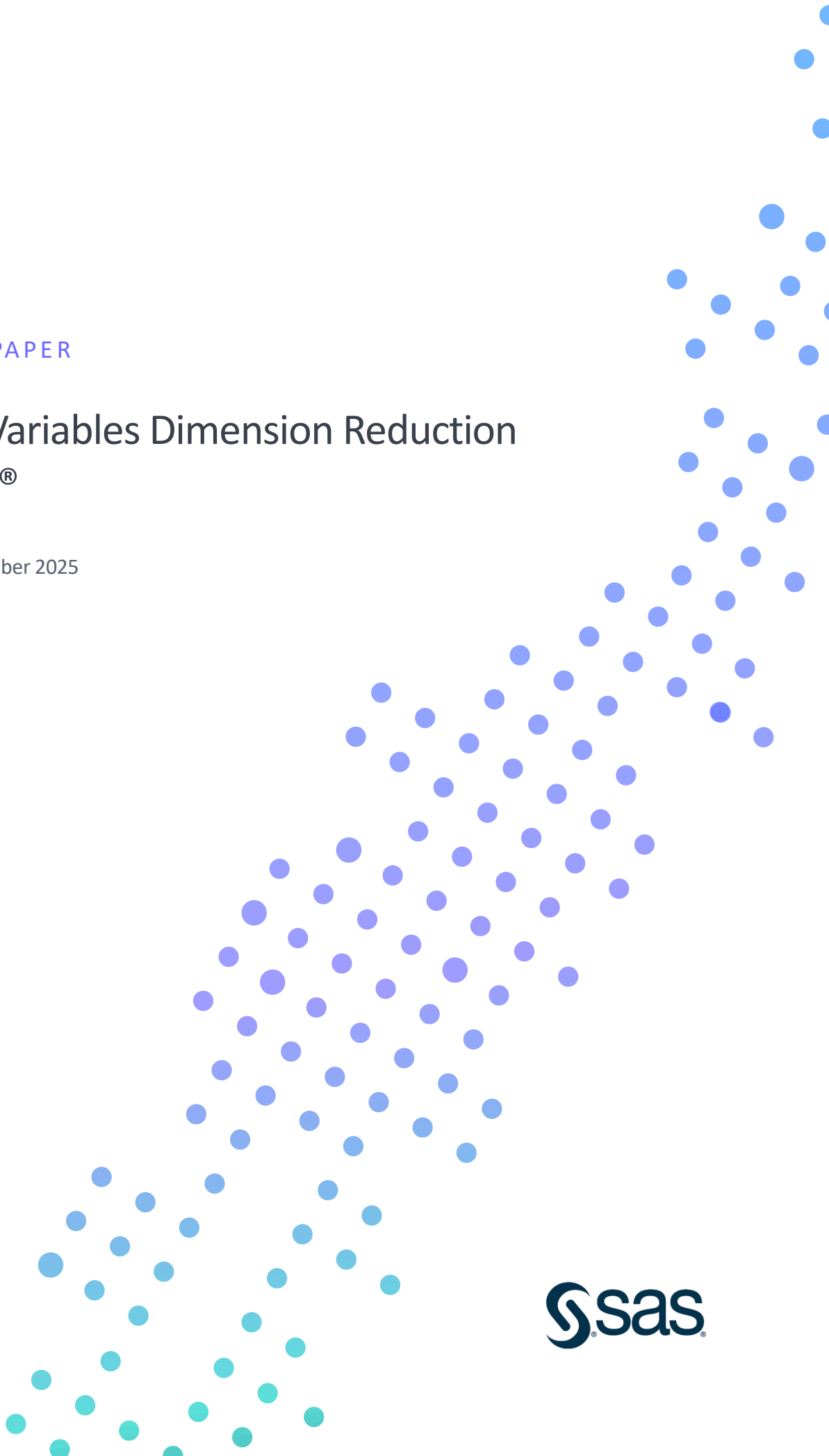
TECHNICAL PAPER

Nominal Variables Dimension Reduction Using SAS®

Last update: December 2025



Contents

Relevant Products and Releases

- SAS® Viya®
  - NOMINALDR Procedure

Introduction

Data Growth and Dimension Reduction

Modern data sets are expanding in size so rapidly that traditional tools struggle to store and analyze them. The significant disk space, large amount of memory, and heavy computational cost that they require can be burdensome. Even when the huge quantity of data in these data sets can be loaded into memory, processing the data can be extremely time-consuming. This growth in size is driven by both the increasing number of observations and the rising dimensionality of observations, which in fields such as health care and marketing can contain hundreds of variables. Although high dimensionality enriches insights, it also increases memory demands and the risk of overfitting. Mitigation strategies include feature selection, regularization, sparse modeling, and dimension reduction. Dimension reduction techniques decrease the size of a data set while preserving its essential structure, and they also serve as preprocessing steps for more efficient analysis. These techniques are especially useful when features are highly correlated or when the data can be well represented by only a few underlying variables. However, most dimension reduction techniques assume that the input is numerical and cannot be directly applied to nominal variables.

Nominal Variables and Their Dimension Reduction

Many real-world data sets contain a substantial portion—often a majority—of nominal variables, which represent unordered categories such as gender, product type, occupation, or zip code. These variables appear across many domains: health care records include diagnosis codes and genetic variants; marketing profiles capture geographic region and membership tiers; financial data record transaction types and fraud labels; industrial logs track equipment types and fault codes; and internet of things (IoT) devices emit event labels like device status and location identifiers.

As the number of nominal variables or their categories increases, dimensionality grows rapidly, especially when each category is treated as a separate variable. This expansion leads to higher memory demands, greater computational cost, and increased risk of overfitting. Dimension reduction for nominal variables addresses these issues by transforming high-dimensional nominal variables into a lower-dimensional representation that preserves essential