

An Introduction to the Process of Improving a Neural Network

YuTing Tian , M.S. Statistics, WCU PA

ABSTRACT

The top-level goal of this paper is to lay out a process for building a neural network in SAS®. It is hoped that a reader can use the process, shown in this paper, as a template for building a Deep Neural Network. A lower level goal is to build a network that can outperform a network in a paper by one of my professors.

Deep learning is a kind of neural network and a specific kind of machine learning (e.g. artificial intelligence). Deep learning is a recent and powerful machine learning algorithm that enables a computer to build a multi-layer non-linear model.

Even though deep neural networks are popular, not many papers discuss the overall process of building a neural network in SAS. This paper explores a practical application, associated with the process of deep neural network in SAS Enterprise Miner.

INTRODUCTION

This paper is divided into the following 9 sections. These correspond to the steps in our suggested process build a neural network.

Section 1) A brief review of model history

Section 2) Clean/trim variables using the replacement tab

Section 3) Impute missing values with tree based replacement logic using the impute tab

Section 4) Select non-redundant variables using the variable clustering tab in SAS Enterprise Miner

Section 5) Select predicting variables using the variable selection tab in SAS Enterprise Miner

Section 6) Build 22 different Neural Network models using the AutoNeural node to evaluate many different structures and transform functions.

Section 7) Use PROC Neural to train one deep-learning feedforward neural network using a structure that we thought would be effective.

Section 8) Use a decision tree to create a benchmark for comparison.

Section 9) Model Comparison node

SECTION 1) A BRIEF REVIEW OF MODEL HISTORY

A consumer credit company wants to take over the process for approval of loans and to automate the approval process. In order to attain this target, we use the Home Equity dataset, which contains 5960 observations recording re-payment. The target (Y) is BAD (a binary variable), indicating whether an applicant paid a loan or was delinquent. For each applicant, 12 input (X) variables were recorded.

Name	Model Role	Measurement Level	Description
BAD	Target	Binary	1=client defaulted on loan 0=loan repaid
CLAGE	Input	Interval	Age of oldest trade line in months
CLNO	Input	Interval	Number of credit lines
DEBTINC	Input	Interval	Debt-to-income ratio
DELINQ	Input	Interval	Number of delinquent credit lines
DEROG	Input	Interval	Number of major derogatory reports
JOB	Input	Nominal	Six occupational categories
LOAN	Input	Interval	Amount of the loan request
MORTDUE	Input	Interval	Amount due on existing mortgage
NINQ	Input	Interval	Number of recent credit inquiries
REASON	Input	Binary	DebtCon=debt consolidation, HomeImp=home improvement
VALUE	Input	Interval	Value of current property
YOJ	Input	Interval	Years at present job

Figure 1

The loan approval model will create a probability of a given loan applicant defaulting loan repayment. A threshold will be selected such that all loan applications whose probability of default is in excess of the threshold will be recommended for rejection.

SAS Enterprise Miner has been a proven data mining workbench for many years. Using it, an analyst, can create models, assess models, and create the scoring code for a “final” model. SAS Enterprise Miner is a very convenient and quick method to perform the process of creating a neural network and using SAS Enterprise Miner can significantly reduce development costs when compared to a process of modeling using the SAS display manager. The diagram of our process is shown in in Figure 2, below:

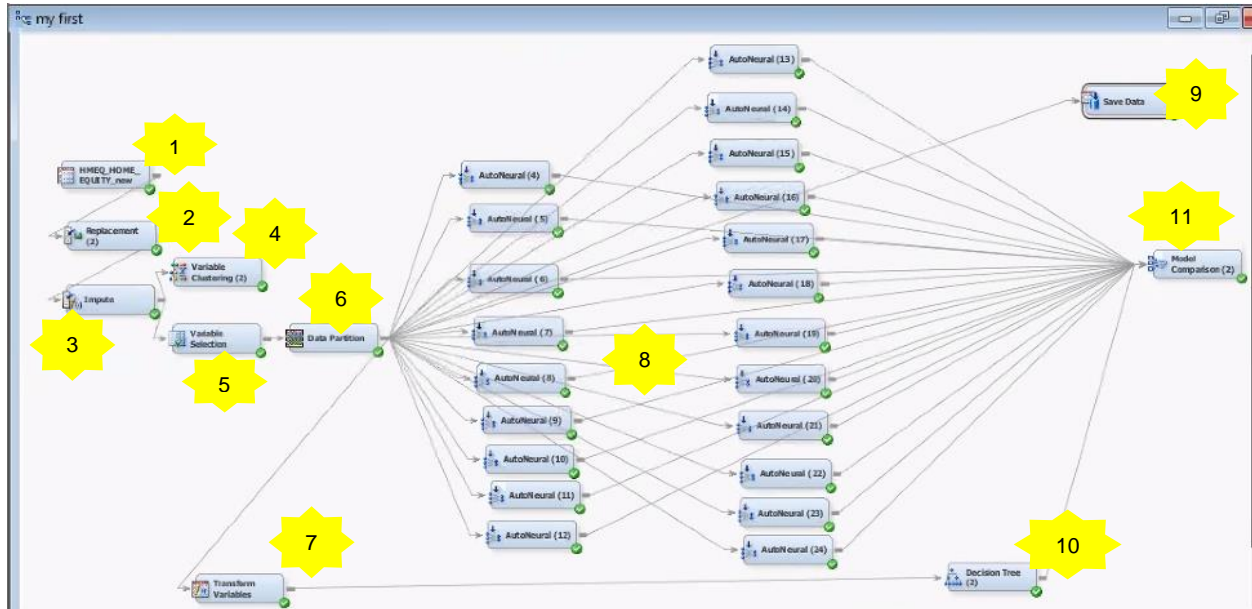


Figure 2

In Figure 2, object one brings data into SAS Enterprise Miner.

Object two is a replacement node and in this node we reduced the impact of outliers by replacing their high values with the upper limit of “normal activity”.

Object three is an impute node and we used this to replace missing values.

Object four is variable clustering and we used this tab to identify redundant variables. We use this tab as a benchmark to judge the reasonableness of the results of the variable selection node.

Object five is the variable selection tab and we used this powerful SAS functionality to identify variables that seem to predict y. The node, automatically, does several sophisticated techniques like creating all possible interaction terms and optimal binning before testing to see if variables are likely to predict Y.

Object six is a partition node. We use this note to split the data into train, validate and test. The ratio was .5, .25 and .25. We split the data after we had finished replacing missing values and transforming to assure that the three partition datasets contained very similar data and were created using identical business rules.

Object seven is the transform tab, which is not part of the main flow of data. We used this node to rename the variables before sending them into a decision tree. The default names, produced by the variable selection tab were so long that they could not be displayed in a decision tree. We use this node to shorten the variable names so that the names fit well on axis of graphs.


Object eight identifies the 22 different auto neural nodes that we ran. One auto neural will identify many different structures and transform functions , then select the best one. In the auto neural tab, to some extent, a particular auto neural analyses a particular network structure.

Object nine is a data export node. We use that because we want to build a deep neural network in the display manager – not in SAS Enterprise Miner.

Object ten is a decision tree. It is good to create many different models and let them compete. Competition will allow us to find the solution that saves the company the most amount of money. The author feels that SAS Enterprise Miner is an incredible tool and adding analysis options is fast and easy.

Object eleven is a model comparison node. This is an incredibly convenient feature in SAS Enterprise Miner and allows us to compare different types of models and to rank them by their predictive ability. The deep neural net that we did by hand will not be compared using this node. We will calculate the performance metrics for our deep neural net by hand and “merge” that result in the SAS Enterprise Miner model comparison output.

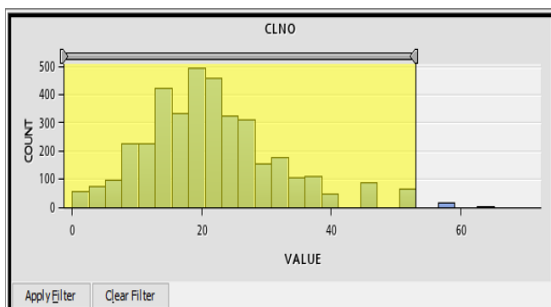
SECTION 2) CLEAN VARIABLES USING THE REPLACEMENT NODE

We used the replacement node  to interactively specify replacement values for class and interval levels and to reassign some specified non-missing values before performing imputation calculations for the missing value.

