Paper 155-2011

The Anti-Curse: Creating an Extension to SAS® Enterprise Miner™ Using PROC ARBORETUM

Andrew Cathie, SAS Institute (NZ) Ltd, Auckland, New Zealand

ABSTRACT

SAS Enterprise Miner introduced a toolkit to enable programmers to create their own nodes in SAS Enterprise Miner 5.3. These nodes can then be used in the same manner as regular SAS Enterprise Miner nodes under the Sample, Explore, Modify, Model, Assess, or Utility menus. The user-created nodes make use of the flow features and metadata in SAS Enterprise Miner and allow you to implement your own algorithms, perform special-purpose data manipulation or reporting, and so on. This paper covers my experience of developing my own node, the nominal to interval node, which converts high-cardinality nominal inputs (ZIP codes, for example) to lower-dimensional interval inputs using decision trees. The paper goes some way toward addressing a persistent data mining problem, the curse of dimensionality.

During this I describe some of the tools available, hints, and tips, and how the finished result looks. Code is available from the author.

INTRODUCTION – THE CURSE OF DIMENSIONALITY

A persistent issue in data mining is dealing with highly dimensional data. Paradoxically adding more data (in terms of numbers of variables) does not necessarily improve the ability to develop a reliably predictive model, as this also increases the dimensions of the input vector. This problem has become known as the *curse of dimensionality*[3]. Reducing the number of dimensions can in some situations help the modeler develop a more reliable result.

Highly dimensional data is frequently encountered, very often in dealing with nominal values that are used as inputs to models. A nominal value has one of a finite number of discrete values, such as a product code or a region code. Typically, character variables used as inputs in data mining are nominal variables. Numeric variables may be nominal as well; a classic example being a zip code.

REDUCING DIMENSIONALITY WITH A DECISION TREE

The modeling technique used here is to use a decision tree where the input vector is just a single nominal variable, and the target variable is that of the main model. Using the decision tree's leaf predicted values, expressed as the probability of the target outcome, we have an interval value. Having calculated this interval value, we use this for subsequent modeling in place of the nominal variable.

For example, using the PVA 97 data set[5], there is a variable called *DemCluster* which is nominal with 54 distinct values, representing a cluster identifier for cases. This potentially has usefulness in a model, as there appears to be a relationship to the target variable *TargetB* as shown in Figure 1. However, the high number of distinct values introduces a lot of extra dimensionality for some techniques like regression or neural networks, which at their heart require interval values.

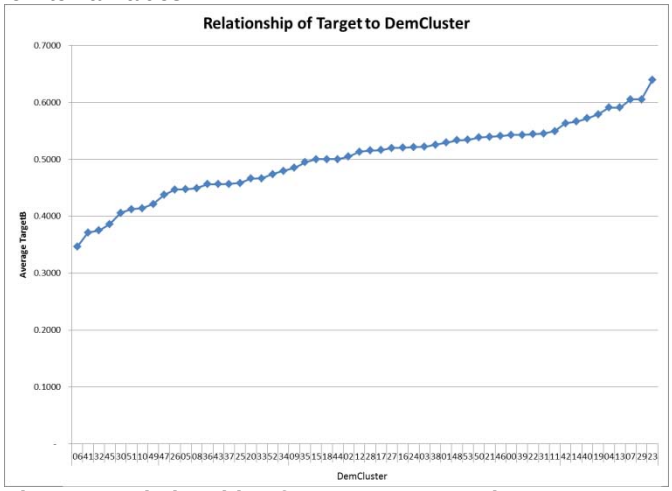

Figure 1: Relationship of *TargetB* to *DemCluster*

Using SAS Enterprise Miner to create a decision tree using just the nominal input variable of interest and the target variable, we can create a simple model to predict the target from the *DemCluster* variable. The following diagram shows an example of this:

Figure 2: SAS EnterpriseMiner process flow

The Variables property of the Decision Tree node is changed so that only the variables *DemCluster* (Input and Nominal) and *TargetB* (Target and Binary) are set to USE. Other changes to the Decision Tree properties are:

- The Bonferroni Adjustment is set to NO
  This suppresses the default penalty which would be applied to the input variable for having a large number of values
- Subtree is set to LARGEST
  This means the tree will not be pruned

Running the process flow results in the decision tree, shown in Figure 3.