patterns. The benefits include improved efficiency, reduced memory usage, and shorter run times. Dimension reduction also eliminates redundancy and noise, leading to better model generalization and predictive accuracy. In addition, because many analytical procedures accept only continuous variables, transforming nominal variables into continuous reduced-dimension variables expands the range of techniques available for downstream analysis.

Dimension Reduction of Nominal Variables in SAS® Viya®

This paper introduces the new functionality available in SAS Viya, which enables you to reduce the dimension of nominal variables. In SAS Viya, the NOMINALDR procedure (SAS Institute Inc. 2025d) implements two nominal variable dimension reduction methods: multiple correspondence analysis (MCA) and logistic principal component analysis (LPCA).

Audience for This Paper

The audience for this paper includes data scientists and engineers who work with data sets that contain high-dimensional nominal variables. The paper demonstrates how to apply the NOMINALDR procedure to reduce the dimension of nominal variables and improve the efficiency of downstream modeling and analysis.

NOMINALDR Procedure in SAS Viya

In SAS Viya, the NOMINALDR procedure (SAS Institute Inc. 2025d) includes two methods that you can use for dimension reduction of nominal variables: multiple correspondence analysis (MCA) and logistic principal component analysis (LPCA). You can use MCA or LPCA by specifying METHOD=MCA or METHOD=LPCA, respectively, in the PROC NOMINALDR statement to reduce the dimension of nominal data. This procedure accepts as its input a table, where each row is a training sample and each column is a nominal variable. The nominal variables can be either numeric or character.

Multiple Correspondence Analysis (MCA)

Principal component analysis (PCA) is a widely used method of reducing the dimension of continuous variables. Conceptually based on PCA, multiple correspondence analysis (MCA) is designed specifically for nominal data. MCA reduces the dimension by analyzing the relationships between categories of all the nominal variables. The data are first transformed into an indicator matrix, where each category is represented as a binary column. If you assume that the data include I observations, and that the nominal variables include J categories in total, the binary indicator matrix has the size $I \times J$ and is denoted as X. Let f represent the frequency of the I observations, expressed as an I-dimensional vector. This indicator matrix X is normalized as the probability matrix P,

