The Gretl `fsboost` function package for running forward stagewise regressions

Artur Tarassow

July 12, 2023

Github project page

Version 0.5

Changelog

- Version 0.5 (July, 2023)
  - add cross-references in section 1 to other packages.
- Version 0.4 (March, 2023)
  - Improve printing of coefficient estimates
- Version 0.3 (June, 2022)
  - Bugfix: Final set of features was not correctly identified when calling the function `plot_coefficient_paths()`
  - Internal changes: Remove dependency on "ridge" package for computing statistics of fit
- Version 0.2 (March, 2021)
  - update docs on early stopping and the learning rate
  - add two new configurable early stopping rules: "residual_corr_rel" and "residual_corr_abs"
  - add MSE (insample) statistics
  - required gretl version is 2020e now
  - internal refactoring
- Version 0.1 (September, 2020)
  - initial release

Contents

1 Introduction

Selecting the relevant predictors is a crucial task when working with a large pool of potentially relevant ones. Neglecting relevant predictors may lead to inconsistent parameter estimates while considering irrelevant regressors leads to inefficient estimates. Furthermore, including highly correlated predictors within a standard least square regression setting most probably suffers from multicollinearity issues. Lastly, standard least squares cannot handle the case when the number of observations, T, exceeds the number of potential regressors, k.

So called shrinkage and/ or selection estimators such as Ridge or Lasso or stepwise-OLS among others are known to handle such issues by imposing an additional restriction to an otherwise ordinary least square setting.[1] Another alternative estimation approach is the so called forward stagewise regression approach (*fsboost* henceforth).

fsboost is a simple strategy for constructing a sequence of sparse regression estimates: Initially set all coefficients to zero, and iteratively update the coefficient (by a small amount, depending on the *learning rate* ϵ) of the variable that achieves (under quadratic loss) the maximal absolute correlation with the current residual. *Learning* from the residuals has some connection to an approach known as *boosting* in the machine-learning community.

[1] For running both the Ridge and Lasso (and hybrids) estimators, consider the gretl addon "regls". For sequential addition of variables to a model, you may be interested in the function package "addlist".

The *fsboost* procedure also has some interesting connection to the Lasso under some conditions (Hastie et al. 2007). As $\varepsilon \to 0$, the sequence of forward stagewise estimates exactly coincides with the lasso path. While, this equivalence holds outside of least squares regression (Tibshirani, 2015), currently we only support minimization of squared loss (RMSE). Furthermore, as shown by Tibshirani, the *fsboost* algorithm also covers the Poisson or logistics regression losses. These cases may be covered in future versions of this package.

2 The fsboost algorithm

2.1 The algorithm

The fsboost algorithm works as follows:[2]

1. Start with $r = y$ and $\beta_1 = \beta_2 = \ldots = \beta_k = 0$.[3]
2. Find the predictorx_j most correlated with r.
3. Update the$j$-th predictor $\beta_j^i \leftarrow \beta_j^{i-1} + \delta_j$ where$\delta_j = \epsilon \times \text{sign} < r^{i-1}, x_j >$
4. Update the residuals $r^i \leftarrow r^{i-1} - \delta_j \times x_j$ and repeat steps 2 and 3 many times.

Here y, β, ϵ, $< r, x_j >$ and i refer to the endogenous variable, the unknown regression coefficients, the learning rate, the correlation between the current residuals and the$j$-th regressor and the i-th iteration. The learning rate ϵ is a hyper-parameter and set to a fixed constant (e.g., $\epsilon = 0.01$). The only computational intense task is to compute the correlation between r for all k predictors. We make use of Gretl's `mcorr()` function for this which is written in C and computationally very fast.

The idea behind the stagewise updates is simple: at each iteration greedily select the predictor j that has the largest absolute inner product (or correlation, for standardized variables under quadratic loss) with the residual. As the residuals refer to the yet unexplained part of the model, we are searching for any variable that still has some information content for explaining 'something' left unexplained.

