Paper SAS313-2014

An Overview of Machine Learning with SAS® Enterprise Miner™

Patrick Hall, Jared Dean, Ilknur Kaynar Kabul, Jorge Silva
SAS Institute Inc.

ABSTRACT

SAS® and SAS® Enterprise Miner™ have provided advanced data mining and machine learning capabilities for years—beginning long before the current buzz. Moreover, SAS has continually incorporated advances in machine learning research into its classification, prediction, and segmentation procedures. SAS Enterprise Miner now includes many proven machine learning algorithms in its high-performance environment and is introducing new leading-edge scalable technologies. This paper provides an overview of machine learning and presents several supervised and unsupervised machine learning examples that use SAS Enterprise Miner. So, come back to the future to see machine learning in action with SAS!

INTRODUCTION

Machine learning is a branch of artificial intelligence that is concerned with building systems that require minimal human intervention in order to learn from data and make accurate predictions. SAS Enterprise Miner offers many machine learning procedures and resources. The power tools for prediction and classification include high-performance Bayesian networks, neural networks, random forests, and support vector machines. High-performance clustering enables you to segment your biggest data and offers a novel algorithm for choosing the best number of clusters: the aligned box criterion. As a rigorously tested domain-specific fourth-generation programming language (4GPL) that offers native performance, the SAS language is a powerful machine learning research tool and is an ideal platform for numerically sensitive applications and larger data sources.

The following sections provide a short history and overview of machine learning along with a taxonomy of prominent machine learning algorithms and several practical examples. Each example uses machine learning algorithms from SAS Enterprise Miner and data from the Kaggle predictive modeling competitions: document classification with the EMC Israel Data Science Challenge data, *k*-means clustering with the Claim Prediction Challenge data, and deep learning with the MNIST Digit Recognizer data. SAS source code for the examples can be downloaded from http://support.sas.com/saspresents. Appendix A provides references for further reading and a guide for implementing machine learning algorithms in SAS Enterprise Miner.

MACHINE LEARNING BACKGROUND

Throughout the 1940s, breakthroughs in psychology and mathematical biophysics laid the foundation for early computer-like machine simulation of neural processes (Rashevsky 1948; McCulloch and Pitts 1943). In 1958, the perceptron algorithm, a neural learning technique, was introduced (Rosenblatt 1958). In the following years, machine learning researchers created similar neural modeling algorithms and hardware. After the initial flurry of theoretical and practical research, Minsky and Papert (1969) defined the rather serious limitations of single layer neural networks. In the face of high expectations, minimal computing power, and theoretical limitations, subsequent neural network research languished for years.

Symbolic concept acquisition approaches grew from fundamental research in artificial intelligence (Hunt, Marin, and Stone 1966). These symbolic logic systems used logical and graph structures to represent higher-level knowledge, as opposed to the numerical functions in neural networks. Early successes of symbolic logic in the 1960s eventually fell prey to the same unrealistic expectations and computational intractability that thwarted early neural network research (Lighthill 1973). However, rule-building successes in specific problem domains throughout the 1970s, along with the PROLOG computer language and dialects of the LISP computer language, led to commercially viable decision-support products, known as expert systems, in the 1980s (Waterman and Hayes-Roth 1978; Giarratano and Riley 1998).

Neural network research was also revitalized in the 1980s with the introduction of back-propagation of errors and ensuing multilayer perceptrons (Rumelhart, Hinton, and Williams 1986). Classification and regression tree (CART) methods were also embraced widely during the 1980s (Brieman et al. 1984). Such innovations, along with the advent of large digital data sources and relatively inexpensive yet powerful computers, became the foundation for many aspects of contemporary machine learning. Through the next decade, neural networks and decision trees were assimilated into the larger fields of data mining and statistics, while an academically diverse group of researchers pursued the next generation of learning algorithms. Breiman (2001a) observes:

> In the past fifteen years, the growth in algorithmic modeling applications and methodology has been rapid. It has occurred largely outside statistics in a new community—often called machine learning.

Machine learning today is a mathematically rigorous discipline that encompasses sophisticated modeling, optimization, and learning research; it has concrete applications in medicine, software, robotics, and traditional business problems. Particularly in the business problem domain, there is significant overlap among the fields of data science, data mining, and machine learning. Figure 1 illustrates the multidisciplinary nature of data mining, data science, machine learning, and related areas. This graphic was originally created by SAS in a 1998 primer about data mining, and the new field of data science was added for this paper.

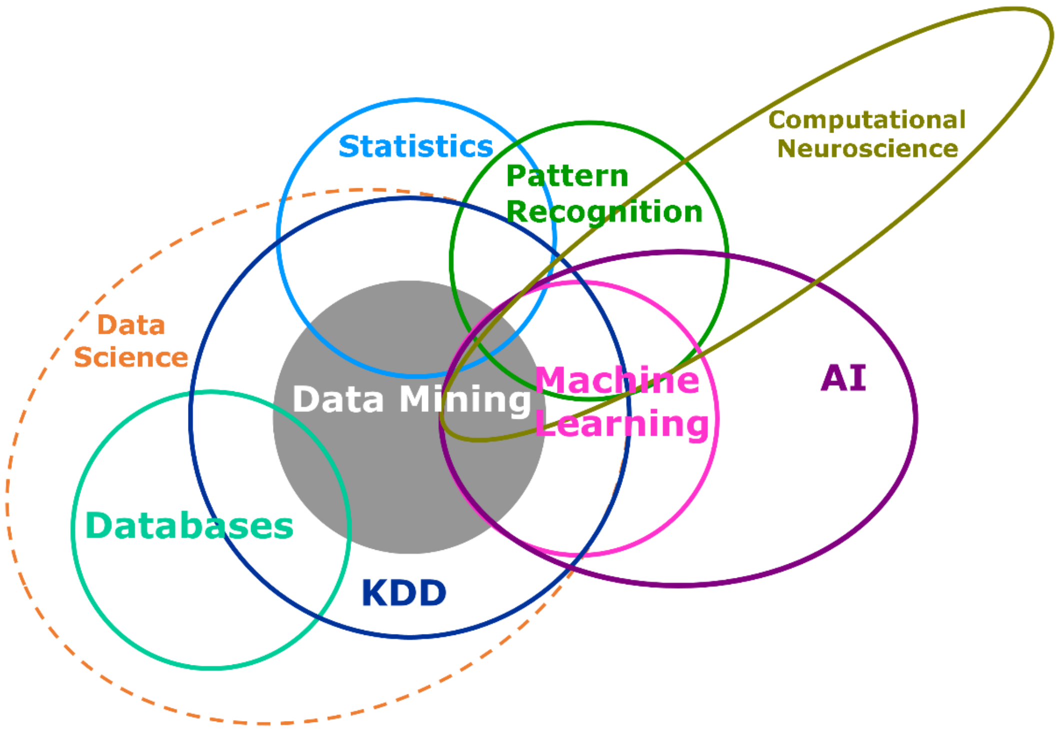

Figure 1. Multidisciplinary Nature of Machine Learning

MACHINE LEARNING, SAS, AND SAS ENTERPRISE MINER

This section introduces machine learning for SAS users and briefly discusses the primary SAS products that customers can use for machine learning. Many topics in machine learning will be familiar to experienced statisticians and data miners. However, dissimilar terminology among data mining, data science, machine learning, and statistics might cause confusion. Table 1 maps machine learning terms to terms that are used in other disciplines.

Machine Learning Term	Multidisciplinary Synonyms
Feature, input	Independent variable, variable, column
Case, instance, example	Observation, record, row
Label	Dependent variable, target
Train	Fit
Class	Categorical target variable level

Table 1: Mapping between Common Vocabulary Terms in Data Analysis Fields

In contrast to many statistical modeling approaches, which can value inference over prediction, the focus of machine learning is predictive accuracy (Breiman 2001a). High predictive accuracy is usually achieved by training complex models, which often involve advanced numerical optimization routines, on a large number of training examples. Because of their almost uninterpretable internal mechanisms, some machine learning algorithms have been labeled "black box" techniques. Yet algorithms such as neural networks, random forests, and support vector machines can learn faint and nonlinear patterns from training data that generalize well in test data. Outlining these prominent types of learning algorithms is the emphasis of this section, but there are numerous other important facets of machine learning, such as preprocessing training examples to reduce noise and cross-validating models to prevent overfitting.

