

Exploring SAS® Enterprise Miner™ Special Collection



Foreword by
Jared Dean

The correct bibliographic citation for this manual is as follows: Dean, Jared. 2018. *Exploring SAS® Enterprise Miner Special Collection*. Cary, NC: SAS Institute Inc.

Exploring SAS® Enterprise Miner: Special Collection

Copyright © 2018, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-63526-886-7 (PDF)

All Rights Reserved. Produced in the United States of America.

For a hard-copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

June 2018

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.

SAS software may be provided with certain third-party software, including but not limited to open-source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with SAS software, refer to <http://support.sas.com/thirdpartylicenses>.

Table of Contents

Top 10 Tips for SAS Enterprise Miner Based on 20 Years' Experience
Melodie Rush, SAS Institute, Inc.

Interpreting Black-Box Machine Learning Models Using Partial Dependence and Individual Conditional Expectation Plots
Ray Wright, SAS Institute, Inc.

Building Bayesian Network Classifiers Using the HPBNET Procedure
Ye Liu, Weihua Shi, and Wendy Czika, SAS Institute, Inc.

Methods of Multinomial Classification Using Support Vector Machines
Ralph Abbey, Taiping He, and Tao Wang, SAS Institute, Inc.

Random Forests with Approximate Bayesian Model Averaging
Tiny Du Toit, North-West University, and Andre De Waal, SAS Institute, Inc.

Using Vibration Spectral Analysis to Predict Failures by Integrating R into SAS® Asset Performance Analytics
Adriaan Van Horenbeek, SAS Institute, Inc.

Ready to take your SAS® Enterprise Miner™ skills up a notch?



Discover these titles and more at sas.com/books.
Use **EM35** at checkout for a 35% discount on ebook
and hard copy formats.

 sas.com/books
for additional books and resources.


THE POWER TO KNOW®

About This Book

What Does This Collection Cover?

SAS Enterprise Miner was first released in 1999. The interface has changed, and the capabilities have increased, but what remains the same is the potential Enterprise Miner gives its users to answer some of their most difficult predictive modeling challenges.

The papers included in this special collection have been selected to broaden your knowledge of Enterprise Miner; its utility and productivity.

The following papers are excerpts from the SAS Global Users Group *Proceedings*. For more SUGI and SAS Global Forum *Proceedings*, visit [the online versions of the Proceedings](#).

For many more helpful resources, please visit [support.sas.com](#) and [sas.com/books](#).

We Want to Hear from You

SAS Press books are written by SAS users *for* SAS users. We welcome your participation in their development and your feedback on SAS Press books that you are using. Please visit [sas.com/books](#) to

- Sign up to review a book
- Request information on how to become a SAS Press author
- Recommend a topic
- Provide feedback on a book

Do you have questions about a SAS Press book that you are reading? Contact the author through saspress@sas.com.

Foreword

SAS® Enterprise Miner™ was initially released in 1999. I first began using and developing the software when I was a Mathematical Statistician at the US Census Bureau in 2002. While the interface has changed and its capabilities have grown, what remains constant is the potential SAS Enterprise Miner gives users to answer some of their most perplexing predictive modeling challenges.

Perhaps you think of SAS Enterprise Miner as just a graphical user interface (GUI). While that it is, it is so much more—SAS Enterprise Miner offers a set of algorithms and analytical methods that make performing routine predictive tasks easy and tackling hard tasks possible. SAS Enterprise Miner developed some of the first high-performance computing applications at SAS. Today data scientists likely refer to most of these methods as *machine learning*, but at the time they were introduced, they were simply called *data mining*. Over the many years I've used the software, I've seen companies thrive with data-driven predictions and analytics that affect positive social change, improve health outcomes, and help government respond to the needs of its citizens.

These following topics are intended to broaden your knowledge of SAS Enterprise Miner's usability and the enhanced productivity it affords users.

[Top 10 Tips for SAS Enterprise Miner Based on 20 Years' Experience](#)

Melodie Rush, SAS

Over the past 20 years that I have been using SAS Enterprise Miner and helping analysts with it, I have learned and developed many tips and tricks for ease of use, productivity, and just plain clever implementation. In this presentation, I cover the evolution of SAS Enterprise Miner from the original SAS/AF® software application to the current version that integrates with both open-source software and with SAS® Viya®. I share my top 10 tips for getting the most from using SAS Enterprise Miner, including identifying my favorite node that no one seems to know about and how to implement more complex modeling techniques.

[Interpreting Black-Box Machine Learning Models Using Partial Dependence and Individual Conditional Expectation Plots](#)

Ray Wright, SAS

One of the key questions a data scientist asks when interpreting a predictive model is: How do the model inputs work? Variable importance rankings are helpful for identifying the strongest drivers, but these rankings provide no insight into the functional relationship between the drivers and the model predictions. Partial dependence (PD) and individual conditional expectation (ICE) plots are visual, model-agnostic techniques that depict the functional relationships between one or more input variables and the predictions of a black-box model. ICE plots enable data scientists to drill much deeper to explore individual differences and identify subgroups and interactions between model inputs. This paper shows how PD and ICE plots can be used to gain insight from and compare machine learning models, particularly so-called black-box algorithms such as random forest, neural network, and gradient boosting. It also discusses the limitations of PD plots and offers recommendations on how to generate scalable plots for big data.

[Methods of Multinomial Classification Using Support Vector Machines](#)

Ralph Abbey, Taiping He, and Tao Wang, SAS

Many practitioners of machine learning are familiar with support vector machines (SVMs) for solving binary classification problems. Two established methods of using SVMs in multinomial classification are the one-versus-all approach and the one-versus-one approach. This paper describes how to use SAS® software to implement these two methods of multinomial classification, with emphasis on both training the model and scoring new data. A variety of data sets are used to illustrate the pros and cons of each method.

[Building Bayesian Network Classifiers Using the HPBNET Procedure](#)

Ye Liu, Weihua Shi, and Wendy Czika, SAS

A Bayesian network is a directed acyclic graphical model that represents probability relationships and conditional independence structures between random variables. SAS Enterprise Miner implements a Bayesian network primarily as a classification tool; it includes naive Bayes, tree-augmented naive Bayes, Bayesian-network-augmented naive Bayes, parent-child Bayesian networks, and Markov blanket Bayesian network classifiers. The HPBNET procedure uses both a score-based approach and a constraint-based approach to model network structures. This paper compares the performance of Bayesian network classifiers to other popular classification methods, such as classification trees, neural networks, logistic regression, and support vector machines. The paper also shows some real-world applications of the implemented Bayesian network classifiers and a useful visualization of the results.

[Random Forests with Approximate Bayesian Model Averaging](#)

Tiny Du Toit, North-West University, and Andre De Waal, SAS

A random forest is an ensemble of decision trees that often produce more accurate results than a single decision tree. The predictions of the individual trees in the forest are averaged to produce a final prediction. The question now arises whether a better or more accurate final prediction cannot be obtained by a more intelligent use of the trees in the forest. In particular, in the way random forests are currently defined, every tree contributes the same fraction to the final result (for example, if there are 50 trees, each tree contributes 1/50th to the final result). This ignores model uncertainty as less accurate trees are treated exactly like more accurate trees. Replacing averaging with Bayesian Model Averaging will give better trees the opportunity to contribute more to the final result, which might lead to more accurate predictions. However, there are several complications to this approach that have to be resolved, such as the computation of an SBC value for a decision tree. Two novel approaches to solving this problem are presented and the results compared to that obtained with the standard random forest approach.

[Using Vibration Spectral Analysis to Predict Failures by Integrating R into SAS® Asset Performance Analytics](#)

Adriaan Van Horenbeek, SAS

In industrial systems, vibration signals are the most important measurements for indicating asset health. Based on these measurements, an engineer with expert knowledge about the assets, industrial process, and vibration monitoring can perform spectral analysis to identify failure modes. However, this is still a manual process that heavily depends on the experience and knowledge of the engineer analyzing the vibration data. Moreover, when measurements are performed continuously, it becomes impossible to act in real time on this data. The objective of this paper is to examine using analytics to perform vibration spectral analysis in real time to predict asset failures. The first step in this approach is to translate engineering knowledge and features into analytic features in order to perform predictive modeling. This process involves converting the time signal into the frequency domain by applying a fast Fourier transform (FFT). Based on the specific design characteristics of the asset, it is possible to derive the relevant features of the vibration signal to predict asset failures. This approach is illustrated using a bearing data set available from the Prognostics Data Repository of the National Aeronautics and Space Administration (NASA). Modeling is done using R and is integrated within SAS® Asset Performance Analytics.

I hope you enjoy this collection of informative papers. I know many of the authors personally and can attest to their deep domain knowledge and their passion to share what they know with you. As you read these papers, remember that you are only limited by your imagination.

Jared Dean

Principal Data Scientist and Business Knowledge Series instructor at SAS and author of [Big Data, Data Mining, and Machine Learning: Value Creation for Business Leaders and Practitioners](#)



Jared Dean is a Principal Data Scientist and Business Knowledge Series instructor at SAS. He has developed leading-edge analytics for banking, entertainment, and mobile data. He has developed and maintains projects in Python and R including the SAS kernel for Jupyter and SASPy. Outside SAS, he is an adjunct professor in the MBA program for North Carolina State University as well as a frequent speaker and presenter. Jared has several patents in data mining.

Previously, Jared was a Mathematical Statistician for the US Census Bureau and Chief Technology Officer (CTO) of Reveal Mobile, a mobile marketing analytics startup. He was also a Senior Director of Research and Development for SAS Enterprise Miner. He holds an MS degree in computational statistics from George Mason University and is an advisory board member to their statistics department.

Jared is the author of *Big Data, Data Mining, and Machine Learning: Value Creation for Business Leaders and Practitioners*. Recently, he was named to a list of leading analytics bloggers.

When not working, he spends time with his wife and four children. Their favorite family activities include travel, sports, enjoying good food, cheering for the Broncos, and playing games at home.

Top 10 Tips for SAS® Enterprise Miner™ Based on 20 Years' Experience

Melodie Rush, SAS Institute Inc., Cary, NC

ABSTRACT

Over the past 20 years that I have been using SAS® Enterprise Miner™ and helping analysts with it, I have learned and developed many tips and tricks for ease of use, productivity, and just plain clever implementation. In this presentation, I cover the evolution of SAS Enterprise Miner from the original SAS/AF® software application to the current version that integrates with both open-source software and with SAS® Viya®. I share my top 10 tips for getting the most from using SAS Enterprise Miner, including sharing my favorite node that no one seems to know about and how to implement more complex modeling techniques.

INTRODUCTION

SAS® Enterprise Miner™ has been the proven data mining workbench for the past 20 years. Using it enables you to quickly create models, compare models, and create the score code for the winning model. In this paper, I cover 10 quick tips to help the novice to the expert user gain more insight about their data using SAS Enterprise Miner. These tips help with increasing productivity, learning about new nodes, and leveraging options to expand the functionality and knowledge gained.

BACKGROUND AND HISTORY

SAS Enterprise Miner was released in 1998 with the interface built in SAS/AF® (Figure 1). The first version of SAS Enterprise Miner was 2.01 released with SAS 6.12. The first release included Client Server for both Windows and UNIX, process flow diagrams with drag-and-drop capabilities based on the SEMMA (Sample, Explore, Modify, Model, and Assess) model development process, integrated model comparison, and the creation of SAS Score Code, including transformations. This first release had 15 nodes versus the 80+ that are available in the current version. This release even included nodes for decision trees, neural networks, and ensemble models.

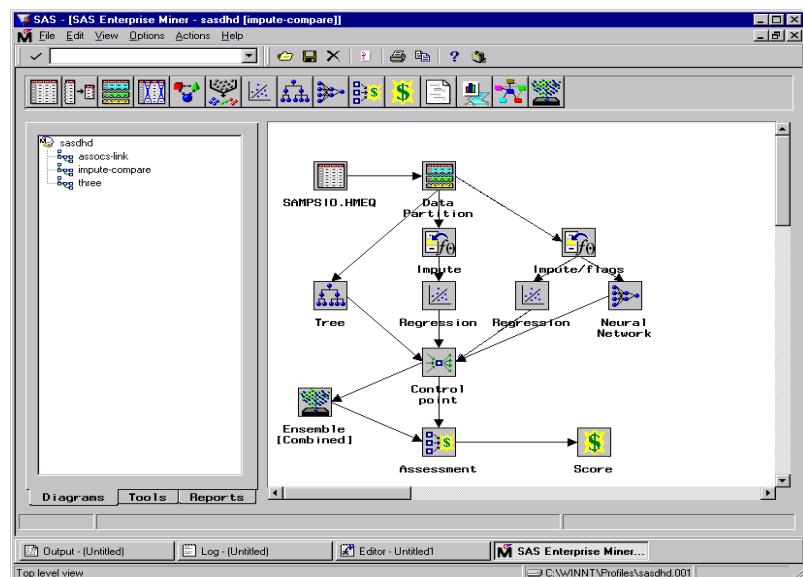


Figure 1. SAS Enterprise Miner Original Interface

Over the last 20 years, many milestones have been reached. Here are some of the highlights by year:

- 2000: EM 4.0 – C and Java score code added, and the Tree Desktop Viewer (Figure 2).

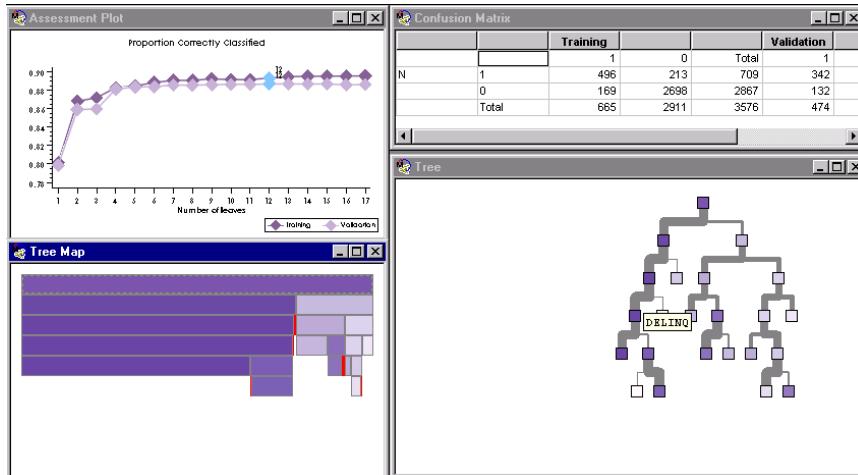


Figure 2. Tree Desktop Viewer

- 2001: EM 4.1 – SAS 8.2 Link Analysis, Memory-Based Reasoning (MBR), and Time Series added.
- 2002: SAS® Text Miner Add-on released (Figure 3).
- 2003: EM 5.1 – SAS 9.1 interface rewritten to a rich Java client; parallel and batch processing, XML diagram exchange, model packages, graph explorer, credit scoring nodes.

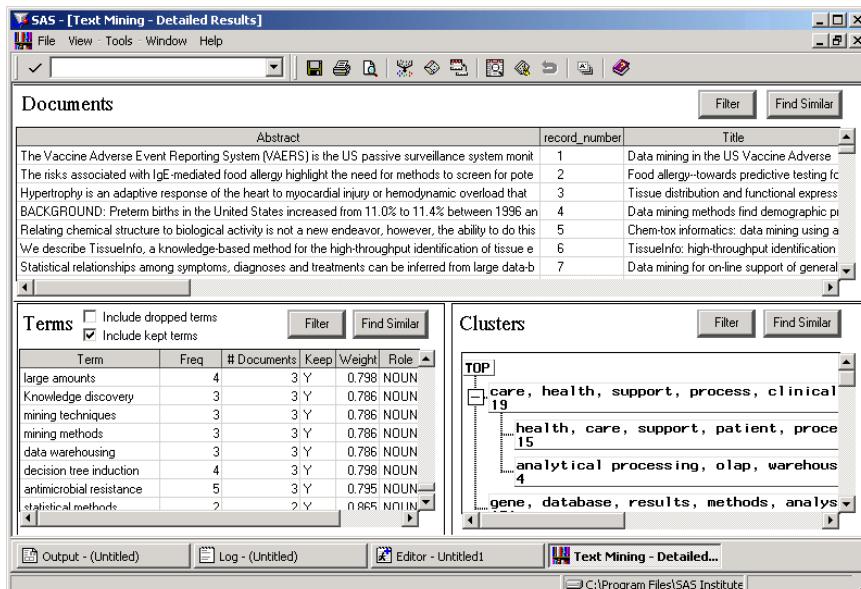


Figure 3. Text Miner Add-on

- 2005: EM 5.2 – SAS 9.1 Decision, Replacement, and SOM/Kohonen nodes, GRID processing, desktop release.
- 2007: EM 5.3 – SAS 9.1.3 group processing, gradient boosting, variable clustering, and hierarchical associations.
- 2009: EM 6.1 – SAS 9.2 File Import node, LARS, optimized score code generation, and native interactive decision trees. Rapid Predictive Modeling task introduced as experimental.
- 2010: EM 6.2 – SAS 9.2 Rapid Predictive Modeling task, SAS® Analytics Accelerator for Teradata.
- 2011: EM 7.1 – SAS 9.3 Survival node, Incremental Response node, Support Vector Machine (SVM), and creation of PMML score code.
- 2012: EM 12.1 – SAS 9.3 Time Series Data Mining nodes (TS Similarity, TS Exponential Smoothing, and TS Data Preparation) production and redesigned Interactive Grouping node.
- 2013: EM 12.3 – SAS 9.4 High-Performance tab added with several HP nodes, including nodes for Random Forest, Neural Networks, Decision Tree, Regression (Logistic and Linear), and GLM (Generalized Linear Model).
- 2013: EM 13.1 – More high-performance nodes (SVM, Principal Components, and Clustering), three New Time Series nodes (TS Dimension Reduction, Time Series Correlation, and TS Decomposition), Open Source Integration node, Register Model node, and Save Data node.
- 2014: EM 13.2 – HP Regression creates VIF (Variance Inflation Factor), support for SAP Hana.
- 2015: EM 14.1 – HP Bayesian Network node, HP Cluster supports automatic selection for number of clusters.
- 2016: EM 14.2 – SAS Viya Code node and support of Analytic Item Store (ASTORE).
- 2017: EM 14.3 – SAS Viya Code node rewritten to support CAS (SAS® Cloud Analytic Services).

The complete list of nodes available in the current release of SAS Enterprise Miner 14.3 is in Figure 4.

SAMPLE	Append	Data Partition	File Import	Filter	Merge	Sample	Input Data	
EXPLORE	Association Cluster	Graph Explore	Variable Clustering	DMDB MultiPlot	Market Basket StatExplore	Link Analysis Path Analysis	Variable Selection	SOM/Kohonen
MODIFY	Drop	Impute	Interactive Binning	Principal Components	Replacement	Rules Builder	Transform Variables	
MODEL	Decision Tree	AutoNeural Regression	Neural Network	Partial Least Squares	Dmine Regression	DM Neural Ensemble	Rule Induction	Gradient Boosting
	Incremental Response	Survival Analysis	Credit Scoring*	TS Correlation	TS Data Prep	TS Dimension Reduction	TS Decomp.	TS Similarity
	HP Explore	HP Bayesian Network	HP Regression	HP Transform	HP Variable Selection	HP Neural	HP Decision Tree	HP GLM
						HP Forest	HP Data Partition	HP SVM
ASSESS	Cutoff	Decisions	Model Comparison	Score	Segment Profile			
UTILITY	Control Point	End Groups	Open Source Integration	Reporter	Score Code Export	Metadata	SAS Code Ext Demo	Save Data
		Start Groups						Register Metadata
								SAS Viya Code

Figure 4. Nodes Available in EM 14.3

TIPS FOR PRODUCTIVITY

Three quick tips for productivity include how to find the node you want, what the available properties are for that node, and how to clone a diagram. Each of these tips accelerates model development.

TIP 1: HOW TO FIND THE NODE I WANT

Inexperienced users often struggle to find the nodes they need to build their data mining flow or diagram. The nodes are organized in the proven data mining process called SEMMA, which stands for Sample, Explore, Modify, Model, and Assess. Each tab on the toolbar at the top of the diagram workspace includes the appropriate nodes (Figure 5).



Figure 5. Sample Tab Nodes

For example, to add a decision tree to your diagram, click the **Model** tab (Figure 6).



Figure 6. Model Tab Nodes

To discern which icon is for the decision tree, scroll across the nodes and position your pointer over the node to see a brief description. The first node is the AutoNeural (Figure 7).

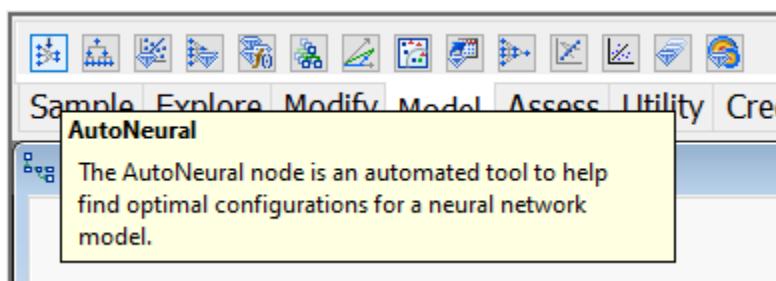


Figure 7. Tooltip for Each Node; AutoNeural Description Displayed

The second node is the Decision Tree (Figure 8).

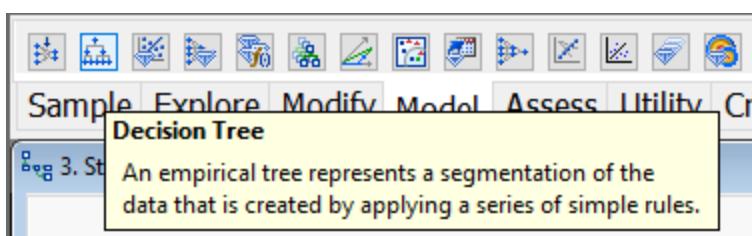


Figure 8. Tooltip for Each Node; Decision Tree Description Displayed

An additional tip: The nodes on each tab are in alphabetical order.

Another way to add a node is to right-click within the diagram you are building (Figure 9). At the top of the menu, select **Add Node**, and then select from the nodes organized by SEMMA. Note that the nodes are also in alphabetical order.

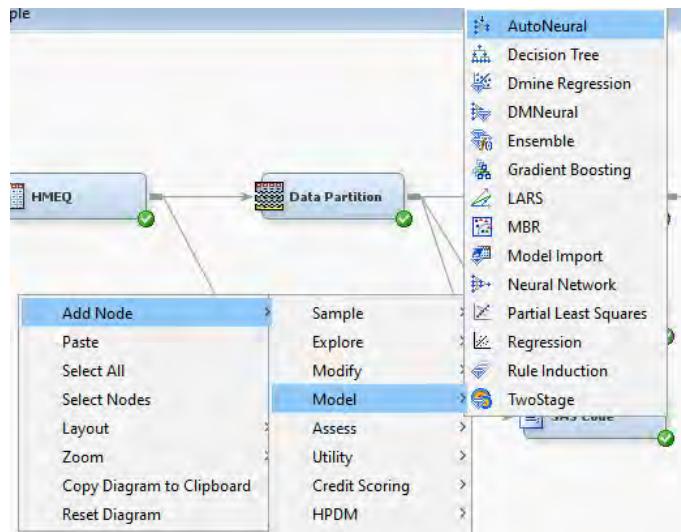


Figure 9. Add Node from Diagram Workspace

TIP 2: WHAT ARE THE AVAILABLE PROPERTIES FOR EACH NODE?

Now that you know how to find a node, you might want to know which properties are available for each node. Simply double-click a node on the toolbar. For example, double-click **Data Partition** on the **Sample** tab (Figure 10).

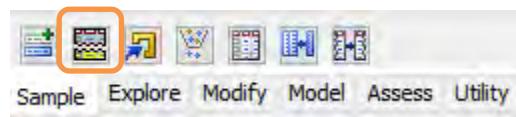


Figure 10. Data Partition Node

The properties for the Data Partition node open in a separate window (Figure 11). This window enables you to see all the current property values and whether the property can be edited.

The properties window for the 'Data Partition' node displays the following properties:

View	Property	Batch Name	Description	Type	Editable	Valid Values	Initial Value
General	Node ID	NODEID	Node Identifier	String	No		
	Imported Data	ImportSet	Set of tables imported by this node.	String	Yes		
	Exported Data	ExportSet	Set of tables exported by this node.	String	Yes		
	Notes	NotesFile	Enter notes for this node.	String	Yes		
	Variables	VariableSet	Variable Properties	String	Yes		
	Output Type	OutputType	Indicates if the node should create data set(s) or DATA step view(s).	String	Yes	Data, View	Data

Figure 11. Properties of the Data Partition Node

An additional tip: More details are outlined in the SAS Data Mining and Machine Learning Community: <https://communities.sas.com/t5/SAS-Communities-Library/SAS-Enterprise-Miner-shortcut-How-to-quickly-see-node-properties/ta-p/375805>.

TIP 3: CLONE A PROCESS FLOW

Do you have a process flow that you want to reuse within your project? It doesn't have to be perfect to make a copy; sometimes we make copies because we are conducting trial and error or another team member would like to use a copy to build a new model faster based on what's already been defined and vetted. It's very easy to clone your process flow and replicate it in the same diagram workspace or a new diagram workspace in three easy steps:

1. Highlight the process flow by dragging your mouse across the process flow (Figure 12).

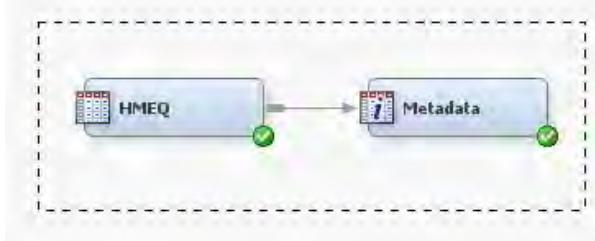


Figure 12. Simple Process Flow Selected

2. Right-click and select **Copy** or select CTRL+C to copy (Figure 13).



Figure 13. Right-Click to Copy

3. Click where you want to insert the process flow, and then right-click and select **Paste** or use CTRL+V to paste (Figure 14).

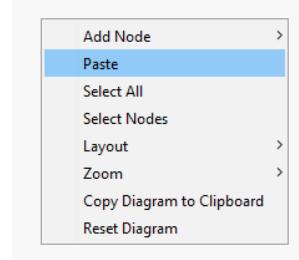


Figure 14. Right-Click to Paste

TIPS ON NODES

SAS Enterprise Miner currently has 80 nodes in the standard installation. If you also have the SAS Text Miner add-on and the Credit Scoring Add-on for SAS® Enterprise Miner™, these two together add an additional 11 nodes. The next three tips cover some of the new nodes, my favorite node that no one knows about, and the node that changes everything.

TIP 4: WHAT'S NEW

You might be like me and stick with what you know, so sometimes I miss new features and functionality when a new release becomes available. With each new release of SAS Enterprise Miner, new nodes are added. This tip is about the new nodes. The current version of SAS Enterprise Miner is 14.3 on SAS 9.4M5.

HPDM Tab and Nodes

Starting with SAS Enterprise Miner 12.3, there is a brand-new tab, HPDM (which stands for High-Performance Data Mining) with several new nodes (Figure 15). These nodes are optimized to run in a distributed environment, meaning the processing can be split among many processors to help minimize processing time. Nodes cover both data mining and machine learning algorithms.

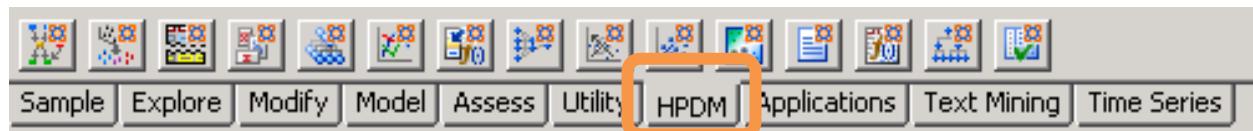


Figure 15. High-Performance Data Mining Tab

In Version 14.3, these nodes are included:

- HP Bayesian Network Classifier
- HP Cluster
- HP Data Partition
- HP Explore
- HP Forest
- HP GLM
- HP Impute
- HP Neural
- HP Principal Components
- HP Regression
- HP SVM
- HP Text Miner
- HP Transform
- HP Tree
- HP Variable Selection

Programming Code Nodes

Two new nodes appear on the Utility tab to help incorporate programming code from both Open Source (R) and SAS® Viya® (Figure 16). These two programming nodes join the SAS Code node, which has been available in SAS Enterprise Miner for several releases.



Figure 16. New Nodes for Open Source and SAS Viya Code

Open Source Integration Node

The Open Source Integration node highlighted first in Figure 16 enables you to use code from the R language inside SAS Enterprise Miner diagrams. This node allows for both supervised and unsupervised algorithms and PMML (Predictive Model Markup Language) and non-PMML R packages. You can compare SAS Enterprise Miner models with R models (Figure 17), ensemble SAS Enterprise models with R models, and create the corresponding SAS DATA step scoring code if the R model comes from a PMML-supported package. This node transfers data, metadata, and results automatically between SAS Enterprise Miner and R.

An additional tip: For more information, watch this video: [Using R in SAS Enterprise Miner](#).

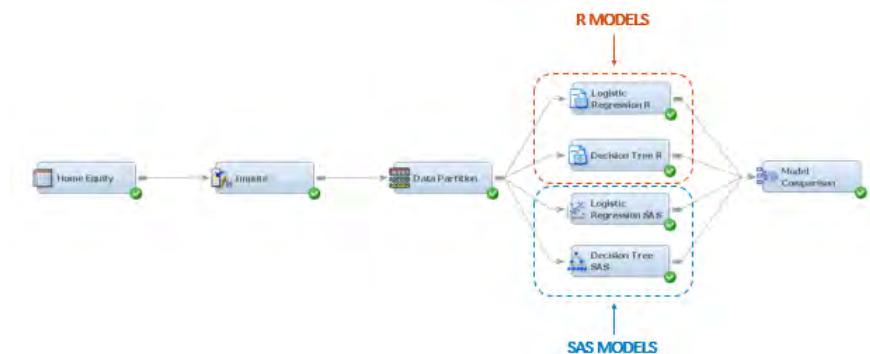


Figure 17. Comparing SAS and R Models Diagram

SAS Viya Code Node

The SAS Viya Code node (the second highlighted node in Figure 16) is created to incorporate code that will be executed in SAS Viya and CAS (SAS Cloud Analytic Services). It allows you to include the new data mining and machine algorithms available in SAS® Visual Data Mining and Machine Learning as part of your SAS Enterprise Miner diagrams (Figure 18).

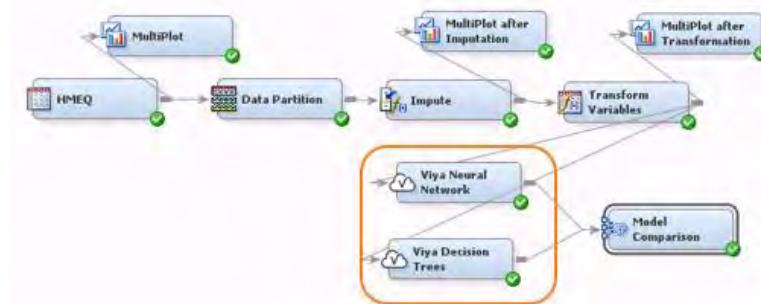


Figure 18. SAS Viya Code Nodes in Diagram

Saving and Sharing Results

Two more nodes recently added are the Register Model node and the Save Data node. Both nodes are located on the Utility tab (Figure 19). These nodes enable you to save and share your output and results.



Figure 19. Register Model and Save Data Nodes

Register Model Node

The Register Model node is highlighted first in Figure 19. The node enables you to register segmentation, classification, or prediction models to the SAS Metadata Server. Why register your models? Registered models can be used and monitored by SAS® Decision Manager and SAS® Model Manager; they can easily score data in SAS® Enterprise Guide®; and they can score or compare models in SAS Enterprise Miner. Using the Register Model node extends and expands your models' intelligence. In previous versions of SAS Enterprise Miner, registering models took several steps; now registering can be done within the diagram using the Register Model node.

An additional tip: The Register Model node provides a model registration mechanism that can run in batch code.

The Register Model node enables users to select the path to register, the name of the model, a model description, and the data mining function of segmentation, classification, or prediction (Figure 20).

Train	
Repository Path	[...]
Model Name	Propensity Model
Model Description	[...]
Mining Function	Classification

Figure 20. Properties of the Register Model Node

Save Data Node

The Save Data node is highlighted second in Figure 19. This node can be used after any node in the process flow diagram to save the training, validation, test, score, or transaction data. The data can be saved as a SAS data set, a JMP data set, and an Excel, CSV, or tab-delimited file. You can also opt to replace existing files, include all or a subset of observations, and all or selected data sets (Figure 21).

Property	Value
Train	
Variables	[...]
Filename Prefix	
Replace Existing Files	Yes
All Observations	Yes
Number of Observations	1000
Output Format	
File Format	SAS (.sas7bdat)
SAS Library Name	DM
Directory	[...]
Output Data	
All Roles	No
Select Roles	[...]

Figure 21. Properties of the Save Data Node

TIP 5: MY FAVORITE NODE THAT NO ONE KNOWS ABOUT

Over the years, I have asked many SAS Enterprise Miner users if they use this one node and usually the response is no. Do you like to document your SAS processes? For some of you, the answer is yes, but for most the answer is no. Either way, this node helps you easily document your SAS Enterprise Miner process flow diagrams. Which node is it? It's the Reporter node located on the **Utility** tab (Figure 22).

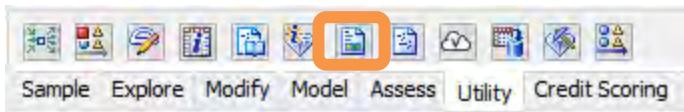


Figure 22. Reporter Node

The Reporter node creates a .pdf or .rtf file to document the entire process flow. It includes an image of the diagram (Figure 23), detailed information about each node included in the diagram (Figure 24), and output from each node.



