MWSUG 2016 - Paper AA20

An Introduction to the HPFOREST Procedure and its Options

Carl Nord, Grand Valley State University, Grand Rapids, MI
Jacob Keeley, Grand Valley State University, Grand Rapids, MI

ABSTRACT

The acquisition of big data into usable formats can be quite a challenge. There is a large emphasis being placed on the efficiency and scope of data acquisition in many industries. With the increasing amount of data available to analyse, the best methods for creating predictive models from big data banks is becoming a desire of many sectors. In particular, sectors where prediction is the sole goal of the model. Decision tree techniques are a common and effective approach for creating optimal predictive models. A procedure, the HPFOREST procedure, creates random forests models in a high performance environment. There is limited amount of information on this procedure, which makes it a prime candidate for discussion. This procedure allows for a predictive model to be created based on decision tree methodology. This method of model averaging has been known to produce predictive models that generalize quite favourably (Breiman, 2001). This paper will include an outline of basic code structure for the procedure as well as options such as specifying maximum trees, outputting fit statistics, etc. Scoring new data with a model file and generating helpful figures will be discussed as well.

INTRODUCTION

The interest in this topic was sparked from a lecture on random forests in a survival analysis course. This course utilized SAS® but in the lecture, the random forest models were not generated in SAS software. This sparked interest in searching for how to conduct random forests in SAS. The initial search was surprisingly sparse of information. Most of the information that can be located is found on blogs or SAS community threads. There is documentation on the HP Forest Node in SAS Enterprise Miner 13.2. While this is very informative information, the documentation does not cite how to perform a random forest in a hard-code format. A breakthrough occurred when PROC HPFOREST was discovered. This solution provided the more intuitive and organized solution that was desired. With all that said, it seemed useful to generate a paper that points out some of the basics of PROC HPFOREST in the hopes of sparking more awareness and discussion on this procedure. The topics that will be discussed in this paper include a very quick mention of the SAS product PROC HPFOREST is available in, basic code format for the procedure, examples of different model tuning options available, discussion of saving the model file to score data sets, and also a few useful figures to generate when utilizing random forest models. This overview should provide users with the basic knowledge to get started with PROC HPFOREST and begin tuning random forest models.

SAS SOFTWARE FOR PROC HPFOREST

After some searching for information on PROC HPFOREST it seemed necessary to discuss the software that PROC HPFOREST may be utilized in. The random forest algorithm associated with PROC HPFOREST is provided through SAS Enterprise Miner. A SAS High-Performance Data Mining license is also required ("SAS High-Performance Analytics", 2016). After a good amount of time searching, the documentation for PROC HPFOREST was found in SAS Enterprise Miner 14.1 High-Performance Procedures Documentation. This is secure documentation.

The authors utilized the SAS software available from Grand Valley State University. This included SAS Enterprise Guide 7.1 and SAS Enterprise Miner Workstation 13.2. The authors utilized SAS Enterprise Guide 7.1 for all the PROC HPFOREST code in the following stages of the paper. It should be noted that this was only possible with SAS Enterprise Miner being available as well. Enterprise Guide 7.1 does not have the stand alone capability of running PROC HPFOREST. However, due to the lack of openly available online documentation, it was useful to be able to utilize this procedure in Enterprise Guide 7.1 as it allowed for quick and easy assessment of procedure options and code structure.

BASIC PROC HPFOREST CODE OUTLINE

This section will outline the basic code for running PROC HPFOREST. This includes the basic code to define the response variable and predictors. Further options for tuning the random forest model will be discussed in the next section.

The following provides a generalized code outline for PROC HPFOREST:

```
proc hpforest data= <data>;

      target <response variable> /

            level= <binary, nominal, interval>;

      input <predictor 1, predictor 2…>/

            level= <binary, nominal, ordinal, interval> ;

run;
```

The TARGET statement defines the response variable. The LEVEL option defines the response variable as binary, nominal, or interval (continuous). The INPUT statement performs a similar action, just for the predictor variables. The LEVEL option for the INPUT statement defines the predictors as binary, nominal, ordinal, or interval.