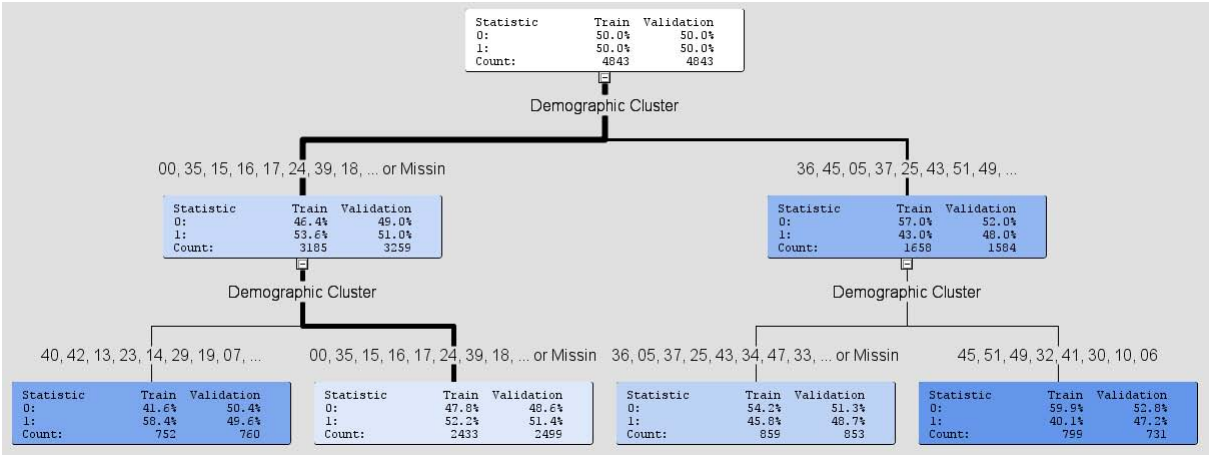

Figure 3: Decision Tree created on *DemCluster* only

This process flow has created new variables to the exported data set, *P_TargetB1* and *P_TargetB0*, which are respectively the predicted values for the target variable binary events based on the input *DemCluster* value. Using these predicted values, we have a transformation of the nominal value in *DemCluster* to an interval value, *P_TargetB1*. Figure 4 shows this relationship:

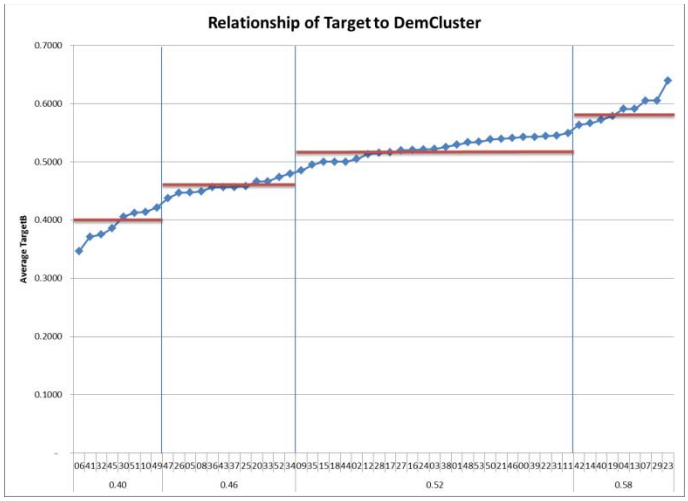

Figure 4: Relationship of TargetB toDemCluster, with decision tree leaf predictions overlaid

I could see myself needing to do this process again and again, so started to think about creating a tool to do this. This became the basis of the *Nominal-to-Interval* extension node.

DESIGNING FOR AN EXTENSION NODE

I found the easiest way to construct the tool was to build the code for one situation, then work out what variable elements needed to be parameterized for a generic situation, then to extend that to fit within the SAS Enterprise Miner context.

The starting point was PROC ARBORETUM (ARBOR). Examining the SAS Log from a SAS Enterprise Miner process flow such as the one above gives a fair idea of what it is doing behind the scenes. The procedure PROC ARBORETUM is behind the Decision Tree and Gradient Boosting nodes. PROC ARBOR is a topic to itself, however the key elements to achieve the work above are something like this:

```
PROC ARBOR DATA = aaem61.pva97nk ;                    /* Input data set          */
   INPUT    DemCluster / LEVEL = Nominal ;            /* Input variable          */
   TARGET   TargetB    / LEVEL = Binary               /* Target variable         */
                         CRITERION = PROBCHISQ        /* Split rule criterion    */
                         ;

   ASSESS   MEASURE = MISC ;                          /*Assessment measure       */
   SUBTREE  LARGEST ;

   SAVE     NODESTAT    = nodes
            PATH        = Path ;
QUIT ;
```

The meaningful output collectively describing the tree structure and the leaf predictions is found in the output data sets defined here as *nodes* and *path*.

