the optimal solution with arbitrarily high probability (Hawkins, 1994). Thus for large p, we argue that **rFSA** is a practical solution for researchers who wish to consider high-dimensional data, higher-order terms, generalized linear or mixed models, or other non-traditional statistical methods and criterion functions.

Example

Two- and three-way interactions in Census Data

The 2014 Planning Database Block Group Data (PDB) from the Census Bureau is publicly available at http://www.census.gov/research/data/planning_database/2014/. Descriptions of the variables can be found from PDB documentation on the aforementioned website. Herein we examine only the census blocks associated with Kentucky, with several variables removed because they were transformations of other variables. The final dataset contained 3285 observations and 67 quantitative explanatory variables in addition to the quantitative response variable, mail response rate. The revised dataset can be downloaded from https://github.com/joshuawlambert/data/raw/master/census_data_nopct.csv.

In this example, we search for the best linear model containing (a) two main effects and their interaction, or (b) three main effects, as well as their pairwise and three-way interactions. To fit linear models using the `FSA` function, we set `fitfunc` equal to `lm`. For simplicity, we choose not to fix any variables in the models, which we achieve by setting `fixvar` equal to `NULL`. We provide a null model regressing the response variable y (Mail Response Rate) solely on an intercept term, thereby restricting **rFSA** to use y as the response variable throughout the procedure. The criterion function is taken to be `int.p.val` (interaction p-value) and minimized on each iteration. To specify a desire for two-way interactions, we set parameters `interactions = TRUE` and `m = 2`; for three-way interactions, `interactions = TRUE` and `m = 3`. We request 50 random starts, which **rFSA** completed in approximately one minute for `m = 2` or five minutes for `m = 3` on a Windows 7 machine with Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz and 24.00 GB of memory.

R Code follows for reproduction of these results:

```
install.packages("rFSA")
library(rFSA)
download.file("http://raw.githubusercontent.com/joshuawlambert/data/master/
               census_data_nopct.csv",destfile = "tmp.csv")
census_data_nopct <- read.csv(file = "tmp.csv")

# find two-way interactions
set.seed(123)
fit_2_way <- FSA(formula = "y~1", data = census_data_nopct, fitfunc = lm,
                 fixvar = NULL, quad = F, m = 2, numrs = 50, cores = 1,
                 interactions = T, criterion = int.p.val, minmax = "min")

print(fit_2_way)    # summary of solutions found
summary(fit_2_way)  # list of summaries from each lm fit
plot(fit_2_way)     # diagnostic plots


# find three-way interactions
set.seed(1234)
fit_3_way <- FSA(formula = "y~1", data = census_data_nopct, fitfunc = lm,
                 fixvar = NULL, quad = F, m = 3, numrs = 50, cores = 1,
```

```
                interactions = T, criterion = int.p.val, minmax = "min")

print(fit_3_way)    # summary of solutions found
summary(fit_3_way)  # list of summaries from each lm fit
plot(fit_3_way)     # diagnostic plots
```

Two-way interactions

Across 50 random starts seeking two-way interactions, **rFSA** discovered three unique feasible solutions. The solution with the smallest two-way interaction p-value occurred 24 times, ranking second in prevalence. The three interactions suggest relationships between (a) number of households with both spouses as members, and number of persons aged 18 to 24 ($p = 6.19 \times 10^{-38}$); (b) average household income and average aggregate house value in dollars ($p = 1.68 \times 10^{-43}$); and (c) number of mobile homes and total vacant housing units ($p = 7.26 \times 10^{-19}$). All three interaction p-values are significant using a Bonferroni-adjusted cutoff of $\frac{0.05}{\binom{67}{2}} \approx 0.00045$.

Having identified a set of feasible solutions, an investigator generally wishes to examine a summary of each model fit. Given an object, `fit`, returned by `FSA`, the function call `summary(fit)` returns a list of summaries for each model fit (including the original). Assessing the fit of each feasible solution can be facilitated with `plot(fit)` to display diagnostic plots for the original and feasible models. Each solution should be considered in a practical manner, with reasonable interpretations of the interaction term being considered prior to its inclusion in a final model.

The three interactions listed above are informative for understanding the types of results returned by the algorithm. For example, one might expect that a larger average household income would be correlated positively with the average value of houses in a census block. Thus, as in any linear regression setting, multicollinearity may account for apparent relationships in the data. The second interaction, between number of households with both spouses as members and the number of persons aged 18 to 24, may be more meaningful in the description of mail response rate (y). Because this interaction is both justifiable and statistically significant, an investigator would have considerable evidence to warrant its inclusion in a final model.