Figure 23. Process Flow Diagram in a Report

Role	Level	Frequency Count	Name
TARGET	BINARY	1	BAD
INPUT	BINARY	1	REASON
INPUT	INTERVAL	7	CLAGE CLNO DEBTINC LOAN MORTDUE VALUE YOJ
INPUT	NOMINAL	4	DELINQ DEROG JOB NINQ

Figure 24. Data Source Node Settings in a Report

An additional tip: If you have included notes in your nodes or used the SAS Code node to create output in report or graphic format, those notes are included in the report as well. The reports created can range from 30 to 100+ pages depending on the complexity of your process flow diagram. I recommend that you end each flow with a Reporter node so that automatic documentation is created.

TIP 6: THE NODE THAT CHANGES EVERYTHING

One of the most valuable nodes in SAS Enterprise Miner is the Metadata node. This node is on the **Utility** tab (Figure 25).

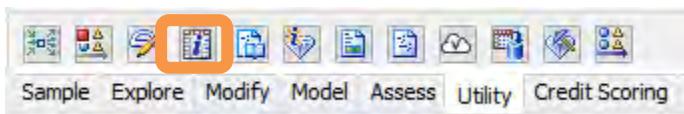


Figure 25. Metadata Node

This node enables you to change the metadata information in your process flow diagram. You can modify any attribute such as variable roles, measurement levels, and so on. You can also use it to merge predecessor nodes. An example of this is in Tip 9.

Have you ever wanted to use your settings from one data set in another data set? I discovered one of the best tips for the Metadata node in the [SAS Data Mining and Machine Learning Community](#). The tip is to always use a Metadata node after a data set (Figure 26). Doing this enables you to capture the settings

for your variables so that you can apply them to new data or to data in a different diagram. This allows for repeatability and consistency when you set up and use your data. It's also a great time saver.



Figure 26. Metadata Node Example

To use the Metadata node in your flow:

1. Create a new diagram.
2. Add your data source using basic settings in the Data Source Wizard.
3. Add a Metadata node:
 - Set up all your roles and levels.
4. Copy and paste the Metadata node to another data set.

TIPS FOR USING OPTIONS

One of the most powerful capabilities of SAS Enterprise Miner is its ability to change options and properties for the process flow diagram and nodes. All the nodes come with what I like to call “smart properties” so that they will run without making any changes. The next four tips are about changing the options or properties to gain even more insight from your data and models.

TIP 7: HOW TO GENERATE A SCORECARD

Did you know that you can create scorecards for your models? With just a couple of modifications to the Reporter node, you can generate a scorecard that emphasizes which variables and values are important (and which are not).

First, what is a scorecard? A scorecard displays your model in such a way that quickly reveals which variables are important and which values are important (Figure 27). The summary ranges from 0 to 1,000. The closer to 1,000 the more likely your event will happen. The closer to 0 the less likely. In the following scorecard, if the customer purchased two or more blankets we would assign them 61 points; 2 domestic products 18 points; 4 or more Heat products 106, and so on. Add all the highlighted numbers and you get $61 + 18 + 106 + 32 + 74 + 77 + 113 + 296 = 777$. In this case, we would say that the customer was likely to purchase from our new campaign (the event we are predicting in this model).

		Scorecard Points
Blankets Purch.	1: LOW - 0.5	0.00
	2: 0.5 - 1.5	13.00
	3: 1.5 - HIGH	81.00
Domestic Prod.	1: LOW - 0.5	0.00
	2: 0.5 - 1.5	18.00
	3: 1.5 - 4.5	46.00
	4: 4.5 - HIGH	195.00
HEAT	1: LOW - 1.5	0.00
	2: 1.5 - 2.5	36.00
	3: 2.5 - 3.5	77.00
	4: 3.5 - HIGH	106.00
Kitchen Prod.	1: LOW - 0.5	0.00
	2: 0.5 - 1.5	32.00
	3: 1.5 - HIGH	62.00
Outdoor Prod.	1: LOW - 0.5	0.00
	2: 0.5 - 3.5	19.00
	3: 3.5 - HIGH	74.00
Promo: 1-7 mon.	1: LOW - 6.5	93.00
	2: 6.5 - 10.5	77.00
	3: 10.5 - 67.5	48.00
	4: 67.5 - HIGH	0.00
Recency	1: LOW - 71.5	113.00
	2: 71.5 - 142	84.00
	3: 142 - 393.5	45.00
	4: 393.5 - HIGH	0.00
Telemarket Ind.	NO	298.00
	YES	0.00

Figure 27. Scorecard

Also, the model tells us the more Blankets, Domestic, Heat, Kitchen, and Outdoor products purchased, the more likely the customer will purchase from our next campaign. It also indicates the more recently the customer received a promotion and the more recently they purchased, the more likely they are to purchase from our next campaign. The model also indicates that if the customer has not received a telemarketing call, they are more likely to purchase from the next campaign.

How do you produce a scorecard in SAS Enterprise Miner? Simply change the properties on the Reporter node. First, the Reporter node needs to follow a Score node. Second, change the Nodes property to Summary (Figure 28) in the Reporter node properties.

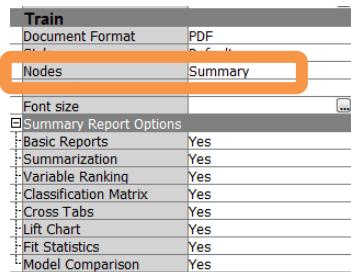


Figure 28. Reporter Node Properties for Scorecard

TIP 8: HELP, I HAVE MORE THAN 512 LEVELS

One of the more common error messages in SAS Enterprise Miner is “Maximum target levels of 512 exceeded” (Figure 29). Here are the two questions that it generates:

1. What does this error message mean?
2. How can I override or overcome it?

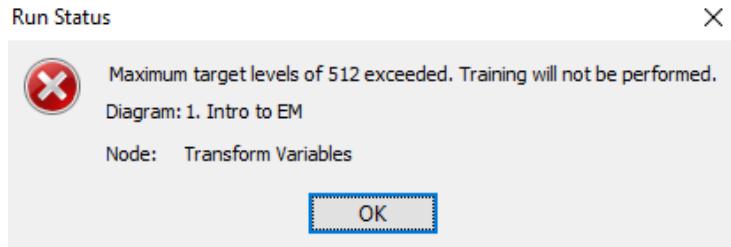


Figure 29. Error Message for Maximum Levels Exceeded

This error occurs when you have a categorical input variable (nominal or ordinal) that has 512 or more distinct values (called cardinality). An example might be a ZIP code. SAS Enterprise Miner set this default for a couple of reasons. It prevents novice users from accidentally using a variable with a bunch of levels because this takes additional processing time and is often unintentional by the user (that is, using a unique ID variable as in input). For example, a ZIP code might have as many as 40,000 levels. If a ZIP code is used as a categorical input into our regression model, the model would create 39,999 parameters to represent the 40,000 levels. Using a neural network model, the number of parameters increases quickly depending on the architecture and number of hidden layers. Having this many parameters to estimate also causes additional issues with sparsity and convergence.

Sometimes it might make sense to use these high cardinality variables in our models. The default can be overridden by changing the EM_TRAIN_MAXLEVELS macro variable to a higher value. There are two ways to do this:

Change macro variable in properties

1. Click your project name in the project window.
2. Scroll down to the project properties.
3. Click the Project Macro Variables ellipsis (Figure 30).

.. Property	Value
Name	Tips for EM
Project Start Code	[...]
Project Macro Variables	[...]
Created	9/20/17 9:15 AM
Server	
Grid Available	No
Path	C:\EMProjects\Tips for I
Metadata Folder Path	
Max. Concurrent Tasks	Default

Figure 30. Project Properties

4. Change the value for EM_TRAIN_MAXLEVELS (Figure 31).

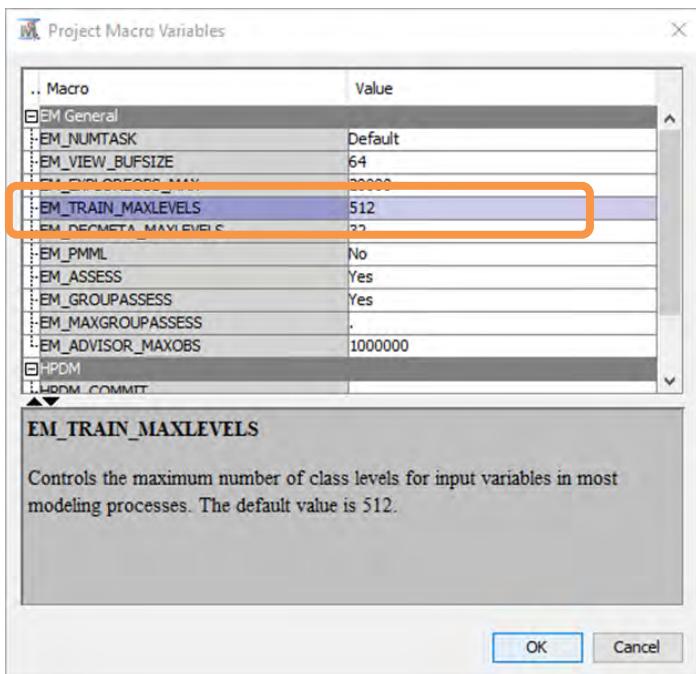


Figure 31. Project Macro Variable Values

Change macro variable in project start code

1. Click your project name in the project window.
2. Scroll down to the project properties.
3. Click the Project Start Code (Figure 32).

... Property	Value
Name	Tips for EM
Project Start Code	<input type="button" value="..."/>
Project Macro variables	<input type="button" value="..."/>
Created	9/20/17 9:15 AM
Server	
Grid Available	No
Path	C:\EMProjects\Tips for EM
Metadata Folder Path	
Max. Concurrent Tasks	Default

Figure 32. Project Properties

4. Add the statement %let EM_TRAIN_MAXLEVELS = *MYVALUE*; (Figure 33).
5. Click **Run Now**.

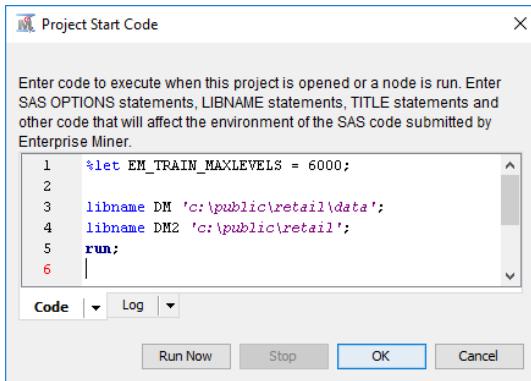


Figure 33. Project Start Code Window

TIP 9: WHICH VARIABLE SELECTION METHOD SHOULD I USE?

SAS Enterprise Miner has several variable selection methods such as Stepwise, Forward, Backward, Decision Trees, R², Chi-square, Random Forest, and more. The question becomes which one should be used. The good news is you don't have to choose just one. You can use multiple methods and combine the results using the Metadata node from Tip 6 (Figure 34).



Figure 34. Example of Metadata Node Variable Selection

The preceding example shows using the LARS, Variable Selection, Variable Clustering, HP Variable Selection, and Decision Tree for variable selection. Connect all the nodes to the Metadata node and navigate to the properties to specify how you want to combine the results (Figure 35). Here are some of the choices:

- None – keeps the original metadata and makes no changes based on the variable selection methods of the previous nodes.
- Any – a variable is set to rejected if any of the previous variable selection nodes rejected it.
- All – a variable is set to rejected if all of the previous variable selection nodes rejected it.
- Majority – a variable is set to rejected if the majority of the previous variable selection nodes rejected it.

.. Property	Value
General	
Node ID	Meta
Imported Data	[...]
Exported Data	[...]
Notes	[...]
Train	
Import Selection	[...]
Summarize	No
Advanced Advisor	No
Rejected Variables	
-- Hide Rejected Variable	No
Combine Rule	Majority
Variables	None
-- Train	Any
-- Transaction	All
-- Validate	Majority
-- Test	[...]

Figure 35. Metadata Node Properties

An additional tip: More details are outlined in the [SAS Data Mining and Machine Learning Community](#). There is also a [SAS Ask the Expert Session on Variable Selection Using SAS Enterprise Guide and SAS Enterprise Miner](#).

TIP 10: HOW DO I INTERPRET MY NEURAL NETWORK?

Neural networks are notoriously hard to interpret. This tip shows how to use a decision tree to create an alternate or proxy interpretation.

First, run your Neural Network, connect a Metadata node, and then connect a Decision Tree node (Figure 36).



Figure 36. Example of Metadata Node Neural Network

Click the Metadata node, and then click the ellipsis next to **Variables→Train** (Figure 37).

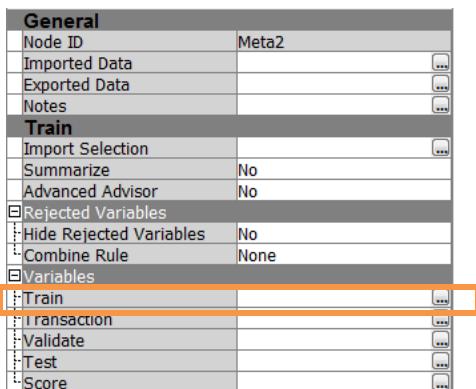


Figure 37. Metadata Node Properties

Change the Prediction variable to be your Target and the original Target variable to be rejected (Figure 38). By doing this, the decision tree is using the predicted values from the neural network as the Y or Target variable (what it is predicting).

P_BAD1	N	Default	Prediction	Target
BAD	N	Default	Target	Rejected
DEBTINC	Y	Default	Rejected	Default
CLAGE	Y	Default	Rejected	Default
CLNO	Y	Default	Rejected	Default
DELINQ	Y	Default	Rejected	Default
DEROG	Y	Default	Rejected	Default
IMP_CLNO	N	Default	Input	Default
F_BAD	N	Default	Classification	Default
IMP_CLAGE	N	Default	Input	Default
TMD_DEBTINC	N	Default	Input	Default

Figure 38. Settings for Metadata Node Variable

The resulting decision tree (Figure 39) shows variables that are important to the predictive value of the neural network. A simplified tree is shown at the bottom of Figure 39. Credit Line Age (CLAGE) and Debt to Income Ratio (DEBTINC) are the two most important variables.

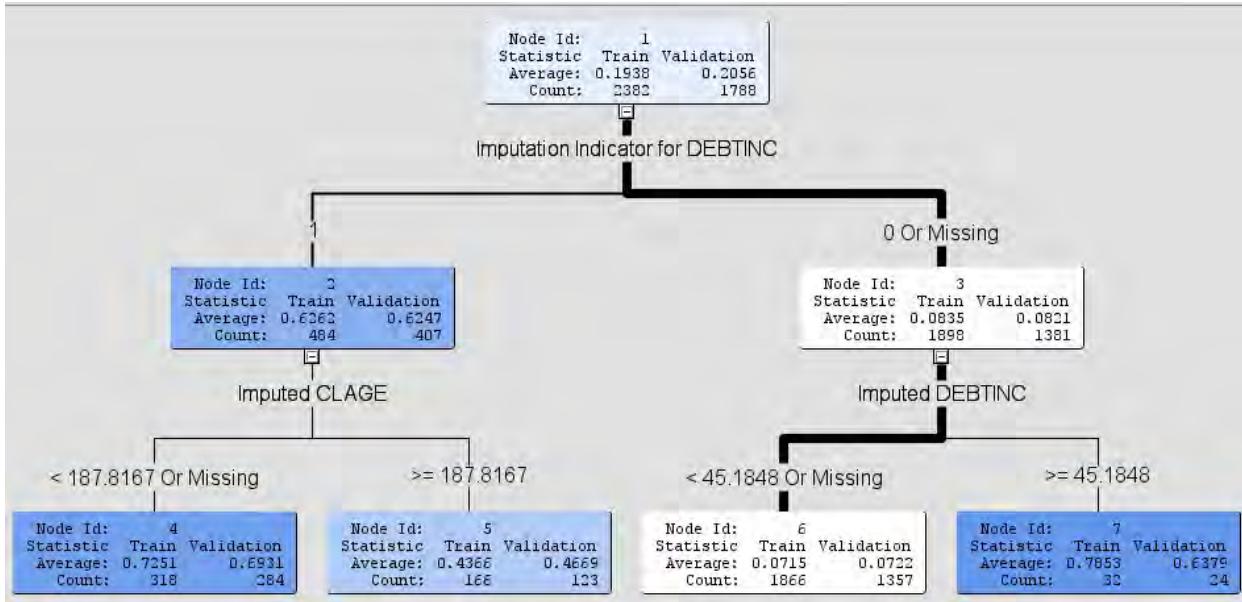


Figure 39. Decision Tree Based on Neural Network Predictors

BONUS TIP

Because SAS Enterprise Miner is loaded with functionality, there is a wealth of resources to help you learn and exploit it. Here are some of my favorites:

- The SAS Data Mining and Machine Learning Community available at https://communities.sas.com/t5/SAS-Data-Mining-and-Machine/bd-p/data_mining has new tips added each month. Plus, it's a great place to ask any questions you have and see what others are asking and solving.
- The SAS Enterprise Miner Learn page available at https://www.sas.com/en_us/learn/software/enterprise-miner.html has resources for new users and advanced tips for more experienced users, including videos, documentation, and examples.
- The Ask the Expert series available at <http://support.sas.com/training/askexpert.html> includes live session and recorded videos where you can witness SAS Enterprise Miner in action. In these one-hour sessions, attendees can ask SAS analysts questions. Past recorded sessions are available on demand. Here are the current sessions (with new ones added often):
 - SAS Enterprise Miner: Getting Started
 - Ensemble Models and Partitioning Algorithms in SAS Enterprise Miner
 - Model Selection Techniques in SAS Enterprise Guide & SAS Enterprise Miner
 - Variable Selection Using SAS Enterprise Guide and SAS Enterprise Miner
 - Data Mining Tasks with SAS Enterprise Guide
 - SAS Text Miner: Getting Started
- Help within SAS Enterprise Miner is available by clicking the book with a ? icon or selecting **Help→Contents** from the menu. This Help includes a node reference guide that gives detailed information about all the nodes, including examples (Figure 40).

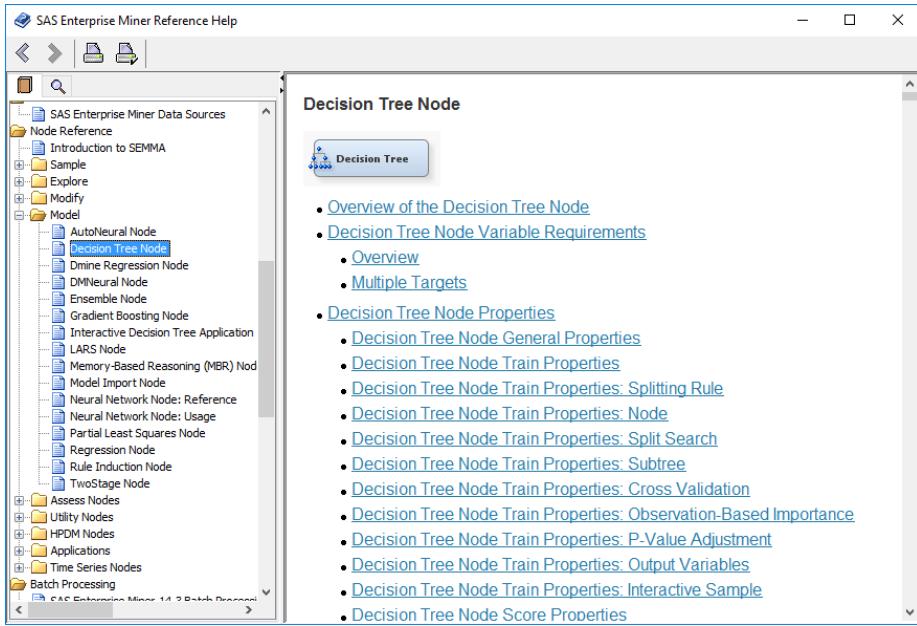


Figure 40. In-Product Node Reference Help

- The SAS Enterprise Miner documentation includes What's New, Getting Started, and administration information available at <http://go.documentation.sas.com/?docsetId=emref&docsetTarget=titlepage.htm&docsetVersion=14.3&locale=en>.
- Github.com is a wonderful place to find and share process flow diagrams. SAS has a library of process flow diagrams to help you learn by example available at <https://github.com/sassoftware/dm-flow>. Here is a video with instructions for using these process flow diagrams: <https://www.youtube.com/watch?v=oSLrkvQH7iU>.

CONCLUSION

SAS Enterprise Miner is a powerful tool for conducting data mining and machine learning projects. The tips shared in this paper enable users to gain more insight quicker by being more productive, to use new nodes, and to modify options and properties to leverage even more efficiency and knowledge.

The 10 tips shared in this paper are just the tip of the iceberg. You can find more tips by referencing the links provided in the Bonus Tip section. Becoming active on communities.sas.com yields even more tips, and you can share your tips too.

REFERENCES

- Ask the Expert series available at <http://support.sas.com/training/askexpert.html>
- Github.com available at <https://github.com/sassoftware/dm-flow>
- SAS Data Mining and Machine Learning Community available at https://communities.sas.com/t5/SAS-Data-Mining-and-Machine/bd-p/data_mining
- SAS Enterprise Miner documentation available at <http://go.documentation.sas.com/?docsetId=emref&docsetTarget=titlepage.htm&docsetVersion=14.3&locale=en>
- SAS Enterprise Miner Learn page available at https://www.sas.com/en_us/learn/software/enterprise-miner.html

VIDEOS

- Deep Learning in SAS Enterprise Miner
 - <https://www.youtube.com/watch?v=HOEqvyyuPrk>
- Getting Started with SAS Enterprise Miner Tutorial Videos
 - https://www.youtube.com/playlist?list=PLVBcK_IpFVi-xzvJiOlf33UvVbRoLRu0z
- How to Execute a Python Script in SAS Enterprise Miner
 - <https://www.youtube.com/watch?v=GROwni8nw64>
- Learn by Example with SAS Enterprise Miner Templates
 - <https://www.youtube.com/watch?v=oSLrkvQH7iU>
- The New HP GLM Node
 - <https://www.youtube.com/watch?v=88qWDc1pGUU>
- Random Forest and Support Vector Machines
 - <https://www.youtube.com/watch?v=EOxwpnbFqIU>
- Using R in SAS Enterprise Miner
 - <https://www.youtube.com/watch?v=TbXo0xQCqDw>

RECOMMENDED READING

- Chu, Robert, David Duling, and Wayne Thompson. 2007. "Best Practices for Managing Predictive Models in a Production Environment." *Proceedings of the 2007 SAS Global Forum Conference*. Cary, NC: SAS Institute Inc. Available <http://www2.sas.com/proceedings/forum2007/076-2007.pdf>.
- Collica, Randall S. 2017. *Customer Segmentation and Clustering Using SAS Enterprise Miner, Third Edition*. Cary, NC: SAS Institute Inc.
- Dean, Jared. 2014. *Big Data, Data Mining, and Machine Learning: Value Creation for Business Leaders and Practitioners*. New York: Wiley.
- de Ville, Barry, and Padraig Neville. 2013. *Decision Trees for Analytics Using SAS Enterprise Miner*. Cary, NC: SAS Institute Inc.
- Duling, David, Howard Plemmons, and Nancy Rausch. 2008. "From Soup to Nuts: Practices in Data Management for Analytical Performance. *Proceedings of the 2008 SAS Global Forum Conference*. Cary, NC: SAS Institute Inc. Available <http://www2.sas.com/proceedings/forum2008/129-2008.pdf>.
- Hall, Patrick. "The Open Source Integration node installation cheat sheet." <https://communities.sas.com/docs/DOC-9988>. 2014.
- Huang, Shunping. "Spectral Clustering in SAS Enterprise Miner Using Open Source Integration Node." Available <https://communities.sas.com/docs/DOC-8011>. 2014.
- Linoff, Gordon S., and Michael J. A. Berry. 2011. *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management, Third Edition*. New York: Wiley.
- Wielenga, Doug. 2007. "Identifying and Overcoming Common Data Mining Mistakes." *Proceedings of the 2007 SAS Global Forum Conference*. Cary, NC: SAS Institute Inc. Available <http://www2.sas.com/proceedings/forum2007/073-2007.pdf>.

ACKNOWLEDGMENTS

Much appreciation goes to Kate Schwarz, Twanda Baker, and Wayne Thompson. Thank you, Kate and Twanda, for your support for this paper by allowing me to bounce ideas off you. And thank you, Kate, for

reviewing this paper and providing constructive comments. Both of you have made this paper better. Wayne, thank you for your reference material on the history of SAS Enterprise Miner and for all your support over the years in answering my crazy and sometimes frantic questions.

I would also like to thank SAS Enterprise Miner R&D, Data Mining Technical Support staff, and all the coworkers and customers who have helped, supported, and walked beside me on this journey over the past 20 years. Thank you!

CONTACT INFORMATION

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

Melodie Rush
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Melodie.Rush@sas.com
<http://www.sas.com>

Twitter: https://twitter.com/Melodie_Rush
LinkedIn: <https://www.linkedin.com/in/melodierush>

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.

Interpreting Black-Box Machine Learning Models Using Partial Dependence and Individual Conditional Expectation Plots

Ray Wright, SAS Institute Inc.

ABSTRACT

One of the key questions a data scientist asks when interpreting a predictive model is “How do the model inputs work?” Variable importance rankings are helpful for identifying the strongest drivers, but these rankings provide no insight into the functional relationship between the drivers and the model’s predictions.

Partial dependence (PD) and individual conditional expectation (ICE) plots are visual, model-agnostic techniques that depict the functional relationships between one or more input variables and the predictions of a black-box model. For example, a PD plot can show whether estimated car price increases linearly with horsepower or whether the relationship is another type, such as a step function, curvilinear, and so on. ICE plots enable data scientists to drill much deeper to explore individual differences and identify subgroups and interactions between model inputs.

This paper shows how PD and ICE plots can be used to gain insight from and compare machine learning models, particularly so-called “black-box” algorithms such as random forest, neural network, and gradient boosting. It also discusses limitations of PD plots and offers recommendations about how to generate scalable plots for big data. The paper includes SAS® code for both types of plots.

INTRODUCTION

After assessing a model’s accuracy, data scientists often want to know how the model’s predictions vary depending on the values of the inputs. This knowledge can help data scientists identify flaws in their models, select from among competing models, and explain their models to stakeholders such as consulting clients, credit card applicants, and medical patients.

In the days of small data sets and relatively simple models, interpreting predictive models was fairly straightforward. For example, the coefficients from a linear regression model indicate the strength and direction of the relationship between a model input and the model’s predictions. Small decision trees are also easily understood by data analysts. But although so-called “black box” machine learning algorithms such as neural network, gradient boosting, and random forest are capable of highly accurate predictions, their inner workings can be very difficult to grasp because these algorithms are enormously complex.

Partial dependence (PD) plots (Friedman 2001) and individual conditional expectation (ICE) plots (Goldstein et al. 2014) are highly visual, model-agnostic tools that can help you interpret modern machine learning models. PD plots show how values of model inputs affect the model’s predictions. ICE plots, which are closely related to PD plots, let you drill down further to identify individual differences, interesting subgroups, and interactions among model variables.

PD and ICE are post hoc methods of model interpretation, meaning that they do not reveal a model’s inner workings; rather, they show how the model behaves in response to changing inputs. The difference is similar to the difference between looking under the hood of a sports car and observing how the car responds when you operate the driver’s controls. Nonetheless, PD and ICE plots are popular and highly visual tools for obtaining a working understanding of increasingly complicated machine learning models.

PARTIAL DEPENDENCE PLOTS

A partial dependence plot depicts the functional relationship between a small number of model inputs (generally one or two inputs) and a model’s predictions. PD plots are thus named because they show how the model’s predictions partially depend on values of the input variables of interest.

ONE-WAY PD PLOTS

The simplest PD plots are one-way plots, which show how a model's predictions depend on a single input to the model. For example, Figure 1 shows the relationship between horsepower and predicted MSRP (manufacturer's suggested retail price in US dollars) for automobile models.

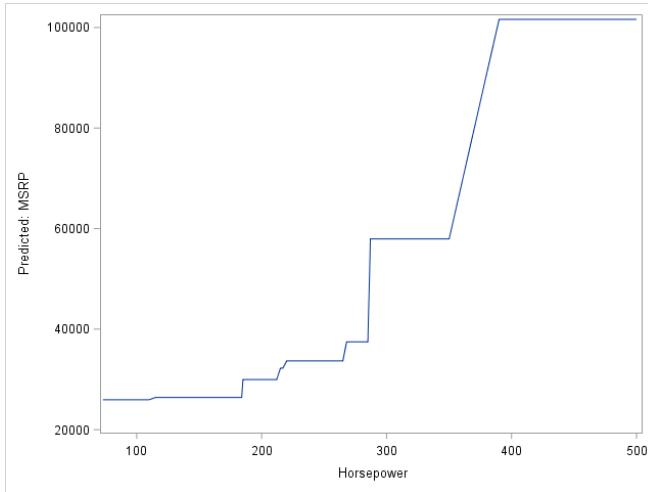


Figure 1. Partial Dependence Plot for Horsepower

Here the model's estimate of MSRP is a step function: MSRP tends to increase with horsepower, but there are sharp increases in expected MSRP for certain values of horsepower. There is a price premium of approximately \$20,000 for cars that have more than 285 horsepower, and an additional (and even greater) premium of about \$40,000 for cars that have about 400 horsepower. Expected MSRP levels off for higher values of horsepower.

PD plots can be used with any supervised learning algorithm. The following block of code uses a decision tree to predict MSRP. As is often true, the model includes several other inputs such as engine size, number of cylinders, and origin. Each value shown in a PD plot represents a prediction for a particular value of horsepower while averaging out the effects of the other (complementary) model variables.

```
proc hpsplit data=sashelp.cars leafsize = 10;
  target MSRP / level = interval;
  input horsepower engineSize length cylinders weight
    MPG_highway MPG_city wheelbase / level = int;
  input make driveTrain type / level = nominal;
  code file="treeCode.sas";
run;
```

The next block calls the %PDFunction macro (described in detail in a later section) to compute the partial dependence function for horsepower:

```
%PDFunction(
  dataset=sashelp.cars,
  target=MSRP,
  PDVars=horsepower,
  otherIntervalInputs=engineSize length cylinders weight
    MPG_highway MPG_city wheelbase,
  otherClassInputs=origin make driveTrain type,
  scorecodeFile=treeCode.sas,
  outPD=partialDependence
);
```

Finally, the next block calls the SGPOINT procedure to plot the partial dependence function, which is shown as a series plot in Figure 1:

```
proc sgplot data=partialDependence;
  series x = horsepower y = AvgYHat;
run;
quit;
```

You can create PD plots for model inputs of both interval and classification variables. Figure 2 shows the partial dependence for the nominal (classification) variable Make.

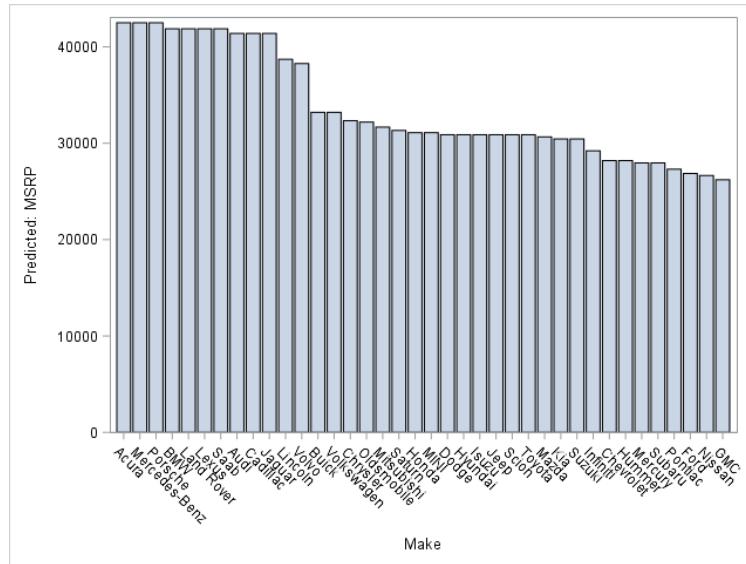


Figure 2. Partial Dependence Function for Make

As you might expect, predicted MSRP is highest for luxury brands such as Acura, Mercedes-Benz, and Porsche. The following code generates the bar chart:

```
%PDFunction(

  dataset=sashelp.cars,
  target=MSRP,
  PDVars=make,
  otherIntervalInputs=horsepower engineSize length cylinders
    weight MPG_highway MPG_city wheelbase,
  otherClassInputs=origin driveTrain type,
  scorecodeFile=treeCode.sas,
  outPD=partialDependence

);

proc sgplot data=partialDependence;
  vbar make / response = AvgYHat categoryorder = respdesc;
run;
quit;
```

TWO-WAY PD PLOTS

Because one-way PD plots display one variable at a time, they are valid only if the variable of interest does not interact strongly with other model inputs. However, interactions are common in actual practice; in these cases, you can use higher-order (such as two- and three-way) partial dependence plots to check for interactions among specific model variables. For example, Figure 3 shows an interaction between the model variables Horsepower and Origin:

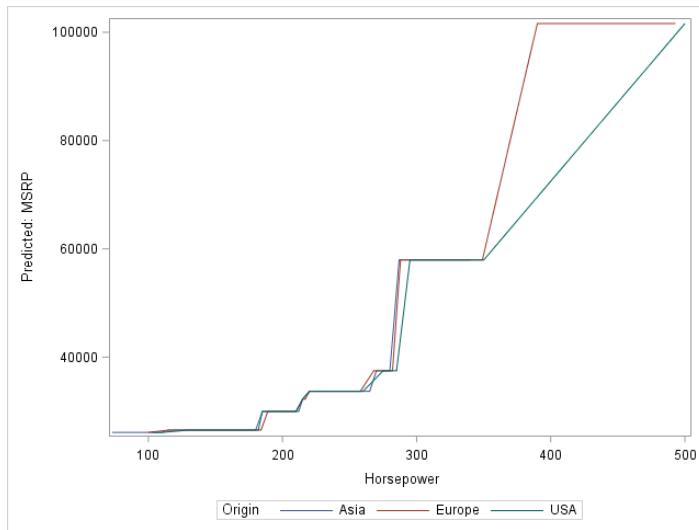


Figure 3. Partial Dependence for Horsepower by Origin

Figure 3 shows a separate PD function for each region (Asia, Europe, and USA). Consistent with the one-way plot for Horsepower in Figure 1, MSRP increases monotonically with horsepower for each region. But the two-way plot shows that powerful European cars (more than 350 horsepower) have much higher expected prices than their American counterparts, an interaction that was not apparent in the one-way plot.