To construct a generic solution, we need to use the SAS Enterprise Miner facilities to supply the following variable data to the above code:

- Input data set:
  This will be the data set supplied by the prior node in the process flow
- Input nominal variable(s)
  Any eligible nominal variable in the input data set
- Target variable name
  For simplicity, I chose to restrict the node to only support one target variable
- Splitting rule criterion (depends on whether target is interval or not)
- Assessment measure
  For example, miss-classification rate or average squared error.

This becomes the following code segment when made as part of the extension node. :

```
PROC ARBOR DATA        = &EM_IMPORT_DATA
           LEAFSIZE    = 5
           MINCATSIZE  = 5
           MAXBRANCH   = 2
           MAXDEPTH    = 6
           ALPHA       = 0.2
           PADJUST     = DEPTH
           MAXRULES    = 1
           MAXSURRS    = 0
           MISSING     = USEINSEARCH
           EXHAUSTIVE  = 5000
           ;
   INPUT  &InputVar   / LEVEL     = Nominal ;
   TARGET %EM_TARGET / LEVEL     = %EM_TARGET_LEVEL

%If %EM_TARGET_LEVEL = INTERVAL %Then
   %Do ;
                       CRITERION = PROBF ;
   %End ;
%Else
   %Do ;
                       CRITERION = PROBCHISQ ;
   %End ;



%If %EM_TARGET_LEVEL = INTERVAL %Then
   %Do ;
   ASSESS      MEASURE     = ASE ;
   %End ;
%Else
   %Do ;
   ASSESS      MEASURE     = MISC ;
   %End ;

   SUBTREE     LARGEST ;

   SAVE        NODESTAT    = Nodes
               PATH        = Path
               ;

RUN ;
QUIT ;
```

The macros and macro variables referenced above are mostly managed and populated from the SAS Enterprise Miner environment and are described in the reference documentation[4].

During the course of this development, I realised it would also be useful to have parameters to control:

- The number of distinct values eligible for a nominal variable:
  This is so nominal variables such as Gender with few values could be excluded from this processing if required.
  This subsequently became the *threshold* parameter.
- Whether the original nominal input variables were passed along the process flow or were rejected.
  This subsequently became the *HideOriginal* parameter.

Note that this node depends on PROC Arboretum in SAS, which is currently described as experimental and may change in future releases.

CREATING AN EXTENSION NODE

To get a node into SAS Enterprise Miner, I suggest first reading the very good developers guide[4] and building the worked example.

The main elements needed to create a node are:

- Some icon graphics to display in the toolbar and diagram workspace
- An XML file to declare some of the node's metadata
- The SAS code for the Create, Train, and Score functions of the node

ICON GRAPHICS

This is possibly the easiest stage, as there is no coding involved, or the hardest, if your proficiency with a bitmap editor is like mine. Using a tool such as Microsoft Paint, edit two simple icon images, one 16 x 16 pixels, and one 32 x 32 pixels. I found the easiest way was to copy one of the SAS supplied icons and adapt that. Each icon file needs to take the same name. Depending on your SAS installation, you will find a gif16 and gif32 folder within the SAS Enterprise Miner installation. On my development system (Windows 7) these are subfolders of C:\SAS\EMTM\Lev1\AnalyticsPlatform\apps\EnterpriseMiner\conf\components\.