For each variable in this dataset, the process of this step is to trim the right side of the distribution to create a tighter, more centralized distribution, and replace all the values over the “upper limit bounds” with an upper limit value.

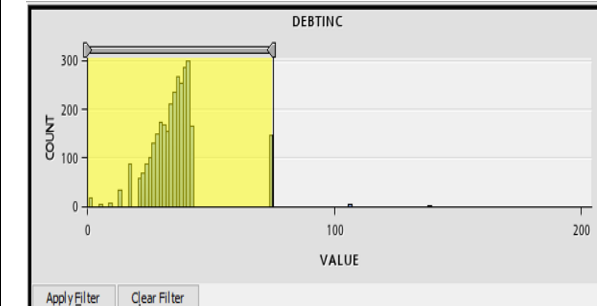
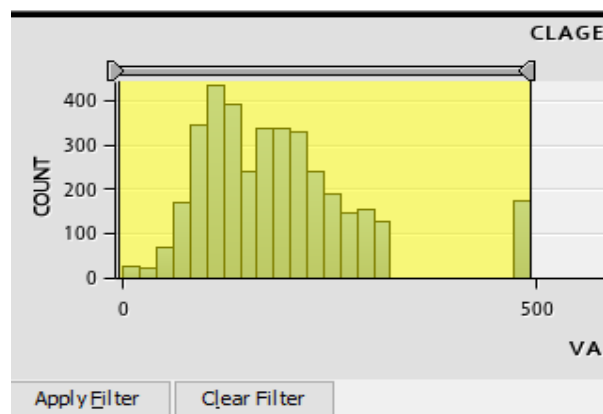
The CLAGE (see right) represents age of oldest credit account in months.

In this example, we trim the right edge of the distribution when value equals to 492, then we will replace all values in the outliers of the distribution (values > 492) with the upper limited value 492. Although the point about 492 is far away from the middle of the distribution, but there are almost 200 applicants' value is 492. So we assume this point is probably to have an important impact on BAD.



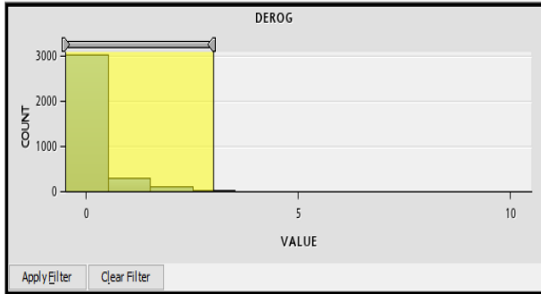
The CLNO (see above) represents the number of credit accounts.

For CLNO, we trim the right edge of the distribution when value equals to 53. We will replace all outliers of the distribution (values > 53) with the upper limited value 53.



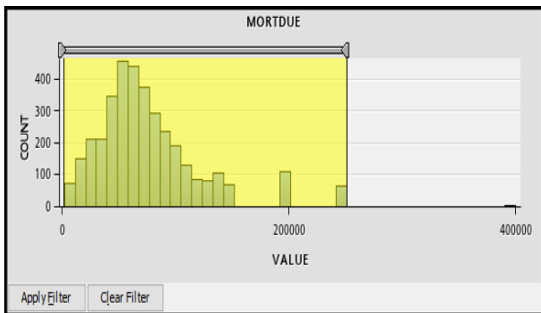
The DEBTINC (see above) represents debt-to-income ratio

We trim values greater than 75 and we will replace all values above 75 with a 75. Look at the point 74.18. It is far away from the distribution, but almost 150 applicants have a debt-to-income ratio close to 74.18. It probably has an important effect on BAD prediction, so we will keep this special value in the dataset.



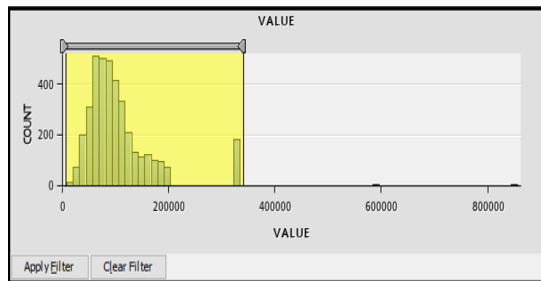
The DEROG (see above) represents the number of major derogatory reports

For DEROG, we trim the right edge of the distribution when value is equal to 3, we will replace all values in the outliers of (values > 3) with the upper limited value 3.



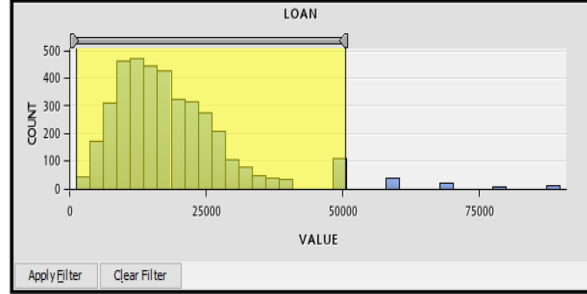
The MORTDUE (see above) represents amount of the loan due on an existing mortgage

For MORTDUE, we trim the right edge of the distribution when value equals to 250,437. We will replace all values greater than 250,437 with the upper limited value 250,437.



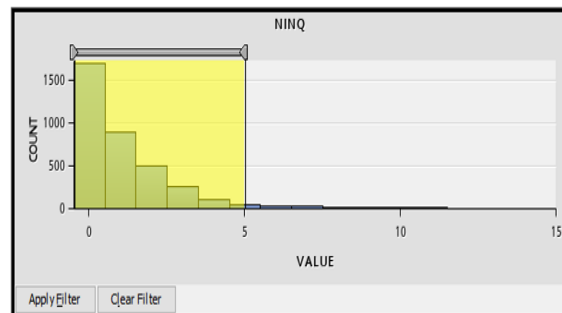
The VALUE (see above) represents the value of current property

For VALUE, we trim the right edge of the distribution when value equals to 340,000. We will replace values greater than 340,000 with a value 340,000.



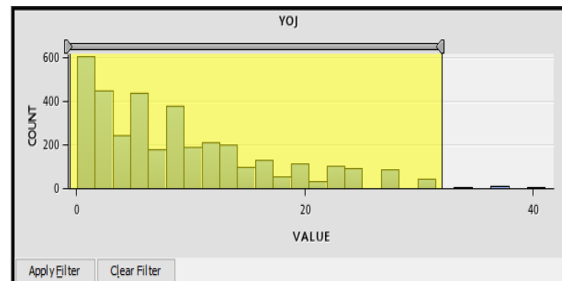
The LOAN (see above) represents amount of the loan requests.

For LOAN, we trim the right edge of the distribution when value equals 50,500. We will replace all values greater than 50,500 with an upper limit value 50,500.



The NINQ (see above) represents number of recent credit inquiries

For NINQ, we trim the right edge of the distribution when value equals to 5. We will replace all values greater than 5 with a 5.



The YOJ (see above) represents year at present job

For YOJ, we trim the right edge of the distribution. We will replace all values greater than 32 a 32.

Figure 3

SECTION 3) IMPUTE MISSING VALUES USING TREE BASED REPLACEMENT LOGIC

Figure 4 shows how SAS handled the class variables. The impute tab, in SAS Enterprise Miner, that imputes missing values has an incredibly convenient feature.

It allows an analyst to have SAS Enterprise Miner, automatically, create a decision tree to predict missing values.

Anyone who's had to introduce inaccuracy in an X variable by replacing missing values with a simple mean will appreciate this feature.

Anyone who's had to build ANCOVA models to predict missing values will appreciate this feature. It's very fast and saves many person-hours.

We would like to impute the reasonable values for observations that have a missing value. Many modeling techniques will discard a complete row of data if one of the variables in the model has a missing value.

The fact that SAS Enterprise Miner will build a separate decision tree to impute missing values for each of the X variables that have missing values is a great timesaver and increases the number of observations that we can use in the model.