Machine learning algorithms are divided into several subcategories, of which supervised and unsupervised learning techniques are the most widely applied in other disciplines, particularly in data mining (see Figure 1). Additional machine learning research areas include semi-supervised learning, transduction, reinforcement learning, and developmental learning. Supervised, unsupervised, and semi-supervised learning algorithms are deployed extensively in business applications and are the subject of the discussions and examples in this paper.

The remainder of this section presents a taxonomy of machine learning algorithms in which supervised, unsupervised, and semi-supervised learning algorithms lie within the intersection of the machine learning and data mining fields (see Figure 2). In this taxonomy, other machine learning research areas such as transduction, reinforcement learning, and developmental learning are outside the scope of data mining. Because machine learning is closely associated with data mining, SAS Enterprise Miner, a data mining product built on the SAS platform, offers many scalable solutions in the important common space that is shared between data mining and machine learning: supervised, unsupervised and semi-supervised learning.

Supervised learning refers to techniques that use labeled data to train a model. Supervised learning consists of prediction ("regression") algorithms for interval labels and classification algorithms for class labels. Generally, supervised learning algorithms are trained on labeled, preprocessed examples and assessed by their performance on test data. Example 1 describes a typical supervised learning task (document classification), in which an extremely large set of input features are preprocessed into a smaller set of inputs, different learning algorithms are trained, and the algorithms are assessed by their performance on test data. Other common business applications of supervised learning include voice recognition and insurance claims prediction.

Unsupervised learning occurs when a model is trained on unlabeled data. Unsupervised learning algorithms usually segment data into groups of examples or groups of features. Groups of examples are often called "clusters." Combining training features into a smaller, more representative group of features is called "feature extraction," and finding the most important subset of input features is called "feature selection." Unsupervised learning can be the end goal of a machine learning task (as it is in the market segmentation in Example 2), or it can be a preliminary or preprocessing step in a supervised learning task in which clusters or preprocessed features are used as inputs to a supervised learning algorithm.

Semi-supervised learning algorithms generally train on small portions of labeled data that are then combined with larger amounts of unlabeled data, combining supervised and unsupervised learning in situations where labeled data are difficult or expensive to procure. Several stand-alone machine learning algorithms have also been described as semi-supervised because they use both labeled and unlabeled training examples as inputs (Belkin, Niyogi, and Sindhwani 2006; Bengio 2009; Joachims 1999; Nigam et al. 2000). In Example 3, a semi-supervised algorithm is used for image recognition. Although they are less common, semi-supervised algorithms are garnering acceptance by business practitioners.

Figure 2 lists some of the most common algorithms in supervised, unsupervised, and semi-supervised learning. Tables in Appendix A list the algorithms that are used in each type of learning, show the names of corresponding SAS procedures and SAS Enterprise Miner nodes, and provide references where you can find more detailed information. For the few mainstay algorithms that are not currently implemented by SAS, you can pursue custom solutions by using Base SAS®, SAS/STAT®, SAS/OR®, and SAS/IML® tools. SAS integration with R and Java add even greater extensibility to the SAS platform. Integration with R is available through SAS/IML and the Open Source Integration node in SAS Enterprise Miner. A Java API is provided by the Base SAS Java object.

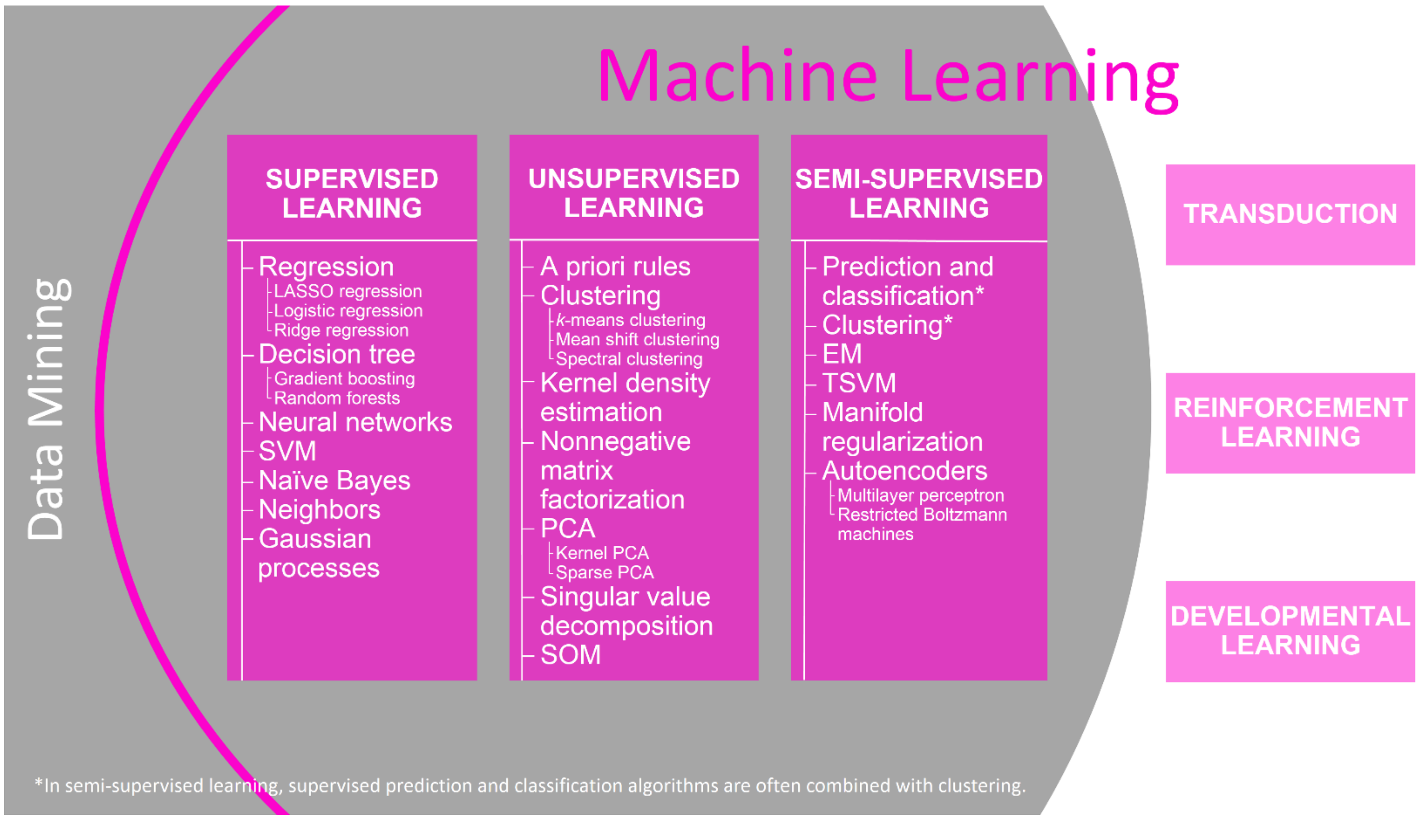

Figure 2. Machine Learning Taxonomy

EXAMPLES

EXAMPLE 1: SUPERVISED LEARNING WITH THE KAGGLE EMC ISRAEL DATA SCIENCE CHALLENGE DATA

Document classification is a popular area of machine learning research that has numerous applications (Sebastiani 2002). The EMC Israel Data Science Challenge data represent source code files as raw term counts, and they use the repository to which each file belongs as a class label. The data contain 592,158 input features and approximately 175,000 training examples. Each input feature represents a source code token, and each example contains the raw token counts from a particular source code file. The 97 levels of the class label represent 97 different source code repositories. In this example, predictive features are selected from the training data and several supervised models are trained to classify examples. The EMC Israel Data Science Challenge data are available from the Kaggle website. As in most text frequency data, the input features are sparsely populated with nonzero values. However, no special measures are taken to treat the training data as sparse, and this example essentially demonstrates the ability of Base SAS and the high-performance SAS Enterprise Miner procedures to handle wide tables in a straightforward supervised learning task. Advanced functionality for scalable text mining is available through SAS® Text Miner.

The example data are stored as plain text in compressed sparse row (CSR) format. Appendix B shows a SAS DATA step approach to building the training data. Code for this example is also available from http://support.sas.com/saspresents. After building the training data and setting the grid system options, you can use the entire set of input features and the HPBNET procedure to establish a naïve Bayes classification benchmark and to quickly determine a highly predictive subset of input features as follows:

```
proc hpbnet data= emcIsraelFull structure= naive maxparents= 2
            indeptest= mi mialpha= 0.05;
     target target;
     input token:;
     output  pred= emcIsraelFullPred
             varlevel= emcIsraelFullVarLev
             varselect= emcIsraelFullVarSel;
     performance nodes= all;
run;
```

Multiclass logarithmic loss is used to assess predictive models in high-cardinality classification problems. Multiclass logarithmic loss is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}y_{i,j}\log(p_{i,j})$$

where *N* is the number of examples, *M* is the number of features, $y_{i,j}$ is equal to 1 when the label of observation *i* is equal to *j* and is 0 otherwise, and $p_{i,j}$ is the probability that the predicted label is equal to *j*.