Given the greediness that only a single predictor is selected each iteration, updating the coefficient of variable j by a large amount is problematic. Instead, the parameter ϵ slows down the learning process only changing the coefficient by a tiny amount. Thus, many iterations are required to yield reasonable parameter estimates among a large set of potential predictor variables. Figure1 illustrates the coefficient path of the coefficient estimates over 2000 iterations.

2.2 The learning rate

In practice one of the main problems is how to set the learning rate, ϵ. Whenϵ is too small, the algorithm is less efficient, and when it is too large, estimates can fail to span the full regularization path. On heuristic mentioned by Tibshirani (2015) is to start with a large value of ϵ, and to plot

[2]Also note, that this is analogous to least squares boosting, with the number of trees equal to the number of predictors.

[3]Note that some references initialize r as $r = y - \bar{y}$ where $\bar{y}$ refers to the mean of y.

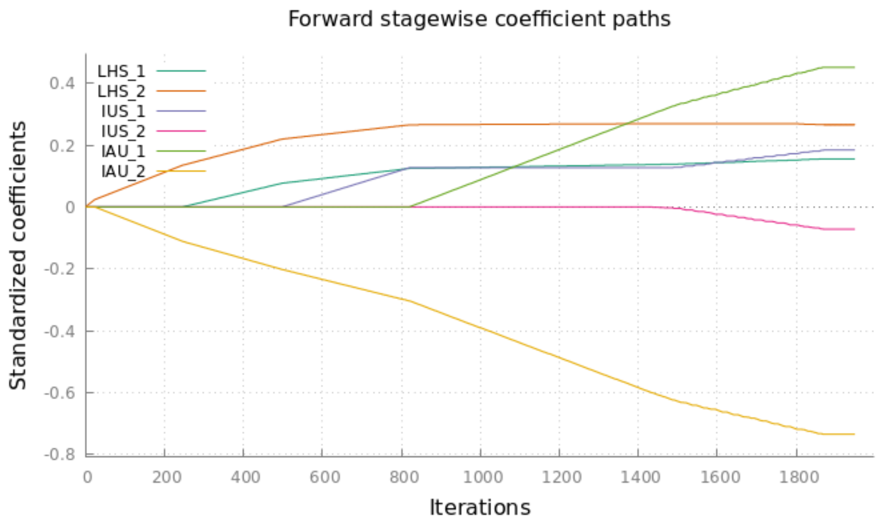

Figure 1: Coefficient paths

the progress of the achieved loss. With a reasonable choice of ϵ, one should see a monotonic decline in loss with the number of steps realized. If ϵ is too large, one observes in practice that the achieved loss stops its monotone progress and starts to fluctuate up and down.

In principle one could lower the learning rate, and continue the stagewise algorithm from that last step until some stopping criteria is achieved. This continuation, however, is not supported by this package, yet. Also note, that the 'optimal' choice of ϵ can be determined by cross-validation.

2.3 Early stopping

Early stopping rules are important for two reasons: First, one wants to avoid over-fitting meaning that the model learns the training set well but terrifically fails on the test set. Second, it may be unnecessary to run $\bar{N}$ iterations if no improvement (in terms of model fit) can be seen after $N \ll \bar{N}$ iterations.

Before version 0.2, the implemented *early stopping* strategy checks the absolute correlation $|<r, x_j^i>|$: In case the absolute correlation does not improve for n (e.g., $n = 30$) iterations, we assume that the coefficient estimates have converged and stop the algorithm. Figure 2 illustrates the development of the absolute correlation between the residuals and the selected variables. As one can see, after about 250 iterations the improvement in the correlation coefficient becomes marginal.