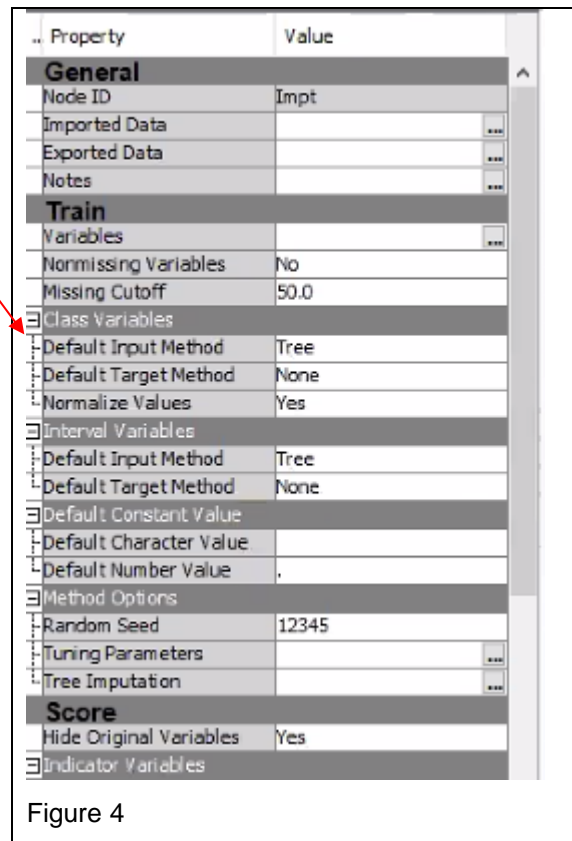


Figure 4

SECTION 4) SELECT NON-REDUNDANT VARIABLES USING THE VARIABLE CLUSTERING TAB IN SAS ENTERPRISE MINER

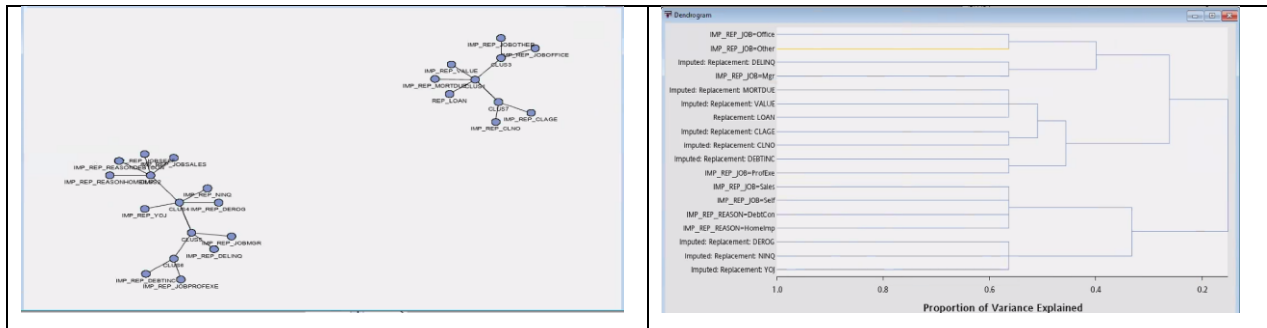
Every modeler operating in the age of big data encounters two problems. When a data has many columns the raw data contains X variables that have two types of problems.

Some of the possible X variables are redundant with other X variables. They should be removed before the modeling process starts because they introduce multicollinearity, increase the run time, and violate our goal of parsimony.

The other problem is that some of the X variables are not likely to help us predict Y.

Every modeling process should have two explicit steps. The first step should be an attempt to remove redundant variables and we will do that using the variable clustering TAB and enterprise Miner and our own intuition. The second step is to remove non-predicting X variables and we will do that in Section 5 using the variable selection node.

There is a step that is not on the flowchart because it is manual. We expect an analyst will take a look at the output from the variable clustering and see which of those variables are selected by SAS as part of the variable selection process. There's nothing very formal about this comparison, it is just a "sniff test".



The figures above give some idea of how SAS clusters the variables and which variables SAS considers to be close to each other (multi-collinear).

Figure 5

The figure below shows the details of how SAS selects variables from clusters. It looks for variables that are highly correlated with variables that are in its own cluster and uncorrelated with variables in other clusters. To come up with a solution, this tab implements a very complicated algorithm involving an iterative process.

The final table is shown below and the selected variables are highlighted in yellow.

7 Clusters		R-squared with			Variable Label
Cluster	Variable	Own Cluster	Next Closest	1-R**2 Ratio	
Cluster 1	IMP_REP_MORTDUE	0.8350	0.0943	0.1822	Imputed: Replacement: MORTDUE
	IMP_REP_VALUE	0.8967	0.0857	0.1130	Imputed: Replacement: VALUE
	REP_LOAN	0.2994	0.0241	0.7179	Replacement: LOAN
Cluster 2	IMP_REP_JOBSales	0.0147	0.0072	0.9924	IMP_REP_JOB=Sales
	IMP_REP_JOBSelf	0.0877	0.0295	0.9400	IMP_REP_JOB=Self
	IMP_REP_REASONDebtCon	0.9760	0.0102	0.0243	IMP_REP_REASON=DebtCon
	IMP_REP_REASONHomeImp	0.9760	0.0102	0.0243	IMP_REP_REASON=HomeImp
Cluster 3	IMP_REP_JOBOffice	0.6964	0.0330	0.3139	IMP_REP_JOB=Office
	IMP_REP_JOBOther	0.6964	0.0912	0.3340	IMP_REP_JOB=Other
Cluster 4	IMP_REP_DEROG	0.4510	0.0521	0.5792	Imputed: Replacement: DEROG
	IMP_REP_NINQ	0.4884	0.0235	0.5239	Imputed: Replacement: NINQ
	IMP_REP_YOJ	0.3102	0.0185	0.7028	Imputed: Replacement: YOJ
Cluster 5	IMP_REP_DELINQ	0.5315	0.0336	0.4848	Imputed: Replacement: DELINQ
	IMP_REP_JOBMgr	0.5315	0.0301	0.4831	IMP_REP_JOB=Mgr
Cluster 6	IMP_REP_DEBTINC	0.5562	0.0347	0.4597	Imputed: Replacement: DEBTINC
	IMP_REP_JOBProfExe	0.5562	0.0547	0.4695	IMP_REP_JOB=ProfExe
Cluster 7	IMP_REP_CLAGE	0.6151	0.0431	0.4022	Imputed: Replacement: CLAGE
	IMP_REP_CLNO	0.6151	0.1144	0.4346	Imputed: Replacement: CLNO