The following statements use the prediction outputs from PROC HPBNET and a SAS DATA step to calculate the multiclass logarithmic loss:

```
*** OUTPUT MULTI-CLASS LOGARITHMIC LOSS TO LOG;
data _null_;
    set emcIsraelFullPred end= eof;
    array posteriorProbs p_:;
    retain logloss 0;
    logloss + log(posteriorProbs[(96-target)+1]);
    if eof then do;
           logloss= (-1*logloss)/_n_;
           put logloss= ;
    end;
run;
```

The example naïve Bayes model yields a multiclass logarithmic loss of 4.19 for the training examples.

PROC HBNET also computes useful quantities for feature selection. You can use the variable-level output to find the tokens that appear in the most source code files. By modifying the SAS DATA step code in Appendix B, you can generate a new training set that contains only these most common tokens. The variable selection output from PROC HPBNET also contains mutual information criterion (MIC) values for each input feature. MIC measures the amount of information that is shared between the input feature vector and the target label vector. You can also adapt the DATA step code in Appendix B to select the input features that contain the tokens that have the highest MIC.

After you have selected an appropriate subset of features from the training data, you can apply a stronger learning algorithm. In this example, the 200 most frequently occurring tokens and the 200 tokens that have the highest MIC are selected as separate sets of preprocessed inputs. Both sets of selected features are imported into SAS Enterprise Miner to train high-performance forest and high-performance neural network classifiers. The forest algorithm is run with the default settings, 50 trees, and three random input features to be tried in each split search. The neural algorithm is also run with a basic configuration, 500 hidden neurons in a single layer, a maximum of 1,000 iterations, and a validation set. Such an analysis produces a diagram similar to Figure 3. For simplicity, 200 input features and straightforward node property settings are chosen. Experienced users are encouraged to experiment with a larger number of input features and the many options available in the various high-performance modeling nodes.

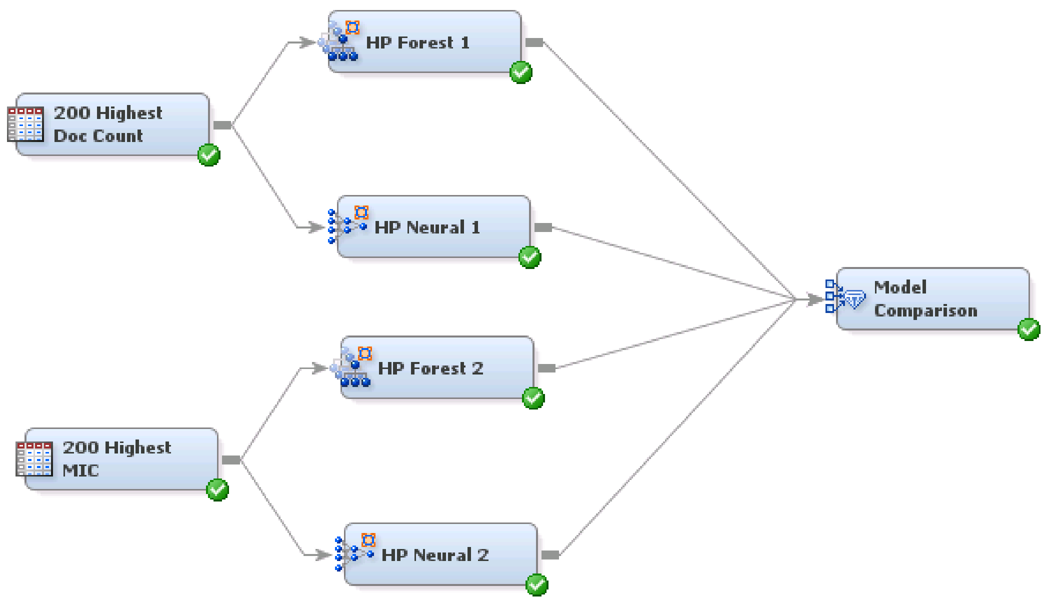

Figure 3: SAS Enterprise Miner Flow Comparing Selected Features and Different Classifiers

The misclassification rate on a holdout set of examples is another standard measurement of classification accuracy for supervised learning. The high-performance forest node reports out-of-bag (OOB) misclassification, and the high-performance neural node reports validation misclassification when a validation set is generated. You can use logarithmic loss, along with misclassification rate, to assess the features you have selected and the models you have trained. The modeling node results and the Model Comparison node results contain misclassification rates. You can enter the previous DATA step in a SAS Code node to calculate logarithmic loss. Table 2 summarizes the findings of the comparison, including multiclass logarithmic loss, which is calculated for both training sets in their entirety. By default, OOB misclassification is calculated from a 40% out-of-bag sample, and validation misclassification is reported for a 25% validation set.

Random Forest	200 Highest Document Count	200 Highest MIC
Logarithmic loss	2.11	2.91
OOB misclassification	0.5	0.54
Neural Network	**200 Highest Document Count**	**200 Highest MIC**
Logarithmic loss	0.09	0.22
Validation misclassification	0.23	0.28

Table 2: EMC Israel Data Science Challenge Data Classification Results

For the input features and algorithm settings in this example, results indicate that neural networks are the strongest predictors and that features selected by document count are more representative than features selected by MIC. Based on these results, neural networks that are trained on features selected by high document count are the preferred classifiers for the example data. You could probably improve the results in this example by ensembling different strong learning algorithms and by using more inputs and advanced options with the learning algorithms. Ensembling is a common approach for increasing the accuracy of predictive models and is easy to implement by using the Ensemble node in SAS Enterprise Miner.

EXAMPLE 2: UNSUPERVISED LEARNING WITH THE KAGGLE CLAIM PREDICTION CHALLENGE DATA

One of the most prominent business applications of unsupervised learning is market segmentation. Although market segmentation is a broad topic, many techniques involve clustering demographic and product data into groups of representative customers. Perhaps the most difficult aspect of any clustering task is determining the number of clusters, *k*. This example compares three approaches for determining *k* for the Kaggle Claim Prediction Challenge data. These data provide an adequately realistic set of input features at a reasonable example scale—approximately 13 million cases.

Although many methods have been proposed to estimate the number of clusters in a data set, this example uses the cubic clustering criterion (CCC), gap statistic, and aligned box criterion (ABC). Introduced by SAS in 1983 (Sarle 1983), CCC uses heuristic approximation to compare the R^2 statistic of a *k*-cluster solution for the input features to the R^2 statistic in a uniform hyperbox. The gap statistic, introduced in the early 2000s (Tibshirani, Walther, and Hastie 2001), compares the within-cluster sum of squares (WSS) between a *k*-cluster solution for the input features to the WSS of a Monte Carlo–simulated reference distribution. The HPCLUS procedure, new in SAS Enterprise Miner 13.1, implements a new algorithm for determining *k*: the aligned box criterion (ABC). With ABC, SAS has increased the accuracy of previous approaches while using advances in distributed computing to retain scalability. ABC compares WSS between a *k*-cluster solution for the input features to the WSS of a data-adaptive, Monte Carlo–simulated reference distribution.

The example Claim Prediction Challenge data are available from the Kaggle website. After you download the data, import it into SAS. Then you can use SAS Enterprise Miner to preprocess the original training features. You can use the CCC, gap statistic, and ABC with any clustering technique that minimizes a WSS measure. This example uses the traditional *k*-means approach. Data preprocessing is required because the *k*-means method is valid only for interval inputs and is highly susceptible to outliers, differences in input feature scales, and high correlation between input features. In addition to aggregating the separate cases to a suitable customer or household level, the following preprocessing steps are strongly recommended before you attempt to cluster or determine *k*:

- replacement or removal of interval outliers
- binning of rare categorical input levels
- standardization of interval inputs
- appropriate numeric encoding of categorical inputs
- combining or removing highly correlated features

The results in this example are obtained by using a binary encoding of categorical inputs; therefore, categorical variables that have high cardinality are dropped from the input feature set prior to encoding. This example uses the FASTCLUS procedure in SAS/STAT and the Open Source Integration node and HPCLUS procedure in SAS Enterprise Miner to generate the CCC, gap statistic, and ABC values, respectively. The Open Source Integration node enables comparison with the gap statistic method through the R language `cluster` package. To avoid memory restrictions in the gap statistic `clusGap()` function, the preprocessed data are sampled to approximately 13,000 examples (0.1%). The CCC, gap statistic, and ABC for the sample are calculated on a single machine. ABC is also calculated in distributed mode on a Teradata appliance for the entire training data set.