Since version 0.2 two additional more sophisticated early stopping rules are implemented:

1. **residual_corr_abs**: Check that the absolute correlation, $|<r, x_j^i>|$, decreases at least by some minimum value, $\underline{c}$. The early stopping indicator function follows the following rule:

$$I = \begin{cases} 0 & \text{if } \Delta|<r, x_j^i>| \geq \underline{c} \\ 1 & \text{if } \Delta|<r, x_j^i>| < \underline{c} \end{cases}$$

where Δ refers to the change in the statistics between the stagewise iterations i and $i-1$,

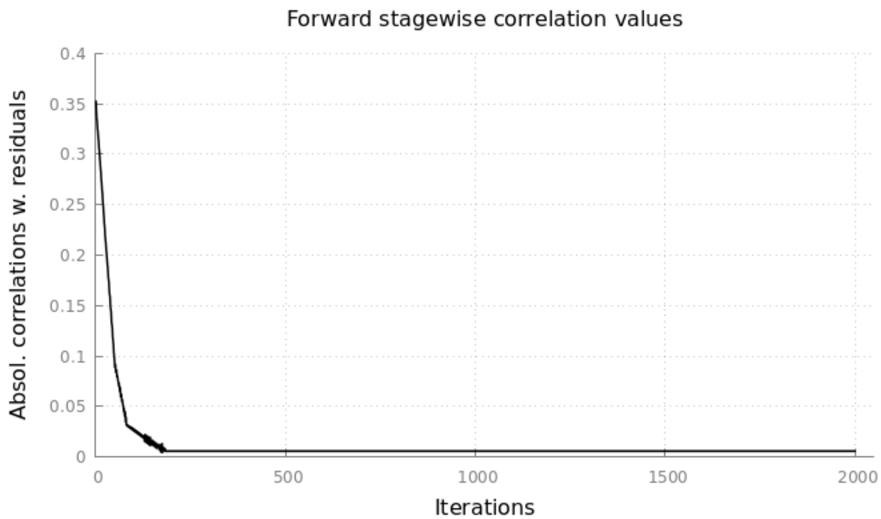

Figure 2: Correlation dynamics between the updated residuals and the most correlated predictor x_j^i.

respectively. The default value for $\underline{c}$ is 0.01.

2. **residual_corr_rel**: Check that the relative change in the absolute correlation, $| < r, x_j^i > |$, decreases at least by some minimum value, $\underline{c}$. The early stopping indicator function follows the following rule:

$$I = \begin{cases} 0 & \text{if } \frac{\Delta|<r,x_j^i>|}{|<r,x_j^{i-1}>|} \geq \underline{c} \\ 1 & \text{if } \frac{\Delta|<r,x_j^i>|}{|<r,x_j^{i-1}>|} < \underline{c} \end{cases}$$

The default value for $\underline{c}$ is 0.05.

Both early stopping rules yield much faster convergence.

3 Install and load the package

The `fsboost` package is publicly available on the gretl server. The package must be downloaded once, and loaded into memory each time gretl is started.

```
clear
set verbose off

pkg install fsboost         # Download package (only once needed)
include fsboost.gfn         # Load the package into memory
help fsboost                # Open the help file
```

4 Example

For illustration we use the "mroz87" cross-sectional data set comprising 753 observations from a study of income dynamics. The endogenous variable is named `WW` and refers to the a wife's 1975 average hourly earnings (in 1975 dollars). The data set includes additional 17 potential predictors.

4.1 OLS benchmark

The sample script opens the data set, sets the right-hand-side list of predictors and computes standard least square estimates first.

```
open mroz87.gdt --quiet
list RHS = const dataset
RHS -= WW              # drop lhs variable
ols LHS RHS             # run ols as benchmark
```