Figure 5: 16-bit icon

Figure 6: 32-bit icon

XML FILE AND NODE PROPERTIES

The key purpose of the XML file is to define for SAS Enterprise Miner the location of your icons and source code, and to define user parameters of the node you are creating. Key points are:

- *name* Used as the node ID within a diagram, e.g. "N2I"
  Keep this relatively short as this is used as a prefix in the creation of object names within a SAS Enterprise Miner processflow.
- *displayName* Tool-tip name, e.g. "Nominal to Interval"
- *description* A short description also used in the tool-tip.

```
name        = "N2I"
displayName = "Nominal to Interval"
description = "Transform nominal input variables to interval input variables based on
their relationship to target."
```

- *group* This defines what sub-menu tab this node should appear in. Your options are SAMPLE, EXPLORE, MODIFY, MODEL, ASSESS and UTILITY. My node has a transformational type of functionality so I placed it in the MODIFY menu.
- *icon* Names the icon file created previously
- *prefix* A short string that will be used to prefix objects created as part of the process flow (e.g. data sets and so on). Needs to be reasonably short because of the SAS name restriction of 32 characters.

```
group       = "MODIFY"
icon        = "NominalToInterval.gif"
prefix      = "N2I"
```

Finally, the Property Descriptors section lists the source code location (or the entry point at least), which is a required property, and user-properties of the node. In my case I kept it simple and have just two properties:

- *threshold* How many distinct values there need to be before the Nominal to Interval node operates on an input. Default is 10, so a low cardinality variable like *Gender* would not be processed, but a high-cardinality variable like *City* would be.
- *HideOriginal* *Yes/No*, indicating whether the original nominal inputs should be hidden subsequently in the process flow (default is Yes)

```
<PropertyDescriptors>
      <Property type        = "String"
                name        = "Location"
                initial     = "CATALOG" />
      <Property type        = "String"
                name        = "Catalog"
```

```
                  initial      = "SASHELP.EXTN_N2I.N2I.SOURCE" />
       <Property type          = "int"
                  name         = "Threshold"
                  displayName  = "Threshold"
                  description  = "Number of distinct values required in nominal variable
before transforming to an interval value."
                edit           = "Y"
                initial        = "10" />
       <Property type           = "boolean"
                  name          = "HideOriginal"
                  displayName   = "Hide Original Variables"
                  description   = "Specifies whether to hide the original variables in
the data exported from this node."
                initial        = "Y" />
</PropertyDescriptors>
```

These properties are subsequently displayed as properties of the node when it is introduced to the SAS Enterprise Miner diagram:

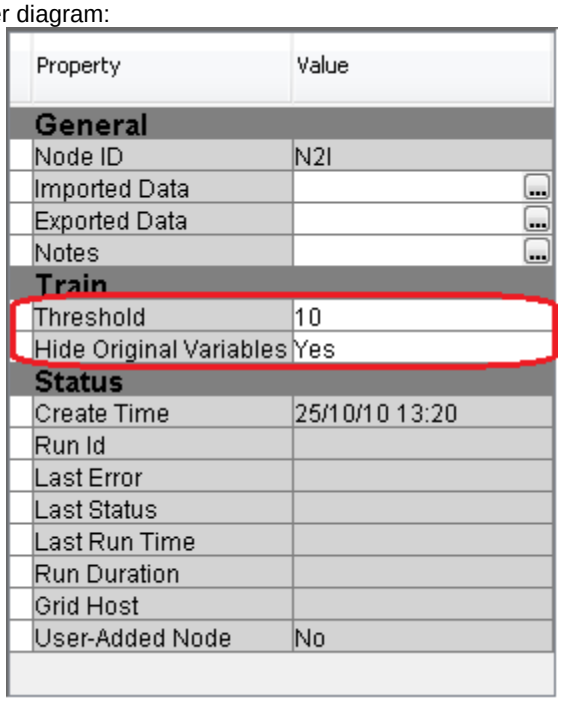

Figure 7: Properties of the Nominal to Interval node

Note that the errors within the XML file are not clearly reported, so you will need to take care with the syntax. In particular I discovered that key words within the XML file are case sensitive, so for example *`Property type = "String"`* is valid (first letter uppercase for *`String`*) but *`Property type = "string"`* is not (all lower case). However *`Property type = "Int"`* is invalid, *`Property type = "int"`* is valid. Follow the examples in the Developers Guide[4] and take particular care with case.

Once everything is prepared and copied to the appropriate folders for the SAS installation the new node should appear in the tool bar of SAS Enterprise Miner as shown:

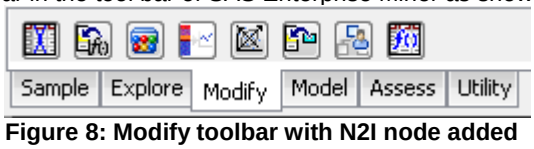

Figure 8: Modify toolbar with N2I node added

SAS CODE FOR THE EXTENSION NODE

There are three main pieces of code that will need writing, corresponding to equivalent SAS Enterprise Miner process flow actions, governed by a SAS Enterprise Miner macro variable &EM_ACTION:

- CREATE
  This code runs when the node is first placed on the diagram. Tasks done here are to define properties with defaults and to register objects such as data sets that are needed by the node.