Two-way PD plots are computed much like one-way PD plots. The difference is that two-way plots compute average prediction for each *combination* of values of the plot variables. Here are the %PDFunction and PROC SGPlot calls that generate Figure 3:

```
%PDFunction()

dataset=sashelp.cars,
target=MSRP,
PDVars=horsepower origin,
otherIntervalInputs=engineSize length cylinders weight
    MPG_highway MPG_city wheelbase,
otherClassInputs=make driveTrain type,
scorecodeFile=treeCode.sas,
outPD=partialDependence

);

proc sgplot data=partialDependence;
    series x = horsepower y = AvgYHat / group = origin;
run;
quit;
```

Figure 3 plots an interval input by a nominal input. Figure 4 is a scatter plot for two interval variables, Horsepower and MPG_City. PD plots of two interval variables are typically gradient scatter plots, response surfaces, or 3-D plots.

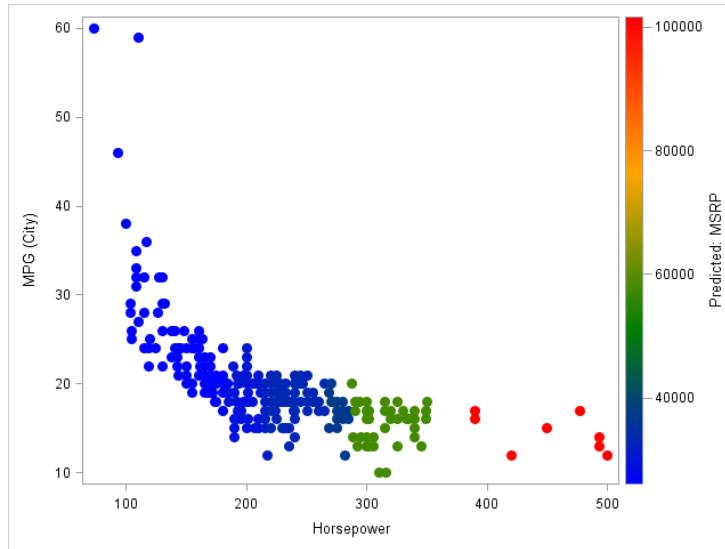


Figure 4. Partial Dependence for Horsepower by MPG_City

No interaction is apparent among the plot variables in Figure 4: MSRP appears to increase with horsepower regardless of the car's mileage estimate. The following %PDFunction macro call and PROC SGPOINT code generate the PD plot:

```
%PDFunction(
  dataset=sashelp.cars,
  target=MSRP,
  PDVars= horsepower MPG_City,
  otherIntervalInputs= engineSize cylinders MPG_highway wheelbase
    length weight,
  otherClassInputs= make origin type driveTrain,
  scorecodeFile=treeCode.sas,
  outPD=partialDependence
);

proc sgplot data=partialDependence;
  scatter x = horsepower y = MPG_City /
    colorresponse = avgYHat
    colormodel=(blue green orange red)
    markerattrs=(symbol=CircleFilled size=10)    ;
run;
quit;
```

COMPUTING THE PD FUNCTION

To create a traditional PD plot (such as Figure 1), you must first compute the PD function by using the following steps. These steps are illustrated using hypothetical data in Figure 5 for a one-way plot.

1. Find the unique values of the plot variable (for a one-way plot) or variables (for a higher-order plot) in the training set and identify the complementary variables.
2. Create one replicate of the training set for each unique value of the plot variable, and fix the value of the plot variable. For complementary variables, use the same values as in the training set.
3. Score each replicate by using your predictive model.
4. Compute the average predicted value within each replicate.

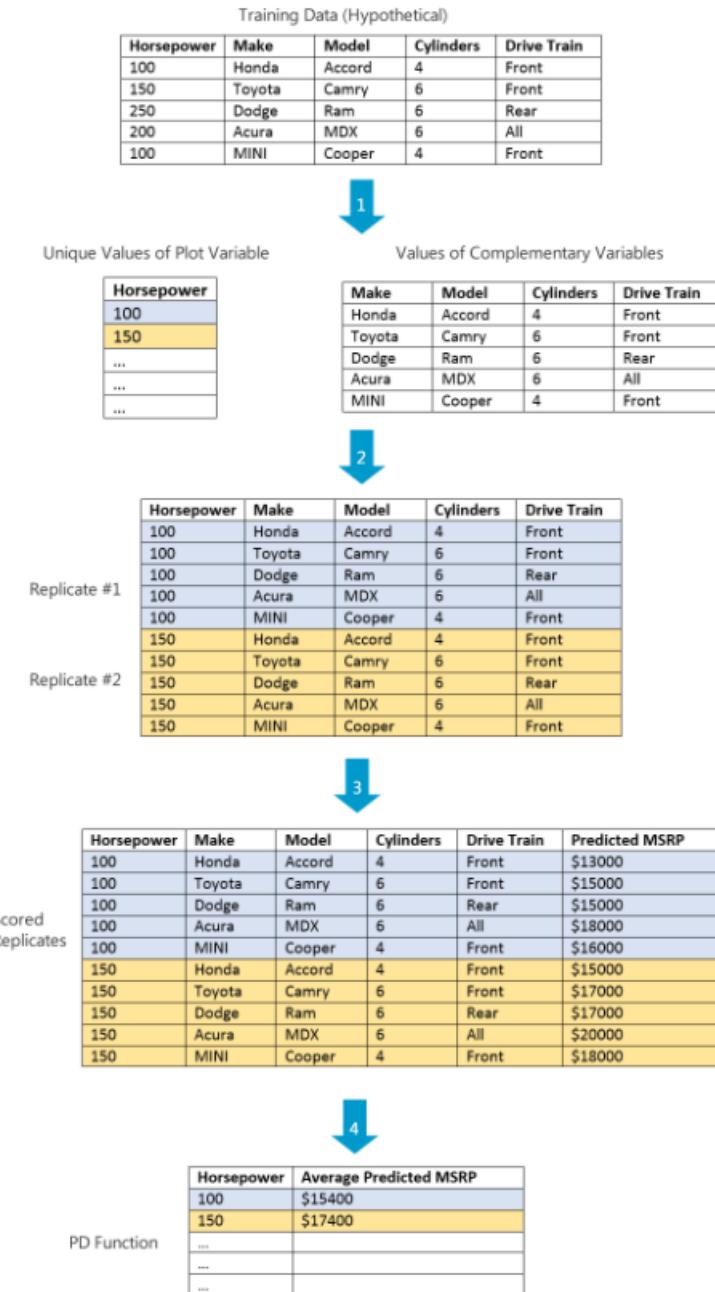


Figure 5. Computing the PD Function

As you can see from the last step in Figure 5, each value of the PD function represents an average prediction for a particular value of the plot variable. A simpler way to depict the functional relationship between the horsepower and MSRP variables might be to fix the values of the complementary variables at their average (or modal) values. But PD functions have the advantage that each value reflects the actual joint distribution of the complementary variables in the training set.

THE %PDFUNCTION MACRO

The following %PDFunction macro is called in the examples:

```
%macro PDfunction(
    dataset=,
    target=,
    PDVars=,
    otherIntervalInputs=,
    otherClassInputs=,
    scoreCodeFile=,
    outPD=
);
%let PDVar1 = %sysfunc(scan(&PDVars,1));
%let PDVar2 = %sysfunc(scan(&PDVars,2));

%let numPDVars = 1;
%if &PDVar2 ne %str() %then %let numPDVars = 2;

/*Obtain the unique values of the PD variable */
proc summary data = &dataset.;
    class &PDVar1. &PDVar2.;
    output out=uniqueXs
        %if &numPDVars = 1 %then
            %do;
        (where=_type_ = 1)
            %end;
        %if &numPDVars = 2 %then
            %do;
        (where=_type_ = 3)
            %end;
    ;
run;

/*Create data set of complementary Xs */
data complementaryXs;
    set &dataset(keep= &otherIntervalInputs. &otherClassInputs.);
    obsID = _n_;
run;

/*For every observation in uniqueXs, read in each observation
   from complementaryXs */
data replicates;
    set uniqueXs (drop=_type_ _freq_);
    do i=1 to n;
        set complementaryXs point=i nobs=n;
        %include "&scoreCodeFile.";
        output;
    end;
run;
```

```

/*Compute average yHat by replicate*/
proc summary data = replicates;
  class &PDVar1. &PDVar2.;
  output out=&outPD.
    %if &numPDVars = 1 %then
      %do;
      (where=(_type_ = 1))
      %end;
    %if &numPDVars = 2 %then
      %do;
      (where=(_type_ = 3))
      %end;
    mean (p_&target.) = AvgYHat;
  run;

%mend PDFunction;

```

The macro requires the following input parameters:

- dataset: Specify the training set.
- target: Specify the target variable to use in the predictive model.
- PDVars: For one-way plots, specify one variable; for two-way plots, specify two model variables.
- otherIntervalInputs: Specify the complementary model variables whose measurement level is interval.
- otherClassInputs: Specify the complementary model variables whose measurement level is nominal or binary.
- scoreCodeFile: Specify the score code from the machine learning model.
- outPD: Name the output data set to contain the PD function.

The %PDFunction macro is intended to introduce the concept of partial dependence and might not scale well to large data sets. As the number of unique values and observations increase, the number of replicated observations can grow out of hand. To avoid creating too many replicates for larger data sets, you can consider alternative approaches such as the following:

- Bin unique values of high-cardinality inputs such as income.
- Sample or cluster observations.
- Avoid stacking the replicates altogether: process the replicates one at a time, keeping only the average predicted value for each replicate.

Subsequent examples use one or more of these strategies to greatly reduce the number of rows that are replicated.

INDIVIDUAL CONDITIONAL EXPECTATION PLOTS

Whereas PD plots provide a coarse view of a model's workings, ICE plots enable you to drill down to the level of individual observations. Essentially, ICE plots disaggregate the PD function (which, after all, is an average) to reveal interactions and individual differences. To avoid visualization overload, ICE plots show one model variable at a time.

This section shows an example that uses generated data. The PD function in Figure 6 is essentially flat, giving the impression that there is no relationship between X1 and the model's predictions.

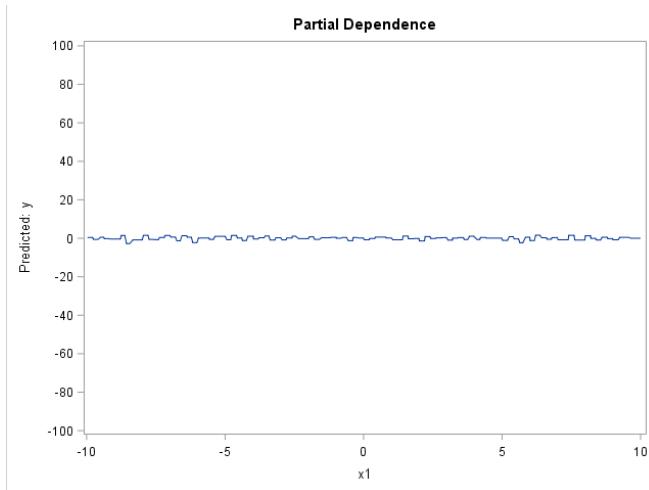


Figure 6. PD Plot for X1

Figure 7 is an ICE plot for two observations in the same data set.

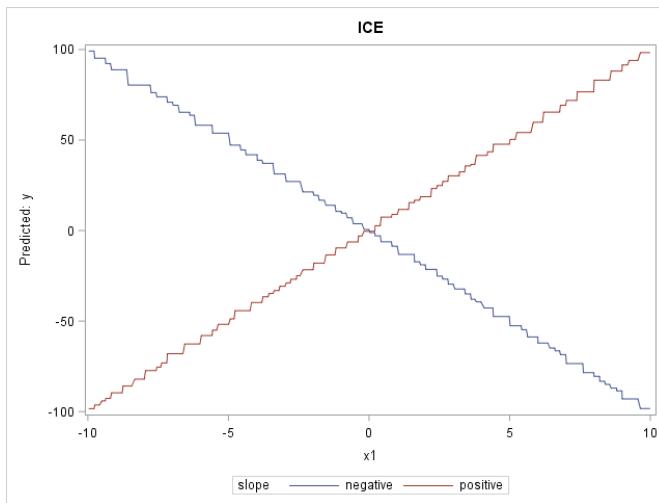


Figure 7. ICE Plot for X1

The ICE plot presents a much different picture: the relationship is strongly positive for one observation, but strongly negative for the other observation. So despite the PD plot, the ICE plot shows that X1 is actually related to the target; it's just that there are strong individual differences in the nature of that relationship.

Traditional ICE plots display one curve for each observation in the training set, but plotting a curve for every observation can result in visualization overload even for data sets of moderate size. Fortunately, you can manage the number of curves that are displayed by sampling or clustering.

Figure 8 shows the dependence of predicted MSRP on horsepower for 10 randomly selected observations.

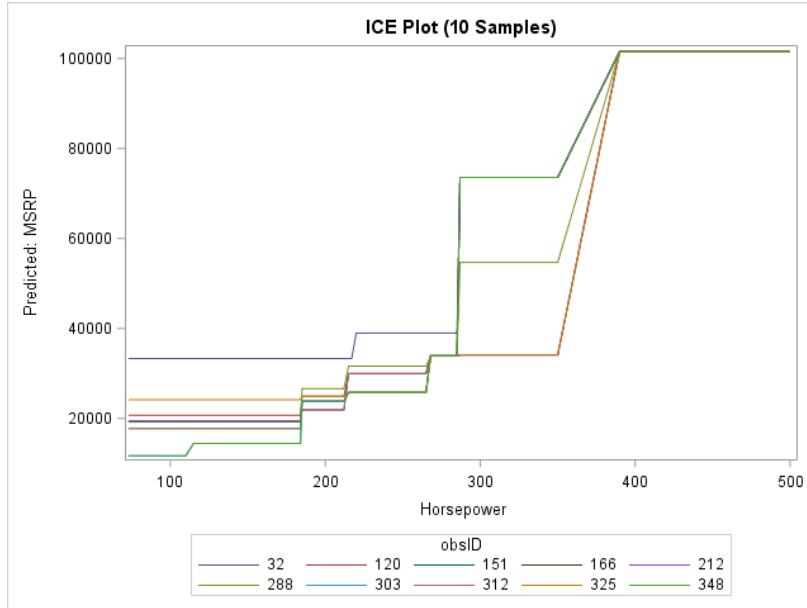


Figure 8. ICE Plot for Horsepower

Each series in the plot represents a different car model. Although some overlap is apparent among the individual curves, expected MSRP clearly diverges in the 280–350 horsepower range. The predictive model indicates that if horsepower were varied (leaving the other characteristics of each car unchanged), the same change in horsepower would have a different effect on the MSRP of the various car models. Although testing this in real life might not be practical, the model's prediction makes intuitive sense. It suggests that car manufacturers believe that customers are willing to pay for increased horsepower for some car models but not others.

COMPUTING THE ICE FUNCTION

You can think of each ICE curve as a kind of simulation that shows what would happen to the model's prediction if you varied one characteristic of a particular observation. As illustrated in Figure 9, the ICE curve for one observation is obtained by replicating the individual observation over the unique values of the plot variable and scoring each replicate.

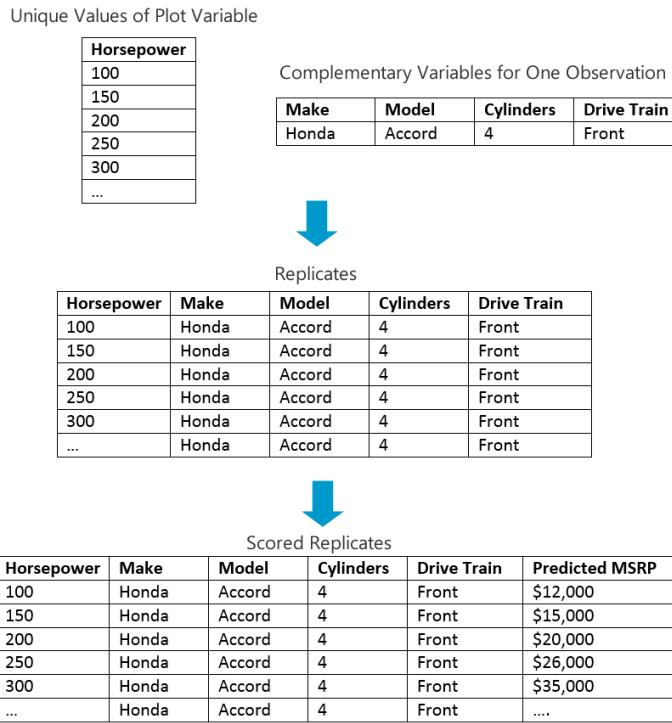


Figure 9. Computing an ICE Curve for One Observation

THE %ICEPLOT MACRO

ICE plots are essentially plots of raw replicates. Thus, if you have computed the PD function for a single variable, you need to add only a few more steps to produce an ICE plot for that variable:

5. Sample individuals from the replicates data set that is created by the %PDFunction macro.
6. Select the replicates that correspond to the sampled individuals.
7. Plot the sampled replicates as overlaid series.

The following %ICEPlot macro is used to plot the ICE curves:

```
%macro ICEPlot (
  ICEVar=,
  samples=10,
  YHatVar=
);

/*Select a small number of individuals at random*/
proc summary data = replicates;
  class obsID;
  output out=individuals (where=(_type_ = 1));
run;

data individuals;
  set individuals;
  random = ranuni(12345);
run;
```

```

proc sort data = individuals;
    by random;
run;

data sampledIndividuals;
    set individuals;
    if _N_ LE &samples.;
run;

proc sort data = sampledIndividuals;
    by obsID;
run;

proc sort data = replicates;
    by obsID;
run;

data ICEReplicates ;
    merge replicates sampledIndividuals (in = s);
    by obsID;
    if s;
run;

/*Plot the ICE curves for the sampled individuals*/
title "ICE Plot (&samples. Samples)";
proc sgplot data = ICEReplicates;
    series x=&ICEVar. y = &yHatVar. / group=obsID;
run;

%mend ICEPlot;

```

Figure 8 is created by the following call of the %ICEPlot macro, which specifies a plot variable, the number of individual curves to sample, and the variable to contain the model's predicted values:

```
%ICEPlot(
    ICEVar=horsepower,
    samples=10,
    YHatVar=p_MSRP
);
```

EXAMPLE: NBA BASKETBALL SHOTS

This section presents an extended example that uses a larger data set and more complicated machine learning models. The data are a sample of 16,934 basketball shots taken in the NBA 2015–2016 regular season.

An autotuned neural network model is used to predict shot success. As shown in Table 1, model inputs include both shot and player characteristics.

Variable	Role	Measurement Level	Values
Shot outcome	Target	Binary	0=made,1=missed
Distance to basket	Input	Interval	In feet
Player experience	Input	Interval	In years
Player height	Input	Interval	In inches
Player weight	Input	Interval	In pounds
Player position	Input	Nominal	Center, guard, or forward
Shot style	Input	Nominal	Jump, layup, hook, or dunk
Shot location	Input	Nominal	Right, left, center, left center, or right center
Shot area	Input	Nominal	Mid-range, restricted area (RA), in the paint (non-RA), above the break 3, right corner 3, left corner 3

Table 1. Model Variables

Because predictive models can have many inputs, it is customary to focus on the most important inputs when interpreting the model. According to a decision tree model, the most important predictors for the shot success model are distance to basket, shot style, and shot location (in that order). Figure 10 through

Figure 12 show partial dependence for the top model inputs in order of their relative importance.

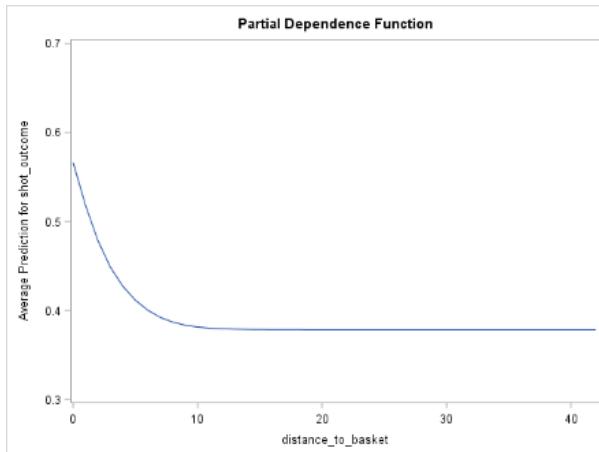


Figure 10. Distance to Basket

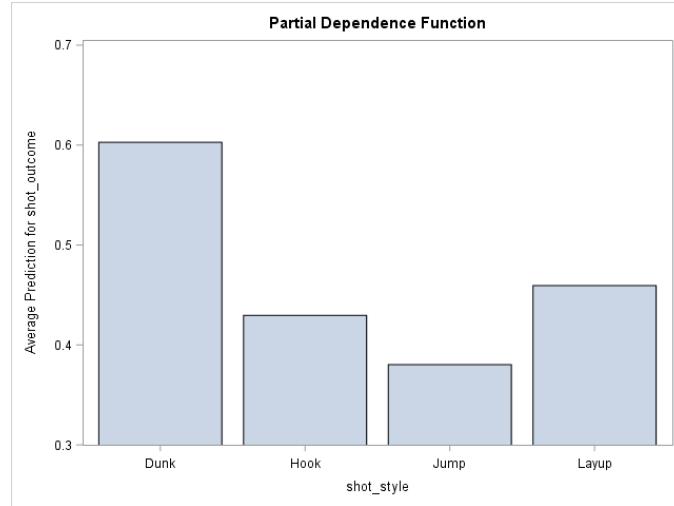


Figure 11. Shot Style

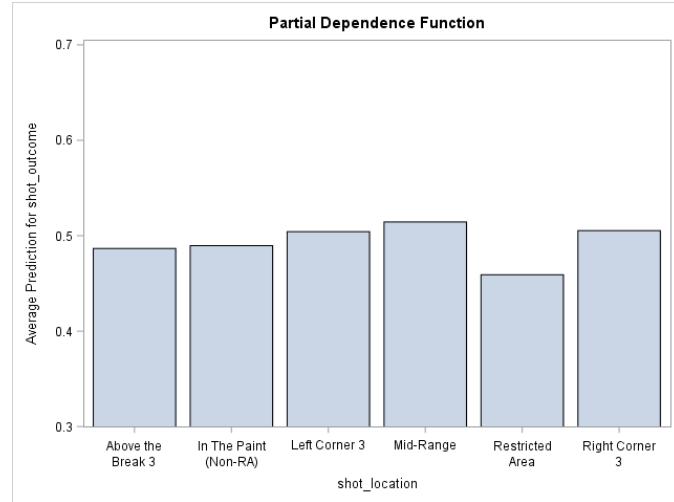


Figure 12. Shot Location

Figure 10 and

Figure 11 show that there is a success advantage for shots taken near the basket and for dunk shots, respectively. Because the plots use a consistent Y-axis scale, you can compare the success rate for each type of shot. For example, shots taken near the basket are approximately 20 percentage points more successful than those taken 10 feet away. This is about the same advantage as for dunk shots over hook shots.

Figure 12 shows a disadvantage for shots taken in the restricted area, although the effect of shot location is small compared to the difference among the different shot styles.

Typically data scientists build multiple candidate models and choose a champion based on both its accuracy and some assessment of its validity. Figure 13 compares PD functions for a neural network, an autotuned gradient boosting model, and an autotuned random forest.

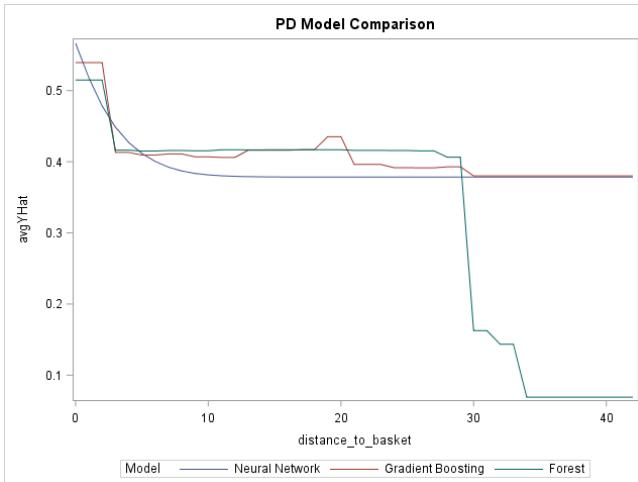


Figure 13. Comparison of Three Models

The models represent three different perspectives: For the neural network, the predicted effect of distance to basket on shot success is nonlinear and asymptotic, whereas the effect is more steplike for the other two models. Only one candidate, the forest, predicts a steep drop-off in success for shots taken from around 28 feet or more. That seems intuitive because long-distance shots are typically taken out of desperation. Overlaying PD functions in this manner can help you choose models that not only are accurate but also make intuitive sense and are acceptable to consumers of the model.

Let's see whether ICE plots reveal any interesting interactions or subgroups. Figure 14 is an ICE plot for distance to basket for the neural network model.

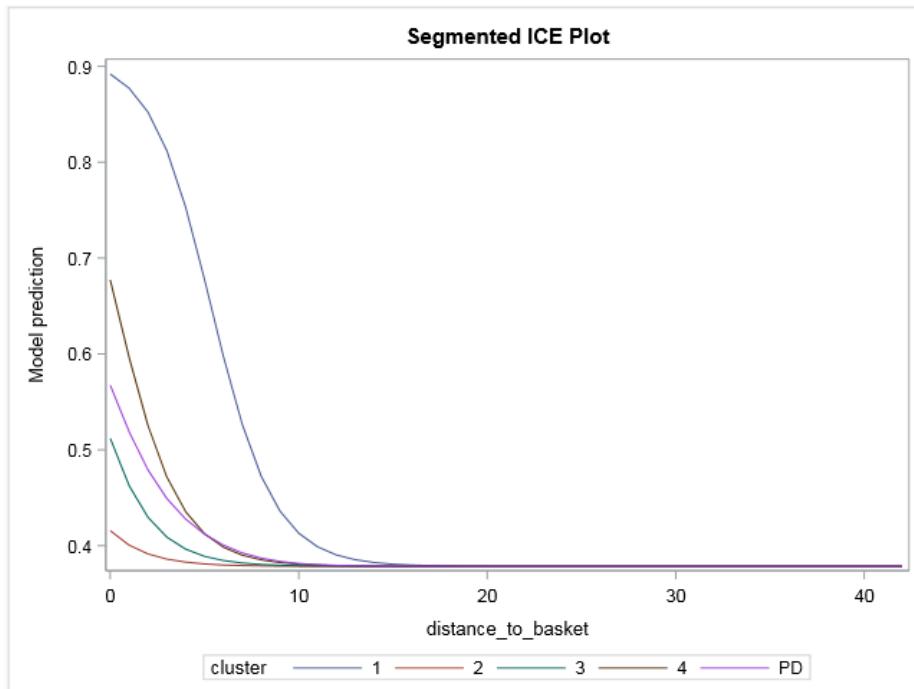


Figure 14. Distance to Basket

This plot actually is a *segmented* ICE plot. Unlike a traditional ICE plot, in which each curve represents a single observation, each curve in a segmented plot represents a cluster of observations whose ICE curves have a similar shape. By plotting representative clusters rather than observations, the number of curves is greatly reduced, making the plot easier to digest. Figure 14 also includes the PD function (the overall average) for reference.

There are two things to look for in ICE plots: intersecting slopes, which indicate interactions between the plot variable and one or more complementary variables, and level differences, which indicate group effects. The slopes of the various clusters are nearly parallel, indicating a lack of strong interaction between distance to basket and the other model variable. But notice the dark blue line (cluster 1): it indicates a marked success advantage for certain types of shots compared to other shots taken at the same distance to basket. It turns out that this cluster includes a disproportionate number of dunk shots taken in the restricted area at center court. These shots had an extremely high success rate of 84.6%.

An alternative to clustering the individual curves is to group them by values of other model variables.

Figure 15 and Figure 16 show ICE curves for distance to basket for 250 randomly selected shots that are grouped by shot style and location, respectively:

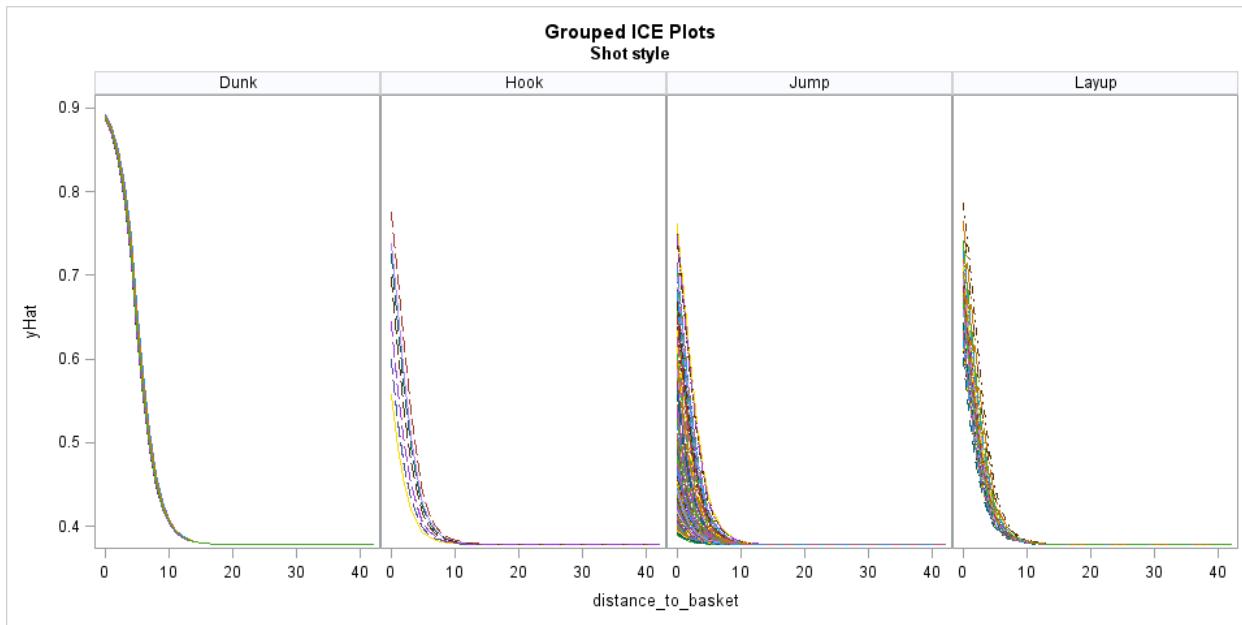


Figure 15. Grouped ICE Plot: Shot Style

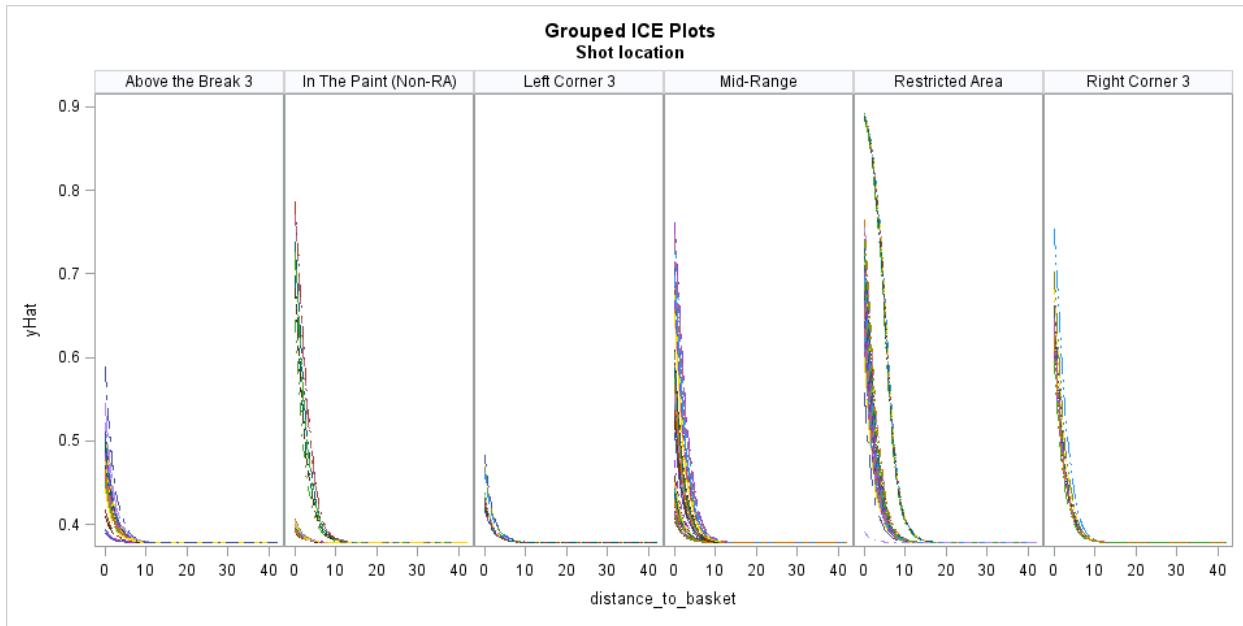


Figure 16. Grouped ICE Plot: Shot Location

When ICE curves are grouped by shot style, it appears that dunk shots are more advantageous than other shot styles. This is consistent with the PD plot for shot style. But when ICE curves are grouped by shot location, it appears that shots taken in the restricted area include high-, medium-, and low-probability shots. In other words, some shots in the restricted area are actually advantageous—a finding that is not apparent from the PD plot. After segregating the ICE curves in this manner, you could drill even deeper to explore why some shots in the restricted area are more successful than others.

SAS® VISUAL DATA MINING AND MACHINE LEARNING

SAS Visual Data Mining and Machine Learning enables you to run machine learning pipelines that build powerful supervised learning models. Figure 17 shows a pipeline that runs three autotuned black-box algorithms to predict shot success: neural network, gradient boosting, and forest.