A question may be how to set up the code if there are multiple predictors of different class. For example, predictor 1 and predictor 2 are interval, predictor 3 is nominal, and predictor 4 is binary. The following provides an example of how this situation would be properly coded:

```
proc hpforest data= <data>;

      target <response variable> /

            level= <binary, nominal, interval>;

      input predictor 1, predictor 2/

            level= interval;

      input predictor 3/

            level= nominal;

      input predictor 4/

            level= binary;

run;
```

Each INPUT statement may only be associated with one unique LEVEL option. However, multiple predictors may be defined for each INPUT statement, as long as they are of the same INTERVAL specification.

This is a very basic outline of the procedure but a necessary step in the process, simply due to the lack of online documentation. The next section will delve into more options of the procedure for tuning the random forest model.

USEFUL OPTIONS IN PROC HPFOREST

This topic of the paper delves deeper into the model tuning options of PROC HPFOREST. Not all of the options are addressed but the most common are outlined. The following is an example of a more complete random forest model. Descriptions of the options will be outlined below the code. Note that the purpose of this section of the paper is to highlight the options associated with this procedure. This model was not intended to provide a template for optimal fitting.

An automobile developer may have a Miles Per Gallon City target for a new car they are developing. They want to be able to predict the Miles Per Gallon City based on a number of attributes associated with the automobile they are developing. Utilizing the SASHELP.CARS data set, the developer decides that the predictors of MPG_City they want included are as follows:

- Number of Cylinders
- Wheelbase Distance
- Size of the Engine
- Weight of the Vehicle
- Length of the Vehicle
- Horsepower

The PROC HPFOREST Code to perform an initial model run for this situation is below:

```
proc hpforest data=sashelp.cars

     maxtrees= 500 vars_to_try=4

     seed=600 trainfraction=0.6

     maxdepth=50 leafsize=6

     alpha= 0.1;

          target MPG_City/ level=interval;

          input  cylinders wheelbase enginesize weight length

                 horsepower/ level=interval;

     ods output fitstatistics = fit;

run;
```

Overview of various options in SAS Enterprise Miner 14.1 High-Performance Procedures Documentation:

- TRAINFRACTION specifies the fraction of the original observations used for bootstrapping each tree.
- SEED sets the randomization seed for bootstrapping and feature selection.

- MAXTREES specifies the maximum number of trees.
- VARS_TO-TRY specifies the randomized number inputs to select at each node.
- LEAFSIZE indicates the minimum number of observations allowed in each branch.
- ALPHA specifies the p-value threshold a candidate variable must meet for a node to be split.
- MAXDEPTH specifies the number of splitting rules for the nodes.
- PRESELECT indicates the method of selecting a splitting feature.

Also, notice the ODS OUTPUT statement. This provides the model fit statistics in an output data set. This is where the out of bag average square errors are found in output format. A snapshot example of a subset of the model fit statistics output is shown in Table 1 below.

NTrees	**NLeaves**	**PredAll**	**PredOob**
1	25	8.71949	13.6515
2	53	8.26232	14.2445
3	76	7.24522	11.6454
4	98	6.67597	10.4466
5	115	6.49442	9.3086
6	138	6.15208	9.1756

Table 1. Example of Fitstatistics output

SAVING AND SCORING A BINARY MODEL FILE

This section of the paper deals with a way of saving the model file for future data set scoring. This was an interesting portion of code found on a SAS community thread (“Scoring PROC HPForest”, 2015). The SAVE statement appears in red, indicating an error. However, this code works perfectly. This will save the model file as a binary image to be utilized to score future data sets or samples.

```
proc hpforest data=sashelp.cars

     maxtrees= 500 vars_to_try=4

     seed=600 trainfraction=0.6

     maxdepth=50 leafsize=6

     alpha= 0.1;

          target MPG_City/ level= interval;

          input weight length horsepower/ level=interval;

ods output fitstatistics = fitstats;

save file = "FilePath\model_fit.bin";
run;
```

The method to score new data utilizes PROC HP4SCORE along with the saved binary model file. The topic of scoring data has been a very common question seen on blogs and threads, which makes the topic relevant to discuss. Sticking with the Miles Per Gallon City example, say that the MPG_City is sought to be predicted for a sample of other cars not included in the sashelp.cars data set.