- TRAIN
  Code that runs when you run a process flow as part of a training process over the input data set.
  This typically encompasses the largest part of the development effort. TRAIN code is required for the node.
  In this example, the TRAIN process loops over each eligible nominal input variable, builds a tree, and saves the result. Once all trees are built, data is scored using those trees and new variables created.
- SCORE
  Score executes as part of the training process and also when a node's score properties are changed. In this case there was no separate Score action.
- REPORT
  Report executes as part of the training process and also when a node's Report properties are changed. In this case there was no separate Report action.

This code is typically written in SOURCE entries in a catalog that is promoted to the SASHELP library. The entry point is the location referred to in the XML file.

CREATE CODE

This code runs only once, when the node is first placed on the diagram. Tasks done here are to define properties with defaults and to register objects such as data sets that are needed by the node.

My code is show below. It uses the SAS Enterprise Miner supplied macro `%EM_PROPERTY` to set up the attributes of the node that is being placed on the diagram. These can subsequently be accessed as macro variables, `&EM_PROPERTY_Threshold` and `&EM_PROPERTY_HideOriginal`. The other function is to use the `%EM_REGISTER` macro to register several data sets, `PATH`, `NODES`, `NominalVarStats`, `TreeNode` and `TargetVariables`. These are subsequently referenced using the macro variables `&EM_USER_NODES`, `&EM_USER_PATH`, etc in the context of SAS data sets. The `%Put` statements are not required, but do help with debugging.

It is important to let the Enterprise Miner macro tools perform the management of creating and registering objects. Bear in mind that a process flow diagram can have multiple instances of the same node so unless this is done there is the potential for name conflicts between nodes.

```
%Macro CREATE ;

%Put N2I_Create: Starting CREATE code... ;

/* Training Properties */
%Put N2I_Create: Defining THRESHOLD EM_PROPERTY. ;
%EM_PROPERTY(name=Threshold,    value=10);


%Put N2I_Create: Defining HideOriginal EM_PROPERTY. ;
%EM_PROPERTY(name=HideOriginal, value=Y );



/* Register Data Sets:
   - PATH, NODES        Proc Arbor output
   - NominalVarStats    Summary stats on the Nominal Vars processed, e.g. Number of Distinct
                        values
   - TreeNode           Proc Arbor output, join of PATH & NODES data sets
   - TargetVariables    Prediction variables created by Proc Arbor
*/
%Put N2I_Create: Registering data sets. ;

%EM_REGISTER(key=PATH,            type=DATA)
%EM_REGISTER(key=NODES,           type=DATA)
%EM_REGISTER(key=NominalVarStats, type=DATA)
%EM_REGISTER(key=TreeNode,        type=DATA)
%EM_REGISTER(key=TargetVariables, type=DATA)
```

```
%Put N2I_Create: ...Ending CREATE code. ;

%MEnd CREATE ;
```

TRAIN CODE

The meat of the code is in the TRAIN section. This is where input data is processed using PROC ARBOR, the results collected and applied, or Scored, on the data. The key steps of the TRAIN code are:

1. Validate the input data and parameters, and tidily exit if there are problems
2. Loop over each nominal input variable and build a decision tree
3. Use the saved node/leaf data to build DATA step score code
4. Execute the DATA step score code to calculate the new interval variables
5. Adjust metadata as required
6. Report results

VALIDATE THE INPUT DATA

Steps required are to:

- Verify that the input data set exists
- Check a target variable is defined in the metadata
- Check that only one Target variable is defined
- Check that there is at least one nominal input variable