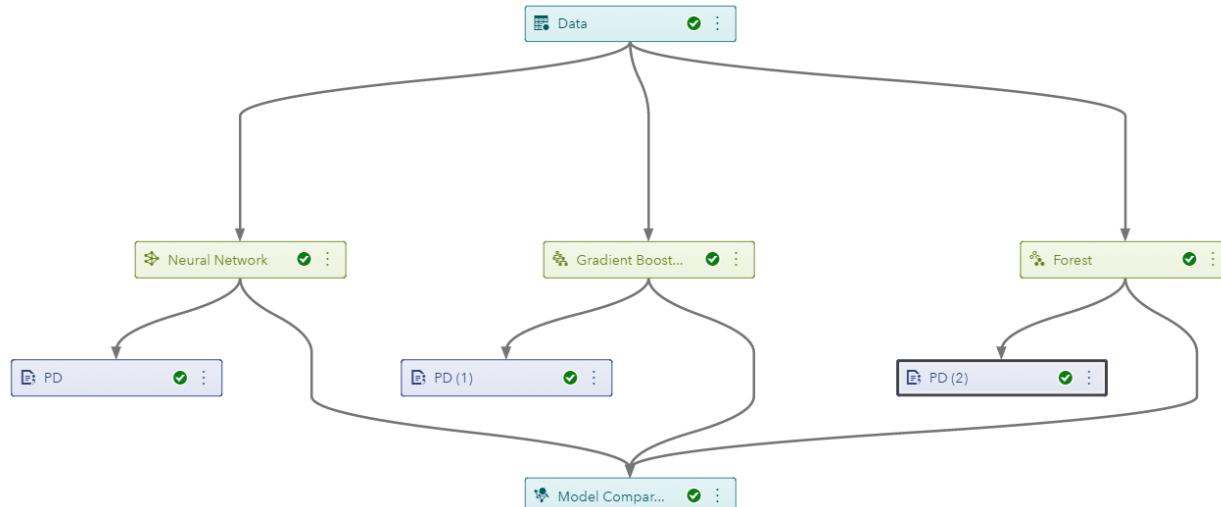


Figure 17. Model Studio Pipeline

Model Studio, the visual interface to SAS Visual Data Mining and Machine Learning, enables you to use a Code node to generate partial dependence plots. Each node that immediately follows a modeling node is a Code node that produces PD plots for the predecessor model. (The Code nodes are identified by the  symbol.) For example, Figure 18 shows Code node output for the forest model.

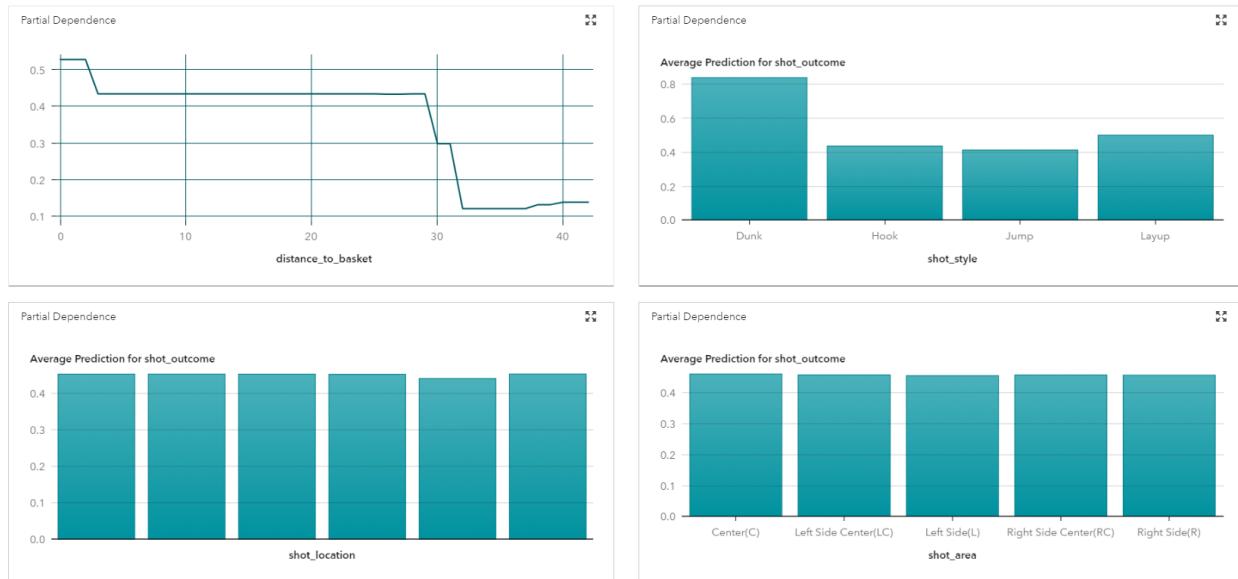


Figure 18. Partial Dependence Results

Each Code node calls the %partialDep macro (available at <https://github.com/sassoftware/sas-viya-machine-learning/tree/master/interpretability>) to create for each requested input a PD plot that is appropriate to that input's measurement level. Depending on the modeling algorithm used in the pipeline, the macro obtains either score code or the analytics store from the preceding modeling node. To limit the number of replicated observations, the macro samples observations by default.

Using the macro is straightforward: simply download the macro, paste it into a Code node, and then call it, specifying the number of important inputs to plot and the proportion of observations to sample. For example, the following macro call requests PD plots for four inputs and uses a 10% sample of the observations:

```
%partialDep(
    importantInputs=distance_to_basket shot_style shot_location,
    obsSampProp=.10
);
```

CONCLUSION

Modern machine learning algorithms can be incredibly powerful predictors but their inner workings are often difficult for data scientists to digest. PD plots help you understand how your model works by depicting how changes in input values affect a model's predictions, and you can use them to evaluate competing models. ICE plots enable you to drill deeper to find interactions among model variables and unusual subgroups in your data.

Although you can easily compute traditional PD and ICE plots, you might need to make some adjustments for efficient computation with large data sets. Several options are available, including binning of plot variables, sampling of rows, and sampling or clustering of observations and ICE curves. You can use these techniques individually or in combination to produce reasonable approximations of traditional plots in a fraction of the time.

Although PD and ICE plots provide only indirect approximations of a model's workings, they are popular and highly visual tools for obtaining a working understanding of increasingly complicated machine learning models.

REFERENCES

Friedman, J.H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, 29, pp. 1189–1232.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2014). "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation." arXiv:1309.6392v2.

ACKNOWLEDGMENTS

Thanks to Funda Gunes for critical feedback.

CONTACT INFORMATION

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

Ray Wright
SAS Institute Inc.
Ray.wright@sas.com

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.

Building Bayesian Network Classifiers Using the HPBNET Procedure

Ye Liu, Weihua Shi, and Wendy Czika, SAS Institute Inc.

ABSTRACT

A Bayesian network is a directed acyclic graphical model that represents probability relationships and conditional independence structure between random variables. SAS® Enterprise Miner™ implements a Bayesian network primarily as a classification tool; it supports naïve Bayes, tree-augmented naïve Bayes, Bayesian-network-augmented naïve Bayes, parent-child Bayesian network, and Markov blanket Bayesian network classifiers. The HPBNET procedure uses a score-based approach and a constraint-based approach to model network structures. This paper compares the performance of Bayesian network classifiers to other popular classification methods such as classification tree, neural network, logistic regression, and support vector machines. The paper also shows some real-world applications of the implemented Bayesian network classifiers and a useful visualization of the results.

INTRODUCTION

Bayesian network (BN) classifiers are one of the newest supervised learning algorithms available in SAS Enterprise Miner. The HP BN Classifier node is a high-performance data mining node that you can select from the HPDM toolbar; it uses the HPBNET procedure in SAS® High-Performance Data Mining to learn a BN structure from a training data set. This paper shows how the various BN structures that are available in PROC HPBNET can be used as a predictive model for classifying a binary or nominal target.

Because of the practical importance of classification, many other classifiers besides BN classifiers are commonly applied. These classifiers include logistic regression, decision tree, support vector machines, and neural network classifiers. Recent research in supervised learning has shown that the prediction performance of the BN classifiers is competitive when compared to these other classifiers. However, BN classifiers can surpass these competitors in terms of interpretability. A BN can explicitly represent distributional dependency relationships among all available random variables; thus it enables you to discover and interpret the dependency and causality relationships among variables in addition to the target's conditional distribution. In contrast, support vector machines and neural network classifiers are black boxes and logistic regression and decision tree classifiers only estimate the conditional distribution of the target. Therefore, BN classifiers have great potential in real-world classification applications, especially in fields where interpretability is a concern.

SAS Enterprise Miner implements PROC HPBNET to build BN classifiers that can take advantage of modern multithreaded distributed-computing platforms. The HPBNET procedure can build five types of BN classifiers: naïve BN, tree-augmented naïve BN, BN-augmented naïve BN, parent-child BN, and Markov blanket BN. This paper introduces the basic structure of these five types of BN classifiers, explains the key programming techniques and outputs of the HPBNET procedure, and demonstrates useful visualization methods for displaying the structures of the output BN classifiers. This paper also compares the prediction performance of BN classifiers to that of the previously mentioned competitor classifiers by using 25 data sets in the UCI Machine Learning Repository (Lichman 2013).

BAYESIAN NETWORKS

A Bayesian network is a graphical model that consists of two parts, $\langle \mathbf{G}, \mathbf{P} \rangle$:

- \mathbf{G} is a directed acyclic graph (DAG) in which nodes represent random variables and arcs between nodes represent conditional dependency of the random variables.
- \mathbf{P} is a set of conditional probability distributions, one for each node conditional on its parents.

The following example explains these terms in greater detail.

EXAMPLE OF A SIMPLE BAYESIAN NETWORK

Figure 1 shows a Bayesian network for a house alarm from Russell and Norvig (2010). It describes the following scenario: Your house has an alarm system against burglary. You live in a seismically active area, and the alarm system can be set off occasionally by an earthquake. You have two neighbors, Mary and John, who do not know each other. If they hear the alarm, they might or might not call you.

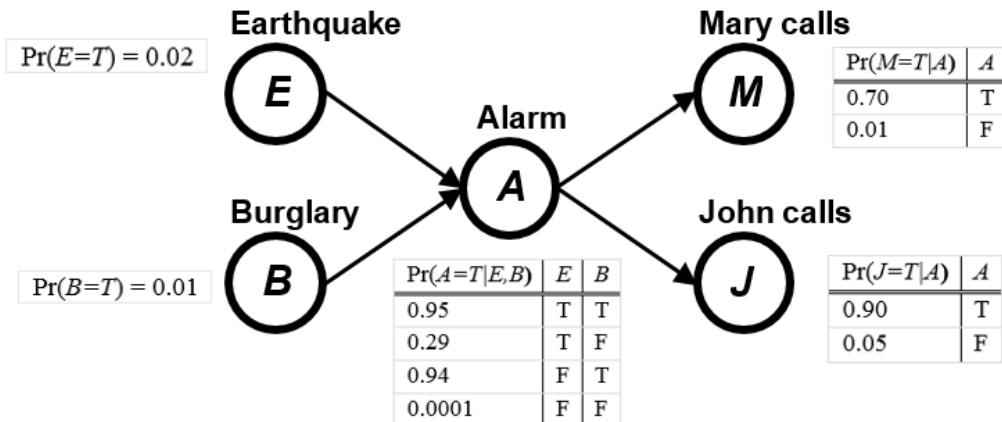


Figure 1. House Alarm Bayesian Network

In the house alarm Bayesian network, E , B , A , M , and J are called nodes, and the links between those five nodes are called edges or arcs. Node A is the parent of nodes J and M because the links point from A to J and M ; nodes J and M are called the children of node A . Similarly, nodes E and B are the parents of node A ; node A is the child of nodes E and B . Those nodes and edges constitute the graph (\mathbf{G}) part of the Bayesian network model. The conditional probability tables (CPTs) that are associated with the nodes are the probability distribution (\mathbf{P}) part of the Bayesian network model.

PROPERTIES OF BAYESIAN NETWORK

Two important properties of a Bayesian network are the following:

- Edges (arcs between nodes) represent “causation,” so no directed cycles are allowed.
- Each node is conditionally independent of its ancestors given its parents. This is called Markov property.

According to the Markov property, the joint probability distribution of all nodes in the network can be factored to the product of the conditional probability distributions of each node given its parents. That is,

$$\Pr(G) = \Pr(X_1, X_2, \dots, X_p) = \prod_{i=1}^p \Pr(X_i | \pi(X_i))$$

where $\pi(X_i)$ are the parents of node X_i .

In the simplest case, where all the X_i are discrete variables as in the following example, conditional distribution is represented as CPTs, each of which lists the probability that the child node takes on each of its different values for each combination of values of its parents.

In the house alarm example, observe that whether Mary or John calls is conditionally dependent only on the state of the alarm (that is, their parent node). Based on the graph, the joint probability distribution of the events (E, B, A, M, and J) is

$$\Pr(E, B, A, M, J) = \Pr(J|A) \cdot \Pr(M|A) \cdot \Pr(A|E, B) \cdot \Pr(B) \cdot \Pr(E)$$

The network structure together with the conditional probability distributions completely determine the Bayesian network model.

SUPERVISED LEARNING USING A BAYESIAN NETWORK MODEL

Now consider this question:

Suppose you are at work, the house is burglarized ($B = \text{True}$), there is no earthquake ($E = \text{False}$), your neighbor Mary calls to say your alarm is ringing ($M = \text{True}$), but neighbor John doesn't call ($J = \text{False}$). What is the probability that the alarm went off ($A = \text{True}$)?

In other words, what is the value of

$$\Pr(A = T | B = T, E = F, M = T, J = F)$$

To simplify the appearance of these equations, T and F are used to represent True and False, respectively.

From the definition of conditional probability,

$$\Pr(A = T | B = T, E = F, M = T, J = F) = \frac{\Pr(A = T, B = T, E = F, M = T, J = F)}{\Pr(B = T, E = F, M = T, J = F)}$$

According to the equation for $\Pr(E, B, A, M, J)$ from the preceding section and using the values from the conditional probability tables that are shown in Figure 1,

$$\begin{aligned} \Pr(A = T, B = T, E = F, M = T, J = F) &= \Pr(J = F | A = T) \Pr(M = T | A = T) \Pr(A = T | E = F, B = T) \Pr(B = T) \Pr(E = F) \\ &= 0.1 * 0.01 * 0.7 * 0.94 * (1 - 0.02) = 0.00064484 \end{aligned}$$

$$\begin{aligned} \Pr(B = T, E = F, M = T, J = F) &= \Pr(A = T, B = T, E = F, M = T, J = F) + \Pr(A = F, B = T, E = F, M = T, J = F) \\ &= 0.00064484 + \Pr(A = F, B = T, E = F, M = T, J = F) \\ &= 0.00064484 \\ &\quad + \Pr(J = F | A = F) \Pr(B = T) \Pr(M = T | A = F) \Pr(A = F | E = F, B = T) \Pr(E = F) \\ &= 0.00064484 + (1 - 0.05) * 0.01 * 0.01 * (1 - 0.94) * (1 - 0.02) = 0.000650426 \end{aligned}$$

$$\Pr(A = T | B = T, E = F, M = T, J = F) = \frac{0.00064484}{0.000650426} \approx 0.99$$

Thus, the conditional probability of the alarm having gone off in this situation is about 0.99. This value can be used to classify (predict) whether the alarm went off.

In general, based on a Bayesian network model, a new observation $X = (x_1, x_2, \dots, x_p)$ is classified by determining the classification of the target Y that has the largest conditional probability,

$$\arg \max_k \Pr(Y = k | x_1, x_2, \dots, x_p)$$

where

$$\Pr(Y = k | x_1, x_2, \dots, x_p) \propto \Pr(Y = k, x_1, x_2, \dots, x_p) = \prod_i \Pr(x_i | \pi(X_i)) \Pr(Y = k | \pi(Y))$$

Because the target is binary (True or False) in this example, when the value of the preceding equation is greater than 0.5, the prediction is that the alarm went off (A = True).

HPBNET PROCEDURE

The HPBNET procedure is a high-performance procedure that can learn different types of Bayesian networks—naïve, tree-augmented naïve (TAN), Bayesian network-augmented naïve (BAN), parent-child Bayesian network (PC), or Markov blanket (MB)—from an input data set. PROC HPBNET runs in either single-machine mode or distributed-computing mode. In this era of big data, where computation performance is crucial for many real-world applications, the HPBNET procedure's distributed-computing mode is very efficient in processing large data sets.

The HPBNET procedure supports two types of variable selection: one by independence tests between each input variable and the target (when PRESCREENING=1), and the other by conditional independence tests between each input variable and the target given any subset of other input variables (when VARSELECT=1, 2, or 3). PROC HPBNET uses specialized data structures to efficiently compute the contingency tables for any variable combination, and it uses dynamic candidate generation to reduce the number of false candidates for variable combinations. If you have many input variables, structure learning can be time-consuming because the number of variable combinations is exponential. Therefore, variable selection is strongly recommended.

To learn a TAN structure, the HPBNET procedure constructs a maximum spanning tree in which the weight for an edge is the mutual information between the two nodes. A maximum spanning tree is a spanning tree of a weighted graph that has maximum weight. If there are K variables in a system, then the corresponding tree structure will have K nodes, and K–1 edges should be added to create a tree structure that connects all the nodes in the graph. Also, the sum of the weights of all the edges needs to be the maximum weight among all such tree structures.

To learn the other BN types, PROC HPBNET uses both of the following approaches:

- The score-based approach uses the BIC (Bayesian information criterion) score to measure how well a structure fits the training data and then tries to find the structure that has the best score. The BIC is defined as

$$\text{BIC}(G, D) = N \sum_{i=1}^n \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} p(\pi_{ij}) p(X_i = v_{ik} | \pi_{ij}) \ln p(X_i = v_{ik} | \pi_{ij}) - \frac{M}{2} \ln N$$

where G is a network, D is the training data set, N is the number of observations in D , n is the number of variables, X_i is a random variable, r_i is the number of levels for X_i , v_{ik} is the k th value of X_i , q_i is the number of value combinations of X_i 's parents, π_{ij} is the j th value combination of X_i 's parents, and $M = \sum_{i=1}^n (r_i - 1) \times q_i$ is the number of parameters for the probability distributions.

- The constraint-based approach uses independence tests (such as a chi-square test or mutual information test) to determine the edges and directions among the nodes as follows: Assume that you have three variables, X , Y and Z , and that it has been determined (using independence tests) that there are edges between X and Z and Y and Z , but no edge between X and Y . If X is conditionally independent of Y given any subset of variables $S = \{Z\} \cup S'$, $S' \subseteq \{X, Y, Z\}$, then the directions between X and Z and between Y and Z are $X \rightarrow Z$ and $Y \rightarrow Z$, respectively. Notice that using only independence tests might not be able to orient all edges because some structures are equivalent with respect to conditional independence tests. For example, $X \leftarrow Y \leftarrow Z$, $X \rightarrow Y \rightarrow Z$, and $X \leftarrow Y \rightarrow Z$ belong to the same equivalence class. In these cases, PROC HPBNET uses the BIC score to determine the directions of the edges.

For the PC and MB structures, PROC HPBNET learns the parents of the target first. Then it learns the parents of the input variable that has the highest BIC score with the target. It continues learning the parents of the input variable that has the next highest BIC score, and so on. When learning the parents of a node, it first determines the edges by using independence tests. Then it orients the edges by using both independence tests and the BIC score. PROC HPBNET uses the BIC score not only for orienting the edges but also for controlling the network complexity, because a complex network that has more parents is penalized in the BIC score. Both the BESTONE and BESTSET values of the PARENTING= option try to find the local optimum structure for each node. BESTONE adds the best candidate variable to the parents at each iteration, whereas BESTSET tries to choose the best set of variables among the candidate sets.

TYPES OF BAYESIAN NETWORK CLASSIFIERS SUPPORTED BY THE HPBNET PROCEDURE

The HPBNET procedure supports the following types of Bayesian network classifiers:

- Naïve Bayesian network classifier:** As shown in Figure 2, the target node (Y) has a direct edge to each input variable, the target node is the only parent for all other nodes, and there are no other edges. This structure assumes that all input variables are conditionally independent of each other given the target.

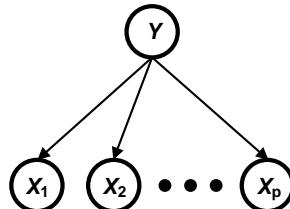


Figure 2. Naïve Bayesian Network Classifier

- **Tree-augmented naïve Bayesian network classifier:** As shown in Figure 3, in addition to the edges from the target node Y to each input node, the edges among the input nodes form a tree. This structure is less restrictive than the naïve Bayes structure.

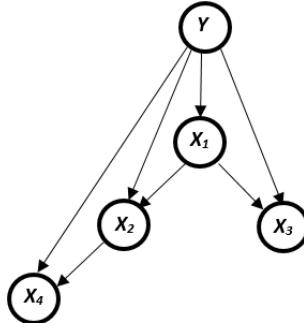


Figure 3. Tree-Augmented Naïve Bayesian Network Classifier

- **Bayesian network-augmented naïve Bayesian network classifier:** As shown in Figure 4, the target node Y has a direct edge to each input node, and the edges among the input nodes form a Bayesian network.

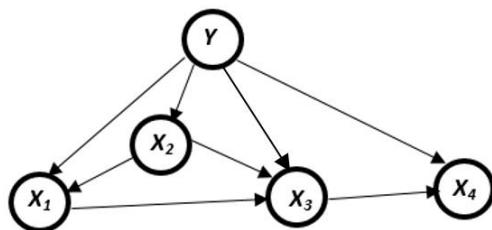


Figure 4. Bayesian Network-Augmented Naïve Bayesian Network Classifier

- **Parent-child Bayesian network classifier:** As shown in Figure 5, input variables can be the parents of the target variable Y. In addition, edges from the parents of the target to the children of the target

and among the children of the target are also possible.

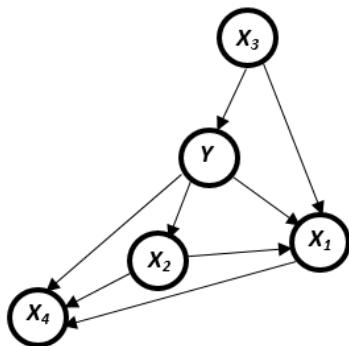


Figure 5. Parent-Child Bayesian Network Classifier

- **Markov blanket Bayesian network classifier:** As shown in Figure 6, the Markov blanket includes the target's parents, children, and spouses (the other parents of the target's children).

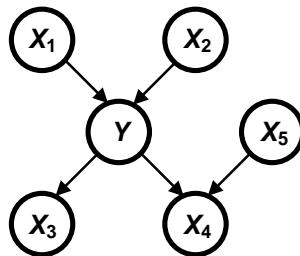


Figure 6. Markov Blanket Bayesian Network Classifier

One advantage of PROC HPBNET is that you can specify all the structures that you want to consider for training and request (by specifying the BESTMODEL option) that the procedure automatically choose the best structure based on each model's performance on validation data.

EXAMPLE OF USING PROC HPBNET TO ANALYZE DATA

This example uses PROC HPBNET to diagnose whether a patient has breast cancer, based on the Breast Cancer Wisconsin data set from the UCI Machine Learning Repository (Lichman 2013).

Table 1 lists the details of the attributes found in this data set.

Variables	Attribute	Domain	Description of Benign Cells	Description of Cancerous Cells
1	Sample code number	ID number	N/A	N/A
2	Clump thickness	1–10	Tend to be grouped in monolayers	Often grouped in multiple layers
3	Uniformity of cell size	1–10	Evenly distributed	Unevenly distributed
4	Uniformity of cell shape	1–10	Evenly distributed	Unevenly distributed
5	Marginal adhesion	1–10	Tend to stick together	Tend not to stick together
6	Single epithelial cell size	1–10	Tend to be normal-sized	Tend to be significantly enlarged
7	Bare nuclei	1–10	Typically nuclei are not surrounded by cytoplasm of benign cells	Nuclei might be surrounded by cytoplasm
8	Bland chromatin	1–10	Uniform “texture” of nucleus	Coarser “texture” of nucleus
9	Normal nucleoli	1–10	Very small, if visible	More prominent, and greater in number
10	Mitoses	1–10	Grade of cancer determined by counting the number of mitoses (nuclear division, the process by which the cell divides and replicates)	
11	Class	2 or 4	2	4

Table 1. Attributes of Breast Cancer Wisconsin Data Set

The RENAME statement in the following DATA step enables you to assign a name to each variable so that you can understand it more easily:

```
data BreastCancer;
set BreastCancer;
rename var1=ID
      var2=Clump_Thickness
      var3=Uniformity_of_Cell_Size
      var4=Uniformity_of_Cell_Shape
      var5=Marginal_Adhesion
      var6=Single_Epithelial_Cell_Size
      var7=Bare_Nuclei
      var8=Bland_Chromatin
      var9=Normal_Nucleoli
      var10=Mitoses
      var11=Class;
run;
```

The following SAS program shows how you can use PROC HPBNET to analyze the BreastCancer data set:

```
proc hpbnet data=BreastCancer nbins=5 structure=Naive TAN PC MB bestmodel;
target Class;
id ID;
input Clump_Thickness Uniformity_of_Cell_Size Uniformity_of_Cell_Shape
Marginal_Adhesion Single_Epithelial_Cell_Size Bare_Nuclei Bland_Chromatin
Normal_Nucleoli Mitoses/level=INT;
output network=net validinfo=vi varselect=vs
      varlevel=var1 parameter=parm fit=fitstats pred=prediction;
partition fraction(validate=0.3 seed=12345);
code file="c:\hpbnetscorecode.sas" ;
run;
```

The TARGET statement specifies **Class** as the target variable. The ID statement specifies **ID** as the ID variable. The INPUT statement specifies that all the other variables are to be used as interval inputs. The NBIN= option in the PROC HPBNET statement specifies 5 for the number of equal-width bins for interval inputs. Four different structures are specified in the STRUCTURE= option (so each structure is trained), and the BESTMODEL option requests that PROC HPBNET automatically choose the best model to minimize the validation misclassification rate. The FRACTION option in the PARTITION statement requests that 30% of the data be used for validation (leaving 70% to be used for training). The OUTPUT statement specifies multiple output tables to be saved in the Work directory. The CODE statement specifies a filename (**hpbnetscorecode.sas**) where the generated score code is to be stored.

After you run PROC HPBNET, you can visualize the final model by using the **%createBNCdiagram** macro in the Appendix to view the selected Bayesian network structure. This macro takes the target variable and the output network data as arguments.

Figure 7 shows the generated diagram, which indicates that the naïve Bayes network is selected as the best structure for this data set, because the input variables are all conditionally independent of each other given the target.

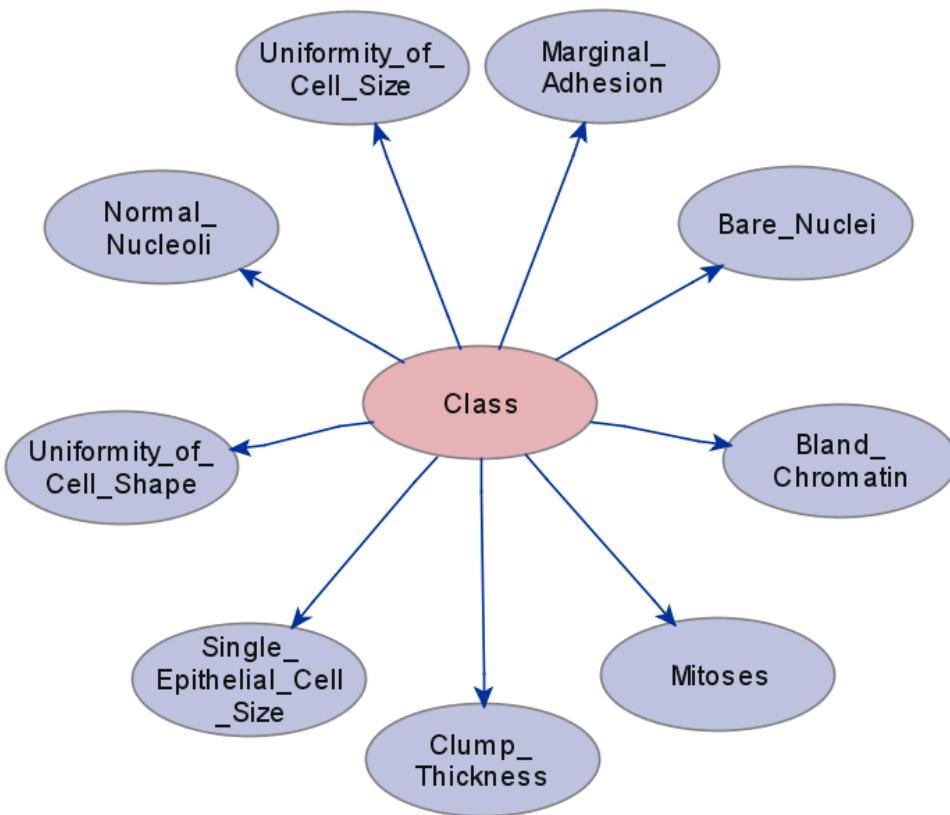


Figure 7. Bayesian Network Diagram

Table 2 through Table 7 show all the other output tables, which are stored in the Work directory.

The Best Model column in Table 2 shows that a naïve Bayesian network model with a maximum of one parent is selected, and the Misclassification Errors column shows that five validation observations are misclassified.

	Best Model	Misclassification Errors	Significance Threshold	Input Parameter: Prescreening	Input Parameter: Variable Selection	Input Parameter: Structure	Input Parameter: Parenting Method	Input Parameter: Maximum Number of Parents
1		5	0.05 1	1	PC	BESTSET		1
2		5	0.05 1	1	PC	BESTSET		2
3		5	0.05 1	1	PC	BESTSET		3
4		5	0.05 1	1	PC	BESTSET		4
5		5	0.05 1	1	PC	BESTSET		5
6		5	0.05 1	1	TAN	BESTSET		2
7	YES	5	0.05 1	1	NAIVE	BESTSET		1
8		5	0.05 1	1	NAIVE	BESTSET		2
9		5	0.05 1	1	NAIVE	BESTSET		3
10		5	0.05 1	1	NAIVE	BESTSET		4
11		5	0.05 1	1	NAIVE	BESTSET		5
12		44	0.05 1	3	MB	BESTSET		1
13		44	0.05 1	3	MB	BESTSET		2
14		44	0.05 1	3	MB	BESTSET		3
15		44	0.05 1	3	MB	BESTSET		4
16		44	0.05 1	3	MB	BESTSET		5

Table 2. Validation Information Table

Table 3 shows that the number of observations for validation is 178. Together with the misclassification errors shown in Table 2, you can calculate the validation accuracy as $1 - 5/178 = 97.19\%$. In PROC HPBNET, continuous variables are binned to equal-width discrete levels in order to simplify the model. If you want to improve this accuracy, you can discretize the interval inputs differently. For example, you could use entropy binning instead of equal-width binning.

	Number of Observations for Training	Number of Observations for Validation	Number of Observations Ignored	Sum of Frequencies for Training	Sum of Frequencies for Validation	Sum of Frequencies Ignored	Number of Nodes	Number of Links	Average Degree	Maximum Number of Parents in Network	Number of Parameters	Score
1	506	178	15	506	178	15	10	9	1.8	1	73	-4101.349933

Table 3. Fit Statistics Table

Table 4 shows the variable selection results. In the preceding PROC HPBNET call, the VARSELECT= option is not specified in the PROC statement, so its default value is applied. By default, each input variable is tested for conditional independence of the target variable given any other input variable, and only the variables that are conditionally dependent on the target given any other input variable are selected. Table 4 shows that all the nine input variables are selected into the model.

	Variable Name	Selected	Chi-Square Statistics	G-Square Statistics	P-Value of Chi-Square Statistics	P-Value of G-Square Statistics	Mutual Information	Degree of Freedom	Conditional Variables
1	Bare_Nuclei	YES	353.36783823	403.47623668	3.28698E-75	4.933826E-86	0.7412786201	4	
2	Bland_Chromatin	YES	300.12175361	354.16534484	1.019862E-63	2.211028E-75	0.7094932591	4	
3	Clump_Thickness	YES	288.29453088	335.88442772	3.626159E-61	1.955967E-71	0.696498837	4	
4	Marginal_Adhesion	YES	259.38424306	288.34682195	6.189801E-55	3.533217E-61	0.6590847963	4	
5	Mitoses	YES	110.78977073	115.6084875	4.93795E-23	4.62748E-24	0.4519447152	4	
6	Normal_Nucleoli	YES	315.99038507	349.49850642	3.845419E-67	2.250386E-74	0.7062430014	4	
7	Single_Epithelial_Cell_Size	YES	324.76623888	362.06155283	4.910373E-69	4.35991E-77	0.7148918452	4	
8	Uniformity_of_Cell_Shape	YES	381.82995416	458.83620861	2.343084E-81	5.339491E-98	0.772128209	4	
9	Uniformity_of_Cell_Size	YES	378.18381534	449.08950513	1.436784E-80	6.83417E-96	0.7670255046	4	

Table 4. Selected Variables Table

Table 5 shows the details for each level of the target and input variables. The values of 0–4 in the Level Index column indicate that PROC HPBNET bins each interval input variable into five equal-width levels. The number of bins can be specified in the NBIN= option; by default, NBIN=5.