Three-way interactions

Across 50 random starts seeking three-way interactions, **rFSA** identified 14 feasible solutions. The most-frequently observed feasible solution occurred 13 times and boasted the smallest three-way interaction p-value. This interaction contains two variables identified previously in our search for two-way interactions. The three-way interaction among the number of households in which both spouses are members, the number of persons aged 18 to 24, and the number of people who indicate no Hispanic origin and their sole race as "White," affords an interaction p-value of 7.62×10^{-24} and remains significant under a Bonferroni correction. This interaction suggests that the combined effect of married couples and young adults is further modified by the number of people who identify as "White." That is, in locations with a larger population of whites, the effect on mail response rate of having more married couples changes depending on the prevalence of young adults aged 18 and 24. The interaction is reasonably easy both to interpret and to justify, as well as being statistically significant, and thus would warrant additional investigation by an interested researcher.

In sum, leveraging **rFSA** can highlight associations and interactions underlying a dataset that may not be immediately apparent to the researcher. The package is easy to manipulate (as demonstrated by the sparseness of the code provided in this example) as well as highly efficient, and generally returns multiple subsets of variables to permit flexible exploration and validation.

Conclusion

In this paper, we discuss the implementation in R of a complex algorithm originally proposed by Miller (1984) and Hawkins (1994). We further demonstrate its versatility and computational efficiency by comparison with existing subset-selection packages and by execution on a real-life census dataset. Our package, **rFSA**, is available on CRAN and GitHub. Additionally, a light version of the algorithm is available as a Shiny Application for the convenience of users with limited familiarity with R.

rFSA boasts several improvements over existing R packages for finding best subsets, some of the most popular of which include **leaps**, **bestglm**, and **glmulti**. Although these packages all perform well in implementing exhaustive searches without interactions, they offer a limited choice of statistical

technique, and often struggle when confronted with high-dimensional datasets. By contrast, **rFSA** accommodates large datasets, higher-order interaction terms, and a variety of model forms and criterion functions, and can be adapted to work with less-traditional statistical methods such as survey-weighted, penalized, hierarchical, zero-inflated, survival, and many other regression techniques. The results permit facile interpretation and remain flexible to conventional modes of statistical inference. Finally, the algorithm exhibits a considerable speedup on single-core operations for large subsets, and large p, relative to variable-selection packages in common use.

To improve the accessibility and convenience of the FSA for use by the general research community, we plan to incorporate more features of **rFSA** into the existing Shiny Application. Moreover, we intend to improve the data-visualization features of both App and package, and to build an off-line version of the Shiny App for users with secure data. We will seek further improvements to execution speed and resource management to encourage analysis of yet-larger datasets, and incorporate additional model forms and criterion functions to permit efficient analysis by non-traditional methods.

Through its versatility and flexibility, **rFSA** provides an alternative algorithm for model selection that allows users to find statistically optimal subsets and interaction effects in a variety of datasets. Improved model selection may afford better predictive power, which can in turn illuminate relationships underlying large datasets in a variety of research fields.

Summary

This paper outlines an R package, named **rFSA**, for subset selection and discovery of higher-order interactions. **rFSA** is flexible to a variety of statistical models and criterion functions, including those implemented by the user, and boasts execution speeds superior to existing subset-selection packages in context of large datasets. The release version of **rFSA** is hosted on CRAN, and the development version can be accessed from GitHub by calling `devtools::install_github("joshuawlambert/rFSA")`.

Acknowledgements

This research and package creation were supported in part by the Kentucky Biomedical Research Infrastructure Network and INBRE Grant (P20 RR16481) and a National Multiple Sclerosis Society Pilot Grant (PP-1609-25975).