The OLS output is:

```
Model 1: OLS, using observations 1-753
Dependent variable: LHS

             coefficient    std. error    t-ratio    p-value
  ----------------------------------------------------------
  const       3.42796       2.32166        1.477     0.1402
  LFP         3.47221       0.257362      13.49      3.22e-37 ***
  WHRS       -0.00110533    0.000151422   -7.300     7.51e-13 ***
  KL6        -0.0444317     0.178645      -0.2487    0.8037
  K618       -0.0217407     0.0699922     -0.3106    0.7562
  WA         -0.00253127    0.0225216     -0.1124    0.9105
  WE          0.215215      0.0490769      4.385     1.33e-05 ***
  RPWG        0.537172      0.0457955     11.73      3.04e-29 ***
  HHRS       -0.000473085   0.000170096   -2.781     0.0056   ***
  HA          0.000968849   0.0216386      0.04477   0.9643
  HE         -0.0525726     0.0359107     -1.464     0.1436
  HW         -0.115575      0.0377458     -3.062     0.0023   ***
  FAMINC      3.22894e-05   1.51867e-05    2.126     0.0338   **
  MTR        -4.90658       2.38066       -2.061     0.0397   **
  WMED       -0.0289774     0.0298579     -0.9705    0.3321
  WFED       -0.0209198     0.0282231     -0.7412    0.4588
  UN         -0.0177959     0.0260952     -0.6820    0.4955
  CIT         0.0104698     0.178266       0.05873   0.9532
  AX          0.00342378    0.0120605      0.2839    0.7766

Mean dependent var   2.374565   S.D. dependent var   3.241829
Sum squared resid    3379.759   S.E. of regression   2.145828
R-squared            0.572351   Adjusted R-squared   0.561863
F(18, 734)           54.57555   P-value(F)           4.3e-122
Log-likelihood      -1633.773   Akaike criterion     3305.547
Schwarz criterion    3393.404   Hannan-Quinn         3339.394
```

4.2 Forward-stagewise regression

Next, we run the forward stagewise regression by calling the `fsreg()` function. By default the user has two pass only to mandatory arguments: the endogenous series and a list of exogenous. Additional one can pass a bundle comprising optional parameters such as the learning rate. In the following example, we set the learning rate to a rather low value:

```
bundle opts = defbundle("learning_rate", 0.0002)  # optional parameter
bundle B = fsreg(LHS, RHS, opts)
print_fsboost_results(B)      # Print estimation results
```

The regression results are as follows:[4]

```
Forward-stagewise regression results (no inference)
-------------------------------------------------------

             coefficient    std. error   z    p-value
  ---------------------------------------------------
  const      -1.24238           NA       NA     NA
  LFP         2.60587           NA       NA     NA
  WHRS       -0.000284255       NA       NA     NA
  WE          0.132787          NA       NA     NA
  RPWG        0.494871          NA       NA     NA
  FAMINC      1.02652e-05       NA       NA     NA
  MTR        -0.644518          NA       NA     NA

  Learning rate = 0.0002
  Number of iterations = 4964
  Correl. w. residuals = -0.0578633
  S.E. of regression = 2.18792
  R-squared = 0.547703
```

The estimator converged after 4964 iterations and selected only 6 out of 17 potential predictors. The final correlation coefficient between the residuals and the predictor mostly correlated with these residuals is close to zero (-0.057) after the early stopping rule applies. Even though only 6 predictors are selected being relevant, the R^2 statistics is of similar size compared to the OLS-based equivalent. The standard error of the residuals is slightly smaller (2.18) compared to the OLS-based value of 2.14. The average execution time of the `fsreg()` function is 0.8 seconds when repeating the exercise 20 times on a 6 year old i7 Intel notebook CPU.

4.3 Plot the coefficient path

The public function `plot_coefficient_paths()` takes as a single mandatory argument the the returned bundle by the `fsreg()` function.

[4] Note that inference such as a t-test or F-tests is not supported. There is ongoing research in the statistics community on how to conduct inference based on sparse estimates.

```
plot_coefficient_paths(B)
```

The function returns the coefficient paths of the active set (non-zero coefficients) only which is depicted in Figure 3. The plot nicely depicts how the point estimates of the *active set* of variables gradually converge to their final values before the early stopping criteria applies.