	Variable Name	Level Index	Level Value	Frequency
1	Class	0	4	178
2	Class	1	2	328
3	Bare_Nuclei	0	<2.8	323
4	Bare_Nuclei	1	<4.6	32
5	Bare_Nuclei	2	<6.4	21
6	Bare_Nuclei	3	<8.2	23
7	Bare_Nuclei	4	>=8.2	107
8	Bland_Chromatin	0	<2.8	230
9	Bland_Chromatin	1	<4.6	151
10	Bland_Chromatin	2	<6.4	35
11	Bland_Chromatin	3	<8.2	70
12	Bland_Chromatin	4	>=8.2	20
13	Clump_Thickness	0	<2.8	140
14	Clump_Thickness	1	<4.6	138
15	Clump_Thickness	2	<6.4	117
16	Clump_Thickness	3	<8.2	50
17	Clump_Thickness	4	>=8.2	61
18	Marginal_Adhesion	0	<2.8	341
19	Marginal_Adhesion	1	<4.6	62
20	Marginal_Adhesion	2	<6.4	34
21	Marginal_Adhesion	3	<8.2	28
22	Marginal_Adhesion	4	>=8.2	41
23	Mitoses	0	<2.8	442
24	Mitoses	1	<4.6	36
25	Mitoses	2	<6.4	5
26	Mitoses	3	<8.2	15
27	Mitoses	4	>=8.2	8
28	Normal_Nucleoli	0	<2.8	347
29	Normal_Nucleoli	1	<4.6	46
30	Normal_Nucleoli	2	<6.4	31
31	Normal_Nucleoli	3	<8.2	28
32	Normal_Nucleoli	4	>=8.2	54
33	Single_Epithelial_Cell_Size	0	<2.8	313
34	Single_Epithelial_Cell_Size	1	<4.6	86
35	Single_Epithelial_Cell_Size	2	<6.4	61
36	Single_Epithelial_Cell_Size	3	<8.2	24
37	Single_Epithelial_Cell_Size	4	>=8.2	22
38	Uniformity_of_Cell_Shape	0	<2.8	301
39	Uniformity_of_Cell_Shape	1	<4.6	72
40	Uniformity_of_Cell_Shape	2	<6.4	46
41	Uniformity_of_Cell_Shape	3	<8.2	43
42	Uniformity_of_Cell_Shape	4	>=8.2	44
43	Uniformity_of_Cell_Size	0	<2.8	310
44	Uniformity_of_Cell_Size	1	<4.6	70
45	Uniformity_of_Cell_Size	2	<6.4	42
46	Uniformity_of_Cell_Size	3	<8.2	34
47	Uniformity_of_Cell_Size	4	>=8.2	50

Table 5. Variable Levels Table

Table 6 shows the parameter values for the resulting model.

	Parameter Name	Parameter Value
1	ALPHA	0.05
2	PREScreenING	1
3	VARSELECT	1
4	STRUCTURE	NAIVE
5	PARENTING	BESTSET
6	MAXPARENTS	1
7	MISSINGINT	IGNORE
8	MISSINGNOM	IGNORE
9	NBIN	5
10	INDEPTEST	CHIGSQUARE

Table 6. Parameter Table

Table 7 shows the prediction results for the first 20 observations of the training data. The Predicted: Class= columns contain the conditional probabilities for the **Class** variable, where Class=2 indicates a benign cell and Class=4 indicates a malignant cell. The conditional probabilities are then used to predict the target class. Here the target is known because these are the training data, but you can use this information to see how well the model is performing. The model is considered to perform well when the actual target class matches the target class that is predicted based on the conditional probabilities.

	ID	Class	Predicted: Class=4	Predicted: Class=2
1	1000025	2	0.0664101946	0.9335898054
2	1002945	2	0.9064857996	0.0935142004
3	1015425	2	0.0594603149	0.9405396851
4	1017122	4	0.9661046875	0.0338953125
5	1018561	2	0.05589358	0.94410642
6	1033078	2	0.053014663	0.946985337
7	1033078	2	0.0507343909	0.9492656091
8	1035283	2	0.05589358	0.94410642
9	1043999	2	0.0662214198	0.9337785802
10	1044572	4	0.9660008284	0.0339991716
11	1047630	4	0.9196798391	0.0803201609
12	1048672	2	0.0507343909	0.9492656091
13	1054593	4	0.9643091634	0.0356908366
14	1056784	2	0.0507343909	0.9492656091
15	1059552	2	0.05589358	0.94410642
16	1065726	4	0.7499639226	0.2500360774
17	1067444	2	0.048114632	0.951885368
18	1070935	2	0.0507343909	0.9492656091
19	1072179	4	0.9622871871	0.0377128129
20	1074610	2	0.05589358	0.94410642

Table 7. Prediction Results Table

PREDICTION ACCURACY COMPARISON

This section compares the prediction accuracy of Bayesian classifiers to that of their four popular competitor classifiers (decision tree, neural network, logistic regression, and support vector machines) for 25 data sets that were downloaded from the UCI Machine Learning Repository (Lichman 2013). Table 8 summarizes these data sets.

	Data Set	Attributes	Target Levels	Number of Observations	
				Total	Validation
1	Adult	13	2	48,842	16,116
2	Statlog (Australian Credit Approval)	14	2	690	CV-5
3	Breast Cancer Wisconsin (Original) (Mangasarian and Wolberg 1990)	9	2	699	CV-5
4	Car Evaluation	6	4	1,728	CV-5
5	Chess (King-Rook vs. King-Pawn)	36	2	3,196	1,066
6	Diabetes	8	2	768	CV-5
7	Solar Flare	10	2	1,066	CV-5
8	Statlog (German Credit Data)	24	2	1,000	CV-5
9	Glass Identification	9	6	214	CV-5
10	Heart Disease	13	2	270	CV-5
11	Hepatitis	19	2	155	CV-5
12	Iris	4	3	150	CV-5
13	LED Display Domain + 17 Irrelevant Attributes	24	10	3,190	1,057
14	Letter Recognition	16	26	20,000	4,937
15	Lymphography	18	4	148	CV-5
16	Nursery	8	5	12,960	4,319
17	Statlog (Landsat Satellite)	36	6	6,435	1,930
18	Statlog (Image Segmentation)	19	7	2,310	770
19	Soybean (Large)	35	19	683	CV-5
20	SPECT Heart	22	2	267	CV-5
21	Molecular Biology (Splice-Junction Gene Sequences)	60	3	3,190	1,053
22	Tic-Tac-Toe Endgame	9	2	958	CV-5
23	Statlog (Vehicle Silhouettes)	18	4	846	CV-5
24	Congressional Voting Records	16	2	435	CV-5
25	Waveform Database Generator (Version 1)	21	3	5,000	4,700

Table 8 Summary of 25 UCI Data Sets

For the larger data sets, the prediction accuracy was measured by the holdout method (that is, the learning process randomly selected two-thirds of the observations in the data set for building the classifiers, and then evaluated their prediction accuracy on the remaining observations in the data set). For smaller data sets, the prediction accuracy was measured by five-fold cross validation (CV-5). Each process was repeated five times. Observations that have missing values were removed from the data sets. All continuous variables in the data set were discretized with a tree-based binning method. The final average prediction accuracy values and their standard deviations are summarized in Table 9. The best accuracy values for each data set are marked in bold in each row of the table. You can see that PC and TAN in the five BN structures claim most of the wins and are competitive to the other classifiers.

Data Set	BN Classifiers					Competitor Classifiers			
	Naïve Bayes	BAN	TAN	PC	MB	Logistic	NN	Tree	SVM*
1 Adult	78.06+- 0.24	80.93+- 0.34	79.81+- 0.42	85.00+- 0.25	49.61+- 0.37	81.17+- 6.24	85.84+- 0.27	85.28+- 0.13	85.73+- 0.29
2 Statlog (Australian Credit Approval)	86.43+- 0.33	86.29+- 0.30	85.88+- 0.33	86.20+- 0.54	85.51+- 0.00	82.38+- 4.71	85.59+- 0.78	84.96+- 0.42	85.65+- 0.27
3 Breast Cancer Wisconsin (Original) (Mangasarian and Wolberg 1990)	97.42+- 0.00	97.42+- 0.00	96.65+- 0.39	97.17+- 0.12	96.88+- 0.40	95.82+- 0.57	96.54+- 0.45	94.11+- 0.40	96.42+- 0.20
4 Car Evaluation	80.01+- 0.21	86.56+- 1.03	87.52+- 0.10	88.24+- 0.90	86.52+- 1.27	77.26+- 0.26	93.07+- 0.49	96.89+- 0.36	
5 Chess (King-Rook vs. King-Pawn)	90.41+- 0.72	95.31+- 0.38	95.12+- 0.38	95.01+- 0.56	92.25+- 0.91	52.25+- 0.00	96.92+- 0.56	99.04+- 0.39	97.17+- 0.54
6 Diabetes	76.07+- 0.67	76.02+- 0.69	74.97+- 1.17	78.10+- 0.70	72.71+- 1.22	75.86+- 2.98	77.29+- 1.03	75.94+- 0.95	77.63+- 0.89
7 Solar Flare	73.58+- 0.79	73.94+- 0.92	73.60+- 0.78	80.02+- 1.08	77.60+- 1.81	81.54+- 0.22	81.69+- 0.56	81.07+- 0.45	82.18+- 0.42
8 Statlog (German Credit Data)	71.60+- 0.55	71.28+- 1.02	71.94+- 1.29	76.18+- 0.37	66.40+- 1.47	75.24+- 0.50	75.04+- 0.34	72.18+- 0.59	75.86+- 0.76
9 Glass Identification	65.61+- 2.28	65.61+- 2.28	71.68+- 1.02	69.53+- 1.42	69.53+- 1.42	62.80+- 3.70	70.37+- 3.54	69.81+- 1.43	
10 Heart Disease	82.89+- 1.21	83.56+- 1.35	82.74+- 1.07	83.33+- 0.69	80.52+- 1.19	83.26+- 2.05	84.67+- 1.30	81.41+- 1.32	84.15+- 1.66
11 Hepatitis	86.60+- 1.86	86.61+- 1.20	88.73+- 2.60	90.56+- 1.34	92.11+- 1.94	88.69+- 3.25	91.59+- 1.85	92.12+- 1.35	91.06+- 1.22
12 Iris	95.86+- 0.30	95.86+- 0.30	95.19+- 0.74	95.86+- 0.30	95.86+- 0.30	80.37+- 0.72	94.92+- 1.40	94.53+- 0.86	
13 LED Display Domain + 17 Irrelevant Attributes	73.96+- 1.22	73.96+- 1.22	74.25+- 0.88	74.27+- 1.17	74.70+- 1.21	19.79+- 0.73	73.25+- 0.39	74.08+- 0.92	
14 Letter Recognition	68.33+- 0.58	73.19+- 0.77	78.75+- 0.63	72.07+- 0.63	70.80+- 5.37	10.98+- 0.27	78.69+- 0.46	77.66+- 0.43	
15 Lymphography	80.81+- 1.56	81.49+- 1.83	79.32+- 0.77	83.78+- 1.51	74.19+- 3.71	61.62+- 3.89	81.35+- 1.56	74.86+- 0.88	
16 Nursery	82.92+- 0.65	86.46+- 0.69	89.25+- 0.39	91.45+- 0.63	91.02+- 0.25	90.86+- 0.34	92.27+- 0.47	97.41+- 0.16	
17 Statlog (Landsat Satellite)	81.39+- 0.73	86.36+- 0.51	86.31+- 0.79	86.58+- 0.49	84.56+- 0.65	72.78+- 0.29	87.84+- 0.60	85.55+- 0.38	
18 Statlog (Image Segmentation)	89.45+- 0.71	91.09+- 1.71	93.04+- 0.81	91.09+- 1.71	67.01+- 2.34	58.83+- 3.24	92.78+- 0.90	93.56+- 0.74	
19 Soybean (Large)	89.78+- 0.35	89.78+- 0.35	92.97+- 0.99	89.43+- 0.44	60.97+- 2.80	44.22+- 3.67	91.80+- 0.51	91.65+- 1.01	
20 SPECT Heart	72.06+- 1.65	75.36+- 1.04	73.41+- 1.38	80.60+- 1.25	69.96+- 2.74	78.35+- 1.66	82.25+- 1.20	79.33+- 1.51	81.95+- 1.97
21 Molecular Biology (Splice-Junction Gene Sequences)	95.31+- 0.51	95.38+- 0.47	95.71+- 0.71	96.05+- 0.16	92.61+- 7.13	80.46+- 1.61	95.48+- 0.70	94.17+- 0.62	
22 Tic-Tac-Toe Endgame	66.08+- 1.49	79.04+- 1.58	72.03+- 0.70	77.14+- 0.82	75.03+- 3.02	77.10+- 0.80	98.10+- 0.09	93.28+- 0.67	98.33+- 0.00
23 Statlog (Vehicle Silhouettes)	62.01+- 0.84	70.26+- 1.29	71.25+- 0.80	70.26+- 1.39	58.96+- 5.60	63.55+- 1.77	70.09+- 0.91	69.36+- 0.48	
24 Congressional Voting Records	94.80+- 0.53	95.17+- 0.16	95.13+- 0.72	94.90+- 0.10	94.99+- 0.38	93.79+- 2.11	95.82+- 0.99	95.08+- 0.42	95.40+- 0.43
25 Waveform Database Generator(Version 1)	78.31+- 1.48	78.31+- 1.48	73.68+- 1.77	78.35+- 1.33	78.62+- 1.50	62.43+- 3.43	81.78+- 0.85	70.27+- 3.06	

*SVM for binary target only

Table 9. Classification Accuracy on 25 UCI Machine Learning Data Sets

CONCLUSION

This paper describes Bayesian network (BN) classifiers, introduces the HPBNET procedure, and shows how you can use the procedure to build BN classifiers. It also compares the competitive prediction power of BN classifiers with other state-of-the-art classifiers, and shows how you can use a SAS macro to visualize the network structures.

REFERENCES

- Lichman, M. 2013. "UCI Machine Learning Repository." School of Information and Computer Sciences, University of California, Irvine. <http://archive.ics.uci.edu/ml>.
- Russell, S., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, New Jersey: Pearson.
- Mangasarian, O. L., and Wolberg, W. H. 1990. "Cancer Diagnosis via Linear Programming", *SIAM News*, 23 (September): 1, 18.

ACKNOWLEDGMENTS

The authors would like to thank Yongqiao Xiao for reviewing this paper and Anne Baxter for editing it.

The breast cancer database was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg.

The lymphography domain was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Thanks go to M. Zwitter and M. Soklic for providing the data.

The Vehicle Silhouettes data set comes from the Turing Institute, Glasgow, Scotland.

The creators of heart disease data set are:

1. Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
2. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
3. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
4. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation: Robert Detrano, M.D., Ph.D.

CONTACT INFORMATION

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

Ye Liu
ye.liu@sas.com

Weihua Shi
weihua.shi@sas.com

Wendy Czika
wendy.czika@sas.com

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.

APPENDIX

```
%macro createBNCdiagram(target=Class, outnetwork=net);

  data outstruct;
    set &outnetwork;
    if strip(upcase(_TYPE_)) eq 'STRUCTURE' then output;
    keep _nodeid_ _childnode_ _parentnode_;
  run;

  data networklink;
    set outstruct;
    linkid = _N_;
    label linkid ="Link ID";
  run;

  proc sql;
    create table work._node1 as
      select distinct _CHILDNODE_ as node
      from networklink;
    create table work._node2 as
      select distinct _PARENTNODE_ as node
      from networklink;
  quit;

  proc sql;
    create table work._node as
      select node
      from work._node1
      UNION
      select node
      from work._node2;
  quit;

  data bnc_networknode;
    length NodeType $32.;
    set work._node;
    if strip(upcase(node)) eq strip(upcase("&target")) then do;
      NodeType = "TARGET";
      NodeColor=2;
    end;
    else do;
      NodeType = "INPUT";
      NodeColor = 1;
    end;
    label NodeType ="Node Type" ;
    label NodeColor ="Node Color" ;

  run;

  data parents(rename=(_parentnode_ = _node_)) children(rename=(_childnode_
= _node_)) links;
    length _parentnode_ _childnode_ $ 32;
    set networklink;
    keep _parentnode_ _childnode_ ;
  run;
```

```

/*get list of all unique nodes*/
data nodes;
    set parents children;
run;

proc sort data=nodes;
    by _node_;
run;

data nodes;
    set nodes;
    by _node_;
    if first._node_;
    _Parentnode_ = _node_;
    _childnode_ = "";
run;

/*merge node color and type */
data nodes;
    merge nodes bnc_
networknode (rename=(node=_node_ nodeColor=_nodeColor_
nodeType=_nodeType_));
    by _node_;
run;

/*sort color values to ensure consistent color mapping across networks */
/*note that the color mapping is HTML style dependent though */
proc sort data=nodes;
    by _nodeType_;
run;

/*combine nodes and links*/
/* need outsummaryall for model report*/
data bnc_networksummary(drop=_shape_ _nodecolor_ _nodepriority_ _shape_
_nodeID_ _nodetype_ _linkdirection_) bnc_networksummaryall;
length _parentnode_ _childnode_ $ 32;
set nodes links;
drop _node_;
if _childnode_ EQ "" thendo;
    _nodeID_ = _parentnode_;
    _nodepriority_ = 1;
    _shape_ = "OVAL";
end;
else do;
    _linkdirection_ = "TO";
    output bnc_networksummary;
end;
output bnc_networksummaryall;
label _linkdirection_="Link Direction";
run;

proc datasets lib=work nolist nowarn;
    delete _node_ _node1_ _node2_ nodes links parents children;
run;

quit;

```

```

proc template;
  define statgraph bpath;
    begingraph / DesignHeight=720 DesignWidth=720;
      entrytitle "Bayesian Network Diagram";
      layout region;
        pathdiagram fromid=_parentnode_ toid=_childnode_ /
          arrangement=GRIP
          nodeid=_nodeid_
          nodelabel=_nodeID_
          nodeshape=_shape_
          nodepriority=_nodepriority_
          linkdirection=_linkdirection_
          nodeColorGroup=_NodeColor
          textSizeMin = 10
        ;
      endlayout;
    endgraph;
  end;
run;

ods graphics;
proc sgrender data=bnc_networksummaryall template=bpath;
run;

%mend;

%createBNCdiagram;

```

Methods of Multinomial Classification Using Support Vector Machines

Ralph Abbey, Taiping He, and Tao Wang, SAS® Institute Inc.

ABSTRACT

Many practitioners of machine learning are familiar with support vector machines (SVMs) for solving binary classification problems. Two established methods of using SVMs in multinomial classification are the one-versus-all approach and the one-versus-one approach. This paper describes how to use SAS® software to implement these two methods of multinomial classification, with emphasis on both training the model and scoring new data. A variety of data sets are used to illustrate the pros and cons of each method.

INTRODUCTION

The support vector machine (SVM) algorithm is a popular binary classification technique used in the fields of machine learning, data mining, and predictive analytics. Since the introduction of the SVM algorithm in 1995 (Cortes and Vapnik 1995), researchers and practitioners in these fields have shown significant interest in using and improving SVMs.

Support vector machines are supervised learning models that provide a mapping between the feature space and the target labels. The aim of supervised learning is to determine how to classify new or previously unseen data by using labeled training data. Specifically, SVMs are used to solve binary classification problems, in which the target has only one of two possible classes.

The SVM algorithm builds a binary classifier by solving a convex optimization problem during model training. The optimization problem is to find the flat surface (hyperplane) that maximizes the margin between the two classes of the target. SVMs are also known as maximum-margin classifiers, and the training data near the hyperplane are called the support vectors. Thus, the result of training is the support vectors and the weights that are given to them. When new data are to be scored, the support vectors and their weights are used in combination with the new data to assign the new data to one of the two classes.

Many real-world classification problems have more than two target classes. There are several methods in the literature (Hsu and Lin 2002), such as the one-versus-all and one-versus-one methods, that extend the SVM binary classifier to solve multinomial classification problems.

This paper shows how you can use the HPSVM procedure from SAS® Enterprise Miner™ to implement both training and scoring of these multinomial classification extensions to the traditional SVM algorithm. It also demonstrates these implementations on several data sets to illustrate the benefits of these methods.

The paper has three main sections: training, scoring, and experiments, followed by the conclusions. The training section describes how to set up the multinomial SVM training schema. The scoring section discusses how to score new data after you have trained a multinomial SVM. The experiments section illustrates some examples by using real-world data. Finally, the appendices present the SAS macro code that is used to run the experiments.

SUPPORT VECTOR MACHINE TRAINING

Support vector machines (SVMs) are a binary classifier that seeks to find the flat surface (a straight line in two dimensions) that separates the two levels of the target. Figure 1 shows an example of a binary classification problem and the SVM decision surface. In this example, the support vectors consist of the two triangular observations that touch one of the dotted lines and the one hexagonal observation that touches the other dotted line. The dotted lines represent the margin, which indicates the maximum separation between the two classes in the data set.

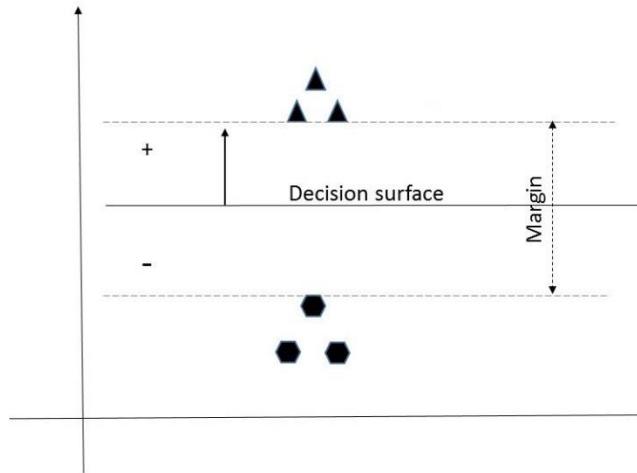


Figure 1. Support Vector Machine Decision Surface and Margin

SVMs also support decision surfaces that are not hyperplanes by using a method called the kernel trick. For the purposes of the examples in this section and the “Support Vector Machine Scoring” section, this paper is limited to referencing only linear SVM models. These sections equally apply to nonlinear SVMs as well. Figure 2 shows an example of two classes that are separated by a nonlinear SVM decision boundary.

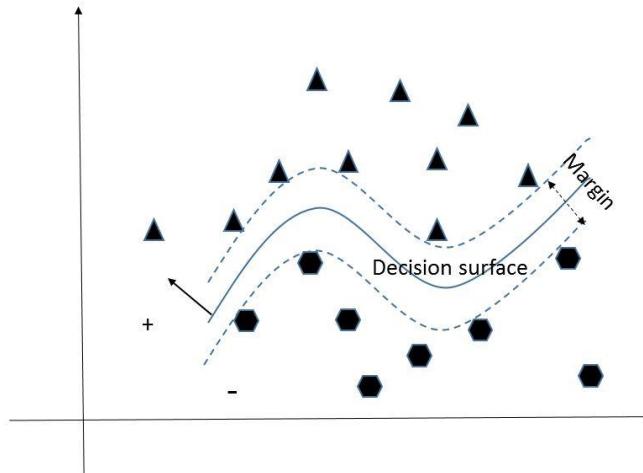


Figure 2. Nonlinear Support Vector Machine Decision Surface and Margin

Multiclass SVMs are used to find the separation when the target has more than two classes. Figure 3 shows an example of a three-class classification problem. Here the classes are triangle, diamond, and hexagon.

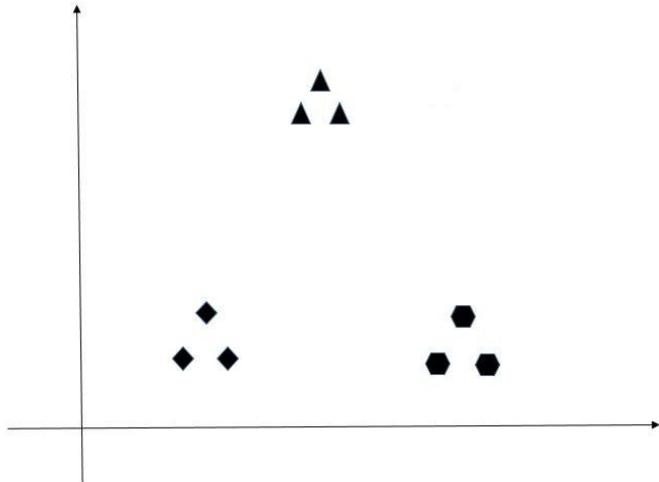


Figure 3. Example of Three Classes: Triangles, Diamonds, and Hexagons

When the data contain more than two classes, a single flat surface cannot separate each group from the others. However, several surfaces can partition the observations from each other. How you find the surfaces depends on your approach in the multiclass SVM: one-versus-all or one-versus-one.

When you are using the HPSVM procedure to solve multinomial classification problems, you first need to create a dummy variable for each class of the target variable. The dummy variable for a particular class is defined to be either 0 when an observation is not of that class or 1 when an observation is of that class. Code to create the dummy variables is presented in the SAS_SVM_ONE_VS_ALL_TRAIN and SAS_SVM_ONE_VS_ONE_TRAIN macros in Appendix B. An example that uses the data in Figure 3 might look something like this:

```

data ModifiedInput;
set Input;
  if (class = "Triangle") then do;
    class_triangle = 1;
  end;
  else do;
    class_triangle = 0;
  end;
  if (class = "Diamond") then do;
    class_diamond = 1;
  end;
  else do;
    class_diamond = 0;
  end;
  if (class = "Hexagon") then do;
    class_hexagon = 1;
  end;
  else do;
    class_hexagon = 0;
  end;
run;

```

When the input data have the dummy variables, the data are ready for you to train using the one-versus-all or one-versus-one method.

One-versus-All Training

The one-versus-all approach to multiclass SVMs is to train k unique SVMs, where you have k classes of the target. Figure 4, Figure 5, and Figure 6 show the three one-versus-all scenarios for training a multiclass SVM on the example from Figure 3.

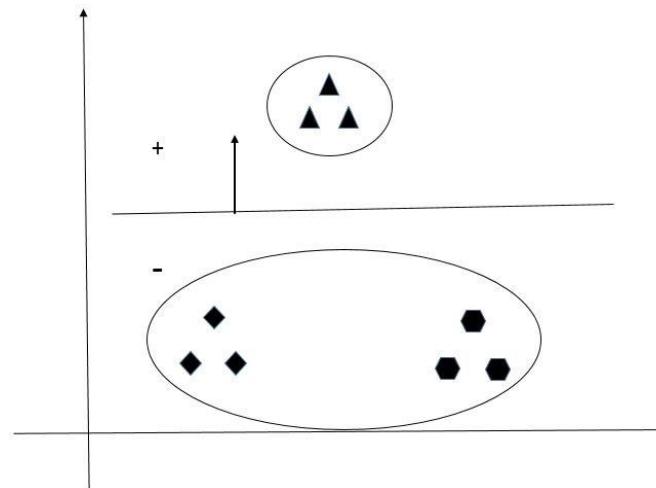


Figure 4. One-versus-All Training: Triangles versus All

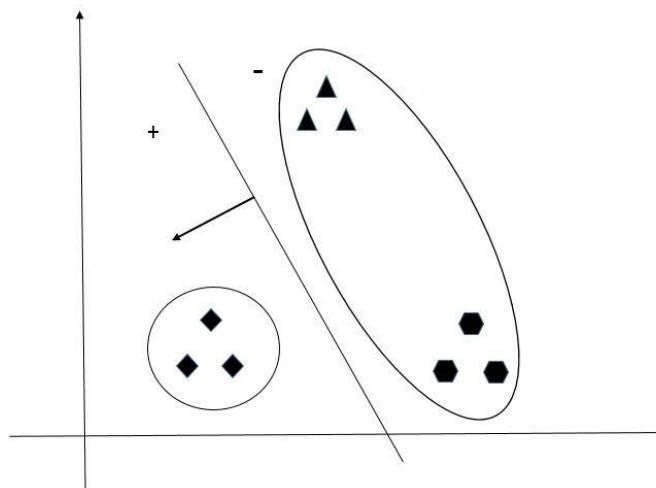


Figure 5. One-versus-All Training: Diamonds versus All

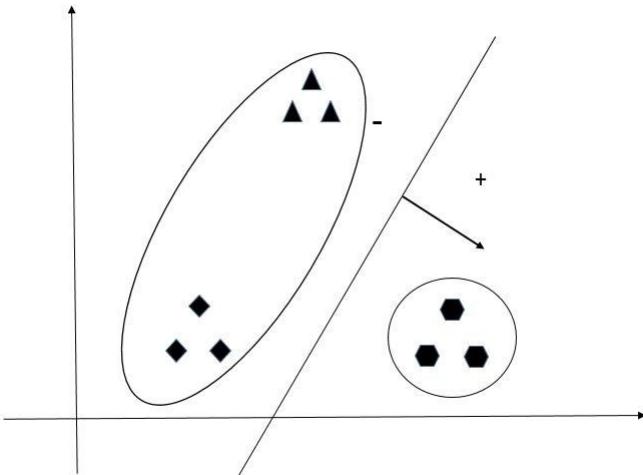


Figure 6. One-versus-All Training: Hexagons versus All

To set up the one-versus-all training by using the HPSVM procedure, you first need to add dummy variables to the input data set, as described previously. The dummy variable that corresponds to Figure 4 has a 1 for each triangular observation and a 0 for each diamond-shaped or hexagonal observation. The HPSVM procedure code for Figure 4 might look like this:

```
proc hpsvm data=ModifiedInput (where=(class=triangle OR class=hexagon));
  input <input variables>;
  target class_triangle;
run;
```

You also need to save the procedure score code by using the CODE statement. This enables you to score new observations based on the training that you have already completed. For the one-versus-all method of multinomial SVM training, you need to run the HPSVM procedure k times, and each run will have a different dummy variable as the target variable for the SVM. The output that you need for scoring is k different DATA step score code files. You can find a discussion of scoring the one-versus-all method in the section “One-versus-All Scoring.”

One-versus-One Training

The one-versus-one approach to multiclass SVMs is to train an SVM for each pair of target classes. When you have k classes, the number of SVMs to be trained is $k*(k-1)/2$. Figure 7, Figure 8, and Figure 9 show the three one-versus-one scenarios for training a multiclass SVM on the example from Figure 3.

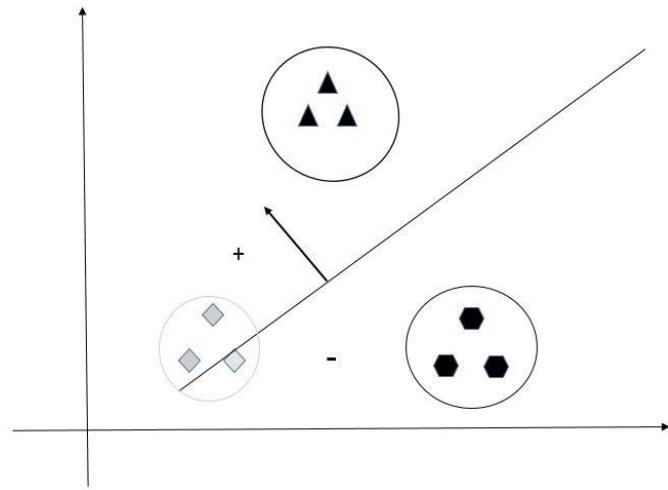


Figure 7. One-versus-One Training: Triangles versus Hexagons

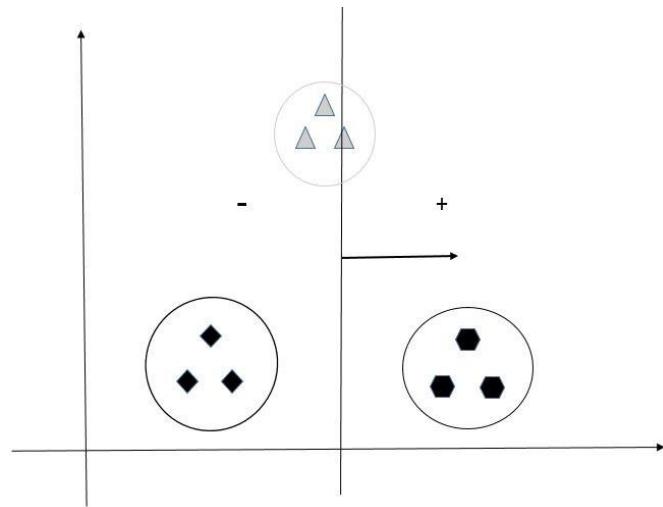


Figure 8. One-versus-One Training: Hexagons versus Diamonds

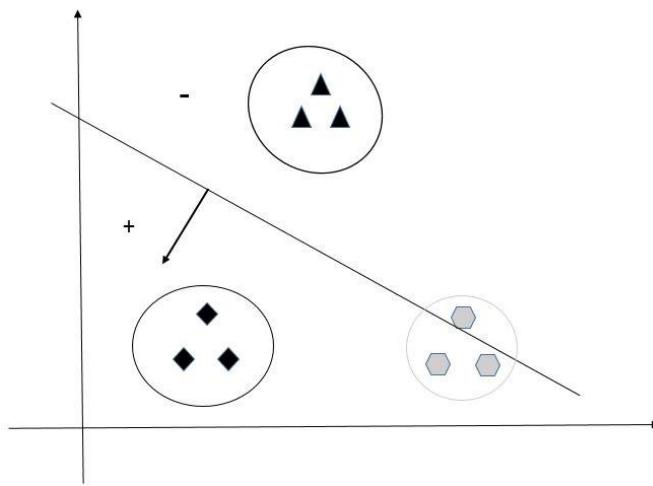


Figure 9. One-versus-One Training: Diamonds versus Triangles

As these three figures show, when you are using the one-versus-one training method of multinomial classification, you ignore any of the data that are not in the current comparison. For this example, you have three comparisons: triangular versus hexagonal observations, hexagonal versus diamond-shaped observations, and diamond-shaped versus triangular observations. In each of these cases, the third class is ignored when you create the SVM model.

To perform this method by using the HPSVM procedure, you first need to create the dummy variables as previously indicated. To ensure that you compare only the proper observations, you also need to subset the input data by using the WHERE= option. An example of the code for triangular versus hexagonal observations might look like this:

```
proc hpsvm data=ModifiedInput (where=(class=triangle OR class=hexagon) ;
  input <input variables>;
  target class_triangle;
run;
```

As in the one-versus-all method, you need to save the procedure score code by using the CODE statement. This enables you to score new observations based on the training that you have already completed. For the one-versus-one method of multinomial SVM training, you need to run PROC HPSVM $k*(k-1)/2$ times. Each run consists of a different pair of target classes that are compared. The output that you need for scoring is $k*(k-1)/2$ different DATA step score code files. There are two ways to score the one-versus-one training; they are detailed in the sections “One-versus-One Scoring” and “Directed Acyclic Graph Scoring.”