LOOP OVER EACH NOMINAL INPUT VARIABLE

This makes use of the Enterprise Miner macro `%EM_NOMINAL_INPUT`, which is a blank-separated list of the nominal variables in the data set. Each one is processed in turn. First check is to see if the number of distinct values in over the threshold, using a small piece of SQL:

```
%Let InputVarIndex = 1 ;       /* Number of Input var within list                    */

%Let InputVar = %Scan(%EM_NOMINAL_INPUT, &InputVarIndex) ;

%Do %Until(&InputVar EQ ) ;


   %Put N2I_TRAIN: N2I_BuildTrees: InputVar value is <&InputVar>. ;
   %Put N2I_TRAIN: N2I_BuildTrees: InputVarIndex value is <&InputVarIndex>. ;

   /* Check distinct values of variable. If less than Threshold, don't
      make a decision tree.                                                  */
   %Put N2I_TRAIN: N2I_BuildTrees: Checking if Count Distinct of <&InputVar> is more
        than threshold, <&EM_PROPERTY_Threshold>... ;

   PROC SQL NOPRINT ;
      SELECT COUNT(DISTINCT &InputVar)
      INTO   :N2ICountDistinct
      FROM   &EM_IMPORT_DATA
      ;
   QUIT ;
```

If it passes, it now runs PROC ARBOR:

```
   %Put N2I_TRAIN: N2I_BuildTrees: ... and Count Distinct of <&InputVar> is
        <&N2ICountDistinct>. ;

   %If %Eval(&N2ICountDistinct GE &EM_PROPERTY_Threshold) %Then
      %Do ;

      %Put N2I_TRAIN: N2I_BuildTrees: Count Distinct <&InputVar>, <&N2ICountDistinct>
is >= <&EM_PROPERTY_Threshold>. ;

      PROC ARBOR DATA       = &EM_IMPORT_DATA
```

The output from the tree is saved in the data sets `Nodes` and `Path`. These are accumulated in to the previously registered data sets referenced using the macro variables `&EM_USER_NODES` and `&EM_USER_PATH`.

BUILD SCORING DATA STEP CODE

We need to use the node/leaf data to construct DATA step statements. This uses macro language (not shown) to write DATA step SELECT statements which encapsulate the content of the output decision trees. An example of the output is below:

```
/* Calculating INTERVAL variable for DemCluster  */
SELECT (DemCluster ) ;
   WHEN ("00") Do ;
               N2I_DemCluster0 = .4784217016029590000000000000000;
               N2I_DemCluster1 = .5215782983970400000000000000000;
               End ;
   WHEN ("01") Do ;
               N2I_DemCluster0 = .4784217016029590000000000000000;
               N2I_DemCluster1 = .5215782983970400000000000000000;
               End ;
   WHEN ("02") Do ;
               N2I_DemCluster0 = .4784217016029590000000000000000;
               N2I_DemCluster1 = .5215782983970400000000000000000;
               End ;
   . . .
   WHEN ("53") Do ;
               N2I_DemCluster0 = .4784217016029590000000000000000;
               N2I_DemCluster1 = .5215782983970400000000000000000;
               End ;
   OTHERWISE ;
END ;
```

EXECUTE THE DATA STEP SCORING CODE

Lastly the code just generated is executed, and so the new variables are created in the data set and immediately are available for use. SAS Enterprise Miner provides a macro to perform this task, as follows:

```
%Put N2I_TRAIN: Calling macro EM_MODEL... ;
%EM_MODEL(TARGET         = %EM_TARGET
         ,ASSESS         = Y
         ,DECSCORECODE   = Y
         ,FITSTATISTICS  = Y
         ,CLASSIFICATION = N
         ,RESIDUALS      = Y)
%Put N2I_TRAIN: ...EM_MODEL done. ;
```

At this point the process flow will have created new interval variables (`N2I_DemCluster0` and `N2I_DemCluster1` in this example) which are now available for down-stream use.

EXAMPLE

So how well does this extension node address the curse of dimensionality? Using the node is straight-forward; drag it in to the process flow at some point before a modeling node executes.

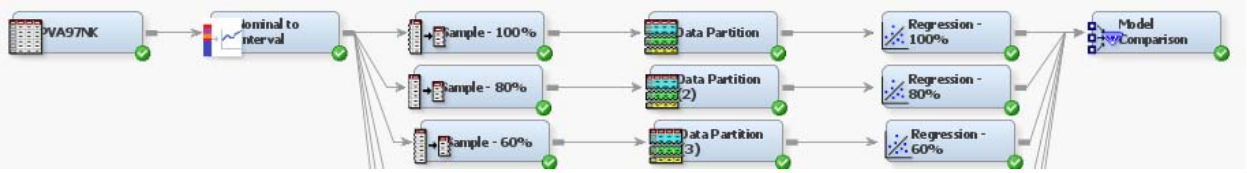

Figure 9 demonstrates its using the PVA data set. In each case a random sample of the original data set is used that is progressively smaller in size. A simple stepwise regression model is run, with and without the Nominal to Interval node inserted. The additional of the node makes a small improvement in reduced the validation error. Note that in this data set only one nominal variable, DemCluster, was replaced.