Code to accomplish this utilizing the established model file above is as follows:

```
proc surveyselect data= other_cars_data

     method=srs N=100 out= samp;


proc hp4score data=samp;

     id MPG_City;

     score file= "FilePath\model_fit.bin"

     out=scored;

run;
```

This code will estimate the MPG_City of cars not included in the sashelp.cars data set utilizing the model file previously created. The first six observations found in the “out = scored” data set is found in Table 2 below.

MPG_City	P_MPG_City	_WARN_
18	17.881384632	
17	18.104731704	
22	22.14511724	
23	21.887448092	
18	18.228784647	
18	18.352983715	

Table 2. Example of scored data output

MPG_City is the observed value from the input data set. P_MPG_City is the predicted MPG_City determined by the saved random forest model.

USEFUL GRAPHICS FROM HPFOREST OUTPUT

There are no direct plot options associated with PROC HPFOREST. However, there are a myriad of graphics and methods associated with generating optimally tuned random forest models. The following examples are common figures associated with the random forest modeling process. These figures were obtained through manipulation of PROC HPFOREST output. Code to generate these figures will be provided in a code appendix due to its length.

Investigating Average Square Error with Different Number of Variables to Try

With the randomized splitting nature of random forests, a common inspection of model performance involves observing the average square error associated with different number of variables to try for splitting. Figure 1 on the next page illustrates this technique for the MPG_City Model grouped by 2, 3, and 4 variables to try.

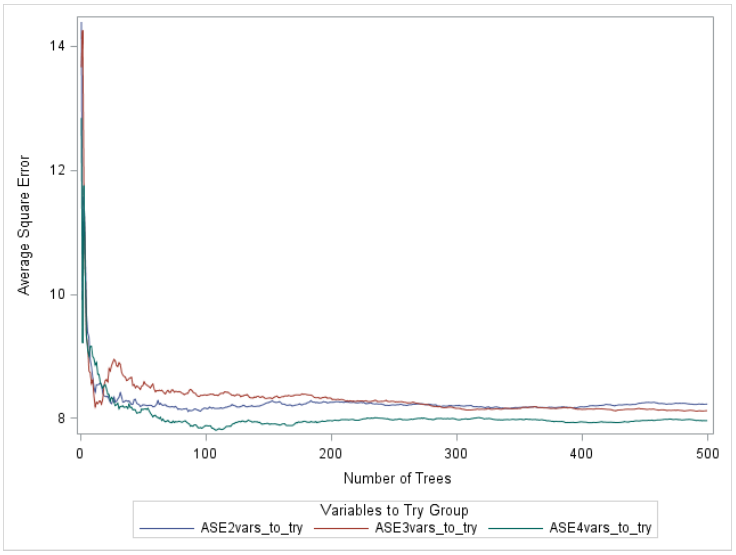

Figure 1. Display of Average Square Error by Number of Trees, Grouped by Number of Variables to Try.

Notice in Figure 1 how the lowest Average Square Error is associated with the ASE4vars_to_try group in this case. Also, this figure is useful as it illustrates how the Average Square Error progresses over the number of trees utilized. For the ASE4vars_to_try group, the Average Square error flattens out at around 250 trees. A general tip is that when the error flattens, the number of trees to utilize is just after this flattening occurs. (Breiman, 2001) This is especially useful when very large numbers of trees are being investigated and computational time is of importance.

Plotting Average Square Error with Different Leaf Size Specifications

The number of observations required in each leaf also effects the average square error of random forest models. As can be seen in Figure 2, the smallest leaf size of four results in the lowest average square error.