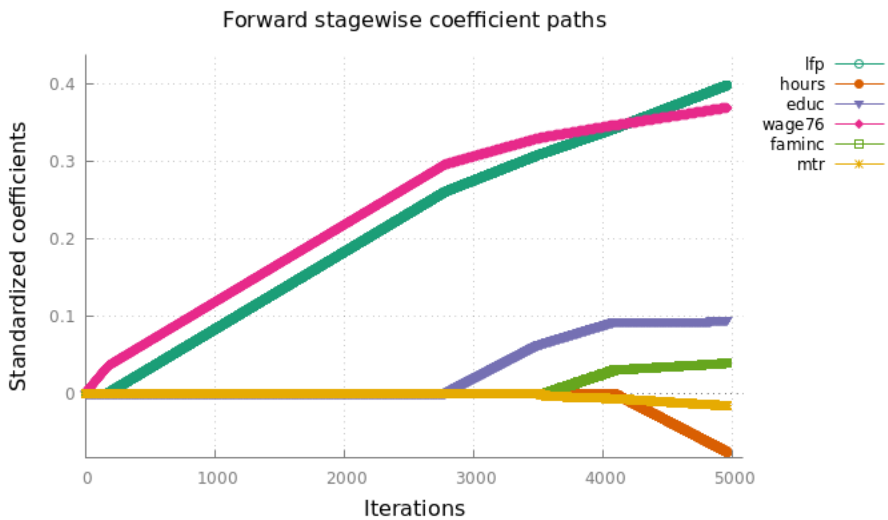

Figure 3: Coefficient paths of the *mroz* model.

4.4 Compute predictions

The function `fsboost_predict()` computes the predictions as $\hat{y} = Xb$ where X is a matrix of dimension $N \times k$ and b is the coefficient vector of dimension $k \times 1$. Note that coefficient vector b also *includes* the zero-coefficients. The following example shows on how to apply the function for predicting some arbitrary five observations:[5]

```
smpl 1 5               # Restrict the data set to the test set
matrix preds = fsboost_predict(RHS, B)
print preds

preds (5 x 1) [t1 = 1, t2 = 5]
      3.5131
      3.5950
      4.1637
      4.0069
      4.4380
```

[5]In practice one 'trains' the model on a separate training data set and predicts on another test set.

5 Using the GUI

Instead of scripting, one may access the `fsboost` procedure by means of the Gretl GUI. Once the package is installed and loaded, simply open the menu"*Model -> Other linear models -> Forward Stagewise*". This a menu window as depicted in Figure4.

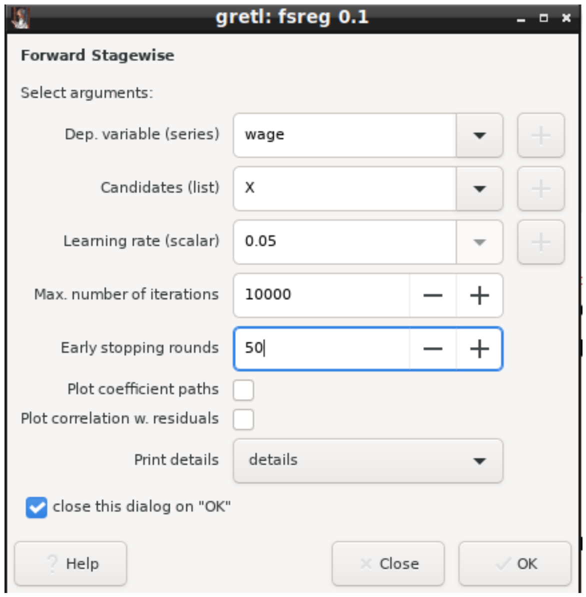

Figure 4: GUI access window

6 Public functions and parameter values

The following public functions currently exist.

6.1 fsreg()

The `fsreg()` function marks the main function for executing the forward-stagewise linear regression. The function arguments are:

```
fsreg(const series y, const list X, bundle opts[null])
```

Return type: `bundle`

Argument	Description
y	series, Endogenous variable
X	list, Non-empty list of predictor variables
opts	bundle, Pass parameters for controlling the algorithm (optional)

Return type: `bundle`

The returned bundle includes various which are listed in Table1.

The additional parameters which can be passed by means of the `opts` bundle to `fsreg()` are shown in Table2.