SUPPORT VECTOR MACHINE SCORING

Scoring by using SVMs is the process of using a trained model to assign a class label to a new observation. In the case of the HPSVM procedure, the DATA step score code contains the information from the SVM model and enables you to score new observations. A new example observation, star, has been added to the previous example to illustrate scoring. This is shown in Figure 10.

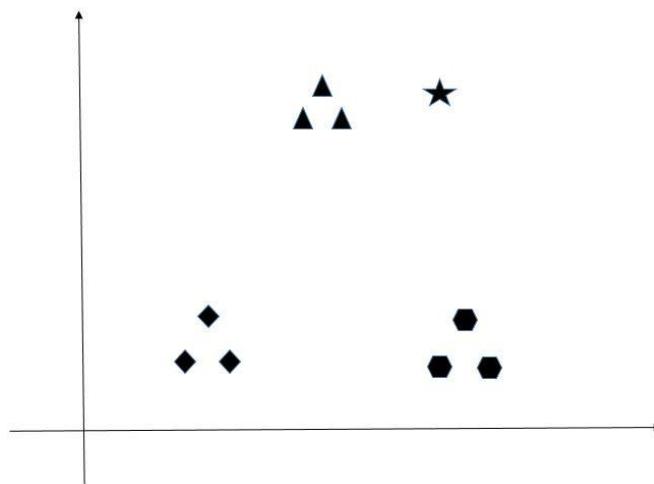


Figure 10. Example Data, with a New Observation (Star) to Be Scored

One-versus-All Scoring

The output from the one-versus-all scoring is k DATA step score code files, one for each class of the multinomial target. When you are determining the class of a new data observation, you need to score the observation by using each saved score code.

To assign a class label to a new observation, you need to score the observation scored according to each SVM model. In this way, the new observation will have an assigned probability for each class of the

target. If the observation is on the negative side of the dividing hyperplane, then its probability is less than 0.5. If it is on the positive side of the dividing hyperplane, then its probability is greater than 0.5.

In this example, the hyperplanes shown in Figure 4, Figure 5, and Figure 6 illustrate that the star point will have the highest probability of assignment to the triangular class.

When you are using the PROC HPSVM score code for each class of the target, new data are assigned a probability that the observation is of that target class. To determine which target class is the correct label, you choose the one that has the highest probability. SAS macro code to do this is presented in the SAS_SVM_ONE_VS_ALL_SCORING macro in Appendix B.

One-versus-One Scoring

The output from one-versus-one scoring is $k*(k-1)/2$ DATA step score code files, one for each pairwise comparison of target classes of the multinomial target. When you are determining the class label of a new data observation, you need to score the observation by using each saved score code.

In one-versus-all scoring, each SVM model answers the question, Does this belong to the class or not? In one-versus-one scoring, each SVM model answers a different question: Is this more of class A or class B? Thus, using the maximum probability, as in one-versus-all scoring, is not the appropriate way to determine the class label assignment.

In one-versus-one scoring, a common method of determining this assignment is by voting. Each observation is assigned a class label for each SVM model that is produced. The label that the observation is assigned the most is considered the true label.

When you are using PROC HPSVM, use the score code to score the new data for each one-versus-one SVM model. Then, for each class of the multinomial target, check to see whether that class has the most votes. If it does, then assign that class as the label for the target. When you have a tie, you can assign the class randomly, or as shown in this paper, you can assign the class by using the first class in the sorted order. SAS macro code to perform one-versus-one scoring is presented in the SAS_SVM_ONE_VS_ONE_SCORING macro in Appendix B.

Directed Acyclic Graph Scoring

The directed acyclic graph (DAG), which was first presented in Platt, Cristianini, and Shawe-Taylor (2000), is a scoring approach that uses the same training as the one-versus-one scoring method. In this case, each observation is scored only $k-1$ times, even though $k*(k-1)/2$ SVM models are trained. The training scheme for a four-class example is shown in Figure 11. In this illustration, a new observation starts at the top of the graph and is scored using the 1 vs. 4 SVM model. Then, depending on the outcome, the observation traverses the graph until it reaches one of the four class labels.

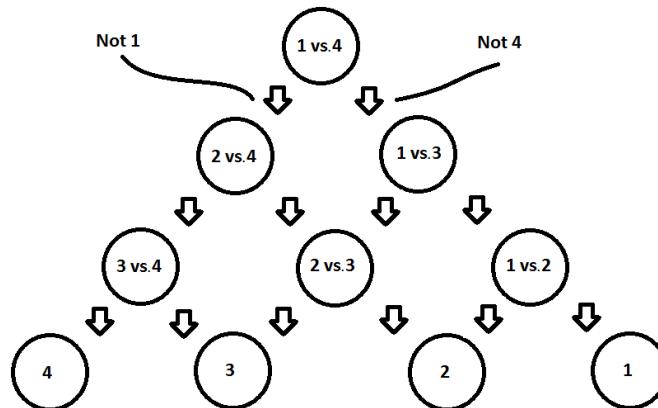


Figure 11. Directed Acyclic Graph Scoring Flow

The DAG method first runs the scoring from the one-versus-one SVM model that compared the first and last classes of the target (the order should be fixed, but the order does not matter). If the SVM model

assigns the observation to the first class, then the last class can be ruled out. Thus the DAG method seeks to recursively exclude possible target class labels until only one label is left. That label becomes the class label for the new observation.

Each one-versus-one model is represented in the DAG. However, the number of times the observation is scored is only $k-1$, because as it is scored, it flows down the graph.

When you are using PROC HPSVM score code to run the DAG method, you need all the score code files from the one-versus-one training. To score recursively, you need to create two output data sets from each input set, in which you assign each observation to one of the two output data sets based on the predicted target class from the SVM model. SAS macro code to perform DAG scoring is presented in the SAS_SVM_DAG_SCORING macro in Appendix B.

EXPERIMENTS

This section presents a few brief examples that were run using the setup code in Appendix A and the macro code in Appendix B. All the runs of the HPSVM procedure use the procedure defaults, except that the kernel is chosen to be polynomial with degree 2 instead of linear, which is the default.

Table 1 lists the data sets that are used in the experiments. Many of these data sets are available in SAS Enterprise Miner. The **Wine** data set is from the UCI Machine Learning Repository (Lichman 2013).

Simulated data were created to run experiments with larger numbers of observations, input variables, and target classes. The target variable in the simulated data has approximately equal class sizes among the seven classes. In addition, only 9 of the 27 variables are correlated with the target levels, but these correlated variable also have large amounts of randomness.

The HPSVM procedure supports multithreading on a single machine as well as distributed computation. These experiments were run using a single desktop machine. Absolute times vary with hardware and setup, but the relative times provide important insight into how the different methods of multinomial classification compare with each other.

Data Set	Number of Observations	Number of Input Variables	Target Variable	Number of Target Classes	Location
Iris	150	4	Species	3	SASHELP.IRIS
Wine	178	13	Cultivar	3	UCI ML Repository
Cars	428	12	Type	6	SASHELP.CARS
German Credit	1000	20	employed	5	SAMPSIO.DMAGECR
Simulated 10K	10000	27	t	7	Simulated data

Table 1. Data Sets Used in the Experiments, along with Table Metadata

Table 2 shows the training and scoring times for each method on each data set. One-versus-one training is used for both one-versus-one scoring and DAG scoring. When the number of target classes is larger, such as in the **Cars** data set or the simulated data, the one-versus-one training requires more time to complete than the one-versus-all training. This is because there are $k*(k-1)/2$ models that require training for the one-versus-one method, compared to only k models for the one-versus-all method. The number of models that are trained is slightly offset by the fact that each of the models trained in the one-versus-one method uses fewer data than the models trained in the one-versus-all method, but as the number of target classes increases, the one-versus-one method takes more time.

Data Set	Training (sec)		Scoring (sec)		
	One-versus-All Method	One-versus-One Method	One-versus-All Method	One-versus-One Method	DAG Method
Iris	1	1	1	1	1
Wine	1	1	< 1	1	< 1
Cars	3	4	1	3	2
German Credit	11	7	1	3	2
Simulated 10K	67	76	2	7	5

Table 2. Timing for Running the Two Training and Three Scoring Methods on the Data Sets

Table 3 shows the misclassification rate for each data set and each scoring method. For each data set, the one-versus-one method has the best classification rate, followed very closely by the DAG method's classification rate. The one-versus-all method's classification rate is lower than that of the one-versus-one and DAG methods, especially on the larger data sets.

Data Set	One-versus-All Classification Rate (%)	One-versus-One Classification Rate (%)	DAG Classification Rate (%)
Iris	96.00	96.67	96.67
Wine	100	100	100
Cars	84.81	87.38	87.38
German Credit	72.5	76.6	76.3
Simulated 10K	70.07	78.06	78.04

Table 3. Classification Rate for Running the Three Different Scoring Methods on the Data Sets

CONCLUSION

This paper explains how to extend the HPSVM procedure for scoring multinomial targets. Two approaches to extending the SVM training are the one-versus-all and one-versus-one methods. When you are scoring the SVM model, you also have the option to use directed acyclic graphs (DAGs) to score the one-versus-one trained models.

The paper applies one-versus-all and one-versus-one training to several data sets to illustrate the strengths and weaknesses of the methods. The one-versus-one method does better at classifying observations than the one-versus-all method. This benefit is balanced by the fact that as the number of target classes increases, one-versus-one training takes longer than one-versus-all training. The scoring times are also longer for the one-versus-one and DAG methods than for the one-versus-all method. The DAG method runs faster than one-versus-one scoring, with only a marginal decrease in accuracy.

The paper also presents SAS macro code to perform the various multinomial classifications.

APPENDIX A

Before running the SAS macro code in Appendix B, you need to run the setup information. The following example does this for the **Iris** data set:

```
*** the training macros create several SAS data sets and some files;
*** to ensure that nothing is overwritten, create a new directory;
*** or point to an existing empty directory;
*** set the output directory below;
%let OutputDir = U:\SGF2017\; *change as needed;
x cd "&OutputDir";
libname l "&OutputDir";

*** set the target variable;
*** also set the input and score data sets;
*** you can change the score data every time you want to score new data;
%let Target      = Species; *case-sensitive;
%let InputData   = sashelp.iris;
%let ScoreData   = sashelp.iris;
proc contents data =&InputData out=names (keep = name type length);
run;
data names;
  set names;
  if name = "&Target" then do;
    call symput("TargetLength", length);
    delete;
  end;
run;
*** manually add names to interval or nominal type;
*** id variables are saved from the input data to the scored output data;
%let ID          = PetalLength PetalWidth SepalLength SepalWidth;
%let INPUT_NOM   = ;
%let INPUT_INT   = PetalLength PetalWidth SepalLength SepalWidth;
%let ID_NUM      = 4;
%let INPUT_NOM_NUM = 0;
%let INPUT_INT_NUM = 4;
*** PROC HPSVM options for the user (optional);
%let Maxiter     = 25;
%let Tolerance   = 0.000001;
%let C           = 1;
```

APPENDIX B

The following macros include the one-versus-all training, one-versus-one training, one-versus-all scoring, one-versus-one scoring, and DAG scoring macros. The dummy variable creation is included in the one-versus-all and one-versus-one training macros and has been commented.

```
%macro SAS_SVM_ONE_VS_ALL_TRAIN();
*** separate the target for information-gathering purposes;
data l.TargetOnly;
  set &InputData;
  keep &Target;
  if MISSING(&Target) then delete;
run;
proc contents data = l.TargetOnly out=l.TType(keep = type);
run;
data _NULL_;
  set l.TType;
```

```

call symput("TargetType", type);
run;
*** get the number of levels of the target;
proc freq data=l.TargetOnly nlevels;
ods output nlevels=l.TargetNLevels OneWayFreqs=l.TargetLevels;
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
  set l.TargetNLevels;
  call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
  set l.TargetLevels;
  i = _N_;
  call symput("level'||left(trim(i)), trim(left(right(&Target.))));;
run;
*** create a column for each level of the target;
*** the value of the column is 1 if the target is that level, 0 otherwise;
data l.ModifiedInput;
  set &InputData;
  _MY_ID_ = _N_;
  %do i=1 %to &n;
    %if (&TargetType = 1) %then %do;
      if MISSING(&Target) then do;
        &Target.&&level&i = .;
      end;
      else if (&Target = &&level&i) then do;
        &Target.&&level&i = 1;
      end;
      else do;
        &Target.&&level&i = 0;
      end;
    %end;
    %else %if (&TargetType = 2) %then %do;
      if MISSING(&Target) then do;
        &Target.&&level&i = .;
      end;
      else if (&Target = "&&level&i") then do;
        &Target.&&level&i = 1;
      end;
      else do;
        &Target.&&level&i = 0;
      end;
    %end;
  %end;
run;
*** run an svm for each target. also save the scoring code for each svm;
%do i=1 %to &n;
  %let Target&i = &Target.&&level&i;

  data _NULL_;
    length svmcode $2000;
    svmcode = "&OutputDir"!!"svmcode"!!"&i"!!".sas";
    call symput("svmcode"||left(trim(&i)), trim(svmcode));
run;

```

```

proc hpsvm data = l.ModifiedInput tolerance = &Tolerance c = &C
maxiter = &Maxiter nomiss;
    target &&Target&i;
    %if &INPUT_INT_NUM > 0 %then %do;
        input &INPUT_INT / level = interval;
    %end;
    %if &INPUT_NOM_NUM > 0 %then %do;
        input &INPUT_NOM / level = nominal;
    %end;
    *kernel linear;
    kernel polynomial / degree = 2;
    id _MY_ID_ &Target;
    code file = "&&svmcode&i";
run;
%end;
*** this table lists all the svm scoring files;
data l.CodeInfoTable;
    length code $2000;
    %do i=1 %to &n;
        code = "&&svmcode&i";
        output;
    %end;
run;
%mend SAS_SVM_ONE_VS_ALL_TRAIN;

%macro SAS_SVM_ONE_VS_ONE_TRAIN();
*** separate the target for information-gathering purposes;
data l.TargetOnly;
    set &InputData;
    keep &Target;
    if MISSING(&Target) then delete;
run;
proc contents data = l.TargetOnly out=l.TType(keep = type);
run;
data _NULL_;
    set l.TType;
    call symput("TargetType", type);
run;
*** get the number of levels of the target;
proc freq data=l.TargetOnly nlevels;
    ods output nlevels=l.TargetNLevels OneWayFreqs=l.TargetLevels;
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
    set l.TargetNLevels;
    call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
    set l.TargetLevels;
    i = _N_;
    call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** create a column for each level of the target;
*** the value of the column is 1 if the target is that level, 0 otherwise;
data l.ModifiedInput;

```

```

set &InputData;
_MY_ID_ = _N_;
%do i=1 %to &n;
  %if (&TargetType = 1) %then %do;
    if MISSING(&Target) then do;
      &Target.&&level&i = .;
    end;
  else if (&Target = &&level&i) then do;
    &Target.&&level&i = 1;
  end;
  else do;
    &Target.&&level&i = 0;
  end;
%end;
%else %if (&TargetType = 2) %then %do;
  if MISSING(&Target) then do;
    &Target.&&level&i = .;
  end;
  else if (&Target = "&&level&i") then do;
    &Target.&&level&i = 1;
  end;
  else do;
    &Target.&&level&i = 0;
  end;
%end;
%end;
run;
*** run an svm for each target. also save the scoring code for each svm;
%do i=1 %to &n;
  %do j=%eval(&i+1) %to &n;
    %let Target&i = &Target.&&level&i;
    %let Target&j = &Target.&&level&j;

    data _NULL_;
      length svmcode $2000;
      svmcode = "&OutputDir"!!"svmcode"!!"&i"!!"_!!"&j"!!".sas";
      call symput("svmcode&i._"||trim(left(&j)), trim(svmcode));
    run;

    proc hpsvm data = l.ModifiedInput(where=(&&Target&i=1 OR
&&Target&j=1)) tolerance = &Tolerance c = &C maxiter = &Maxiter nomiss;
      target &&Target&i;
      %if &INPUT_INT_NUM > 0 %then %do;
        input &INPUT_INT / level = interval;
      %end;
      %if &INPUT_NOM_NUM > 0 %then %do;
        input &INPUT_NOM / level = nominal;
      %end;
      *kernel linear;
      kernel polynomial / degree = 2;
      id _MY_ID_ &Target;
      code file = "&&svmcode&i._&j";
    run;
  %end;
%end;
*** this table lists all the svm scoring files;
data l.CodeInfoTable;

```

```

length code $2000;
%do i=1 %to &n;
  %do j=%eval(&i+1) %to &n;
    code = "&&svmcode&i._&j";
    output;
  %end;
%end;
run;
%mend SAS_SVM_ONE_VS_ONE_TRAIN;

%macro SAS_SVM_ONE_VS_ALL_SCORE();
*** record the target type: 1 = numeric, 2 = character;
data _NULL_;
  set l.TType;
  call symput("TargetType", type);
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
  set l.TargetNLevels;
  call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
  set l.TargetLevels;
  i = _N_;
  call symput("level"||left(trim(i)), trim(left(right(&Target))));
run;
*** read the code info table and create macro variables for each code
file;
data _NULL_;
  set l.CodeInfoTable;
  i = _N_;
  call symput("svmcode"||left(trim(i)), trim(left(right(code))));
run;
%do i=1 %to &n;
  %let Target&i = &Target.&&level&i;
%end;
*** score the data by using each score code;
%MakeScoredOneVsAll();
%mend SAS_SVM_ONE_VS_ALL_SCORE;

%macro MakeScoredOneVsAll();
data l.ScoredOutput;
  set &ScoreData;
  %if (&TargetType = 2) %then %do;
    length I_&Target $ &TargetLength;
  %end;
  %do i=1 %to &n;
    %inc "&&svmcode&i";
  %end;
  keep
  %do i=1 %to &n;
    P_&&Target&i..1
  %end;
  %if (&ID_NUM > 0) %then %do;
    &ID

```

```

%end;
I_&Target &Target;
_P_ = 0;
%do i=1 %to &n;
  %if (&TargetType = 1) %then %do;
    if (P_&&Target&i..1 > _P_) then do;
      _P_ = P_&&Target&i..1;
      I_&Target = &&level&i;
    end;
  %end;
  %else %if (&TargetType = 2) %then %do;
    if (P_&&Target&i..1 > _P_) then do;
      _P_ = P_&&Target&i..1;
      I_&Target = "&&level&i";
    end;
  %end;
%end;
run;
%mend MakeScoredOneVsAll;

%macro SAS_SVM_ONE_VS_ONE_SCORE();
*** record the target type: 1 = numeric, 2 = character;
data _NULL_;
  set l.TType;
  call symput("TargetType", type);
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
  set l.TargetNLevels;
  call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
  set l.TargetLevels;
  i = _N_;
  call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** read the code info table and create macro variables for each code
file;
data _NULL_;
  set l.CodeInfoTable;
  i = _N_;
  call symput("svmcode"||left(trim(i)), trim(left(right(code))));
  call symput("numCode", i);
run;
%let k=1;
%do i=1 %to &n;
  %do j=%eval(&i+1) %to &n;
    %let svmcode&i._&j =&&svmcode&k;
    %let k =%eval(&k+1);
  %end;
%end;
%do i=1 %to &n;
  %let Target&i = &Target.&&level&i;
%end;
*** score the data by using each score code;

```

```

%MakeScoredOneVsOne();
%mend SAS_SVM_ONE_VS_ONE_SCORE;

%macro MakeScoredOneVsOne();
data l.ScoredOutput;
  set &ScoreData;
  %do k=1 %to &n;
    V_&&level&k = 0;
  %end;
  %if (&TargetType = 2) %then %do;
    length I_&Target $ &TargetLength;
  %end;
  %else %do;
    length I_&Target 8;
  %end;
run;
%do i=1 %to &n;
  %do j=%eval(&i+1) %to &n;
    data l.ScoredOutput;
      set l.ScoredOutput;
      %inc "&&svmcode&i._&j";
      if (P_&Target&&level&i..1 >= 0.5) then do;
        V_&&level&i = V_&&level&i+1;
      end;
      else do;
        V_&&level&j = V_&&level&j+1;
      end;
      P_ = 0;
      %if (&TargetType = 1) %then %do;
        %do k=1 %to &n;
          if (V_&&level&k > P_) then do;
            P_ = V_&&level&k;
            I_&Target = &&level&k;
          end;
        %end;
      %end;
      %else %if (&TargetType = 2) %then %do;
        %do k=1 %to &n;
          if (V_&&level&k > P_) then do;
            P_ = V_&&level&k;
            I_&Target = "&&level&k";
          end;
        %end;
      %end;
    %end;
    drop P_&Target&&level&i..1 P_&Target&&level&i..0
    I_&Target&&level&i _P_;
    run;
  %end;
%end;
data l.ScoredOutput;
  set l.ScoredOutput;
  keep
    %do i=1 %to &n;
      V_&&level&i
    %end;
    %if (&ID_NUM > 0) %then %do;

```

```

        &ID
      %end;
      I_&Target &Target;
run;
%mend MakeScoredOneVsOne;

%macro SAS_SVM_DAG_SCORE();
*** record the target type: 1 = numeric, 2 = character;
data _NULL_;
  set l.TType;
  call symput("TargetType", type);
run;
*** create a variable, n, that is the number of levels of the target;
data _NULL_;
  set l.TargetNLevels;
  call symput("n", left(trim(nlevels)));
run;
*** create macro variables for each level of the target;
data _NULL_;
  set l.TargetLevels;
  i = _N_;
  call symput("level"||left(trim(i)), trim(left(right(&Target.))));
run;
*** read the code info table and create macro variables for each code
file;
data _NULL_;
  set l.CodeInfoTable;
  i = _N_;
  call symput("svmcode"||left(trim(i)), trim(left(right(code))));
  call symput("numCode", i);
run;
%let k=1;
%do i=1 %to &n;
  %do j=%eval(&i+1) %to &n;
    %let svmcode&i._&j = &&svmcode&k;
    %let k =%eval(&k+1);
  %end;
%end;
%do i=1 %to &n;
  %let Target&i = &Target.&&level&i;
%end;
*** score the data by using each score code;
%MakeScoredDAG();
%mend SAS_SVM_DAG_SCORE;

%macro MakeScoredDAG();
data ScoredOutput1_&n;
  set &ScoreData;
  _temp_IDvar_ensure_not_existing_ = _N_;
run;
%do k=1 %to %eval(&n-1);
  %let i=&k;
  %let j=&n;
  %do m=1 %to &k;
    %let left =%eval(&i+1);
    %let right=%eval(&j-1);

```

```

data tempL tempR;
  set ScoredOutput&i._&j;
  %inc "&&svmcode&i._&j";
  if (I_&Target&&level&i = 1) then do;
    output tempR;
  end;
  else do;
    output tempL;
  end;
run;
%if &m=1 %then %do;
  data ScoredOutput&left._&j;
    set tempL;
  run;
%end;
%else %do;
  data ScoredOutput&left._&j;
    set ScoredOutput&left._&j tempL;
  run;
%end;
data ScoredOutput&i._&right;
  set tempR;
run;
%let i=%eval(&i-1);
%let j=%eval(&j-1);
%end;
%end;
data ScoredOutput;
  set
    %do i=1 %to &n;
      ScoredOutput&i._&i.(in = in&i.)
    %end;
    ;
%if (&TargetType = 2) %then %do;
  length I_&Target $ &TargetLength;
%end;
%if (&TargetType = 1) %then %do;
  %do i=1 %to &n;
    if (in&i.) then do;
      I_&Target = &&level&i;
    end;
  %end;
%end;
%if (&TargetType = 2) %then %do;
  %do i=1 %to &n;
    if (in&i.) then do;
      I_&Target = "&&level&i";
    end;
  %end;
%end;
keep
%if (&ID_NUM > 0) %then %do;
  &ID
%end;
I_&Target &Target _temp_IDvar.ensure_not_existing_;
run;

```

```

proc sort data=ScoredOutput
out=l.ScoredOutput(drop=_temp_IDvar_ensure_not_existing_);
  by _temp_IDvar_ensure_not_existing_;
run;
%do i=1 %to &n;
  %do j=&i %to &n;
    proc delete data=ScoredOutput&i._&j;
    run;
  %end;
%end;
%mend MakeScoredDAG;

```

REFERENCES

- Cortes, C., and Vapnik, V. (1995). "Support-Vector Network." *Machine Learning* 20:273–297.
- Hsu, C.-W., and Lin, C.-J. (2002). "A Comparison for Multiclass Support Vector Machines." *IEEE Transactions on Neural Networks* 13:415–425.
- Lichman, M. (2013). "UCI Machine Learning Repository." School of Information and Computer Sciences, University of California, Irvine. <http://archive.ics.uci.edu/ml>.
- Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (2000). "Large Margin DAGs for Multiclass Classification." *Advances in Neural Information Processing Systems* 12:547–553.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Ralph Abbey
SAS Institute Inc.
Ralph.Abbey@sas.com

Taiping He
SAS Institute Inc.
Taiping.He@sas.com

Tao Wang
SAS Institute Inc.
T.Wang@sas.com

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.

Random Forests with Approximate Bayesian Model Averaging

Tiny du Toit, North-West University, South Africa; André de Waal, SAS Institute Inc.

ABSTRACT

A random forest is an ensemble of decision trees that often produce more accurate results than a single decision tree. The predictions of the individual trees in the forest are averaged to produce a final prediction. The question now arises whether a better or more accurate final prediction cannot be obtained by a more intelligent use of the trees in the forest. In particular, in the way random forests are currently defined, every tree contributes the same fraction to the final result (e.g. if there are 50 trees, each tree contributes 1/50th to the final result). This ignores model uncertainty as less accurate trees are treated exactly like more accurate trees. Replacing averaging with Bayesian Model Averaging will give better trees the opportunity to contribute more to the final result which may lead to more accurate predictions. However, there are several complications to this approach that have to be resolved, such as the computation of a SBC value for a decision tree. Two novel approaches to solving this problem are presented and the results compared to that obtained with the standard random forest approach.

INTRODUCTION

Random forests (Breiman, 2001; Breiman, 2001b) occupies a leading position amongst ensemble models and have shown to be very successful in data mining and analytics competitions such as KDD Cup (Lichman, 2013) and Kaggle (2016). One of the reasons for its success is that each tree in the forest provides part of the solution which, in the aggregate, produces more accurate results than a single tree.

In the bagging and random forest approaches, multiple decision trees are generated and their predictions are combined into a single prediction. For random forests, the predictions of the individual trees are averaged to obtain a final prediction. All trees are treated equally and each tree makes exactly the same contribution to the final prediction. In this paper we question the supposition as model uncertainty is ignored.

Random forests are successful because the approach is based on the idea that the underlying trees should be different (if the trees were equal only one tree would be needed to represent the forest). This tree mixture is achieved by injecting randomness into the trees (this is explained in more detail in the following section). The resulting trees are diverse (by design) with varying levels of predictive power.

A goodness-of-fit statistic such as misclassification rate or average squared error may be used to judge the quality of each tree. Should the more predictive/accurate trees not carry more weight towards the final prediction? If the answer is affirmative, a second question needs to be answered: how should the trees be aggregated/amalgamated to get the best result?

In the rest of the paper a method of intelligent tree combination/aggregation, based on the theory of Bayesian Model Averaging, is proposed. Forests in SAS® Enterprise Miner is described in Section 2. The theory of Bayesian Model Averaging is explained in Section 3. For the theory of Bayesian Model Averaging to be applicable to decision trees, each tree's SBC value needs to be approximated. Neural networks are used to approximate the decision trees and this is explained in Section 4. A new weighting scheme is introduced in Section 5. Directly computing the degrees of freedom of a tree is reviewed in Section 6. The paper ends with a discussion and conclusions.

FORESTS IN SAS ENTERPRISE MINER

A random forest is an ensemble of decision trees. Multiple decision trees are constructed, each tree based on a random sample of observations from the training data set. The trees are then combined into a final model. For an interval target, the predictions of the individual trees in the forest are averaged. For a categorical target, the posterior probabilities are also averaged over the trees. A second step usually involves some kind of majority voting to predict the target category.

In SAS Enterprise Miner, the HPFOREST procedure (De Ville and Neville, 2013) takes a random sample (without replacement) of the observations in the training data set. This is done for each tree in the forest.

When each node in a tree is constructed, a subset of inputs is selected at random from the set of available inputs. Only the variable with the highest correlation with the target is then selected from this subset and used to split the node. Because many decision trees are grown, the expectation is that the better variables are more likely to be selected and that random errors introduced from overfitting will cancel when the predictions are averaged.

Our first attempt at Bayesian Model Averaging centered on the use of the HP Forest node in SAS Enterprise Miner (see Figure 1).



Figure 1. HP Forest node

However, because the node is “locked down”, the user is unable to get access to the individual trees in the forest. Furthermore, the bagged sample (training data set) as well as the out-of-bag sample (testing data set) constructed by the HP Forest node are inaccessible. It is therefore impossible to use the output of one HP Forest node (in its current state) to implement our new approach. After some experimentation we decided on a different strategy.

The data set that is analyzed in this paper is the HMEQ data set. The data set contains 13 variables with loan default status (BAD) as the dependent variable and 12 independent variables, e.g. years at current job (YOJ), number of derogatory reports (Derogatories), number of delinquent trade lines (Delinquencies), etc. The data partition node was used to partition the raw data set into a training data set containing 80% of the data and a validation data set containing 20% of the data (see Figure 2). As the HMEQ data set is relatively small (only 5960 observations), and the training data set is again going to be divided into bagged and out-of-bag samples, the training data set was kept as large as possible without compromising the various model building steps that will follow.



Figure 2. Data partitioning of the raw data set

N different trees are constructed using N Decision Tree nodes. The trees are then aggregated as needed. But, the trees have to be different (as would have been the case if the HP Forest node was used). The solution is to use the HP Forest node for variable selection.

N HP Forest nodes (with different seeds) are used to construct N different forests, each containing a single tree. As each forest is built using different bagged (0.8) and out-of-bag samples (0.2), the trees in the forests should be very different from each other. Also to restrict the number of variables, the maximum depth of the trees in the forest has been set to three.

Although N trees (one from each forest) have been built, the details of the trees are hidden and access to the bagged and out-of-bag samples that were used to construct the trees are unavailable. But, information on the subsets of variables used to construct the forests (and therefore the trees) is accessible. These subsets are now used to construct N trees using Decision Tree nodes (see Figure 3). This strategy will force the N trees to be different.

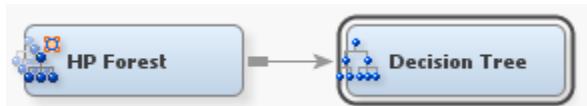


Figure 3. HP Forest node used for variable selection

The HP Forest nodes are basically used as variable selection nodes so that the N decision trees that are constructed will be different from each other (simulating the strategy used by the HP Forest node). The trees will most probably not be exactly the same as that constructed by the HP Forest node (because the

order of the splits are unknown and not all variables in the subset are available at each split), but the trees are built on the same subsets of variables used in the forests. The result is N different trees that can now be used to compare the different weighting schemes.

It might be tempting to use the N trees in the N forests to implement our new weighting scheme on. The problem is that each tree was built on a different bagged sample and also has an associated out-of-bag sample. Because the samples are not known, it is impossible to compute the goodness-of-fit statistics for the bagged or out-of-bag samples. The best that can be done is to consider the union of the different bagged and out-of-bag samples, which is the training data set.

As the HMEQ data set is small, partitioning the raw data set into three data sets to obtain a test data set is not practical (this would have been ideal to obtain an independent estimate of the performance of the models). The scoring data set therefore consists of the union of the training and the validation data sets, thus the raw hmeq data set (see Figure 4).

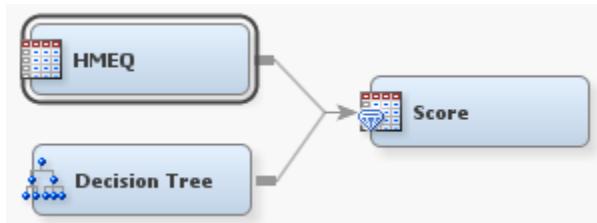


Figure 4. Scoring with a decision tree

This is a compromise and the fit statistics may therefore be optimistic. As only the relative performance of the models are important, all models are treated equally by scoring this data set.

The standard averaging implemented by the HP Forest node is now coded in a SAS Code node (see Figure 5) and the results computed on the scoring (hmeq) data set.



Figure 5. Computing goodness-of-fit measures

In this example, $N=5$ trees are constructed. Details of the number of leaves in each tree, the number of variables used in splitting and the depth of each tree are given in Table 1.

Tree	#Leaves	#Variables	Depth
1	10	3	5
2	5	2	4
3	16	4	6
4	14	4	6
5	16	4	6

Table 1. Five Trees

The c-statistic for the random forest based on these 5 trees is 88.3015, the misclassification rate is 14.89% and the sum of squared errors (sse) is 646.50. This is our *baseline* model and we will demonstrate in the following sections that a more intelligent amalgamation of the trees in the forest could result in a much better model with higher c-statistic, lower misclassification rate and smaller sum of squared errors (an indication of the variance of the errors).

BAYESIAN MODEL AVERAGING

When a single model is selected for predictive modeling, uncertainty about the structure of the model and the variables that must be included are ignored. This leads to uncertainty about the quantities of interest being underestimated (Madigan and Raftery, 1994). Regal and Hook (1991) and Miller (1984) showed in the contexts of regression and contingency tables that this underestimation can be large which can lead to decisions that have too high risk (Hodges, 1987).

In principle, the standard Bayesian formalism (Learner, 1978) provides a universal solution to all these difficulties. Let Δ be the quantity of interest, such as a future observation, a parameter or the utility of a course of action. Given data D , the posterior distribution of Δ is

$$pr(\Delta|D) = \sum_{k=1}^K pr(\Delta|M_k, D)pr(M_k|D) \quad (3.1)$$

The latter expression is the mean of the posterior distributions under each of the models, weighted by their posterior model probabilities. The models that are considered are M_1, M_2, \dots, M_K and

$$pr(M_k|D) = \frac{pr(D|M_k)pr(M_k)}{\sum_{l=1}^K pr(D|M_l)pr(M_l)} \quad (3.2)$$

where

$$pr(D|M_k) = \int pr(D|\theta_k, M_k)pr(\theta_k|M_k)d\theta_k \quad (3.3)$$

is the marginal likelihood of model M_k , θ_k is the parameter vector of M_k , $pr(M_k)$ is the prior probability of M_k , $pr(D|\theta_k, M_k)$ is the likelihood, and $pr(\theta_k|M_k)$ is the prior distribution of θ_k .