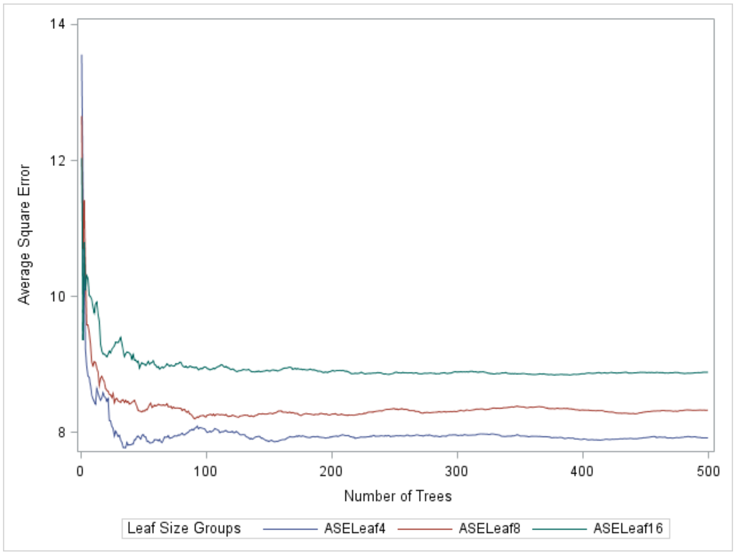

Figure 2. Display of Average Square Error by Number of Trees, Grouped by Leaf Size.

Predicted by Observed Plot

Another common figure associated with Random Forest Models is a standard Predicted by Observed plot. These plots assess the generalizability of a model when the predicted values are from a new data set that is scored through the model. This plot is illustrated in Figure 3 below.

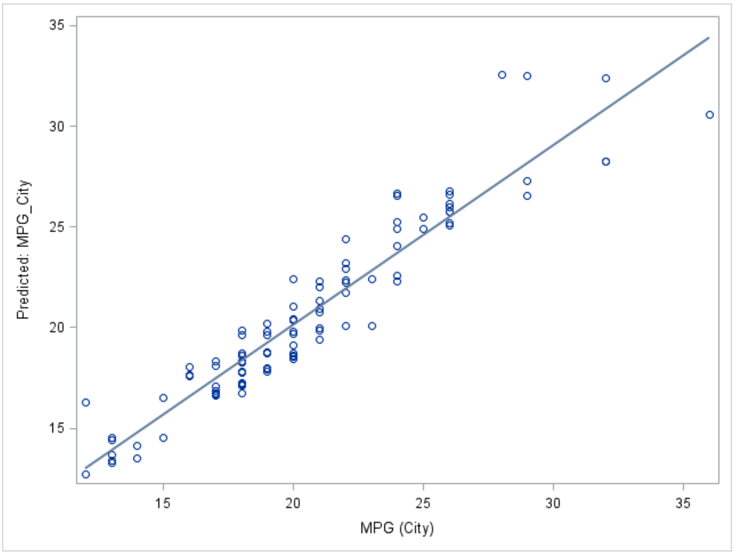

Figure 3. Predicted MPG City by Observed MPG City from a Scored Data set.

These are three basic plots associated with random forest modeling. There are numerous plotting options for both optimal model tuning and assessing model generalizability. As a user becomes more familiar with PROC HPFOREST, more in depth figures regarding variable importance and variance/bias tradeoffs may be explored.

CONCLUSION

PROC HPFOREST provides a quality method for predictive modeling. The information presented in this paper provides a solid foundation for pursuing the use of PROC HPFOREST, as well PROC HP4SCORE for scoring the model. This is especially helpful for individuals not familiar with the HPFOREST NODE in Enterprise Miner. Interpretability is not one of the strengths of random forests and this paper does not delve into the nature of random forest methodology, therefore it is highly recommended that individuals wanting to run PROC HPFOREST study Leo Breiman's work for more information regarding how the model is actually fit. As users become more familiar with tweaking the different options available, model fits should improve. As demonstrated, graphical representations of average square error amongst other statistics can aid when determining the optimal settings under which to run PROC HPFOREST.