Figure 6

SECTION 5) SELECT PREDICTING VARIABLES USING THE VARIABLE SELECTION TAB IN SAS ENTERPRISE MINER

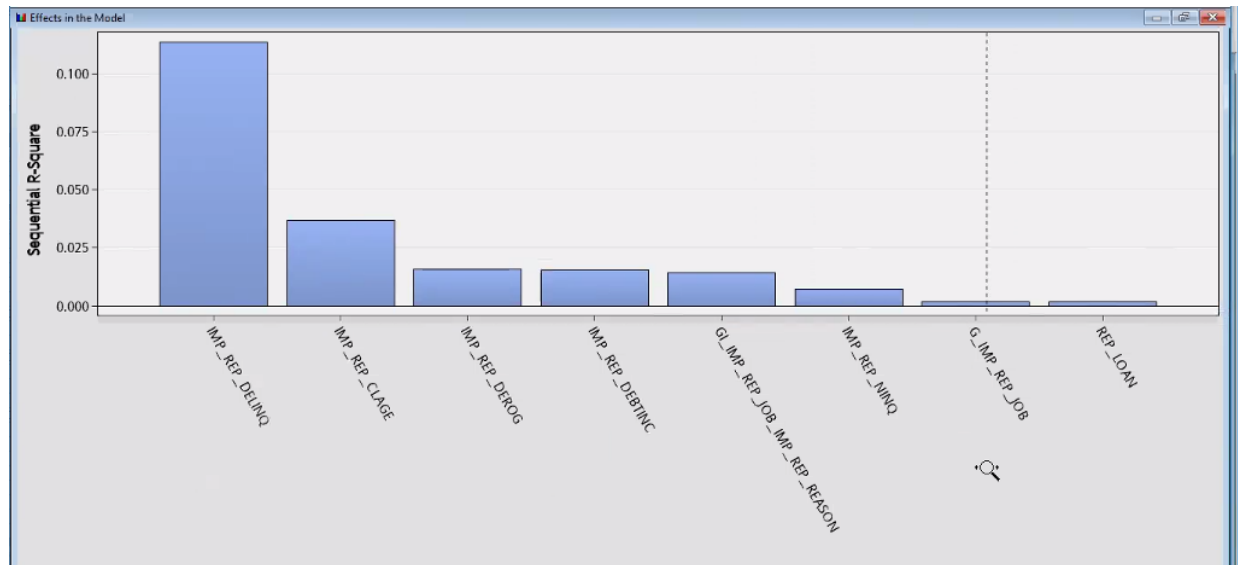


Figure 7

The DMINE Procedure

Effects Chosen for Target: REP_BAD						
Effect	DF	R-Square	F Value	p-Value	Sum of Squares	Error Mean Square
Var: IMP_REP_DELIHQ	1	0.113895	765.810689	<.0001	108.405410	0.141556
Var: IMP_REP_CLAGE	1	0.036860	258.550645	<.0001	35.082946	0.135691
Var: IMP_REP_DEROG	1	0.015647	111.798362	<.0001	14.893005	0.133213
Var: IMP_REP_DEBTINC	1	0.015369	111.852643	<.0001	14.627981	0.130779
Group:IMP_REP_JOB*IMP_REP_REASON	6	0.014155	17.454984	<.0001	13.473101	0.128646
Var: IMP_REP_NINQ	1	0.007339	54.786202	<.0001	6.984883	0.127493
Group:IMP_REP_JOB	2	0.001873	7.005678	0.0009	1.782756	0.127237
Var: REP_LOAN	1	0.001696	12.709266	0.0004	1.613905	0.126986

Figure 8

The variable selection node has a three-step process it applies as it decides whether a variable is likely to predict Y. You can select from several stopping criteria and we used R-Square. If the Y variable is continuous, SAS only does the first two of the steps listed below.

Step One: SAS Enterprise Miner computes the squared correlation between each variable with the Y variable and rejects variables that have a value less than a minimum R squared.

Step Two: SAS Enterprise Miner uses a forward stepwise regression, using R squared is the criteria to determine the importance of variables in a future model. SAS, automatically, creates interactions and can create dummy variables. This is a very convenient, and powerful, node. The fact that other SAS tabs, downstream in our process flow, recognizes the results of this tab makes it very convenient and a large timesaver.

Step Three: If the Y variable is binary, SAS Enterprise Miner will perform a logistic regression, with dynamic binning, as a final step in the process of identifying good predictors.

At this time, the author ran a decision tree just to see what a different algorithm would suggest as important variables for predicting our Y. We will discuss the results of that decision tree later in the paper but think a reader, at this time, might like to see what the decision tree considered to be important variables.

Important variables as determined by different techniques		
variable clustering	variable selection (in order of importance)	decision tree
IMP_REP_CLAGE	IMP_REP_CLAGE	X_Dellnq (importance=1)
	IMP_REP_DEROG	X_DebtInc (importance=2)
	IMP_REP_DEBTINC	X_DEROG(importance=3)
IMP_REP_REASON_DEBTCON	IMP_REP_JOB*IMP_REP_REASON	X_Clage (importance=4)
	IMP_REP_NINQ	X_NINQ(importance=5)
IMP_REP_JOB_PROFEXE IMP_REP_JOBOFFICE IMP_REP_JOB_MGR	IMP_REP_JOB	X_CINO (importance=6)
	REP_LOAN	X_MORTDUE(importance=7)
IMP_REP_VALUE		X_LOAN(importance=8)
		X_VALUE(importance=9)

Figure 9

The Variable Clustering Tab is not a technique that identifies important predictive variables. Variable Clustering technique just checks for redundant variables and is, in some ways, similar to a factor analysis.

The Variable Selection tab does identify variables that predict Y. Variable selection and a decision tree both create measurements of how important a variable is in predicting a Y value. However; they use different geometries and different algorithms and they are not expected to agree 100%. In this step, the author was trying to determine if different techniques, with different underlying assumptions, would identify the same variables as being important.

Because the Variable Selection tab is so powerful, the author decided to use the output from that technique as input into the neural networks to follow.

SECTION 6) AUTO-NEURAL: BUILD 22 DIFFERENT NEURAL NETWORK MODELS USING THE AUTONEURAL NODE TO EVALUATE MANY DIFFERENT STRUCTURES AND TRANSFORM FUNCTIONS.

The auto neural tab/node has promise of being a powerful and huge timesaver for an analyst. The auto-neural tab is really a macro system that calls PROC Neural with many different parameters. One call of the auto-neural tab can evaluate dozens of neural networks.

The auto-neural node promises to evaluate many different structures of neural net and many different transforms inside those structures.

The author found the documentation difficult to read and not to the usual extremely high standards for SAS documentation. Understanding how the parameters that are set when auto-neural is invoked will change the structure of the neural net produced is still not clear. SAS tech support has been very helpful.

Interpreting the output is also a challenge, though the output does feed very nicely into a model comparison node in an SAS Enterprise Miner flowchart.

The author decided to run many (22) different auto-neural nodes for two reasons.

The first reason was to try and create enough examples so that the effect of different parameters could be deduced from the output of the 22 different runs.

The second reason was, because the link between parameters and the neural network structure was hard to determine, the author wanted to do enough neural nets so that the best structure would be selected. This is not very different from a static procedure selecting several random starting processes in hopes of avoiding a local minimum.

The parameter space was explored in the following manner.

Maximum number of nodes in a layer was set to: one, three and eight (the maximum allowed)

Structures for the neural net were set to: funnel, block and cascade (all the structures that allowed more than one layer in the network)

When the author selected the cascade structure, the option for freezing weights was always set to “yes”. This seems to be similar to freezing weights in a deep neural net and seems to be the default.

Tolerance (a setting that requests preliminary estimates of the weights) was set to: medium and High

Proc Auto-Neural will, automatically test many activation functions. The author selected: direct, normal, sign and TanH.

The training options (a setting that has an impact on how the neural nets are grown) were set to: increment and search.

There seems to be an interaction between the training options and the structure option, but this interaction was not in any of the documentations we read and we would encourage the reader to explore more.

Because of the great number of models built by the auto neural tab, and the difficulty in interpreting the output, the output was simply passed to a model comparison node.

SECTION 7) USE PROC NEURAL TO TRAIN TWO DEEP-LEARNING FEEDFORWARD NEURAL NETWORK USING A STRUCTURE THAT WE THOUGHT WOULD BE EFFECTIVE.

The goals of this paper were to create an efficient process for creating a deep neural net and to try and build a net that would outperform one shown in a paper written by one of my professors. This is the step where that deep neural net is built. The difference between the two deep neural networks is that one uses an activation function of linear and one uses an activation function of tanH. Both of the neural networks use a funnel structure starting with eight nodes

The code for the two deep neural networks the author built are included in the appendix to this paper. We is thought that the logic of a deep neural is easier to understand if the code is typed in to the SAS display manager.

A funnel structure was used because we had 8 x input variables and the Y was binary. It was assumed that a “reasonable” structure would start with eight nodes and decrease by 1 node in each layer. The code to do this is in the appendix. The accuracy of the two deep neural networks can be easily understood using a confusion matrix and the two confusion matrices are shown in the model comparison section.

SECTION 8) USE A DECISION TREE TO CREATE A BENCHMARK FOR COMPARISON.

We created this decision tree to provide a benchmark for comparison that we could use to judge the many models we were building using neural net technology.

We paid particular attention to variables that the decision tree considered to be important. We wanted to see if the other SAS techniques would identify the same variables as important.

We also were interested in seeing if we could identify interactions that were high enough up in the tree, so that we might be able to code them manually. We thought that it was possible to create second-order interactions if the decision tree indicated their presence.

We thought that any indication of interaction that showed up on, or below, the third level of the tree would be too difficult to code manually.