When averaging over all the models, a better predictive ability is obtained compared to using any single model M_j , as measured by a logarithmic scoring rule:

$$-E[\log\{\sum_{k=1}^K pr(\Delta|M_k, D)pr(M_k|D)\}] \leq -E[\log\{pr(\Delta|M_j, D)\}] \quad (j = 1, 2, \dots, K)$$

where Δ is the observable to be predicted and the expectation is with respect to

$$\sum_{k=1}^K pr(\Delta|M_k)pr(M_k|D).$$

This latter result follows from the nonnegativity of the Kullback-Leibler information divergence.

In practice, the Bayesian model averaging (BMA) approach in general has not been adapted due to a number of challenges involved (Hoeting, Madigan, Raftery and Volinsky, 1999). Firstly, the posterior model probabilities $pr(M_k|D)$ involve the very high dimensional integrals in (3.3) which typically do not exist in closed form. This makes the probabilities hard to compute. Secondly, as the number of models in the sum of (3.1) can be very large, exhaustive summation is rendered infeasible. Thirdly, as it is challenging, little attention has been given to the specification of $pr(M_k)$, the prior distribution over competing models. The problem of managing the summation in (3.1) for a large number of models has been investigated by a number of researchers. Hoeting (n.d.) discussed the historical developments of BMA, provided an additional description of the challenges of carrying out BMA, and considered solutions to these problems for a number of model classes. More recent research in this area are described by Hoeting (n.d.).

Lee (1999) and Lee (2006) considered a number of methods for estimating the integral of (3.3) and came to the conclusion that the SBC may be the most reliable way of estimating this quantity. The SBC defined as

$$SBC_i = \log(\mathcal{L}(\hat{\theta}|y)) - K_i \log(n)$$

is used. In addition, a noninformative prior is exploited that puts equal mass on each model, i.e. $P(M_i) = P(M_j)$ for all i and j . The SBC approximation to (3.2) then becomes

$$pr(M_i|D) \approx \frac{P(D|M_i)}{\sum_j P(D|M_j)} \approx \frac{e^{SBC_i}}{\sum_j e^{SBC_j}} \quad (3.4)$$

The Bayesian approach automatically manages the balance between improving fit and not overfitting, as additional variables that do not sufficiently improve the fit will dilute the posterior, resulting in a lower posterior probability for the model. This approach is conceptually straightforward and has the advantage of being used simultaneously on both the problem of choosing a subset of explanatory variables, and the problem of choosing the architecture for the neural network (Du Toit, 2006).

When the SBC defined as

$$SBC_i = -2 \log(L(\hat{\theta}|y)) + K_i \log(n)$$

is used, (3.4) becomes

$$pr(M_i|D) \approx \frac{P(D|M_i)}{\sum_j P(D|M_j)} \approx \frac{e^{-SBC_i}}{\sum_j e^{-SBC_j}}.$$

APPROXIMATING SBC WITH A NEURAL NETWORK

It is well-known that a decision tree can be used to approximate a neural network. This is usually done to gain some understanding of the neural network, as a neural network can be a black box. The converse, which is using a neural network to approximate a decision tree, is less obvious.

Just as with surrogate models (a surrogate model approximates an inscrutable model's predictions/decisions in order to facilitate interpretation), a neural network may be used to approximate the decision boundaries of the decision tree. As a neural network retains any non-linear relationships that are present in the decision tree, it is a good candidate for approximating a decision tree.

To apply the Bayesian Model Averaging formula of Section 3 to the trees in a forest, each tree's SBC is needed. This is not available as the degrees of freedom K for a decision tree is in general undefined. It is furthermore known that objective model selection criteria such as SBC cannot be used to compare models across different modeling techniques. It can only be used as a relative measure ranking models based on the same modeling technique. The expectation now is that the ranking of the decision trees from better to worse will be preserved in the SBC values computed by the neural networks.

Assume there are N decision trees in the forest. N neural networks are now constructed: each neural network approximating one tree. The number of hidden nodes in each neural network is adjusted to produce a neural network that closely shadows (in ROC curve and misclassification rate) the relevant tree (see Figure 6). As SBC is defined for neural networks in SAS Enterprise Miner, the neural network models' SBCs are now used as proxies for the decision trees' SBCs. Note also that SBC is only computed for the training data set by the Neural Network node, so this is what is used.

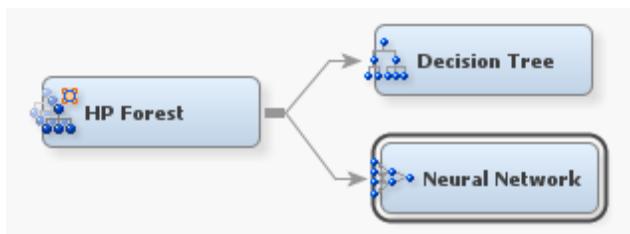


Figure 6. Approximating a decision tree with a neural network

The Neural Network node should be connected in parallel to the Decision Tree node and should have all the variables selected by the HP Forest node as inputs. If the Neural Network node is connected to the Decision Tree node, the Decision Tree node could do additional variable selection which may be undesirable. The training data set is used to construct the decision trees and the neural networks and the validation data set is used to optimize the decision trees as well as for stopped training in the neural networks.

Details of the $N=5$ constructed neural networks sorted by SBC are given in Table 2. All neural networks are multilayer perceptrons with one hidden layer and M hidden nodes.

Rank	Tree	# Hidden Nodes	SBC
1	2	4	3322
2	5	2	4032
3	3	4	4061
4	4	5	4393
5	1	5	4400

Table 2. MLP architectures

As the SBC values computed by the neural networks for the training data set used in this paper are large (3322 and greater), the base (e) used in the Bayesian Model Averaging formula, e.g.

$$\frac{e^{-3322}}{e^{-3322} + e^{-4032} + e^{-4061} + e^{-4393} + e^{-4400}}$$

creates computational difficulties and needs to be adjusted to make the computations viable.

When the base e is replaced by base 1, we get averaging:

$$\frac{1^{-3322}}{1^{-3322} + 1^{-4032} + 1^{-4061} + 1^{-4393} + 1^{-4400}} = \frac{1}{5}$$

as implemented in the HP Forest node in SAS Enterprise Miner (although SAS Enterprise Miner most definitely did not use the above formula to arrive at 1/5).

We therefore need a base greater than 1, but smaller than e to make the computations feasible. In this example, the base is adjusted to 1.002 (some experimentation might be needed to find and to adjust the base used in the formula so that reasonable weights are produced). The final weight computed for the best model is:

$$\frac{1.002^{-3322}}{1.002^{-3322} + 1.002^{-4032} + \dots + 1.002^{-4400}} = 0.5867$$

Table 3 ranks the five models from good to bad giving their SBC values (smaller is better) as well as final weight contribution to the forest.

Rank	SBC	Weight
1	3322	0.5867
2	4032	0.1420
3	4061	0.1340
4	4393	0.0690
5	4400	0.0680

Table 3. Bayesian Model Averaging

The c -statistic for this model is 89.3474, the misclassification rate is 13.37% and the sse is 565.61. This gives an improvement of more than 1.18% in the c -statistic over the standard method used to construct the forest. The misclassification rate is reduced by 10.2% and the sse decreased by 12.5%. Because we do not have the degrees of freedom for the decision tree, we cannot compute the error variance (as is usually done for linear regression), but sse gives a good indication that the size of the errors decreased (the computed probabilities are more precise).

It is worth noting that the SBC is only used to weigh the contributions of each tree and that the underlying trees in the forest are not modified at all. The trees are only amalgamated in a more intelligent way using the computed weights.

A NEW WEIGHTING SCHEME

Approximating a decision tree with a neural network has its drawbacks: extra computation time is needed to train and adjust the neural network and the approximation may be imprecise. Finding the appropriate base to get a reasonable spread of the final weight might be an issue.

As we are only interested in the relative ordering of the models (from good to bad), sse or validation misclassification rate might also suffice. Table 4 lists the SBC and sse on the training data set for the neural networks as well as the sse on the training data set for the decision trees and the validation misclassification rate also for the decision trees.

Rank	TRAIN SBC NN	TRAIN SSE NN	TRAIN SSE DT	VALID MISC DT
1	3322	992	952	12.98
2	4032	1214	1168	15.74
3	4061	1185	1168	15.74
4	4393	1282	1280	16.41
5	4400	1296	1313	16.75

Table 4. SBC, SSE and MISC

Note how closely the neural network sse approximates the decision tree sse. Except for the tie in the 2nd and 3rd models, the decision tree misclassification rate on the validation data set mimics the ordering of SBC computed with the neural network. A simplification of the whole process is therefore to use the validation misclassification rate computed for each decision tree to rank the models.

But, the formula used to compute the final weights depends on SBC and this is now missing if we omit constructing the neural networks. The following weighting scheme can be used as an approximation to the formula of the previous section, and this only depends on the ordering of the models (as given in Table 4), not the absolute values of the computed SBCs.

If there are N trees in the forest, the weight for each tree i ($i = 1, 2, \dots, N$) should be:

$$\left\{ \frac{2^{i-1}}{\sum_{k=0}^{n-1} 2^k} \right\}$$

For the hmeq data set with five trees in the forest, the weights are

$$\left\{ \frac{1}{31}, \frac{2}{31}, \frac{4}{31}, \frac{8}{31}, \frac{16}{31} \right\}$$

which seems reasonable and not that different from the weight computed with Bayesian Model Averaging. The table in the previous section is therefore updated to (see Table 5):

Rank	VALID MISC DT	Weight
1	12.98	0.5161
2	15.74	0.2580
3	15.74	0.1290
4	16.41	0.0645
5	16.75	0.0322

Table 5. Weights based on VALID MISC of DT

Note how closely these weights resemble the weights computed with SBC (see Table 2). The c -statistic for this model is 89.4178, the misclassification rate is 13.65% and the sse is 569.20. This gives a 1.12% improvement in the c -statistic, a 8.32% improvement in the misclassification rate and a 11.9% decrease in the sse.

Although not as good as the previous model, it is still a significant improvement over our baseline model. Furthermore, this is an extremely simple computation that would require very little time to compute.

The formula also generalizes to larger N as the contributions of the inferior models in the forest tend to approach 0. This makes intuitive sense as the effect of random errors are mitigated. If the misclassification rate on the validation data set is replaced with misclassification rate on the out-of-bag sample, it would be a simple step to update the HP Forest node with this new result.

APPROXIMATING THE DEGREES OF FREEDOM K

The problem when trying to compute SBC values for decision trees (highlighted in Section 2) is that we do not have the degrees of freedom K for a decision tree. The AIC and SBC information criteria considers the tradeoff between fit and complexity. The principle is to penalize the fit for the complexity. For a decision tree we need to count the number of independent parameters. In Ritschard and Zighed (2003)

$$K = (r - 1)(c - q)$$

is given as the degrees of freedom for a induced/constructed tree, where q is the number of leaves in the tree, r the number of variables in the tree and c the product of the number of distinct levels for each of the r variables in the tree.

Although the formula for K looks simple, for any tree of reasonable complexity with multiple occurrences of the same variable, and with continuous variables added, the formula became increasingly difficult to apply. K can also become extremely large for a seemingly simple tree.

However, this approach of directly computing K seems promising and will be further investigated in a follow-up paper.

DISCUSSION

The theory of Bayesian Model Averaging is well-developed and provides a coherent mechanism for accounting for model uncertainty. It is therefore surprising that it has not been applied directly to random forests.

Bayesian additive regression (Heranádez, 2016) was an attempt to create a Bayesian version of machine learning tree ensemble methods where decision trees are the base learners. BART-BMA attempted to solve some of the computational issues by incorporating Bayesian model averaging and a greedy search algorithm into a modelling algorithm.

The method proposed in this paper does not attempt to turn an ensemble of decision trees into a statistical model (with corresponding probability estimates and predictions). Furthermore, the base

learners (e.g. decision trees) are only combined in a novel way to produce a more accurate final prediction.

The way the decision trees are combined depends on the ordering of the decision trees from more accurate to less accurate. This was first achieved by building a surrogate neural network model for each tree and using the neural network models' ordering as a proxy for the decision trees' ordering.

The improvement in c-statistic, misclassification rate and sse confirmed our supposition that there is a better way of combining trees than the standard averaging used in random forests. The improvement is summarized in Table 6.

Model	c-statistic	MISC Rate	sse
Random Forest (Ave)	88.30	14.89%	646.50
Random Forest (SBC)	89.34	13.37%	565.61
Random Forest (Scheme)	89.41	13.65%	569.20

Table 6. Results

The Bayesian Model Averaging on surrogate neural networks introduced in this paper elegantly mitigates the reliance on the expectation that random errors introduced from overfitting will cancel when the predictions are averaged. Complex models where overfitting might be an issue are penalized in their SBC values (because of the large degrees of freedom value K in the surrogate neural network) with a resulting reduction in weight or contribution to the final model. Smaller errors are introduced into the system than is the case with random forests and it pays off in a better final model with improved fit statistics.

A Bayesian approach for finding CART models was presented in Chipman, George and McCulloch (1998). The approach consists of two basic components: prior specification and stochastic search. The procedure is a sophisticated heuristic for finding good models, rather than a full Bayesian analysis.

In a sense the procedure outlined in this paper is also a sophisticated heuristic that is used to compute the contribution of each tree to the forest, but with full Bayesian Model Averaging implemented on surrogate neural networks instead of the actual decision trees.

CONCLUSIONS

Although only a small change was proposed to the random forest algorithm, the improvements as shown in this paper could be substantial. However, the method depends on computing SBC values for decision trees which is problematic as a decision tree is not regarded as a statistical model.

The way around this problem is to use the SBC computed by a surrogate neural network. This gave an ordering of the models from good to bad. This information was then used to vary the contribution of each decision tree to the final model. Although a smart approximation, it still required a neural network to be built.

In a further simplification, validation misclassification rate was used to rank models and the contribution of each model to the final prediction was computed with a novel weighting scheme. The last results were still substantially better than that of the standard random forest approach, but not as good as when a neural network was used to approximate SBC.

REFERENCES

- Breiman, L. "Random Forests." Statistics Department, University of California Berkeley, CA. 2001. Available at <http://leg.ufpr.br/lib/exe/fetch.php/wiki:internas:biblioteca:randomforest.pdf>
- Breiman, L. 2001b. "Random Forests." Machine Learning, Vol. 45(1):5-32.
- Chipman, H. A., George, E. I. and McCulloch, R. E. 1998. "Bayesian CART model search (with discussion and rejoinder by the authors)." Journal of the American Statistical Association. Vol. 93:936-960.
- De Ville, B. and Neville, P. 2013. "Decision Trees for Analytics Using SAS Enterprise Miner." SAS Press, Cary, USA.
- Du Toit, J. V. 2006. "Automated Construction of Generalized Additive Neural Networks for Predictive Data Mining." PH.D. thesis. School for Computer, Statistical and Mathematical Sciences, North-West University, South Africa.
- Heranádez, B. "Bayesian additive regression using Bayesian model averaging." 2016. Available at <http://arxiv.org/abs/1507.00181>
- Hodges, J. S. 1987. "Uncertainty, policy analysis and statistics." Statistical Science, Vol. 2(3):259-275.
- Hoeting, J. A. "Methodology for Bayesian model averaging: an update." Colorado State University. n. d. Available at <http://www.stat.colostate.edu/~nsu/starmap/jab.ibcbma.pdf>
- Hoeting, J. A., Madigan, D., Raftery, A. E. and Volinsky, C. T. 1999. "Bayesian model averaging: a tutorial." Statistical Science, Vol. 14(4):382-417.
- Kaggle. 2016. Available at <https://www.kaggle.com/>
- Learner, E. E. 1978. "Specification searches: ad hoc inference with nonexperimental data." Wiley Series in Probability and Mathematical Statistics, John Wiley and Sons, New York.
- Lee, H. K. H. 1999. "Model selection and model averaging for neural networks." PH.D. thesis. Department of Statistics, Carnegie Mellon University.
- Lee, H. K. H. "Model selection for neural network classification." 2006. Available at <http://citeseer.ist.psu.edu/leeeomodel.html>
- Lichman, M. "Machine Learning Repository." University of California, Irvine, School of Information and Computer Sciences. 2013. Available at <http://archive.ics.uci.edu/ml>
- Madigan, D. and Raftery, A. E. 1994. "Model selection and accounting for model uncertainty in graphical models using occam's window." Journal of the American Statistical Association, Vol. 89(428):1535-1546.
- Miller, A. J. 1984. "Selection of subsets of regression variables." Journal of the Royal Statistical Society, Series A, Vol. 147(3):389-425.
- Regal, R. R. and Hook, E. B. 1991. "The effects of model selection on confidence intervals for the size of a closed population." Statistics in Medicine, Vol. 10:717-721.
- Ritschard, G. and Zighed, D. A. 2003. "Goodness-of-fit measures for induction trees." Foundations of Intelligent Systems, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Vol. 27:57-64.

ACKNOWLEDGEMENTS

The authors wish to thank SAS Institute for providing them with Base SAS and SAS Enterprise Miner software used in computing all the results presented in this paper. This work forms part of the research done at the North-West University within the TELKOM CoE research program, funded by TELKOM, GRINTEK TELECOM and THRIP.

CONTACT INFORMATION

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

Tiny du Toit
Tiny.DuToit@nwu.ac.za

Vibration Spectral Analysis to Predict Asset Failures by Integrating R in SAS® Asset Performance Analytics

Adriaan Van Horenbeek, SAS Institute Inc.

ABSTRACT

In industrial systems, vibration signals are the most important measurements indicating asset health. Based on these measurements, an engineer with expert knowledge on the assets, industrial process and vibration monitoring can perform spectral analysis to identify failure modes. However, this is still a manual process that heavily depends on the experience and knowledge of the engineer analyzing the vibration data. Moreover, when measurements are performed continuously, it becomes impossible to act in real time on this data. Therefore, the objective of this paper is to use analytics to perform vibration spectral analysis in real time to predict asset failures. The first step in this approach is to translate engineering knowledge/features into analytic features to perform predictive modeling. This is done by converting the time signal into the frequency domain by applying a Fast Fourier Transform. Based on the specific design characteristics of the asset, it is possible to derive the relevant features of the vibration signal to predict asset failures. This approach is illustrated using a bearing data set available from the Prognostics Data Repository of NASA. The modeling part is done using R and is integrated within SAS® Asset Performance Analytics. In essence, this approach helps the engineers to make better data-driven decisions. This paper illustrates the strength of combining expert engineering knowledge with advanced data analytics techniques to improve asset performance.

INTRODUCTION

Looking at the recent trends within manufacturing like Industry 4.0, smart factories, and the Industrial Internet of Things (IIoT), it is obvious that manufacturing assets and equipment will only generate more data in the coming years. Sensors are becoming cheaper, signal processing capabilities have improved, and wireless data acquisition of these sensor measurements is becoming mainstream. One of the most promising fields within manufacturing to use this data is within asset management, more specifically predictive maintenance. So what does it mean and where did it originate?

Many definitions on maintenance exist, but when considering the bottom line, it can be best defined as a set of activities required to keep equipment, installations, and other physical assets in the desired operating condition or to restore them to this condition (Pintelon and Van Puyvelde 2006). However, this definition is too simple and narrow to define maintenance in all its complexities; therefore, the term asset management has been recently defined. Asset management, even more profound than maintenance management, focuses on the entire life cycle of an asset, including strategy, risk measurement, safety and environment, and human factors. The focus of this paper lies within physical asset management. A publicly available specification for the optimized management of physical assets published by the British Standards Institution (PAS 55:2008) defines physical asset management as follows:

"Asset management can be defined as the systematic and coordinated activities and practices through which an organization optimally and sustainably manages its assets and asset systems, their associated performance, risks and expenditures over their life cycles for the purpose of achieving its organizational strategic plan." (PAS 55:2008)

Maintenance, as such, is thus part of a bigger asset management strategy. In the last several decades, maintenance practice has gone through a process of change due to the increasing awareness of its importance. In the 1950s, corrective or reactive maintenance (run-to-failure) was the predominant maintenance policy. In the 1960s, preventive maintenance (time- or use-based maintenance) became popular. Regular component replacements were scheduled in order to try to avoid any possible—unscheduled—failure, regardless of the health status of a physical asset. In the second half of the 1980s, more and more companies were wondering whether they were not overdoing maintenance by replacing components that could have lasted longer, for example. Therefore, condition monitoring and diagnostic technologies were developed, and consequently, condition-based maintenance (CBM) emerged.

Condition-based maintenance is defined according to the European standard (EN 13306:2010) as follows: "preventive maintenance which includes a combination of condition monitoring and/or inspection and/or testing, analysis and the ensuing maintenance actions". In this way, CBM attempts to avoid unnecessary maintenance activities by triggering these actions only when there is evidence of deterioration or abnormal behavior by monitoring intermittently or continuously. In the beginning, this seemed to be reserved for the high-risk industries, but as it became cheaper, it found its way to the industry at large. Recently, prognostics, which deals with fault prediction before it occurs, made its introduction into maintenance management. Fault prediction determines whether a fault is impending and estimates how soon and how likely a fault will occur (Jardine, D. Lin, et al. 2006). A maintenance policy incorporating prognostics into the decision process is defined as a predictive maintenance policy (PdM). Predictive maintenance can be defined as: "condition-based maintenance carried out following a forecast (that is, remaining useful life [RUL]) derived from repeated analysis or known characteristics and evaluation of the significant parameters of the degradation of the item" (EN 13306:2010). This means that, compared to CBM, PdM incorporates more information into the maintenance decision process as information about future machine or component degradation, in the form of their remaining useful life by prognostics, is taken into account.

For condition monitoring of rotating equipment, the most widely used technology is vibration analysis. Depending on the application, the displacement, velocity, or acceleration is measured. The captured time signal is then converted to a frequency spectrum to identify the frequencies where you can see a change in amplitude to diagnose the specific failure mode of the asset. For more information about vibration analysis, I refer you to the book by Cornelius Scheffer and Paresh Girdhar cited in Recommended Reading. The current state of vibration analysis within manufacturing is that a technician walks around with route-based vibration analyzers and collects data points each month or each quarter. Only the really critical equipment has some continuous measurement systems in place. Moreover, the scope of analysis is limited to a single asset. Some issues arise from this approach:

- After the condition monitoring round is done by the technician, the data is analyzed by spectral analysis to identify failure modes by an engineer with expert knowledge on the assets, industrial process, and vibration monitoring techniques. The engineer compares the current measurement to previous measurements by a mostly manual approach. Therefore, it is a process that heavily depends on the experience and knowledge of the engineer analyzing the vibration data and it is very resource-intensive.
- The features that have to be monitored are derived purely from an engineering perspective. This process is time-consuming as the features are different for each type of equipment.
- Finding the right skilled people to do this analysis is challenging.
- Only intermittent data is available on a weekly, monthly, or quarterly basis. As such, the probability of missing out on failures increases.

Because of the recent evolutions in sensors, data acquisition, and signal processing, and as a result the advent of Industry 4.0 and the Industrial Internet of Things, companies are looking to solve these issues with the traditional condition monitoring approach. Due to the decreasing cost of sensors, it is possible to install permanent sensors on the assets in order to continuously monitor the asset health. By doing so they reduce costs, as no people have to run around in the plant just to collect data. They have the opportunity to continuously monitor their assets rather than only intermittently monitoring them. However, doing so also introduces new challenges within manufacturing, which is, by the way, good news for analytics:

- The challenge of scaling this approach to continuously monitored systems and data spawning sensors is huge. The right skilled people to handle this huge amount of data are not available and even for the right skilled, it is a challenge to analyze this amount of data without the proper tooling.
- Due to the decreasing cost of sensors, it will even become interesting, from a business case perspective, to start monitoring non-critical equipment, which only makes the collected, to-be-analyzed data bigger.
- If all assets are equipped with sensors, you need specific information about each piece of equipment

to determine the relevant features to be monitored. As this has to be done by people with engineering skills, this is a major issue regarding the resources necessary to do so.

- The opportunity to react in real time (which comes by implementing continuous monitoring) to alerts disappears when the data still needs to be analyzed by an engineer.
- Due to the huge amounts of data that will be collected, it becomes even more difficult to find the real issues and prioritize these issues and the generated alerts.
- When you combine vibration data with other condition monitoring data (for example, thermography or current measurements) or even process data, maintenance data, quality data, and so on, the real challenge arises.

The good thing is that analytics provides an answer to all the above challenges that manufacturing companies are facing when making the transition toward the integration of the Internet of Things for condition monitoring. Therefore, the objective of this paper is to provide insight on how analytics can be leveraged to generate value within asset management. More specifically, you will get insight on how SAS® Asset Performance Analytics, which was designed for this purpose, provides a solution for manufacturing companies. Moreover, a detailed and scoped example on how to use the SAS Asset Performance Analytics solution to perform vibration analysis is given. The focus is on how to scale the vibration spectral analysis when continuous measurements become available, on how to make this analysis available to non-experts through SAS Asset Performance Analytics, and on illustrating the strength of combining expert engineering knowledge with advanced data analytics techniques to improve asset performance in the era of the Internet of Things.

The detailed vibration spectral analysis is illustrated using a bearing data set available from the Prognostics Data Repository of NASA. The modeling part is done using R and is integrated within SAS Asset Performance Analytics. I hear you say: "Why do you do this in R and not SAS®?" I played around with the vibration data set in the past, before I started at SAS. I did it in open-source software, more specifically in R. As I already did the work in the past, I didn't want to invest time to just copy my work in SAS code. I wanted to use the R code I developed earlier and industrialize it so that everybody can use it through the SAS Asset Performance Analytics interface. This perfectly mimics a situation we regularly encounter at our manufacturing customers.

By reading this paper I hope you will gain insight into how to use SAS Asset Performance Analytics to improve asset performance. You will learn how to perform vibration spectral analysis, how to integrate R code into SAS Asset Performance Analytics, and finally how to make it available for other people and scalable to an entire manufacturing plant. Moreover, I will give some insight into how to further develop this approach.

ANALYTICS WITHIN ASSET MANAGEMENT

Within asset management it is crucial to consider the different maturity levels as defined, for example, in standards like PAS 55:2008 and ISO 55000. All manufacturing companies use these maturity scales to benchmark themselves compared to a world-class company. Moreover, they use it to build their strategy on how to improve their asset management strategy. These maturity levels can be summarized as shown in Figure 1. It is valuable to note that SAS Asset Performance Analytics (further discussed in SAS Asset Performance Analytics solution) covers the entire spectrum of maturity steps in asset management, from standard reporting capabilities to reliability modeling and trend analysis to advanced predictive modeling and optimization. As such, it provides an end-to-end solution for asset management within manufacturing.

Most manufacturing companies are somewhere between level 4 and 5 of this asset management maturity scale, so there is still quite some improvement potential at stake. What has analytics to do with this asset management maturity scale? Definitely a lot. If we look at the analytics maturity levels defined by Thomas H. Davenport, it is striking to see that these map very well on the maturity levels defined within asset management. When you think about it, this is logical. To get to higher levels on the asset management maturity level, you have to make better use of data. And this is exactly where analytics comes into play.

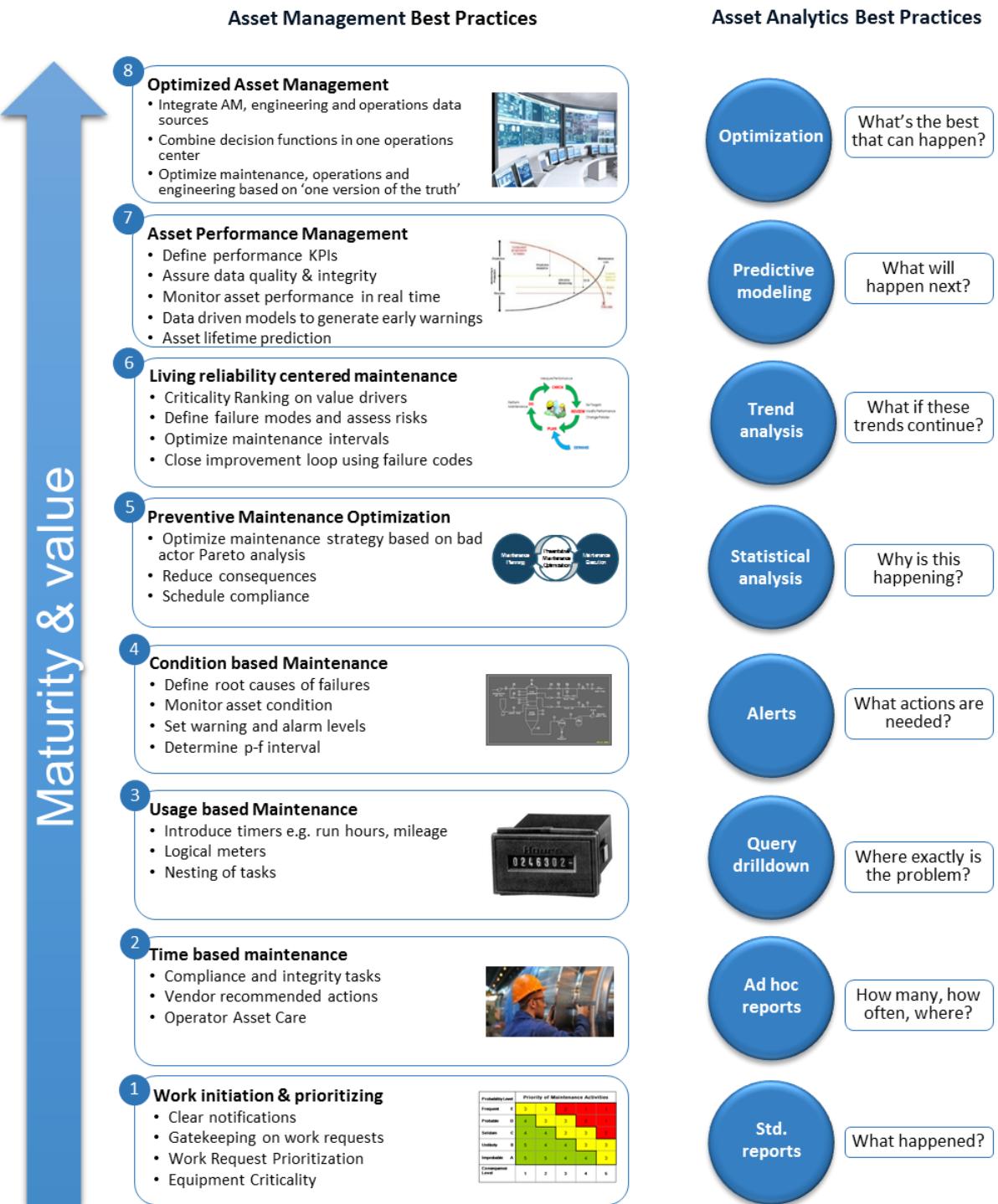


Figure 1. Asset Management Maturity Levels versus Analytics Maturity Levels

So how can analytics be used to help in the transition that is happening within asset management nowadays? There are two development tracks that manufacturers need to take to grow on this maturity scale. First, deploy more sensor measurements and make the measurements continuous in order to monitor and improve asset performance; and second, for the bad actors and critical equipment, use advanced analytical techniques to predict impending failures in order to prevent high failure cost, lost

production, downtime, and safety and environmental issues. These two approaches are shown in Figure 2 and discussed more in detail in the following paragraphs.

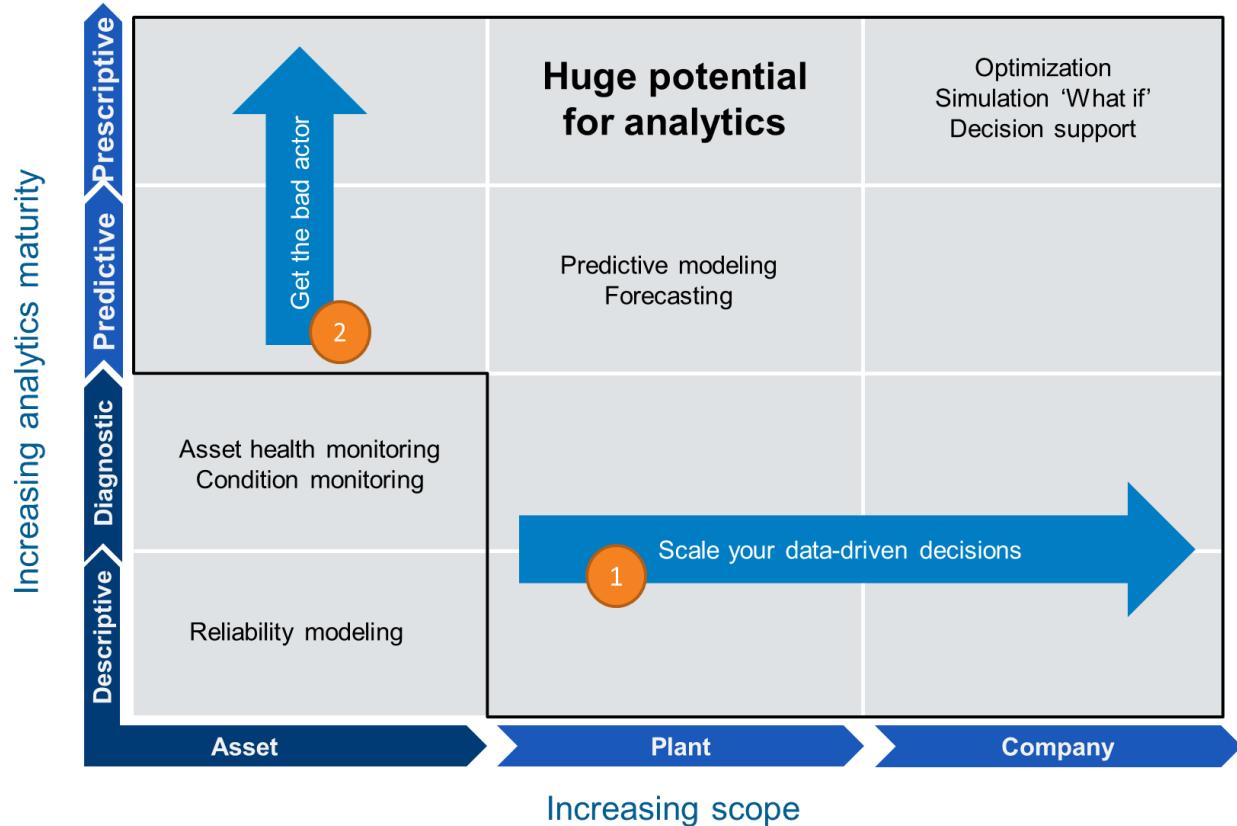


Figure 2. How Analytics Can Be Used to Further Develop Asset Management Maturity

The first path that I will discuss is how to scale your data-driven decisions (indicated by number '1' in Figure 2). Currently in most manufacturing companies, there is some sort of reliability modeling and condition monitoring happening. However, most of the time, it is still limited to a scope of a single asset and based on ad hoc analysis performed by the engineers. There are two major reasons why it stays limited to a single asset. The first reason is that the analysis becomes too complex, cumbersome, and resource-intensive when it is scaled to an entire plant. The second reason is that currently the asset health features that are monitored are defined by engineering knowledge and these must be specified for each asset separately. This becomes challenging when many assets have to be considered. As more sensor data becomes available on a continuous basis, the first step is to automate the analysis of this continuously measured condition monitoring data for a single asset. By automating this, engineers can spend time only on the real issues, rather than wasting time on analyzing all data, and resolve these faster. Furthermore, analytics can be used to automatically derive the relevant features to monitor that makes you scale fast. When issues arise on a specific asset, a deep dive can be made by the engineer, and engineering knowledge-based features can be incorporated into the analysis or advanced predictive models can be developed. As such, a continuous improvement program can be introduced. The automation of the analysis and feature definition becomes even more important when we scale the condition monitoring approach to multiple assets, the entire plant, and even the entire company. Analytics provides an answer to these challenges.