Figure 4 illustrates calculation times for the CCC, gap statistic, and ABC on the 0.1% sample and for ABC on the full training set. The timings in Figure 4 are not a conclusive benchmark; they simply reflect the differences between the algorithms, their use of multithreading, and their use of available computational resources. CCC is extremely fast because it takes advantage of heuristic approximation. Although both the gap statistic and ABC use Monte Carlo techniques, the gap statistic implementation is single-threaded, whereas the ABC calculation uses 12 threads in single-machine mode and several hundred threads in distributed mode.

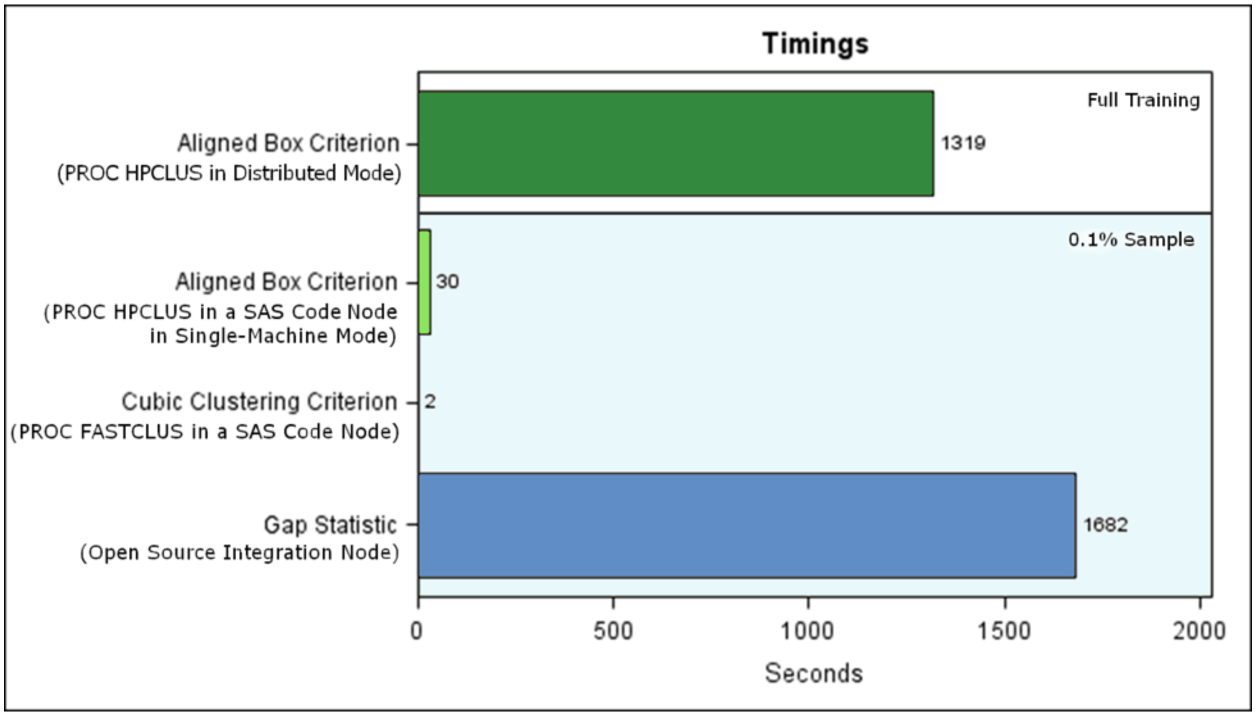

Figure 4. Calculation Times for the CCC, Gap Statistic, and ABC

After preprocessing the example data, you can use PROC FASTCLUS and PROC HPCLUS inside the SAS Code node, along with the Open Source Integration node, to estimate a best number of clusters for the 0.1% sample. Base SAS 9.4 and PROC HPCLUS are used to estimate the best number of clusters for the full training data. Code snippets are presented here and are available from http://support.sas.com/saspresents.

This SAS macro can be used inside a SAS Code node to find CCC values for *k* from 1 to an arbitrary positive integer that is defined by the `maxk` argument:

```
 %macro getCCC(maxK= 20);
    %do i= 1 %to &maxK;
           proc fastclus
                  noprint
                  data= &EM_IMPORT_DATA
                  maxclusters= &i
                  outstat= o(where= (_TYPE_= 'CCC') keep= _TYPE_ OVER_ALL);
                  var %EM_INTERVAL_INPUT;
           run;
           proc append base= CCCOut data= o; run;
    %end;
    proc print data= CCCOut; run;
%mend;
```

The following R code snippet produces gap statistic values for *k* from 1 to 20 when it is submitted in the Open Source Integration node. This code also directs the `clusGap` function to use *k*-means clustering on the node's imported training data and to use 10 Monte Carlo (B=10) draws in the reference distribution.

```
library('cluster')
set.seed(12345)
gskmn <- clusGap(&EMR_IMPORT_DATA, FUN= kmeans, K.max= 20, B= 10)
gskmn
```

When you specify the NOC option in the PROC HPCLUS statement, the procedure outputs ABC values and estimates a best number of clusters. The following NOC suboptions instruct PROC HPCLUS to calculate ABC values by using 25 Monte Carlo draws (B=25) in the reference distribution for each *k* from 1 (MINCLUSTERS=1) to 20 (MAXCLUSTERS=20), to use no reference distribution alignment, and to select the estimated best number of clusters by a combination of all selection criteria. The NTHREADS option in the PERFORMANCE statement instructs PROC HPCLUS to run on 12 threads. (Because no NODES= option is specified in the PERFORMANCE statement and the data are not alongside the database, PROC HPLCUS runs in single-machine mode by default.)

```
proc hpclus
     data= &EM_IMPORT_DATA maxclusters= 20 maxiter= 15
     noc= abc(b= 25 minclusters= 1 align= none criterion= all);
     input %EM_INTERVAL_INPUT;
     performance nthreads= 12;
run;
```

You can use similar settings to execute PROC HPCLUS in distributed mode. After properly configuring your distributed environment, use a distributed data source and the NODES= option in the PEFORMANCE statement to find ABC values for *k* from 1 to 20 and to estimate a best number of clusters, as follows:

```
proc hpclus
     data= gridlib.kaggleClaimPrediction maxclusters= 20 maxiter= 15
     noc= abc(b= 25 minclusters= 1 align= none criterion= all);
     input _ALL_;
     performance nodes= all;
run;
```

Figure 5 illustrates the resulting measurements of each example run. Despite apparent noise in small sample measurements and varying interpretation methodologies, the three algorithms display somewhat parsimonious results. Executing on small samples, the CCC, gap statistic, and ABC point to numerous similar candidate solutions. Operating on the full training set, ABC exhibits local maxima at 2, 4, 9, 14, and 18 clusters, with two clusters as the greatest positive local maxima and thus the suggested best estimate for *k*. Because clusters in highly dimensional, noisy data sources can be overlapping and nonspherical, a true *k*-means solution for *k* might not exist. In practice, ABC enables you to select the most practically viable estimate for *k* from the statistically supported solutions at positive local maxima.