Bibliography

A. Bhattacharyya. On a measure of divergence between two multinomial populations. *Indian Statistical Institute*, 7, 1946. [p301]

J. Bien and R. Tibshirani. *hierNet: A Lasso for Hierarchical Interactions*, 2014. URL https://CRAN.R-project.org/package=hierNet. R package version 1.6. [p297]

J. Bien, J. Taylor, and R. Tibshirani. A lasso for hierarchical interactions. *The Annals of Statistics*, 41, 2013. URL https://doi.org/10.1214/13-aos1096. [p297]

V. Calcagno. **glmulti***: Model Selection and Multimodel Inference Made Easy*, 2013. URL https://cran.r-project.org/web/packages/glmulti/. R package version 1.0.7. [p296, 302]

D. Foster and R. Stine. Variable selection in data mining: Building a predictive model for bankruptcy. *Journal of the American Statistical Association*, 99:303–313, 2004. URL https://doi.org/10.1198/016214504000000287. [p295]

J. Friedman. **glmnet***:Lasso and Elastic-Net Regularized Generalized Linear Models*, 2008. URL https://cran.r-project.org/web/packages/glmnet/. R package version 2.0-5. [p296]

B. Goudey. High performance computing enabling exhaustive analysis of higher order single nucleotide polymorphism interaction in genome wide association studies. *Health Information Science and Systems*, 3, 2015. URL https://doi.org/10.1186/2047-2501-3-s1-s3. [p295, 296]

D. Hawkins. The feasible solution algorithm for least trimmed squares regression. *Computational Statistics & Data Analysis*, 17:185–196, 1994. URL https://doi.org/10.1016/0167-9473(92)00070-8. [p295, 296, 304, 305]

S. Højsgaard, U. Halekoh, and J. Yan. The r package geepack for generalized estimating equations. *Journal of Statistical Software*, 15/2:1–11, 2006. URL https://doi.org/10.18637/jss.v015.i02. [p301]

S. Janse. Inference using bhattacharyya distance to model interaction effects when the number of predictors far exceeds the sample size. *Theses and Dissertations–University of Kentucky Statistics*, 2017. URL https://doi.org/10.13023/etd.2017.455. [p301]

R. Jiang. A random forest approach to the detection of epistatic interactions in case-control studies. *BMC Bioinformatics*, 10, 2009. URL https://doi.org/10.1186/1471-2105-10-s1-s65. [p297]

E. Lampa. The identification of complex interactions in epidemiology and toxicology: a simulation study of boosted regression trees. *Environmental Health*, 13, 2014. URL https://doi.org/10.1186/1476-069x-13-57. [p297]

T. Lumley and A. Miller. **leaps***: Regression Subset Selection*, 2009. URL https://cran.r-project.org/web/packages/leaps/. R package version 2.9. [p296, 302]

A. I. McLeod and C. Xu. *Bestglm: Best Subset GLM*, 2017. URL https://CRAN.R-project.org/package=bestglm. R package version 0.36. [p302]

A. Miller. Selection of subsets of regression variables. *Journal of the Royal Statistical Society*, 147:389–425, 1984. URL https://doi.org/10.2307/2981576. [p295, 296, 302, 305]

W. Pan. Akaike's information criterion in generalized estimating equations. *Biometrics*, 57:120–125, 2001. URL https://doi.org/10.1111/j.0006-341x.2001.00120.x. [p301]

N. Russell. *Hashmap: The Faster Hash Map*, 2017. URL https://CRAN.R-project.org/package=hashmap. R package version 0.2.0. [p300]

J. Yan. Geepack: Yet another package for generalized estimating equations. *R-News*, 2/3:12–14, 2002. [p301]

J. Yan and J. P. Fine. Estimating equations for association structures. *Statistics in Medicine*, 23:859–880, 2004. URL https://doi.org/10.1002/sim.1650. [p301]

Y. Zhang and J. Liu. Bayesian inference of epistatic interactions in case-control studies. *Nature Genetics*, 39, 2007. URL https://doi.org/10.1038/ng2110. [p297]

Dr. Joshua Lambert
College of Nursing
Assistant Professor & Biostatistician
214 Procter Hall
University of Cincinnati
Cincinnati, OH, USA
Joshua.Lambert@uc.edu

Dr. Liyu Gong
Post-Doctoral Scholar
Institute for Biomedical Informatics
University of Kentucky
Lexington, KY, USA
liyu.gong@uky.edu

Corrine F. Elliott, M.S.
Department of Statistics
University of Kentucky
Lexington, KY, USA
ORCiD 0000-0001-7935-9945
corrine.elliott@uky.edu

Dr. Katherine Thompson
Department of Statistics
Assistant Professor
317 Multidisciplinary Science Building
University of Kentucky

Lexington, KY, USA
katherine.thompson@uky.edu

Dr. Arnold Stromberg
Department of Statistics
Department Chair and Professor
313 Multidisciplinary Science Building
University of Kentucky
Lexington, KY, USA
stromberg@uky.edu