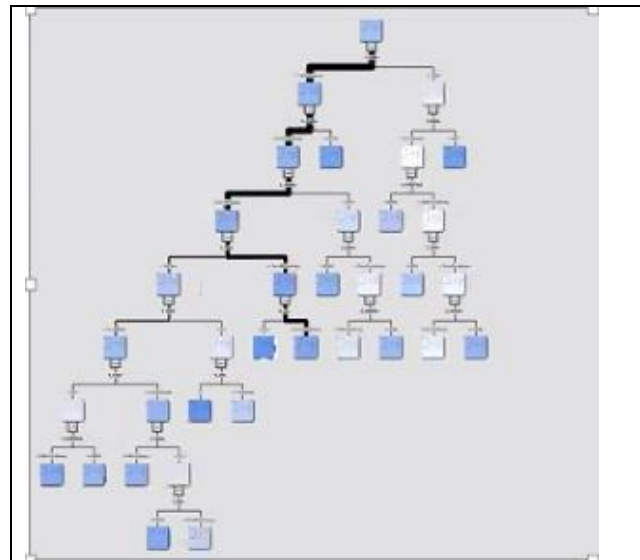


Figure 10

The variable importance table is shown below (the decision tree is very large and very difficult to present). We are very pleased to see the ratio of importance for validation and training to be so high. Mortgage, with its ratio of .5371 is the only variable that shows much difference in importance (see yellow in Figure 11) between training and validate data sets. This suggests, to the author, that the random partitioning of the data into training and validate created data sets that were very similar.

Variable Name	Number of Splitting Rules	Importance	Number of Rules in CV Trees	Relative Importance	Validation Importance	Ratio of Validation To Training Importance
X_DelInq	3	1.0000	37	1.0000	1.0000	1.0000
X_DebtInc	2	0.5925	23	0.5769	0.4902	0.8274
X_DEROG	2	0.5042	21	0.4701	0.4094	0.8120
X_clage	2	0.4667	23	0.4739	0.4196	0.8991
X_NINQ	2	0.4352	16	0.4018	0.4136	0.9503
X_clno	2	0.3610	18	0.3478	0.2973	0.8234
X_MORTDUE	2	0.3595	13	0.2945	0.1931	0.5371
X_LOAN	1	0.3119	12	0.3416	0.2449	0.7854
X_VALUE	1	0.2614	11	0.2397	0.2187	0.8367

Figure 11

Here is an enlargement of the top of the decision tree. The variable “number of delinquencies” is the most important variable in splitting observations into people who pay off the loan and people who do not pay off the loan.

If the number of delinquencies is less than 1.78 a subject would go to the left. If the number of delinquencies was greater than 1.78, the subject would go to the right.

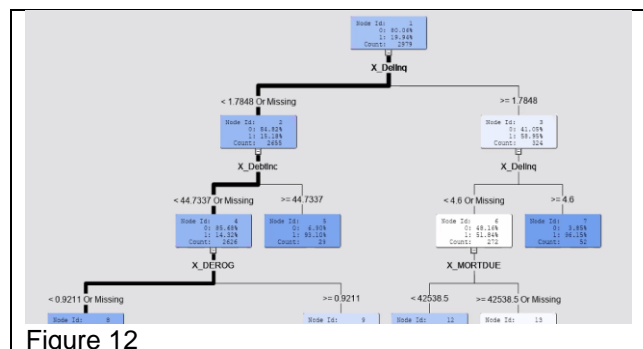


Figure 12

The “model” that is created by a decision tree is a series of if-then statements. A small section of that code is shown to the right.

These if-then statements allow a new observation to be placed in a leaf.

All observations that end up in the same leaf are assigned the same probabilities of being good or bad.

```
***** ASSIGN OBSERVATION TO NODE *****;
IF NOT MISSING(X_Dellnq ) AND
1.78481012658227 <= X_Dellnq THEN DO;
IF NOT MISSING(X_Dellnq ) AND 4.6 <= X_Dellnq
THEN DO;
  _NODE_ = 7;
  _LEAF_ = 18;
  P_REP_BAD1 = 0.96153846153846;
  P_REP_BAD0 = 0.03846153846153;
  Q_REP_BAD1 = 0.96153846153846;
  Q_REP_BAD0 = 0.03846153846153;
  I_REP_BAD = '1';
  U_REP_BAD = 1;
END;
ELSE DO;
```

Figure 13

SECTION 9) MODEL COMPARISON

We use the model comparison node to compare the output from the 22 auto neural nodes and the one decision tree (remember; we had a preference for doing deep neural nets in the SAS display manager).

The fit statistics from the comparison node show that auto neural 13 had the best output (the lowest misclassification rate) – though not much better than auto-neural 22.

As you will see, later on, we do not have 100% faith in misclassification rate and think that a more subtle interpretation is required. Because of limited skills in SAS Enterprise Miner, and limited time, we have to do that subtle investigation manually. We only did that subtle interpretation on auto neural 13 but, if more time had been available, we would have done a detailed examination of the first few auto-neurals

Fit Statistics

Model Selection based on Valid: Misclassification Rate (_VMISC_)

Selected Model	Model Node	Model Description	Valid: Misclassification Rate	Train: Average Squared Error	Train: Misclassification Rate	Valid: Average Squared Error
Y	AutoNeural13	AutoNeural (13)	0.16107	0.09391	0.12756	0.12670
	AutoNeural22	AutoNeural (22)	0.16107	0.09391	0.12756	0.12670
	AutoNeural10	AutoNeural (10)	0.17315	0.12515	0.15374	0.14456
	AutoNeural19	AutoNeural (19)	0.17315	0.12515	0.15374	0.14456
	Tree2	Decision Tree (2)	0.17785	0.10472	0.13394	0.13760
	AutoNeural5	AutoNeural (5)	0.19933	0.15321	0.19940	0.15496
	AutoNeural12	AutoNeural (12)	0.19933	0.15463	0.18731	0.16615
	AutoNeural15	AutoNeural (15)	0.19933	0.15463	0.18731	0.16615
	AutoNeural18	AutoNeural (18)	0.19933	0.15463	0.18731	0.16615
	AutoNeural21	AutoNeural (21)	0.19933	0.15463	0.18731	0.16615
	AutoNeural24	AutoNeural (24)	0.19933	0.15463	0.18731	0.16615
	AutoNeural6	AutoNeural (6)	0.19933	0.15463	0.18731	0.16615
	AutoNeural9	AutoNeural (9)	0.19933	0.15463	0.18731	0.16615
	AutoNeural8	AutoNeural (8)	0.22685	0.14783	0.21920	0.15676
	AutoNeural11	AutoNeural (11)	0.23691	0.16539	0.21954	0.17636
	AutoNeural20	AutoNeural (20)	0.23691	0.16539	0.21954	0.17636
	AutoNeural17	AutoNeural (17)	0.24161	0.15880	0.22457	0.17584
	AutoNeural7	AutoNeural (7)	0.24295	0.16920	0.22054	0.18900
	AutoNeural14	AutoNeural (14)	0.41812	0.26298	0.39275	0.28434
	AutoNeural23	AutoNeural (23)	0.41812	0.26298	0.39275	0.28434
	AutoNeural16	AutoNeural (16)	0.46174	0.40613	0.47264	0.40066
	AutoNeural4	AutoNeural (4)	0.46174	0.40613	0.47264	0.40066

Figure 14

Enterprise Miner also produces some very attractive output that could easily be put into a presentation.

The ROC curves, that we present in Figure 14 show auto-neural13 (green) and auto-neural22 (blue) are the best models for train, test and validate data sets. We were gratified to see that auto-neural models 13 and 22 were high performing models on train, validate and test data sets. This agreement, over all the data sets used, increases our beliefs that we can use these models on new data.