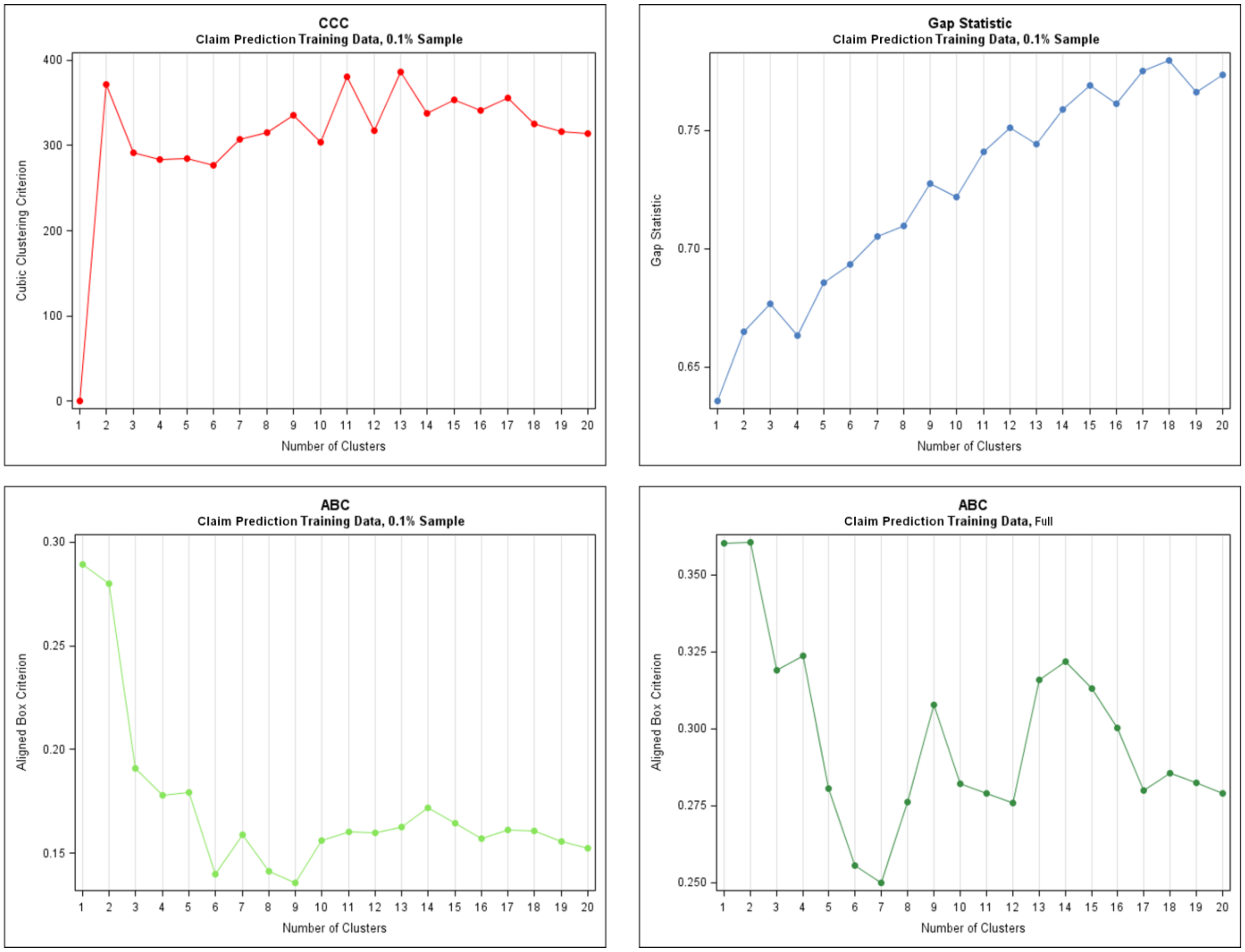

Figure 5. CCC, Gap Statistic, and ABC Values for the Kaggle Claim Prediction Challenge Data

EXAMPLE 3: SEMI-SUPERVISED LEARNING WITH THE MNIST DATA

Deep learning is a machine learning field that is based on endowing large neural networks with multiple hidden layers in order to learn features that have strong predictive abilities. Although deep learning technologies are descendants of earlier neural networks, they take advantage of unsupervised and semi-supervised learning coupled with sophisticated optimization techniques and innovative computational platforms to achieve state-of-the-art accuracy.

To extract representative features from training examples, deep neural networks are often built by stacking autoencoders, which are a special type of single-layer neural network (Hinton and Salakhutdinov 2006). Autoencoders are trained by using the same unlabeled inputs as both training examples and target labels. A denoising autoencoder is trained by randomly corrupting the input matrix of the autoencoder. Because autoencoders do not use the training example labels as targets but instead use the training examples themselves, they have been categorized as a semi-supervised learning technique. Each layer of the deep network is usually trained separately by using the output of the prior layer, or by using the training inputs in the case of the first layer. The weights of the individually trained layers are then used to initialize the entire deep network, and all layers are trained again simultaneously on the original training examples. When a large number of inputs is used in conjunction with a much smaller number of hidden units, the features that are extracted as outputs of the hidden units are a nonlinear projection of the training examples onto a lower-dimensional space. Such features can be highly predictive of a training example's class label. This example uses the NEURAL procedure, which provides multilayer perceptron architectures for autoencoders. Literature sources cite restricted Boltzmann machine architectures for autoencoders (Hinton and Salakhutdinov 2006; Vincent et al. 2008; Bengio 2009), but these architectures are not currently available in PROC NEURAL.

You can train a stacked autoencoder in separate stages by using Base SAS and SAS Enterprise Miner, or you can use the NEURAL procedure, which provides a nuanced syntax that enables a stacked denoising autoencoder to be trained in a single, continuous procedure execution. In this example, PROC NEURAL builds a stacked denoising autoencoder from the Mixed National Institute of Standards and Technologies (MNIST) digits data. Highly predictive features are extracted from the middle layer of the deep autoencoder by using the input data and PROC NEURAL score code, and extracted features are compared to principal components.

Figure 6 illustrates a stacked denoising autoencoder. To train a stacked denoising autoencoder, the labels should be the original, unlabeled training inputs, and the examples should be slightly corrupted copies of the same unlabeled training inputs, as described in the next paragraph. Each layer of hidden units is usually trained separately on the output of the prior layer, and then the entire network is retrained. The features that are output by the middle layer (layer h3 in Figure 6) can be used as inputs to subsequent predictive models. The layer identifiers in Figure 6 correspond to the layer identifiers in the INPUT and HIDDEN statements in the example autoencoder code, which is available in this section and from http://support.sas.com/saspresents.