Key	Description
`rho_values`	Vector holding correlation coefficients with the residuals for each iteration
`max_num_iterations`	Number of the maximum iterations
`actual_num_iterations`	Actual number of iterations ran
`learning_rate`	Learning rate
`early_stopping_strategy`	String referring to some early stopping strategy
`early_stopping_rounds`	Number of iterations of no improvement before stopping
`early_stopping_threshold`	Scalar setting the threshold for a minimum improvement in some score
`yname`	String holding the name of the endogenous variable
`Xnames_wo_constant`	String array holding the names of all predictor variables without the constant
`Xnames`	String array holding the names of all predictor variables incl. the constant
`X_final`	List incl. finally selected predictors.
`betas`	Matrix holding the coefficient point estimates across iterations (rows) for each predictor (columns)
`coeff_nonzero`	k by 1 vector incl. the final coefficient point estimates for all selected predictors (non-zero coefficients)
`coeff`	n by 1 vector incl. the coefficient point estimates of all predictors (incl. zero coefficients)
`with_constant`	Boolean taking zero if the passed list X does not incl. an intercept, otherwise one
`verbose`	Integer indicating the level of verbosity
`T`	Number of effective (non-missing) observations.
`yhat`	Fitted values using final point coefficient estimates
`uhat`	Estimated final residuals
`uhat_variance`	Variance of the estimated final residuals
`r2`	R-squared stats. computed as $1 - \sum(y - \hat{y})^2 / \sum(y - \bar{y})^2$.
`r2_qcorr`	R-squared stats. based on quadratic correlation between y and $\hat{y}$
mse	In-sample based mean squared error statistics
`uhat_first_order_corr`	1st order serial correlation coefficient of final residuals (for time-series data set)

Table 1: Bundle content as returned by `fsreg()`.

Parameter	Data type	Description	Default value
`learning_rate`	scalar	Learning rate; $0 < \epsilon < 1$	0.01
`max_num_iterations`	int	Number of the maximum iterations	10000
`early_stopping_strategy`	string	Early stopping strategy: "residual_corr_abs" "residual_corr_rel"	"residual_corr_abs"
`early_stopping_rounds`	int	Number of iterations of no improvement before stopping	50
`early_stopping_rounds`	scalar	For strategy "residual_corr_abs" For strategy "residual_corr_rel"	0.01 0.05
`verbose`	bool	Print details or not: either 0 gor 1	1 (True)

Table 2: Parameters which can be set through the optional bundle `opts`.

6.2 print_fsboost_results()

The `print_fsboost_results()` function takes the resulting bundle returned by the `fsreg()` function, and prints a summary of the estimation results. The argument is:

```
print_fsboost_results(const bundle B)
```

Return type: `void`

6.3 plot_rho_values()

For plotting the iterative development of the correlation between the residuals, r^i, and the most correlated predictor, x_j^i, call the `plot_rho_values()` function. It takes the resulting bundle returned by the `fsreg()` function. The argument is:

```
plot_rho_values(const bundle B)
```

Return type: `void`

Argument	Description
B	bundle, Bundle returned by `fsreg()`

6.4 plot_coefficient_paths()

For plotting the development of the coefficients (coefficient paths), call the `plot_coefficient_paths()` function. It takes the resulting bundle returned by the `fsreg()` function. The argument is:

```
plot_coefficient_paths(const bundle B)
```

Return type: `void`

Argument	Description
B	bundle, Bundle returned by `fsreg()`

6.5 fsboost_predict()

This function computes the prediction using the point estimates and a list of regressors. This list gmust comprise the same set of regressors as passed to the `fsreg()` function. The function takes two arguments: A list of regressors and the resulting bundle returned by the `fsreg()` function. The argument is:

```
fsboost_predict(const list X, const bundle B)
```

Return type: $N \times 1$ matrix on success, otherwise scalar with NA value.

Argument	Description
X	list, Non-empty list of predictor variables with N observations.
B	bundle, Bundle returned by `fsreg()`

7 References

- Hastie, T., Taylor, J., Tibshirani R. and Walther G. (2007): Forward stagewise regression and the monotone lasso, *Electronic Journal of Statistics,* Vol. 1, 1-29.

- Tibshirani, R. (2015): A General Framework for Fast Stagewise Algorithms, *Journal of Machine Learning Research*, 16, 2543-2588.