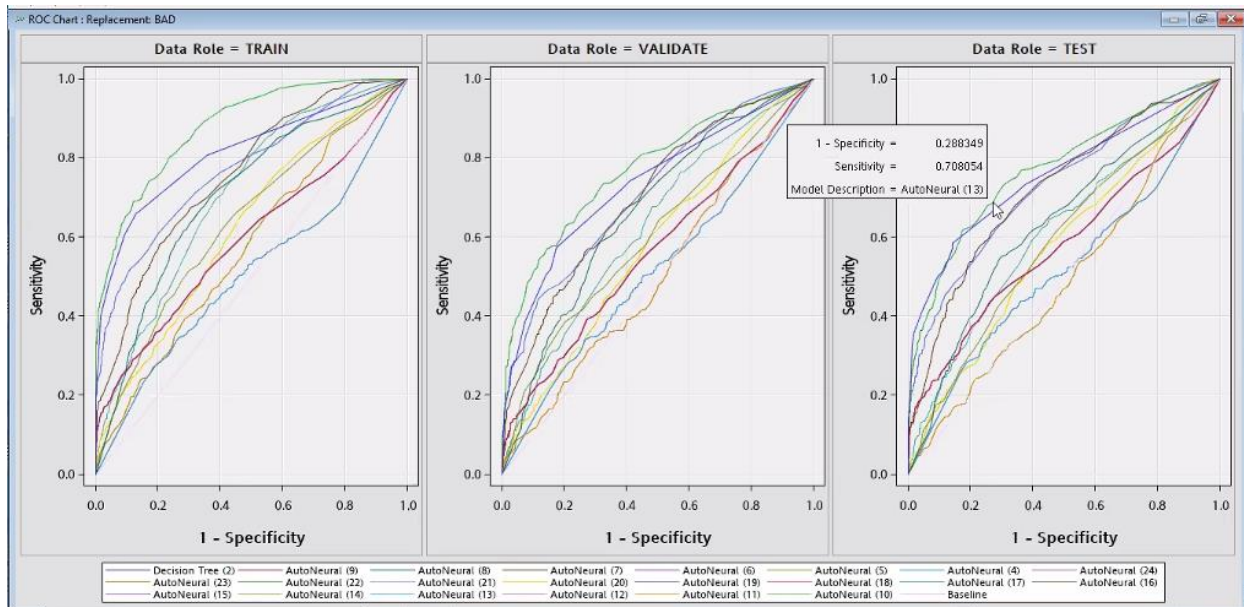


Figure 15

SAS understands that the misclassification rate assumes equal penalties for the two types of mistakes. Several procedures allow an analyst to set different penalties to be associated with creating a bad loan versus not giving a loan to a good customer. We did not have time to explore this.

Below, and apologies for the small font, is the text output for the model comparison node. SAS produces very many fit statistics and we just wanted to use an image give you an idea of how many. Each column is a node that was compared and each row is a different statistic that SAS calculates. We leave, to the readers, the task of studying the statistics that they find most interesting. This paper will use confusion tables to select the best model.

Statistics	Auto Neural11	Auto Neural12	Auto Neural10	Auto Neural19	Auto Tree2	Auto Neural5	Auto Neural12	Auto Neural15	Auto Neural18	Auto Neural21	Auto Neural24	Auto Neural16	Auto Neural19	Auto Neural8	Auto Neural11	Auto Neural20	Auto Neural17	Auto Neural14	Auto Neural13	Auto Neural16	Auto Neural4	
Test: Kolmogorov-Smirnov Statistic	0.44	0.44	0.34	0.34	0.44	0.14	0.17	0.17	0.17	0.17	0.17	0.17	0.25	0.14	0.14	0.19	0.35	0.05	0.05	0.10	0.10	
Test: Average Squared Error	0.13	0.23	0.14	0.14	0.12	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.18	0.18	0.18	0.28	0.28	0.28	0.39	0.39	
Test: Roc Index	0.77	0.77	0.73	0.73	0.76	0.58	0.58	0.58	0.58	0.58	0.58	0.58	0.63	0.58	0.58	0.61	0.72	0.49	0.49	0.52	0.52	
Test: Average Error Function	0.44	0.44	0.47	0.47	0.49	0.51	0.51	0.51	0.51	0.51	0.51	0.52	0.56	0.56	0.59	0.60	0.85	0.85	0.85	2.02	2.02	
Test: Bin-Based Two-Way Kolmogorov-Smirnov Probability Cutoff	0.20	0.20	0.17	0.17	0.20	0.21	0.21	0.21	0.21	0.21	0.21	0.23	0.20	0.20	0.26	0.39	0.86	0.86	0.86	0.98	0.98	
Test: Cumulative Percent Captured Response	35.57	35.57	31.54	31.54	38.12	16.44	22.15	22.15	22.15	22.15	22.15	22.15	16.44	15.77	15.77	18.12	25.17	12.08	12.08	14.11	14.11	
Test: Percent Captured Response	14.09	14.09	11.07	11.07	16.41	8.39	6.04	6.04	6.04	6.04	6.04	6.04	8.39	8.39	8.39	7.72	9.73	5.70	5.70	4.16	4.16	
Test: Divisor for TAEE	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	
Test: Error Function	1337.40	1337.40	1402.48	1402.48	1475.89	1519.83	1519.83	1519.83	1519.83	1519.83	1519.83	1519.83	1536.32	1465.27	1465.27	1705.95	1774.79	2532.43	2532.43	6028.73	6028.73	
Test: Gain	253.57	253.57	213.54	213.54	278.91	63.44	120.15	120.15	120.15	120.15	120.15	120.15	63.44	56.77	56.77	80.12	150.17	20.08	20.08	60.11	60.11	
Test: Gini Coefficient	0.54	0.54	0.46	0.46	0.51	0.17	0.15	0.15	0.15	0.15	0.15	0.15	0.26	0.15	0.15	0.22	0.44	-0.02	-0.02	0.03	0.03	
Test: Bin-Based Two-Way Kolmogorov-Smirnov Statistic	0.43	0.43	0.34	0.34	0.43	0.14	0.18	0.18	0.18	0.18	0.18	0.18	0.25	0.13	0.13	0.19	0.35	0.03	0.03	0.09	0.09	
Test: Kolmogorov-Smirnov Probability Cutoff	0.21	0.21	0.09	0.09	0.15	0.16	0.20	0.20	0.20	0.20	0.20	0.21	0.17	0.17	0.17	0.26	0.35	0.12	0.12	0.97	0.97	
Test: Cumulative Lift	3.54	3.54	3.14	3.14	3.79	1.63	2.20	2.20	2.20	2.20	2.20	2.20	1.63	1.57	1.57	1.85	2.50	1.20	1.20	1.60	1.60	
Test: Lift	2.80	2.80	2.20	2.20	2.20	1.67	1.20	1.20	1.20	1.20	1.20	1.20	1.67	1.67	1.67	1.53	1.93	1.13	1.13	1.87	1.87	
Test: Maximum Absolute Error	1.00	1.00	1.00	1.00	0.93	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.99	0.99	1.00	1.00	1.00	1.00	
Test: Misclassification Rate	0.16	0.16	0.17	0.17	0.14	0.20	0.19	0.19	0.19	0.19	0.19	0.19	0.23	0.24	0.24	0.24	0.25	0.42	0.42	0.46	0.46	
Test: Lower 95% Conf. Limit for TMSC																						
Test: Upper 95% Conf. Limit for TMSC																						
Test: Mean Squared Error	0.13	0.13	0.14	0.14	0.14	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.16	0.18	0.18	0.18	0.18	0.28	0.28	0.39	0.39	
Test: Sum of Frequencies	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	1491.00	
Test: Root Average Squared Error	0.36	0.36	0.37	0.37	0.34	0.40	0.39	0.39	0.39	0.39	0.39	0.39	0.40	0.42	0.42	0.42	0.43	0.53	0.53	0.62	0.62	
Test: Cumulative Percent Response	70.67	70.67	62.67	62.67	75.73	32.67	44.00	44.00	44.00	44.00	44.00	44.00	32.67	31.33	31.33	36.00	50.00	24.00	24.00	32.00	32.00	
Test: Percent Response	56.00	56.00	44.00	44.00	65.19	33.33	24.00	24.00	24.00	24.00	24.00	24.00	33.33	33.33	33.33	30.67	38.67	22.67	22.67	17.33	17.33	
Test: Root Mean Squared Error	0.36	0.36	0.37	0.37	0.40	0.39	0.39	0.39	0.39	0.39	0.39	0.39	0.40	0.42	0.42	0.42	0.43	0.53	0.53	0.62	0.62	
Test: Sum of Squared Errors	377.41	377.41	414.82	414.82	354.25	471.38	464.79	464.79	464.79	464.79	464.79	464.79	464.79	464.79	464.79	478.49	525.18	531.49	547.89	846.62	846.62	
Test: Sum of Weights Times Frags	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	2982.00	
Test: Number of Wrong Classifications	245.00	245.00	254.00	254.00	298.00	277.00	277.00	277.00	277.00	277.00	277.00	277.00	277.00	349.00	351.00	351.00	355.00	368.00	630.00	632.00	680.00	

Figure 16

The confusion matrix for auto-neural 13 is below. Because of our limited experience with SAS Enterprise Miner we had to copy the predicted status dataset (the method used was: click exported data, test, browse, copy-paste to Excel) from auto-neural 13 and pasted into Excel. We created this table manually though We are sure there is a better way.