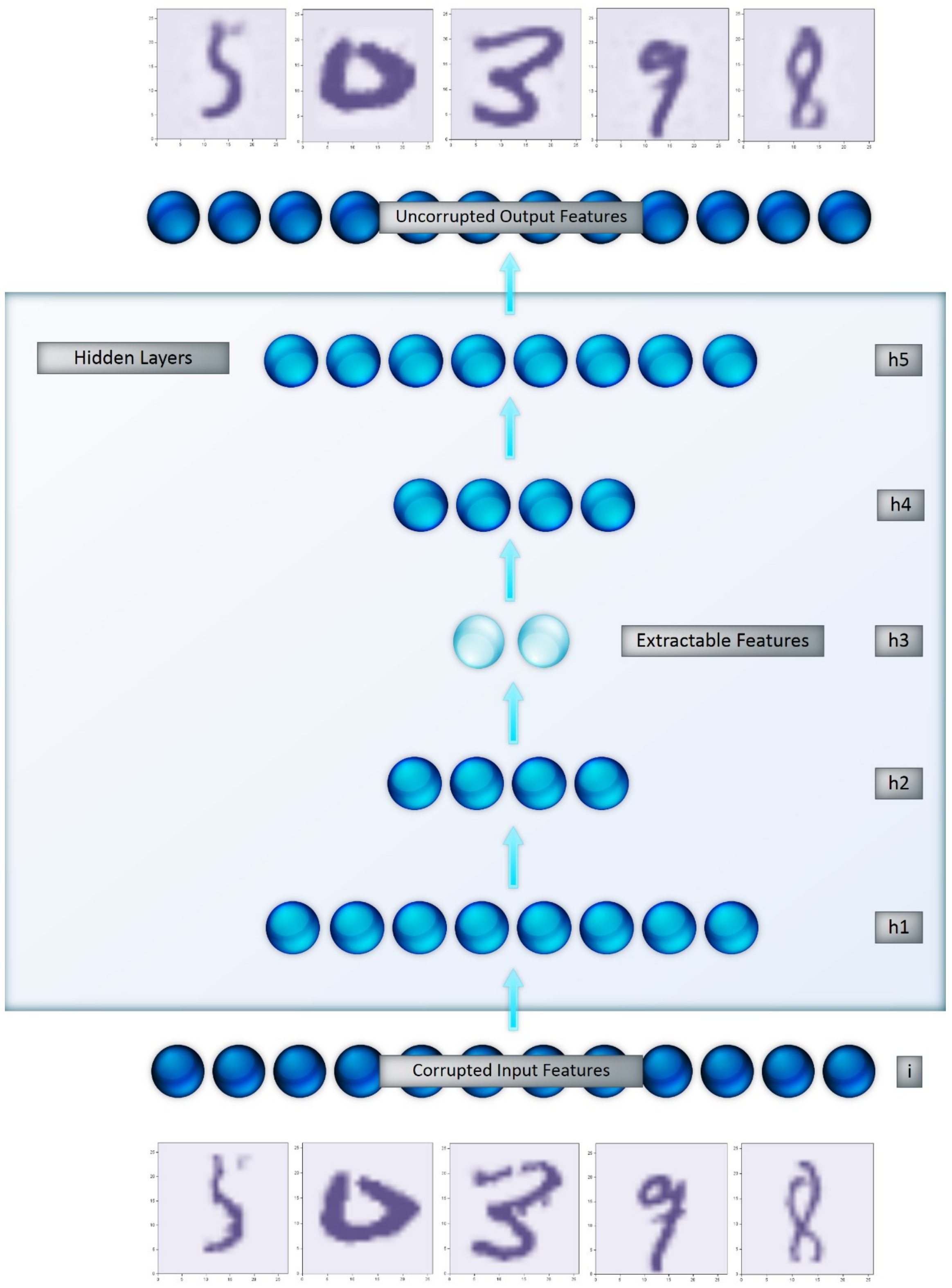

Figure 6. Stacked Denoising Autoencoder

The MNIST data constitute a famous data set that contains thousands of handwritten digit images, with labels from 0 through 9. Correctly labeling these digit images is a classic problem in machine learning research. The MNIST data are widely available. After you procure and import the data into SAS, it is recommended that you center the images (Vogt 1994) and drop the outer pixels that are blank for every training example. To create the training examples as a corrupted copy of the original examples, use a SAS DATA step to copy the approximately 400 remaining inputs and inject them with noise. You can inject them with noise by setting random pixels to 0 in each image. After the original and corrupted pixels are available, use the DMDB procedure in SAS Enterprise Miner to create the required data mining database catalog for PROC NEURAL and use the PRINTTO procedure in Base SAS to redirect the output.

The following PROC NEURAL code defines and trains a denoising autoencoder. The PERFORMANCE statement directs PROC NEURAL to execute in multiple threads during the training process, which dramatically reduces training time. (The number of threads should not exceed the number of physical processors on a machine.) The statements define a stacked network that has five hidden layers, where the corrupted pixels are the training inputs and the original pixels are the training labels. The actual class labels are ignored. The correct numbers of hidden layers and neurons depend on the training data, available optimization routines, and the desired result. The odd number of layers in an hourglass configuration creates a single, low-dimensional middle layer, which is convenient for feature extraction. Generally, the larger the number of input features and examples, the more layers and neurons are required for high accuracy. Increasing the number of layers and neurons can cause overfitting and requires additional sophistication in the optimization routine. The number of layers and neurons in this example strikes the best balance between overfitting and the optimization capabilities of PROC NEURAL.

The PRELIM statement in the example code enables PROC NEURAL to locate suitable initial weights by conducting 10 preliminary trainings from a random initialization. The subsequent FREEZE and THAW statements train individual layers by using the weights that are located by the best initial run. After each layer is trained separately, all the layers are trained again together by using their most recent weights as a starting point for the final training. Conjugate gradient optimization is recommended for all training stages (Ngiam et al. 2011). The CODE statement then creates a score code file before PROC NEURAL terminates.

```
proc neural
       data= autoencoderTraining
       dmdbcat= work.autoencoderTrainingCat;
       performance compile details cpucount= 12 threads= yes;

       /* DEFAULTS: ACT= TANH COMBINE= LINEAR */
       /* IDS ARE USED AS LAYER INDICATORS ╥ SEE FIGURE 6 */
       /* INPUTS AND TARGETS SHOULD BE STANDARDIZED */
       archi MLP hidden= 5;
       hidden 300 / id= h1;
       hidden 100 / id= h2;
       hidden 2 / id= h3 act= linear;
       hidden 100 / id= h4;
       hidden 300 / id= h5;
       input corruptedPixel1-corruptedPixel400 / id= i level= int std= std;
       target pixel1-pixel400 / act= identity id= t level= int std= std;

       /* BEFORE PRELIMINARY TRAINING WEIGHTS WILL BE RANDOM */
       initial random= 123;
       prelim 10 preiter= 10;

       /* TRAIN LAYERS SEPARATELY */
       freeze h1->h2;
       freeze h2->h3;
       freeze h3->h4;
       freeze h4->h5;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze i->h1;
       thaw h1->h2;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze h1->h2;
       thaw h2->h3;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze h2->h3;
       thaw h3->h4;
       train technique= congra maxtime= 129600 maxiter= 1000;

       freeze h3->h4;
       thaw h4->h5;
       train technique= congra maxtime= 129600 maxiter= 1000;

       /* RETRAIN ALL LAYERS SIMULTANEOUSLY */
       thaw i->h1;
       thaw h1->h2;
       thaw h2->h3;
       thaw h3->h4;
       train technique= congra maxtime= 129600 maxiter= 1000;

       code file= 'C:\Path\to\code.sas';
run;
```

The following DATA step extracts features from the middle layer of the autoencoder by applying the generated score code to the training set and keeping only the digit labels and the output from the two middle hidden units:

```
data extractedFeatures;
     set autoencoderTraining;
     %include 'C:\Path\to\code.sas';
     keep label h31 h32;
run;
```

It is instructive to compare the extracted features to the first two principal components. You can use the PRINCOMP procedure in SAS/STAT to calculate the principal components of the pixels. Strikingly different images result from using the SGPLOT procedure in Base SAS to plot both the principal components and the features that are extracted from the deep autoencoder by class label. The classes are largely overlapping and indistinguishable in the principal components space. Conversely, classes are clustered in the extracted feature space, indicating that information has been preserved despite the extreme dimension reduction. Figure 7 shows the first two principal components of the MNIST digit data and features that are extracted from a stacked denoising autoencoder. Given that the raw MNIST data contain 784 features, the clustered nature of the two-dimensional autoencoder projection is remarkable. When generated correctly, the features extracted from denoising autoencoders can be more representative than the features extracted by standard principal components analysis or by singular value decomposition. Experienced users can extract a larger set of denoising autoencoder features to use as preprocessed inputs for a supervised learning algorithm. If you follow a feature extraction methodology similar to the example and use a large ensemble of support vector machines and multilayer neural nets to predict digit class, you can achieve test classification greater 0.99.

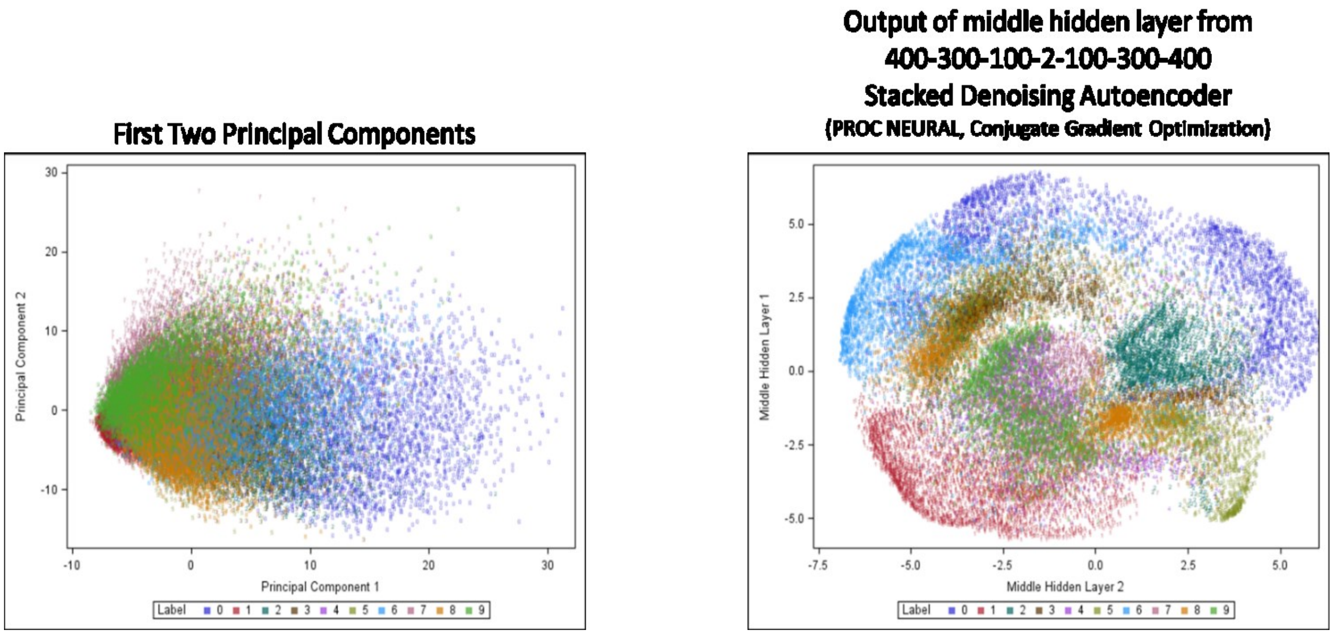

Figure 7. Principal Components and the Nonlinear Features Extracted from a Stacked Denoising Autoencoder, Colored by Class Label