REFERENCES

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. http://doi.org/10.1023/A:1010933404324

SAS High-Performance Analytics Tip #1: How it differs from SAS Grid & SAS In-Memory Analytics. (n.d.). Retrieved September 8, 2016, from https://communities.sas.com/t5/SAS-Communities-Library/SAS-High-Performance-Analytics-tip-1-How-it-differs-from-SAS/ta-p/244538

SAS Institute, Inc. SAS Enterprise Miner High-Performance Procedures. Documentation, Version 14.1. Cary, NC: SAS Institute Inc. (2015).

Scoring PROC HPForest. (2015). Retrieved September 8, 2016, from https://communities.sas.com/t5/SAS-Data-Mining/Scoring-PROC-HPForest/m-p/210979#M2981

ACKNOWLEDGMENTS

We would to thank Dr. Robert Downer at Grand Valley State University for encouraging us to pursue formulating a paper on this topic. Additionally, we would like to thank Dr. Daniel Frobish for inspiring this topic during his lectures on random forests.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Carl Nord
Grand Valley State University
nordc@mail.gvsu.edu

Jacob Keeley
Grand Valley State University
keeleyj@mail.gvsu.edu

CODE APPENDIX

Figure 1 Code:

```
%macro rf(num, num2);

	proc hpforest data=cars

	maxtrees= 500 vars_to_try=&num

	seed=600 trainfraction=0.6

	maxdepth=50 leafsize=6 alpha= 0.1;

	target MPG_City

		/level=interval;

	input  cylinders wheelbase enginesize weight length horsepower

		/level=interval;

	ods output fitstatistics = fitstats&num2:

	run;

%mend rf;

%rf(2,1);
%rf(3,2);
%rf(4,3);


proc sql;

	create table ASE_groups as

		select x.ntrees ,

		x.predoob as ASE2vars_to_try,

		y.predoob as ASE3vars_to_try,

		z.predoob as ASE4vars_to_try

		from fitstats1 x, fitstats2 y, fitstats3 z

		where x.ntrees = y.ntrees

		and x.ntrees =  z.ntrees

		and y.ntrees = z.ntrees;
run;
```

```
proc transpose data= ASE_groups out=ASE_groups1;
     var ASE2vars_to_try ASE3vars_to_try ASE4vars_to_try;
run;


data ASE_groups2;
     set ASE_groups1;
     array RF(1:500) col1- col500;

     do NTREES = 1 to 500;
          ASE = RF(NTREES);
     output;
     end;
     drop col1-col500 _LABEL_;
run;



proc sgplot data=ASE_groups2;


     series x=NTREES y=ASE/ group = _name_ ;

     LABEL ASE = "Average Square Error"

     NTREES = "Number of Trees"

     _name_ = "Variables to Try Group";

run;
```

Figure 2 Code:

```
%macro rf2(num, num2);

     proc hpforest data=cars

          maxtrees= 500 vars_to_try=4

          seed=600 trainfraction=0.6

          maxdepth=50 leafsize=&num alpha= 0.1;

          target MPG_City

               /level=interval;

          input cylinders wheelbase enginesize weight length
          horsepower
          /level=interval;

          ods output fitstatistics = fitstats&num2;

     run;

%mend rf2;

%rf2(4,1);
%rf2(8,2);
%rf2(16,3);

proc sql;

create table ASE_Leaf as

     select x.ntrees ,

          x.predoob as ASELeaf4,

          y.predoob as ASELeaf8,

          z.predoob as ASELeaf16

     from fitstats1 x, fitstats2 y, fitstats3 z

     where x.ntrees = y.ntrees

     and x.ntrees =  z.ntrees

     and y.ntrees = z.ntrees;
run;

proc transpose data= ASE_Leaf out=ASE_Leaf1;
     var ASELeaf4 ASELeaf8 ASELeaf16;
run;


data ASE_Leaf2;
     set ASE_Leaf1;
     array RF(1:500) col1- col500;

     do NTREES = 1 to 500;
          ASE = RF(NTREES);
     output;
     end;
     drop col1-col500 _LABEL_;
run;

proc sgplot data=ASE_Leaf2;

     series x=NTREES y=ASE/ group = _name_;
     LABEL ASE = "Average Square Error"
           NTREES = "Number of Trees"
            _name_ = "Leaf Size Groups";
run;
```