The misclassification rate of .164 was the lowest of all the misclassification rates and suggests that this is an excellent model. However; We thought a deeper analysis was in order. We think confusion matrices are an excellent tool for judging model quality.

Auto-neural 13 would result in our giving a loan to 184 people who would not pay the loan. We think the cost of giving a bad loan is much more expensive than the cost of not giving a loan to a good customer and now say that auto-neural 13 might not be the best model to use.

	Give loan	No loan	Row
Auto-neural 13	Predicted	Predicted	total
	pay (bad=0)	Default (bad=1)	
Actually Paid (bad=0)	1132	61	1193
	0.94887	0.05113	1
	0.759	0.041	
Actually Defaulted bad=1)	184	114	298
	0.61745	0.38255	1
	0.123	0.076	
column totals	1316	175	1491
	0.883	0.117	Errors
			0.164

xz

Figure 17

Below are the confusion matrices for the two neural nets. Both of them use a funnel structure with eight nodes in the first hidden layer. One uses a linear transform function and the other uses tanH.

<u>Deep Neural; Funnel Linear Network</u>					<u>Deep Neural; Funnel TanH Network</u>				
Home Equity and Defaults					Home Equity and Defaults				
test linear LAYER Misclassification Table					test tanh LAYER Misclassification Table				
Table of F_REP_BAD by I_REP_BAD					Table of F_REP_BAD by I_REP_BAD				
F_REP_BAD(From: REP_BAD)					F_REP_BAD(From: REP_BAD)				
I_REP_BAD(Into: REP_BAD)					I_REP_BAD(Into: REP_BAD)				
Frequency					Frequency				
Percent					Percent				
Row Pct					Row Pct				
Col Pct	0	1		Total	Col Pct	0	1		Total
-----+-----+-----+-----+-----					-----+-----+-----+-----+-----				
0	1147	46		1193	0	1134	59		1193
	76.93	3.09		80.01		76.06	3.96		80.01
	96.14	3.86				95.05	4.95		
	83.60	38.66				87.30	30.73		
-----+-----+-----+-----+-----					-----+-----+-----+-----+-----				
1	225	73		298	1	165	133		298
	15.09	4.90		19.99		11.07	8.92		19.99
	75.50	24.50				55.37	44.63		
	16.40	61.34				12.70	69.27		
-----+-----+-----+-----+-----					-----+-----+-----+-----+-----				
Total	1372	119		1491	Total	1299	192		1491
	92.02	7.98		100.00		87.12	12.88		100.00

Figure 18

Below is the confusion matrix for the decision tree.

	Give loan	No loan	Row	
Decision Tree Test	Predicted	Predicted	total	
	pay (bad=0)	Default (bad=1)		
Actually Paid (bad=0)	1170	23	1193	
	0.98072	0.01928	1	
	0.785	0.015		
Actually Defaulted bad=1)	192	106	298	
	0.64430	0.35570	1	
	0.129	0.071		
column totals	1362	129	1491	Errors
	0.913	0.087		0.144

Figure 19

While SAS creates many statistics to measure the performance of these models, we want to focus on minimizing money lost. The expensive mistake to make is to grant somebody a lone when they will not pay it back. In the tables above, the percentage of people who are likely to default on a loan is in red. The model we want to pick is the model that has the lowest probability of giving someone a loan who will then default on the loan.

We would select the deep neural net with a hyperbolic tangent activation function as the model to use.

SECTION 10) CONCLUSION

This paper had two goals. The first goal was to suggest a process, a series of steps, that could be followed to produce deep neural nets that performed well. We suggest that the flow chart above is a process that can be made a “way of working”, though we would appreciate any comments that would improve this.

The second goal for this paper was to create a neural net that performed better than the neural net that was published in a paper written by one of my professors. It should be said, that he was just trying to demonstrate deep neural nets and not to “tune” the net to improve performance. However; the error rate in that paper, on the *training data* set, was .74 and he did not partition the data.74% of the people that would’ve defaulted on a loan were recommended to get a loan.

The models developed above outperforms that metric on the test data set, not the training data set, and therefore can be considered superior.

SECTION 11) REFERENCES

Lavery, Russell. 2016. "An Animated Guide: Deep Neural Networks in SAS® Enterprise Miner." *Proceedings of the 2016 MWSUG Conference*, Available at: <https://www.mwsug.org/proceedings/2016/AA/MWSUG-2016-AA25.pdf>

Book SAS Enterprise Miner 14.3: Reference Help (pdf. available to SAS Enterprise Miner customers)

Brown, Anna. "How to build a deep learning model in SAS Enterprise Miner ." SAS communities library. 11-05-2015 . Available at <https://communities.sas.com/t5/SAS-Communities-Library/How-to-build-a-deep-learning-model-in-SAS-Enterprise-Miner/ta-p/231190>

PROC neural online documentation, available at:

<http://support.sas.com/documentation/onlinedoc/miner/em43/neural.pdf>

Many authors /blog, Why do we use neural networks? Quora, available at

<https://www.quora.com/Why-do-we-use-neural-networks>

SAS(R) LASR(TM) Analytic Server 2.5: Reference Guide; available at

<http://support.sas.com/documentation/cdl/en/inmsref/67629/HTML/default/viewer.htm#p0o8wrmmp8zkisn1riwf74uvagg9.htm>

12) ACKNOWLEDGMENTS

Thanks to the people at SAS Tech Support.

13) CONTACT INFORMATION

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

YuTing Tian YT899963@WCUPA.EDU

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

14) APPENDIX

```
/*Deep Neural;Funnel Linear Network*/  
options nocenter;
```

```
ODS LISTING;
```

```
DATA MWSUG.Em_save_train_Use;  
  SET MWSUG.Em_save_train ;  
RUN; QUIT;
```

```
DATA MWSUG.Em_save_validate_Use;  
  SET MWSUG.Em_save_validate ;  
RUN;QUIT;
```

```
DATA MWSUG.Em_save_test_Use;  
  SET MWSUG.Em_save_test ;  
RUN; QUIT;
```

```
PROC DMDB batch data=MWSUG.Em_save_train_Use  
  out=MWSUG.DMDB_HmEq_train  
  dmdbcat=MWSUG.DMDB_Cat_train_HmEq;  
  var REP_LOAN  
    /*REP_BAD*/  
    IMP_REP_CLAGE  
    IMP_REP_DEBTINC  
    IMP_REP_DELINQ  
    IMP_REP_DEROG  
    IMP_REP_NINQ ;  
  class REP_BAD GI_IMP_REP_JOB_IMP_REP_REASON G_IMP_REP_JOB;  
  target REP_BAD ;  
run;
```

```
PROC DMDB batch data=MWSUG.Em_save_validate_Use  
  out=MWSUG.DMDB_HmEq_validate  
  dmdbcat=MWSUG.DMDB_Cat_Validate_HmEq;  
  var REP_LOAN  
    /*REP_BAD*/  
    IMP_REP_CLAGE  
    IMP_REP_DEBTINC  
    IMP_REP_DELINQ  
    IMP_REP_DEROG  
    IMP_REP_NINQ ;  
  
  class REP_BAD GI_IMP_REP_JOB_IMP_REP_REASON G_IMP_REP_JOB;  
  target REP_BAD ;  
run;
```