CONCLUSION

SAS Enterprise Miner has been at the forefront of data mining and machine learning applications for years, and it continues to evolve and provide new algorithms from the machine learning discipline. A multithreaded implementation of factorization machines will offer an innovative solution for recommendation systems, and model factories that incorporate metadata repositories and analytic lifecycle management will provide fully automated modeling-by-exception functionality. Emerging deep learning methods, such as stacked, denoising autoencoders and deep neural networks, are also becoming available through traditional and high-performance neural network procedures. Behind the scenes, research into novel distributed implementations of scalable optimization methods is ongoing. Moreover, open-source integration features make it easier than ever to incorporate experimental or academic algorithms into SAS Enterprise Miner. If you're seeking a leading-edge machine learning solution—the solution might already be at your fingertips.

APPENDIX A

The following tables list the machine learning algorithms that are available for use in SAS Enterprise Miner nodes and SAS procedures for each category of learning. The tables also provide references where you can find more detailed information.

Algorithm	**SAS Enterprise Miner Nodes**	**SAS Procedures**	**References**
Regression	High Performance Regression LARS Partial Least Squares Regression	ADAPTIVEREG GAM GENMOD GLMSELECT HPGENSELECT HPLOGISTIC HPQUANTSELECT HPREG LOGISTIC QUANTREG QUANTSELECT REG	Panik 2009
Decision tree	Decision Tree High Performance Tree	ARBORETUM HPSPLIT	de Ville and Neville 2013
Random forest	High Performance Forest	HPFOREST	Breiman 2001b
Gradient boosting	Gradient Boosting	ARBORETUM	Friedman 2001
Neural network	AutoNeural DMNeural High Performance Neural Neural Network	HPNEURAL NEURAL	Rumelhart, Hinton, and Williams 1986
Support vector machine	High Performance Support Vector Machine	HPSVM	Cortes and Vapnik 1995
Naïve Bayes		HPBNET*	Friedman, Geiger, and Goldszmidt 1997
Neighbors	Memory Based Reasoning	DISCRIM	Cover and Hart 1967
Gaussian processes			Seeger 2004

Table A.1. Supervised Learning Algorithms

***PROC HPBNET can learn different network structures (naïve, TAN, PC, and MB) and automatically select the best model.**

Algorithm	SAS Enterprise Miner Nodes	SAS Procedures	References
A priori rules	Association Link Analysis		Agrawal, Imieliński, and Swami 1993
***k*-means clustering**	Cluster High Performance Cluster	FASTCLUS HPCLUS	Hartigan and Wong 1979
Mean shift clustering			Cheng 1995
Spectral clustering		Custom solution through Base SAS and the DISTANCE and PRINCOMP procedures	Von Luxburg 2007
Kernel density estimation		KDE	Silverman 1986
Nonnegative matrix factorization			Lee and Seung 1999
Kernel PCA		Custom solution through Base SAS and the CORR, PRINCOMP, and SCORE procedures	Schölkopf, Smola, and Müller 1997
Sparse PCA			Zou, Hastie, and Tibshirani 2006
Singular value decomposition		HPTMINE IML	Golub and Reinsch 1970
Self organizing maps	SOM/Kohonen Node		Kohonen 1984

Table A.2. Unsupervised Learning Algorithms

Algorithm*	SAS Enterprise Miner Node	SAS Procedure	References
Denoising autoencoders		HPNEURAL NEURAL	Vincent et al. 2008
Expectation maximization			Nigam et al. 2000
Manifold regularization			Belkin, Niyogi, and Sindhwani 2006
Transductive support vector machines			Joachims 1999

Table A.3. Semi-Supervised Learning Algorithms

***In semi-supervised learning, supervised prediction and classification algorithms are often combined with clustering. The algorithms noted here provide semi-supervised learning solutions directly.**

APPENDIX B

The SAS code in this appendix can be used to build the complete EMC Israel Data Science Challenge data from CSR format into a SAS data set and to create samples of the training data:

```
%let train_file= \\path\to\raw\data\train_data.csv;
data values columns row_offsets;
          infile "&train_file" recfm= f lrecl= 1 end= eof;
          length accum $32;
          retain accum ' ';                         *** TEXT VARIABLE FOR INPUT VALUES;
          retain recnum 1;                          *** LINE OF INPUT FILE;
          input x $char1.;                          *** PLACEHOLDER FOR ALL INPUT VALUES;
          if x='0d'x then return;
          delim= x in (',' '0a'x);
          if not delim then accum= trimn(accum)||x; *** IF IT'S NOT A DELIMITER IT'S A VALUE;
          if not delim and not eof then return;     *** IF IT'S NOT EOF OR A DELIMITER, CONTINUE;
          nvalues+1;                                *** INCREMENT NUMBER OF NON-ZERO VALUES;
          value= input(accum,best32.);              *** CONVERT ACCUM TO NUMERIC VALUE;
          accum= ' ';                               *** RESET TEXT VARIABLE FOR NEXT VALUE OF X;
          if nvalues<10 then put recnum= value=;    *** OUTPUT A SMALL SAMPLE OF VALUES FOR LOG;
          if recnum= 1 then do;                     *** SPECIAL CASE FOR FIRST ROW OF INPUT FILE;
             if nvalues= 1 then call symputx('nrows',value);
             if nvalues= 2 then call symputx('ncols',value);
          end;
          else if recnum= 2 then output values;     *** SAS DATA SET FOR NON-ZERO VALUES;
          else if recnum= 3 then output columns;    *** SAS DATA SET FOR COLUMN INDEX;
          else if recnum= 4 then output row_offsets;*** SAS DATA SET FOR ROW POINTER;
          if x='0a'x or eof then do;                *** TRUE CARRIAGE RETURN OR EOF, PRINT TO LOG;
             put recnum= nvalues=;
             recnum+1;                              *** INCREMENT TO NEXT INPUT LINE;
             nvalues= 0;                            *** RESET NVALUES;
          end;
          keep value;                               *** KEEP VALUES, NOT TEMP VARS;
run;

*** CREATE A COO FORMAT TABLE;
*** CONTAINS THE ROW NUMBER, COLUMN NUMBER, AND VALUE;
*** ALL INFORMATION NECESSARY TO BUILD FULL TRAINING MATRIX OR JUST SELECTED FEATURES;
data final_coo(keep= rownum colnum value);
         set row_offsets(firstobs= 2) end= eof;    *** 2ND OBS IN ROW_OFFSETS TELLS WHERE ...;
         retain prev 0;                            *** TO STOP THE FIRST ROW IN FINAL;
         retain min_colnum 1e50 max_colnum 0;
         rownum+1;                                 *** INITIALIZE ROWNUM TO ONE;
         count= value-prev;                        *** INDEX FOR END OF ROW;
         prev = value;                             *** INDEX FOR START OF ROW;
         do i=1 to count;
            set values;                            *** GET MATRIX VALUE;
            set columns (rename= (value= colnum)); *** GET COLUMN NUMBER;
            min_colnum= min(min_colnum, colnum);
            max_colnum= max(max_colnum, colnum);
            output;
         end;
         if eof then put _n_= min_colnum= max_colnum= "nrows=&nrows. ncols=&ncols.";
run;

*** IMPORT TRAINING LABELS;
%let label_file= \\path\to\raw\data\train_labels.csv;
data target;
         length hkey 8;
         hkey= _n_;
         infile "&label_file" delimiter= ',' missover dsd lrecl= 32767 firstobs= 1;
         informat target best32. ;
         format target best12. ;
         input target;
         if _n_ <= 10 then put hkey= target=;
run;
```

```
*** EXPAND SUMMARY SET INTO FULL TRAINING MATRIX;
*** THIS WILL TAKE SOME TIME;
data emcIsraelFull;
         set final_coo;
         by rownum;
         array tokens {&ncols} token1-token&ncols;  *** CREATE FULL NUMBER OF COLUMNS;
         retain tokens;
         do i= 1 to &ncols;                         *** POPULATE ARRAY WITH EXPANDED VALUES;
            if i= (colnum+1) then tokens{i}= value; *** COLNUM STARTS AT 0;
            if tokens{i}= . then tokens{i}= 0;
         end;
         keep rownum token1-token&ncols;
         if last.rownum then do;
            output;                                 *** OUTPUT ONE ROW FOR EACH SET OF ROWNUMS;
            if mod(rownum, 1000)= 0 then putlog 'NOTE: currently processing record ' rownum;
            do j = 1 to &ncols;                     *** REINITIALIZE ARRAY;
               tokens{j}= .;
            end;
         end;
run;

*** MERGE LABELS WITH HASH;
data emcIsraelFull;
         declare hash h();
         length hkey target 8;                      *** DEFINE HASH;
         h.defineKey(║hkey=);
         h.defineData(║target=);
         h.defineDone();
         do until(eof1);                            *** FILL WITH TARGET SET;
            set target end= eof1;
            rc1= h.add();
            if rc1 then do;
               putlog 'ERROR: Target not found for line ' _n_=;
               abort;
            end;
         end;
         do until(eof2);                            *** EXECUTE MERGE;
            set emcIsraelFull (rename= (rownum= hkey)) end= eof2;
            rc2= h.find();
            if rc2 then do;
               putlog 'ERROR: Target not found for line ' _n_=;
               abort;
            end;
            output;
         end;
         keep hkey target token1-token&ncols;
run;

*** APPEND TRAINING EXAMPLES THAT ARE ALL ZEROS;
data missing;
         merge target(rename= (hkey= rownum) in= a) final_coo(in= b);
         by rownum;
         if a and ^b;
         keep rownum target;
run;

data missing;
         set missing;
         array tokens token1-token&ncols (&ncols*0);
         do i= 1 to dim(tokens);
                  if tokens{i}= . then abort;
         end;
         drop i;
run;

proc append base= emcIsraelFull data= missing (rename= (rownum= hkey)); run;
```