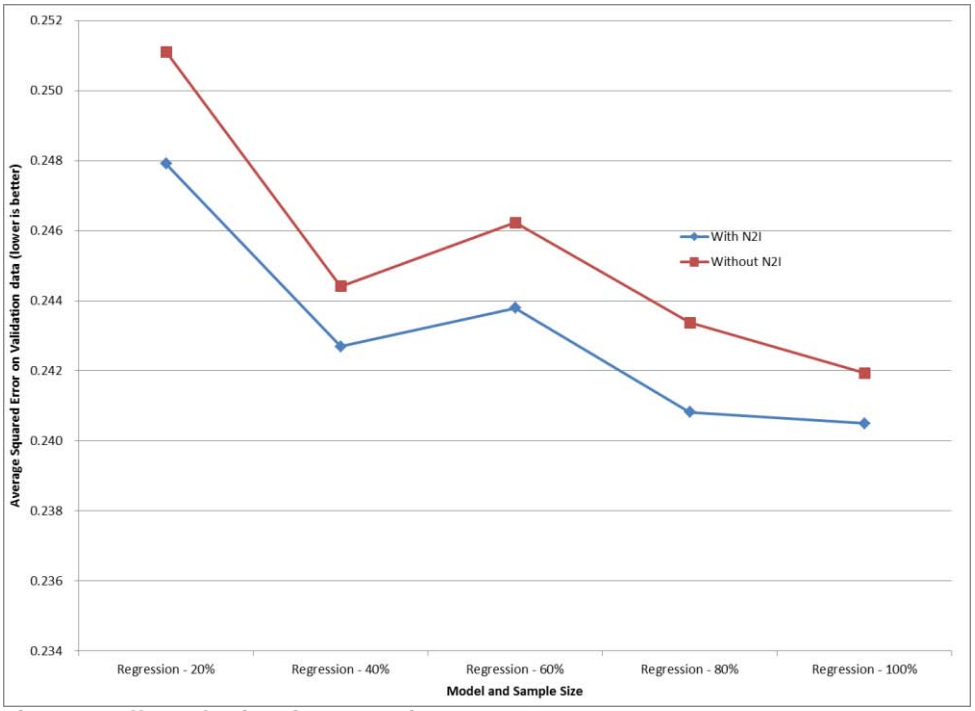

Figure 9: Effect of using the N2I node

CONCLUSION

This won't solve every problem, but it may be of use to quickly transform data to improve a model. In practice in a SAS Enterprise Miner project, high dimensionality will often arise from input variables which have large numbers of discrete values. Nodes such as Regression and Neural Network automatically transform these in to equivalent sets of binary numeric inputs which create these extra dimensions. If there is sufficient data it may still be predictive, so the effect might be more noticeable in smaller data sets.

To create your own extension nodes effectively you will need:

- Macro and SAS programming skills
- A development environment that includes a SAS Analytics Platform Server
- The Extension Nodes guide from SAS

This is an interesting feature of SAS Enterprise Miner that enables the SAS user community to readily extend the supplied functionality within SAS Enterprise Miner.

REFERENCES

1. H. R. Bittencourt, R. T. Clarke, Feature Selection by Using Classification and Regression Trees (CART)
2. Hugh Miller et al., Predicting customer behaviour: The University of Melbourne's KDD Cup report
3. Richard Ernest Bellman; Rand Corporation (1957). Dynamic programming. Princeton University Press. ISBN 9780691079516.
4. SAS Enterprise Miner 6.1 Extension Nodes Developer's Guide
5. Paralyzed Veterans of America (PVA) data set, released as part of the KDD 1998 contest: http://www.kdnuggets.com/meetings/kdd98/kdd-cup-98.html

RECOMMENDED READING

1. Extending SAS® Enterprise Miner™ with User-Written Nodes, SAS Education.

CONTACT INFORMATION

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

Andrew Cathie
SAS Institute (NZ) Ltd
Andrew.Cathie@sas.com