The second path is to develop advanced predictive models to predict remaining useful life for the bad actor assets. For rotating equipment, condition monitoring is performed by measuring specific asset-oriented variables like vibration. Based on the analysis of known failure modes, degradation patterns, and aging, it is possible to develop predictive models. Prognostics is currently only implemented when there is sound knowledge of the failure mechanisms that are likely to cause failure. Although still useful for certain

assets, most prognostic solutions are limited in scope by a considerable narrow scope of data that is used for the analysis and only a limited set of known failure modes that can be detected. The data used will typically only be condition monitoring data like vibration, thermal, and oil analysis results. As such, most manufacturers still ignore the most important factors to predict asset failures like how the asset is operated, changes in product composition, variability in process conditions, and so on. It is only when combining the traditional condition monitoring measurements with additional process data, maintenance data, engineering data, and quality data that the real health of an asset can be determined and predicted. This however introduces additional challenges like the amount of data that has to be integrated and analyzed, the complexity of linking operations and maintenance, and the introduction of asset interdependencies into the analysis. Advanced analytics provides an answer to address these challenges to develop predictive maintenance models considering all relevant asset data and failure modes. This approach introduces predictive maintenance based on evolving knowledge of operation history and anticipated usage of the machinery, as well as the physics and dynamics of material degradation in critical components.

In this paper, I will focus on how to approach the first development path on how to scale your data-driven decisions from a single asset to the entire company by automatically deriving the relevant features to monitor and by making the analysis available through the pre-defined analysis workspace of the SAS Asset Performance Analytics workspace.

SAS ASSET PERFORMANCE ANALYTICS SOLUTION

SAS Asset Performance Analytics is the part of the SAS Quality Analytic Suite that monitors equipment sensors and machine-to-machine (M2M) data to identify hidden patterns that predict slowdowns and failures. The advanced analytics, data mining, and data visualization capabilities allow engineers to identify the real drivers of performance issues out of hundreds or even thousands of measures and conditions. By doing so a predictive maintenance culture is introduced into an organization by implementing SAS Asset Performance Analytics. This foresight gives operations personnel time to bring the equipment down in a planned and controlled way to address the following issues: avoiding unplanned downtime, excessive maintenance costs, revenue losses and environment, and health and safety issues. SAS Asset Performance Analytics also helps you reduce personnel and maintenance costs by pinpointing problems and aligning resources. It also allows you to detect and correct issues earlier to mitigate the risk of failures and downtimes.

Next to the predictive capabilities of SAS Asset Performance Analytics, SAS® Quality Analytic Suite provides common features such as the report and dashboard library, visualization and exploration tools, and pre-defined analytical models (that is, stability monitoring and root cause analysis).

SAS Asset Performance Analytics is packaged with domain-specific data management including:

- data model (single, integrated, and aggregated view of all relevant data and information) and methods to easily build data sets for modeling (for example, to build predictive failure models) and root cause analysis
- unique modeling techniques for assets
- business rules and processes designed to address the complexity of "dirty" sensor data and set threshold based alarms and alerts
- unique user interfaces, including sensor, event and asset visualization
- standard, pre-built reports for asset performance management and to assess and improve data quality.

SAS Asset Performance Analytics is an analytics-driven solution that helps manufacturers to achieve the goal of optimally managing their assets by addressing all asset management maturity levels like shown in Figure 1.

Due to the automated monitoring and alerts, engineers are directed toward issues and can focus on the issues that need fast problem resolution rather than having to focus on analyzing non-relevant data. Automatic monitoring is introduced by implementing basic to advanced alarms and alerts. Basic alerts are

for example, threshold or rate-of-change based, while advanced alerts are triggered by advanced predictive models that predict remaining useful lifetime. Once an alert is triggered, notifications can be sent via email, text, or pager, as part of a workflow or integrated into your operational systems. Alerts guide engineers directly to the supporting control charts and supplemental reports can be supplied for analysis. Users can easily drill down to better understand the root cause of the issue. Workflows are initiated and case management provides a knowledge repository for standardized problem resolution, enabling auditability for asset and process changes. Advanced predictive modeling accurately and reliably finds hidden patterns in the data that indicate an impending failure or performance degradation. New models are developed based on historical events by not only taking into account condition monitoring data (as in the traditional Condition-Based Maintenance [CBM] technologies) but also for example, process parameters. Moreover, traditional CBM technologies focus on one piece of equipment at a time, while the advanced predictive models and analytics are not limited to one piece of equipment because of their capability to analyze massive amounts of data at once. As such, it is possible to take into account data from the entire process line to develop these predictive models. In the dawn of the Internet of Things era, high-performance analytics ensures virtually limitless scalability to continuously monitor the health of your assets. You can test new sensor data against defined rules, thresholds, and predictive scoring models to give you ample time to take the appropriate corrective actions.

Altogether, SAS Asset Performance Analytics provides analytic and predictive capabilities that your organization can use to ensure peak performance and minimal downtime for crucial assets and equipment. AS Asset Performance Analytics provides data integration, advanced analytics, data visualization, and exploration capabilities. It allows engineers, business users, and data scientists to identify the real drivers of performance issues out of hundreds or even thousands of measures and conditions and different data sources. As such it supports both your short-term and daily decision processes as well as your long-term asset management strategy. Moreover, by illustrating how to integrate R programming into SAS Asset Performance Analytics and making the analysis available as a guided point-and-click analysis (this can, of course, also be done for SAS programs), manufacturers are able to extend their analytical knowledge and insights well beyond their core analytics team. Engineers are not statisticians and/or programmers.

In the following paragraphs, I will illustrate how to perform vibration spectral analysis, how to integrate the R code into SAS Asset Performance Analytics and industrialize the analysis so that everybody can use it through the SAS Asset Performance Analytics interface. I will show how to use this to detect and predict failures.

INTRODUCTION TO THE DATA SET

In rotating industrial equipment, vibration signals are the most important measurements indicating asset health. These vibration measurements generate high frequency time series data. Vibration spectral analysis is performed by converting the time signal into the frequency domain by applying a Fast Fourier Transform. Based on this vibration spectrum it is possible to identify potential asset failures. This approach is illustrated using a bearing data set available from the Prognostics Data Repository of NASA. The data was generated by the NSF I/UCR Center for Intelligent Maintenance Systems with support from Rexnord Corp. in Milwaukee, WI.

The data does not allow us to build advanced predictive models as we do not have enough failure events and it is impossible to find an open data source that does. So in this paper I will focus on how you can scale the condition-based activities. However, based on this, I will also give some insight into how to start building predictive models based on the work done.

TEST RIG SETUP

Four bearings were installed on a shaft. The rotation speed was kept constant at 2000 RPM by an AC motor coupled to the shaft via rub belts. A radial load of 6000 lbs is applied onto the shaft and bearing by a spring mechanism. All bearings are force lubricated.

Rexnord ZA-2115 double row bearings were installed on the shaft as shown in Figure 3. PCB 353B33 High Sensitivity Quartz ICP accelerometers were installed on the bearing housing (two accelerometers for each bearing [x- and y-axes] for data set 1, and one accelerometer for each bearing for data sets 2 and

3). Sensor placement is also shown in Figure 3. All failures occurred after exceeding the designed life time of the bearing, which is more than 100 million revolutions. More details can be found in (Qiu, H., J. Lee et al., 2006).

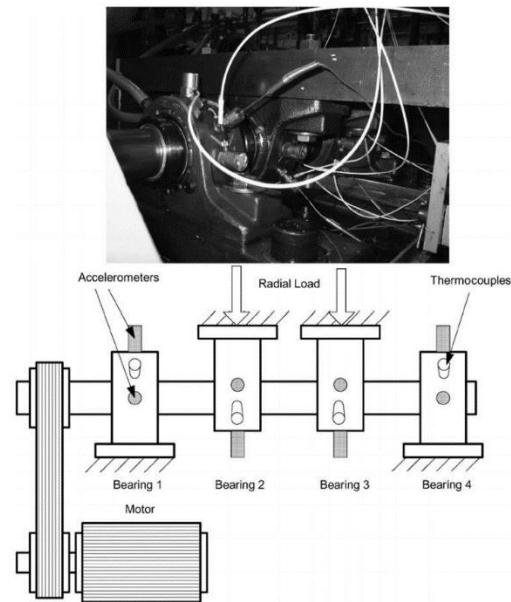


Figure 3. Bearing Test Rig and Sensor Placement (Qiu, H., J. Lee et al., 2006)

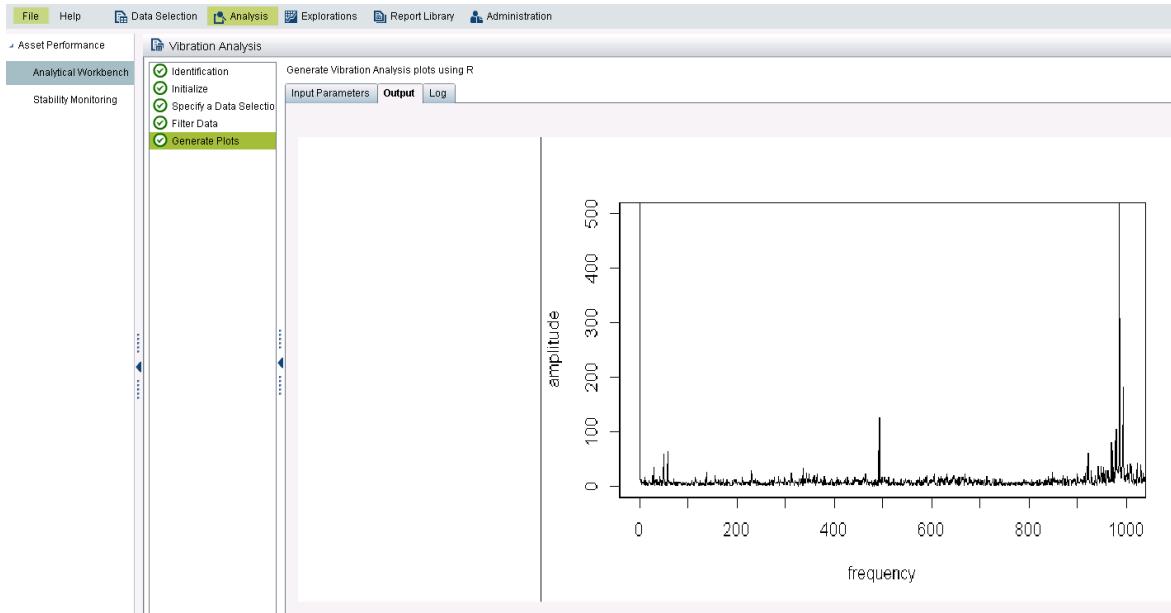
VIBRATION DATA SET

Three data sets are included. Each data set describes a test-to-failure experiment. Each data set consists of individual files (2156 files to be precise for the first test) that are 1,024-second vibration signal snapshots recorded at specific intervals. Each file consists of 20,480 points with the sampling rate set at 20 kHz. The filename indicates when the data was collected. Each record (row) in the data files is a data point. Data collection was facilitated by a National Instruments data acquisition card (NI DAQ Card 6062E). Larger intervals of time stamps indicate resumption of the experiment in the next working day. For this paper, I focus on the first test-to-failure experiment. At the end of the first test-to-failure experiment, an inner race defect occurred in bearing 3 and a roller element defect and outer race defect occurred in bearing 4. For this paper, I will focus on the failure on bearing 4 as an illustration.

VIBRATION SPECTRAL ANALYSIS TO PREDICT ASSET FAILURES BY INTEGRATING R IN SAS ASSET PERFORMANCE ANALYTICS

Display 1 shows the custom SAS Asset Performance Analytics analysis used to demonstrate the SAS open-source integration capabilities using a simple five-step approach. Step 1 identifies which analysis to run. Step 2 initializes the data selections that can be used for the vibration analysis. Step 3 prompts the user to select an SAS Asset Performance Analytics data selection as input to the R analysis. Step 4 prepares the SAS Asset Performance Analytics data for R using simple SAS DATA step code and performs the selected vibration analysis in R. Step 5 generates the saved plots from the R analysis in the SAS Asset Performance Analytics user interface.

The SAS Asset Performance Analytics stored process incorporates R using the IML procedure with the Submit /R statement, and the R output is temporarily saved as a PNG file. Finally, a %include statement and GSLIDE procedure iframe statement is used to call the R code and display analysis results within the SAS Asset Performance Analytics user interface environment. This approach allows users to easily access and modify the PROC IML program and enhance their SAS Asset Performance Analytics analysis workspace with open-source analytics.



Display 1. Vibration Analysis within the SAS Asset Performance Analytics Workbench

The following source code, using PROC IML, is used to integrate the R code:

```

proc iml;
print "R Code Vibration Analysis used to generate output";
run ExportDatasetToR("pamfs.vibr_r","data");
SUBMIT / R;
  summary(data)
  png(filename = "C:/Users/sas/Documents/R_output/rplot1.png")
  plot(data, t="l")
  dev.off()
  b.fft <- fft(data)
  # Ignore the 2nd half, which are complex conjugates of the 1st half,
  # and calculate the Mod (magnitude of each complex number)
  amplitude <- Mod(b.fft[1:(length(b.fft)/2)])
  # Calculate the frequencies
  frequency <- seq(0, 10000, length.out=length(b.fft)/2)
  # Plot!
  png(filename = "C:/Users/sas/Documents/R_output/rplot2.png")
  plot(amplitude ~ frequency, t="l")
  dev.off()

  png(filename = "C:/Users/sas/Documents/R_output/rplot3.png")
  plot(amplitude ~ frequency, t="l", xlim=c(0,1000), ylim=c(0,500))
  axis(1, at=seq(0,1000,100), labels=FALSE) # add more ticks
  dev.off()

  sorted <- sort.int(amplitude, decreasing=TRUE, index.return=TRUE)
  top15 <- sorted$ix[1:15] # indexes of the largest 15 frequencies
  top15f <- frequency[top15] # convert indexes to frequencies
ENDSUBMIT;

```

The submitted R code through PROC IML first plots the time series data of one 1,024-second vibration signal snapshot consisting of 20,480 points. The plot is saved as 'rplot1.png'. Then the Fast Fourier Transformation algorithm (FFT) computes the Discrete Fourier Transform (DFT) of the time series sequence. The Fourier transform for vibration analysis converts the time signal into a representation in the frequency domain. Both amplitude and corresponding frequencies are calculated. The maximal

frequency is 10,000 Hz, which is the Nyquist frequency and equals half the sampling rate (20,000 Hz). A plot of the frequency spectrum is made and saved to 'rplot2.png'. A third plot with a zoom on the frequency range from 0 Hz to 1000 Hz is made and saved to 'rplot3.png'. As the last step, the 15 strongest frequency components from the spectrum are extracted as features. If you would like to add additional analysis within R, this can be easily done by modifying the PROC IML program.

Finally, a %include statement and PROC GSLIDE iframe statement are used to call the result of the R code and display analysis results within the SAS Asset Performance Analytics user interface environment:

```
%include "C:\Program Files\R\R_code.sas";
%let inhtml=" C:/Users/sas/Documents/R_output/rplot1.png";
proc gslide iframe=&inhtml imagestyle=fit; run;
```

PREREQUISITES TO RUN R CODE IN SAS ASSET PERFORMANCE ANALYTICS

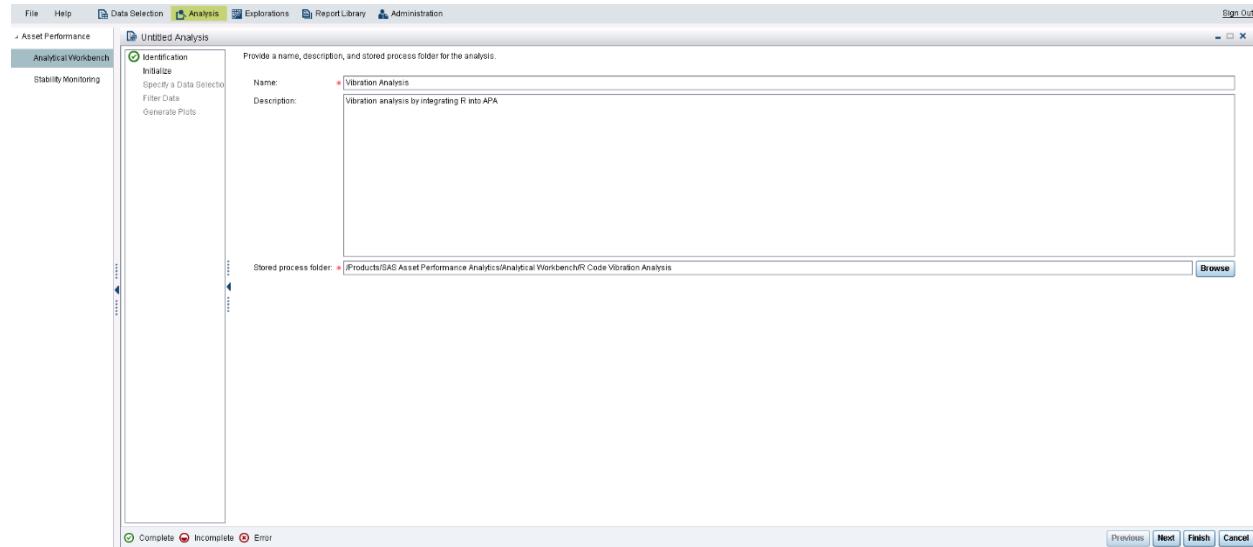
To be able to perform the developed vibration analysis within the SAS Asset Performance Analytics pre-defined analysis workspace, some prerequisite activities need to be done:

1. Download and install R version 2.15.0 or later.
2. Update the sasv9.cfg file to include the R language. To do so, open the sasv9.cfg file with Notepad and add option “-RLANG” to this file. I recommend adding -RLANG below the -TRAINLOC "" statement. Save the updated sasv9.cfg file and then close the file.
3. Import the SAS Asset Performance Analytics vibration analysis R code stored process package into SAS Asset Performance Analytics.

RUN VIBRATION SPECTRAL ANALYSIS IN SAS ASSET PERFORMANCE ANALYTICS

The vibration analysis can now be run through the pre-defined analysis within the SAS Asset Performance Analytics workbench. There are five steps in the vibration analysis.

Step 1 – Identification (Display 2): Provide a name and description for your new analysis and click Browse. Select ‘R Code Vibration Analysis’ from the drop-down list. Select OK. Click Next. This step defines which analysis is performed.



Display 2. Identification Step in the Vibration Analysis

Step 2 – Initialize (Display 3): Click Run and Save Step. This step is looking for SAS Asset Performance Analytics data selections matching the R code stored process requirements. The Output tab lists SAS Asset Performance Analytics data selections containing transposed, interpolated tag-only data with a fixed periodicity. Then click Next.

Display 3. Initialization Step in the Vibration Analysis

Step 3 – Specify a data selection (Display 4): Select an SAS Asset Performance Analytics data selection as input for the R code. Then click Run and Save Step. The Output tab shows a sample of the selected data set. Then click Next.

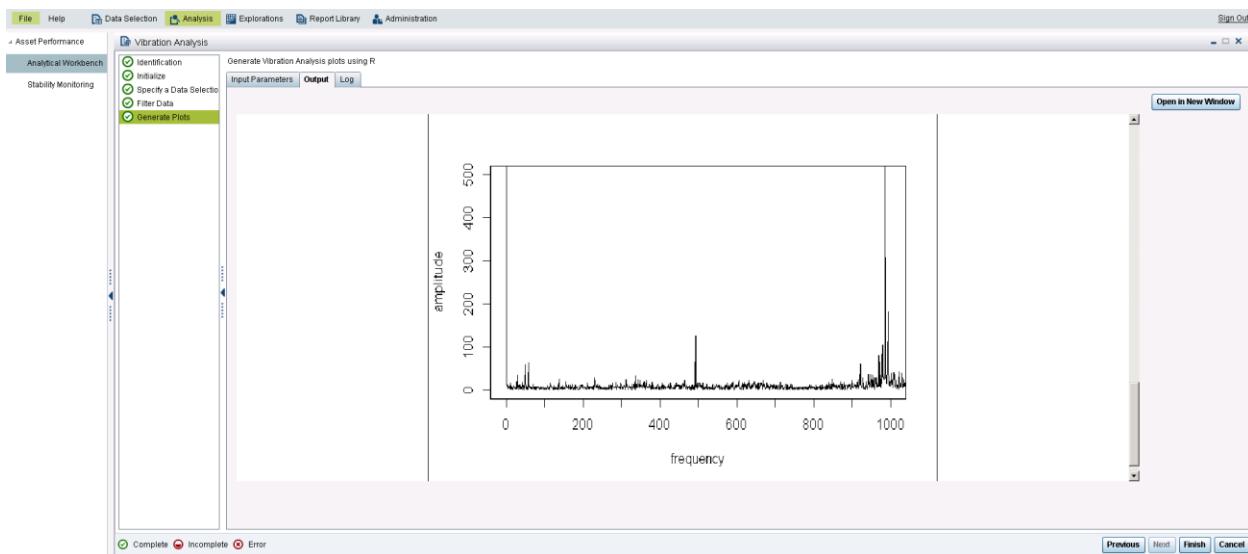
Display 4. Data Selection Step in the Vibration Analysis

Step 4 – Filter data (Display 5): Here you can define through the drop-down menu which data to include in the R code vibration analysis and which type of analysis you want to run. Then click Run and Save Step. The Output tab shows the result of the selected analysis in table format. For example, a means procedure is run on the frequency component with the highest amplitude through time for each sensor on the bearings.

Variable	Minimum	Maximum	Mean	Std Dev	Quartile Range
B1_X	985.417798	985.577778	985.768888	0	0.776657
B2_X	495.212278	1020.05	902.766732	169.873497	7.813263
B3_X	495.212278	1020.05	902.766732	169.873497	7.813263
B4_X	57.0252147	3716.11	943.725033	363.355016	14.649681
B5_X	495.212278	1020.05	902.766732	169.873497	7.813263
B6_X	495.212278	1020.05	902.766732	169.873497	7.813263
B1_Y	0	0	0	0	0
B2_Y	985.417798	985.577778	986.764656	0	0.776657
B3_Y	919.44	919.44	919.44	0	0.776657
B4_Y	495.212278	919.30	1423.07	2171.16	501.0254907
B5_Y	495.212278	919.30	1423.07	2171.16	501.0254907
B6_Y	495.212278	919.30	1648.83	2528.80	501.0254907

Display 5. Filter Data Step in the Vibration Analysis

Step 5 – Generate plots (Display 6): Click Run and Save Step to see the visual output of the analysis. For example, the Output tab shows the frequency spectrum based on a Fast Fourier Transform for one of the bearing measurements.



Display 6. Generate Plots Step in the Vibration Analysis

RESULTS AND NEXT STEPS

Based on the pre-defined vibration analysis, it is possible to identify anomalies and trigger an alert when a potential failure will happen. The anomalies can automatically be identified by monitoring, for example, the frequencies with the highest amplitude and their change, crest factor, true peak, and RMS or engineering-based features.

The analysis still falls short on diagnosing the exact failure mode. (This is a nice opportunity for advanced analytics and future work). The result of the analysis tells you when the normal pattern has changed, but still has no clue on how the specific asset works and therefore can't tell you which failure mode is developing. Fortunately, this is exactly the time for vibration analysis specialists to step in. They can draw on their deep understanding of the asset to identify and solve the specific failure mode. However, all repetitive work of sifting out abnormalities from the billions of data points has already been done. By

scheduling and automating this analysis and anomaly detection in SAS Asset Performance Analytics, engineers only focus on the alerts that are generated, and they do not waste anytime on analyzing irrelevant data as is currently the case in most manufacturing companies. Furthermore, the engineers can run the analysis on an ad hoc basis without needing expert knowledge on the underlying algorithms. When you look at Figure 4, you can clearly see how the vibration spectrum of bearing 4 changes as the degradation evolves through time. The failure happened on 25/11/2003.

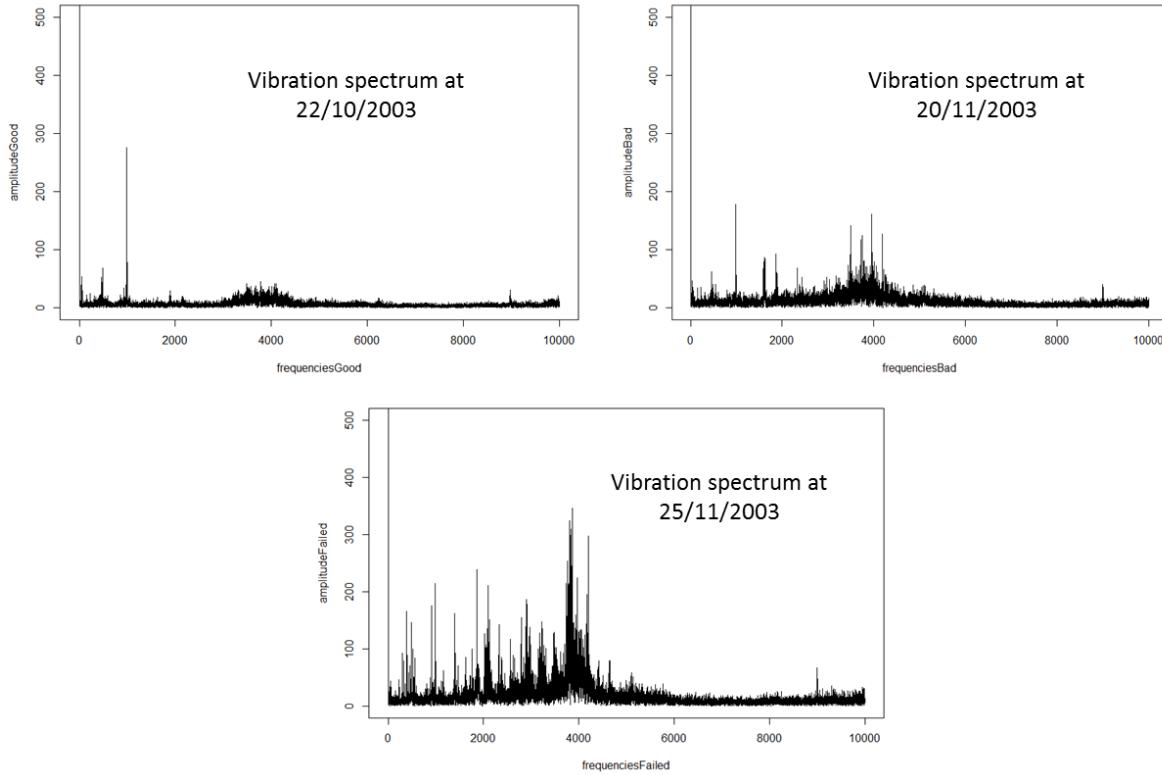


Figure 4. Evolution of Degradation in Vibration Spectrum of Bearing 4

When you look more in detail to the extracted features (15 strongest frequency components in the vibration spectrum), the same conclusion is true. If you look to the second strongest frequency component (strongest one is the DC component) over time for the x-axis of bearing 4 in Figure 4, you can see a clear shift 7 days before the failure happened. Based on this feature, an alert is triggered and a specialist can investigate this more in detail.

For next steps and further development, you can add the extraction of additional features to the vibration analysis. Moreover, you can use these features as an input for advanced analytical models to predict failures, given that you have enough historical failure events. Another option is to define a failure catalog based on the ‘finger prints’ of historical failures and use these on new data to predict and categorize failure modes accordingly. Another important next step is to add process and operational data, quality data and so on to the traditional condition monitoring data to enhance failure prediction.

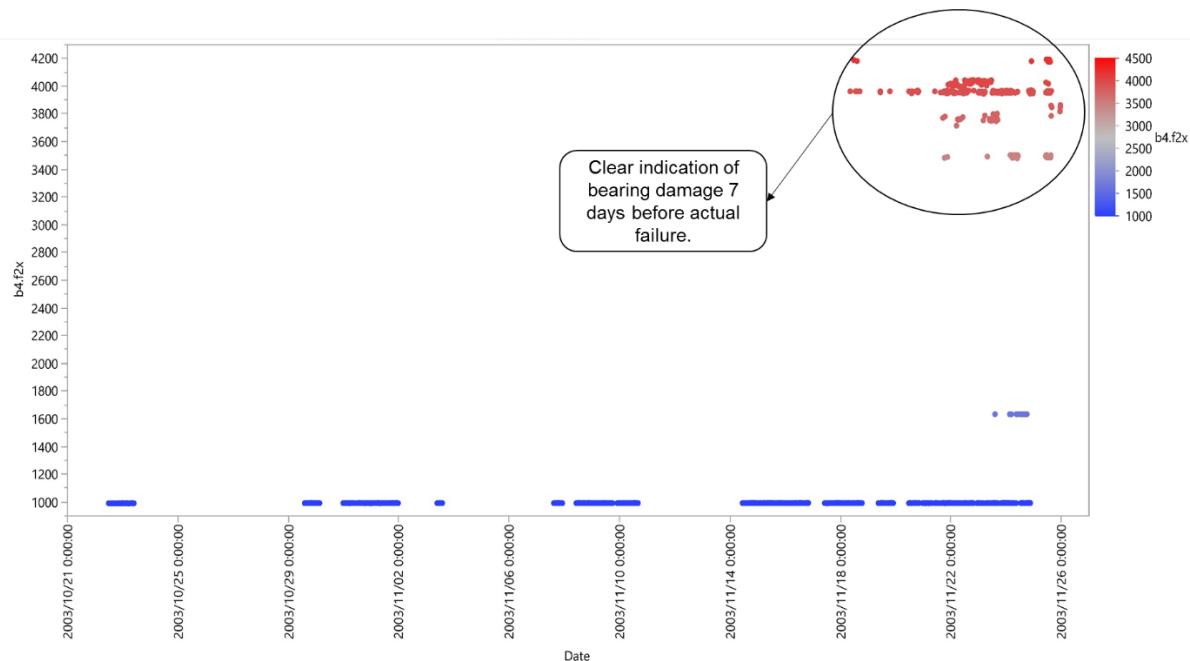


Figure 5. Detailed Analysis of the Second Largest Frequency in the Vibration Spectrum

CONCLUSION

The paper gives insight in how traditional condition-based maintenance strategies can be scaled when more sensor data becomes available due to the emergence of Industry 4.0 and the Industrial Internet of Things. It gives insight into how analytics can be used to face these challenges within asset management. More specifically, a vibration spectral analysis in R code is integrated into SAS Asset Performance Analytics as an example. The paper illustrates the possibility to perform vibration analysis as a guided analysis in the SAS Asset Performance Analytics workbench. This illustrates how data-driven decisions by engineers can be scaled within asset management to give insight into the strength of combining expert engineering knowledge with advanced data analytics techniques to improve asset performance. The performed work can be used as a starting point for the development of more advanced predictive models to predict asset failures.

REFERENCES

- Center for Intelligent Maintenance Systems. Accessed February 27, 2017. <http://www.imscenter.net/>.
- Jardine, A., D. Lin, and D. Banjevic. 2006. "A review on machinery diagnostics and prognostics implementing condition-based maintenance". *Mechanical Systems and Signal Processing* 20:1483-1510.
- Pintelon, L. and F. Van Puyvelde. 2006. Maintenance decision making. Leuven, Belgium: Acco.
- Qiu, H., J. Lee, J. Lin, and G. Yu. 2006. "Wavelet Filter-based Weak Signature Detection Method and its Application on Roller Bearing Prognostics." *Journal of Sound and Vibration* 289(4):1066-1090.

ACKNOWLEDGMENTS

I would like to thank Marcia Walker for reminding me of the opportunity to submit an abstract to the SAS Global Forum 2017. Without this encouragement, there would have never been a paper. I would also like to thank Diana Shaw for the specific help on integrating R code into SAS Asset Performance Analytics and for always being available when I had any questions.

RECOMMENDED READING

- *Practical Machinery Vibration Analysis and Predictive Maintenance* by Cornelius Scheffer and Paresh Girdhar. 2004.

CONTACT INFORMATION

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

Adriaan Van Horenbeek
SAS Institute Inc.
+32 499 56 56 98
Adriaan.van.horenbeek@sas.com

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.

Ready to take your SAS® and JMP® skills up a notch?



Be among the first to know about new books,
special events, and exclusive discounts.

support.sas.com/newbooks

Share your expertise. Write a book with SAS.

support.sas.com/publish

 sas.com/books
for additional books and resources.


THE POWER TO KNOW®