REFERENCES

Agrawal, R., Imieliński, T., and Swami, A. 1993. “Mining Association Rules between Sets of Items in Large Databases.” *ACM SIGMOD Record* 22:207–216.

Belkin, M., Niyogi, P., and Sindhwani, V. 2006. “Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples.” *Journal of Machine Learning Research* 7:2399–2434.

Bengio, Y. 2009. “Learning Deep Architectures for AI.” *Foundations and Trends in Machine Learning* 2:1–127.

Breiman, L. 2001a. “Random Forests.” *Machine Learning* 45:5–32.

Breiman, L. 2001b. “Statistical Modeling: The Two Cultures (with Comments and a Rejoinder by the Author).” *Statistical Science* 16:199–231.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. 1984. *Classification and Regression Trees*. Belmont, CA: Wadsworth.

Cheng, Y. 1995. “Mean Shift, Mode Seeking, and Clustering.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17:790–799.

Cortes, C. and Vapnik, V. 1995. “Support-Vector Networks.” *Machine Learning* 20:273–297.

Cover, T. and Hart, P. 1967. “Nearest Neighbor Pattern Classification.” *IEEE Transactions on Information Theory* 13:21–27.

de Ville, B. and Neville, P. 2013. *Decision Trees for Analytics Using SAS Enterprise Miner*. Cary, NC: SAS Institute Inc.

Friedman, J. H. 2001. “Greedy Function Approximation: A Gradient Boosting Machine.” *Annals of Statistics* 29:1189–1232.

Friedman, N., Geiger, D., and Goldszmidt, M. 1997. “Bayesian Network Classifiers.” *Machine Learning* 29:131–163.

Giarratano, J. C. and Riley, G. 1998. *Expert Systems*. 3rd ed. Boston: PWS.

Golub, G. H. and Reinsch, C. 1970. “Singular Value Decomposition and Least Squares Solutions.” *Numerische Mathematik* 14:403–420.

Hartigan, J. A. and Wong, M. A. 1979. “Algorithm AS 136: A *k*-Means Clustering Algorithm.” *Journal of the Royal Statistical Society, Series C* 28:100–108.

Hinton, G. E. and Salakhutdinov, R. R. 2006. “Reducing the Dimensionality of Data with Neural Networks.” *Science* 313:504–507.

Hunt, E. B., Marin, J., and Stone, P. J. 1966. *Experiments in Induction*. New York: Academic Press.

Joachims, T. 1999. “Transductive Inference for Text Classification Using Support Vector Machines.” *Proceedings of the 16th International Conference on Machine Learning*. New York: ACM.

Kohonen, T. 1984. *Self-Organization and Associative Memory*. Berlin: Springer-Verlag.

Lee, D. D. and Seung, H. S. 1999. “Learning the Parts of Objects by Non-negative Matrix Factorization.” *Nature* 401:788–791.

Lighthill, J. 1973. *Artificial Intelligence: A Paper Symposium*. Swindon, UK: Science Research Council.

McCulloch, W. S. and Pitts, W. 1943. “A Logical Calculus of the Ideas Immanent in Nervous Activity.” *Bulletin of Mathematical Biophysics* 5:115–133.

Minsky, M. and Papert, S. 1969. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press.

Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., Ng, A., and Le, Q. V. 2011. “On Optimization Methods for Deep Learning.” *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. Princeton, NJ: International Machine Learning Society.

Nigam, K., McCallum, A. K., Thrun, S., and Mitchell, T. 2000. “Text Classification from Labeled and Unlabeled Documents Using EM.” *Machine Learning* 39:103–134.

Panik, M. J. 2009. *Regression Modeling: Methods, Theory, and Computation with SAS*. Boca Raton, FL: CRC Press.

Rashevsky, N. 1948. *Mathematical Biophysics*. Rev. ed. Chicago: University of Chicago Press.

Rosenblatt, F. 1958. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” *Psychological Review* 65:386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. “Learning Representations by Back-Propagating Errors.” *Nature* 323:533–536.

Sarle, W. S. 1983. “The Cubic Clustering Criterion.” SAS Technical Report A-108. Cary, NC: SAS Institute Inc.

Schölkopf, B., Smola, A., and Müller, K.-R. 1997. “Kernel Principal Component Analysis.” In *Artificial Neural Networks—ICANN'97*, 583–588. Berlin: Springer.

Sebastiani, F. 2002. “Machine Learning in Automated Text Categorization.” *ACM Computing Surveys* 34:1–47.

Seeger, M. 2004. “Gaussian Processes for Machine Learning.” *International Journal of Neural Systems* 14:69–106.

Silverman, B. W. 1986. *Density Estimation for Statistics and Data Analysis*. Vol. 26. Boca Raton, FL: CRC Press.

Tibshirani, R., Walther, G., and Hastie, T. 2001. “Estimating the Number of Clusters in a Data Set via the Gap Statistic.” *Journal of the Royal Statistical Society, Series B* 63:411–423.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. A. 2008. “Extracting and Composing Robust Features with Denoising Autoencoders.” *Proceedings of the 25th International Conference on Machine Learning*. New York: ACM.

Vogt, R. C., III. 1994. Handwritten digit normalization method. US Patent 5,325,447. Issued June 28.

Von Luxburg, U. 2007. “A Tutorial on Spectral Clustering.” *Statistics and Computing* 17:395–416.

Waterman, D. A. and Hayes-Roth, F. 1978. *Pattern-Directed Inference Systems*. New York: Academic Press.

Zou, H., Hastie, T., and Tibshirani, R. 2006. “Sparse Principal Component Analysis.” *Journal of Computational and Graphical Statistics* 15:265–286.

ACKNOWLEDGMENTS

Georges Guirguis, Chief Numerical Analyst, SAS Enterprise Miner
Susan Haller, Senior Manager, SAS Enterprise Miner
Rick Langston, Senior Manager, Base SAS
Warren Sarle, Senior Manager, SAS Mathematical and Numerical Modeling
Yongqiao Xiao, Principal Research Statistician Developer, SAS Enterprise Miner
Anne Baxter, Senior Technical Editor, Advanced Analytics Division

CONTACT INFORMATION

Patrick Hall
patrick.hall@sas.com

Jared Dean
jared.dean@sas.com

SAS Enterprise Miner
100 SAS Campus Dr.
Cary, NC 27513