PROC Neural

```
data=MWSUG.Em_save_train_Use
validata=MWSUG.Em_save_validate_Use
testdata=MWSUG.Em_save_test_Use
dmdbcat=MWSUG.DMDB_Cat_train_HmEq graph;
performance compile details cpucount= 2
threads= yes; /* ENTER VALUE FOR CPU COUNT */

archi MLP hidden= 7;
/*we set 7 hidden layers because we have 8 input variables ,
for the frist
layer we set 8 neural nodes; we try linear function at first
time*/
hidden 8 / id= h1 act= linear;
hidden 7 / id= h2 act= linear;
hidden 6 / id= h3 act= linear;
hidden 5 / id= h4 act= linear;
hidden 4 / id= h5 act= linear;
hidden 3 / id= h6 act= linear;
hidden 2 / id= h7 act= linear;
input REP_LOAN
/*REP_BAD*/
IMP_REP_CLAGE
IMP_REP_DEBTINC
IMP_REP_DELINQ
IMP_REP_DEROG
IMP_REP_NINQ
/ id= i level= int std= std;
target REP_BAD / act= logistic id=t level= ordinal ;
/* BEFORE PRELIMINARY TRAINING WEIGHTS WILL BE RANDOM */
initial random= 123;
prelim 10 preiter=80;
/* TRAIN LAYERS SEPARATELY */;
*freeze i->h1; /*train the first layer*/
freeze h1->h2;
freeze h2->h3;
freeze h3->h4;
freeze h4->h5;
freeze h5->h6;
freeze h6->h7;
train technique= congra maxtime=10000 maxiter= 10000 ;

freeze i->h1;
thaw h1->h2; /*train the second layer*/
train technique= congra maxtime= 10000 maxiter= 10000;
```

```

freeze h1->h2;
thaw h2->h3; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h2->h3;
thaw h3->h4; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h3->h4;
thaw h4->h5; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h4->h5;
thaw h5->h6; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h5->h6;
thaw h6->h7; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;
/* RETRAIN ALL LAYERS SIMULTANEOUSLY */;
thaw i->h1;
thaw h1->h2;
thaw h2->h3;
thaw h3->h4;
thaw h4->h5;
thaw h5->h6;
thaw h6->h7;
train technique= congra maxtime= 10000 maxiter= 1000;
code file= '';

score data=MWSUG.Em_save_train_Use
outfit=MWSUG.Em_save_train_Use_fit
out=MWSUG.Em_save_train_Use_out role=train;
score data=MWSUG.Em_save_validate_Use
outfit=MWSUG.Em_save_validate_Use_fit
out=MWSUG.Em_save_validate_Use_out role=valid;
score data=MWSUG.Em_save_test_Use
outfit=MWSUG.Em_save_test_Use_fit
out=MWSUG.Em_save_test_Use_out role=test;
run;

proc freq data=MWSUG.Em_save_train_Use_out ;
tables f_REP_BAD *i_REP_BAD ;
title2' train tanh LAYER Misclassification Table';
run;

```

```

proc freq data=MWSUG.Em_save_validate_Use_out ;
  tables f_REP_BAD *i_REP_BAD ;
  title2'validate tanh LAYER Misclassification Table';
run;
proc freq data=MWSUG.Em_save_test_Use_out ;
  tables f_REP_BAD *i_REP_BAD ;
  title3'test tanh LAYER Misclassification Table';
run;

/*Deep Neural;Funnel TanH Network*/
options nocenter;

ODS LISTING;

DATA MWSUG.Em_save_train_Use;
  SET MWSUG.Em_save_train ;
RUN; QUIT;

DATA MWSUG.Em_save_validate_Use;
  SET MWSUG.Em_save_validate ;
RUN; QUIT;

DATA MWSUG.Em_save_test_Use;
  SET MWSUG.Em_save_test ;
RUN; QUIT;

PROC DMDB batch data=MWSUG.Em_save_train_Use
  out=MWSUG.DMDB_HmEq_train
  dmdbcat=MWSUG.DMDB_Cat_train_HmEq;
  var  REP_LOAN
      /*REP_BAD*/
      IMP_REP_CLAGE
      IMP_REP_DEBTINC
      IMP_REP_DELINQ
      IMP_REP_DEROG
      IMP_REP_NINQ ;
  class REP_BAD  GI_IMP_REP_JOB_IMP_REP_REASON G_IMP_REP_JOB;
  target REP_BAD ;
run;

PROC DMDB batch data=MWSUG.Em_save_validate_Use
  out=MWSUG.DMDB_HmEq_validate
  dmdbcat=MWSUG.DMDB_Cat_Validate_HmEq;
  var  REP_LOAN
      /*REP_BAD*/
      IMP_REP_CLAGE
      IMP_REP_DEBTINC
      IMP_REP_DELINQ

```

```

        IMP_REP_DEROG
        IMP_REP_NINQ ;

class REP_BAD  GI_IMP_REP_JOB_IMP_REP_REASON G_IMP_REP_JOB;
target REP_BAD ;
run;

PROC Neural
    data=MWSUG.Em_save_train_Use
    validata=MWSUG.Em_save_validate_Use
    testdata=MWSUG.Em_save_test_Use
    dmdbcat=MWSUG.DMDB_Cat_train_HmEq graph;
    performance compile details cpubcount= 2
    threads= yes; /* ENTER VALUE FOR CPU COUNT */

    archi MLP hidden= 7;
        /*we set 7 hidden layers because we have 8 input variables ,
for the frist
        layer we set 8 neural nodes; we try tanh function at frist
time*/
    hidden 8 / id= h1 act= tanh;
    hidden 7 / id= h2 act= tanh;
    hidden 6 / id= h3 act= tanh;
    hidden 5 / id= h4 act= tanh;
    hidden 4 / id= h5 act= tanh;
    hidden 3 / id= h6 act= tanh;
    hidden 2 / id= h7 act= tanh;
input REP_LOAN
    /*REP_BAD*/
    IMP_REP_CLAGE
    IMP_REP_DEBTINC
    IMP_REP_DELINQ
    IMP_REP_DEROG
    IMP_REP_NINQ
    / id= i level= int std= std;
target REP_BAD / act= logistic id=t level= ordinal ;
/* BEFORE PRELIMINARY TRAINING WEIGHTS WILL BE RANDOM */
initial random= 123;
prelim 10 preiter=80;
/* TRAIN LAYERS SEPARATELY */;
    *freeze i->h1; /*train the first layer*/
    freeze h1->h2;
    freeze h2->h3;
    freeze h3->h4;
    freeze h4->h5;
    freeze h5->h6;
    freeze h6->h7;
    train technique= congra maxtime=10000 maxiter= 10000 ;

```

```

freeze i->h1;
thaw h1->h2; /*train the second layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h1->h2;
thaw h2->h3; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h2->h3;
thaw h3->h4; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h3->h4;
thaw h4->h5; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h4->h5;
thaw h5->h6; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;

freeze h5->h6;
thaw h6->h7; /*train the third layer*/
train technique= congra maxtime= 10000 maxiter= 10000;
/* RETRAIN ALL LAYERS SIMULTANEOUSLY */;
thaw i->h1;
thaw h1->h2;
thaw h2->h3;
thaw h3->h4;
thaw h4->h5;
thaw h5->h6;
thaw h6->h7;
train technique= congra maxtime= 10000 maxiter= 1000;
code file= '';
score data=MWSUG.Em_save_train_Use
outfit=MWSUG.Em_save_train_Use_fit
out=MWSUG.Em_save_train_Use_out role=train;
score data=MWSUG.Em_save_validate_Use
outfit=MWSUG.Em_save_validate_Use_fit
out=MWSUG.Em_save_validate_Use_out role=valid;
score data=MWSUG.Em_save_test_Use
outfit=MWSUG.Em_save_test_Use_fit
out=MWSUG.Em_save_test_Use_out role=test;

run;

```

```
proc freq data=MWSUG.Em_save_train_Use_out ;
  tables f_REP_BAD *i_REP_BAD ;
  title2' train tanh LAYER Misclassification Table';
run;
proc freq data=MWSUG.Em_save_validate_Use_out ;
  tables f_REP_BAD *i_REP_BAD ;
  title2'validate tanh LAYER Misclassification Table';
run;
proc freq data=MWSUG.Em_save_test_Use_out ;
  tables f_REP_BAD *i_REP_BAD ;
  title3'test tanh LAYER Misclassification Table';
run;
```