$$P = \frac{1}{n}X$$

where $n = f'X1$ is the sum of the indicator matrix elements weighted by observation frequencies and 1 is a J-dimensional vector of 1s. The normalized matrix P is then standardized as S to balance contributions from different observations and categories:

$$S = D_r^{-\frac{1}{2}}(P - rc')D_c^{-\frac{1}{2}}$$

where $r = P1$ and $c = P'f$ are row and column marginal proportions, respectively, and D_r and D_c are diagonal matrices with r and c as their diagonal values, respectively. Singular value decomposition is applied to the standardized matrix S: $S = UDV'$, where D is a diagonal matrix whose singular values are arranged in decreasing order of magnitude, and U and V are the associated left and right singular vectors, respectively. The top k singular values and their associated singular vectors are selected to produce the reduced data F:

$$F = D_r^{-\frac{1}{2}}\widetilde{U}\widetilde{D}$$

where $\widetilde{D}$ is the diagonal matrix with the first k singular values and $\widetilde{U}$ contains the first k left singular vectors. You can also obtain the reduced data by using

$$F = D_r^{-\frac{1}{2}}S\widetilde{V}$$

where $\widetilde{V}$ contains the first k right singular vectors. More details about the MCA method can be found in Abdi and Valentin (2007), Khangar and Kamalja (2017), and Appendix A (Theory of Correspondence Analysis) of Greenacre (2017).

Explicitly constructing the indicator matrix is inefficient, because it has the size $I \times J$, where I is the number of observations and J is the total number of categories over all nominal variables. Instead of explicitly constructing this large matrix, PROC NOMINALDR operates on the original nominal data and an internal $J \times J$ matrix, thus improving computational efficiency and reducing memory usage, especially when the number of observations is very large.

Logistic Principal Component Analysis (LPCA)

Principal component analysis (PCA) reduces the dimension of continuous data and can be interpreted as maximizing the Gaussian log likelihood under the assumption that the data follow a Gaussian distribution with constant variance and have a low-rank mean structure. For a data matrix X with I rows and J columns, each element x_{ij} is assumed to follow the Gaussian distribution

$$x_{ij} \sim N\left(\theta_{ij}, \sigma^2\right)$$

where θ_{ij} is the mean and the variance σ^2 is the same for all i, j. The mean matrix Θ, with elements θ_{ij}, is assumed to have rank k and can be factorized as $\Theta = UDV'$, where D is a diagonal matrix that contains k singular values, and U and V are the corresponding left and right singular vectors, respectively.

From the Gaussian density, the log likelihood for x_{ij} is

$$\log P\left(x_{ij} \middle| \theta_{ij}\right) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\,\sigma^2}\left(x_{ij} - \theta_{ij}\right)^2$$

With constant σ^2, maximizing the total log likelihood over all entries is equivalent to minimizing the square reconstruction error:

$$\sum_{i=1}^{I}\sum_{j=1}^{J}\left(x_{ij} - \theta_{ij}\right)^2 = \|X - \Theta\|^2 = \|X - UDV'\|^2$$

Given the observed data X, PCA constructs a low-rank approximation $\tilde{\Theta} \approx \widetilde{U}\widetilde{D}\widetilde{V}'$ that minimizes the preceding reconstruction error (or equivalently, maximizes the total log likelihood). Here, $\widetilde{D}$ is the diagonal matrix that contains the first k singular values of X, and $\widetilde{U}$ and $\widetilde{V}$ are the corresponding left and right singular vectors of X, respectively. The reduced variables are then taken as $F = \widetilde{U}\widetilde{D}$.

However, the Gaussian assumption holds only for continuous data, not for nominal or binary data. Binary data are more appropriately modeled using the Bernoulli distribution. Logistic principal component analysis (LPCA) extends PCA to binary data by maximizing the Bernoulli log likelihood under the assumption that each binary observation follows a Bernoulli distribution (Schein, Saul, and Ungar 2003; De Leeuw 2006; Landgraf and Lee 2020).

PROC NOMINALDR further extends LPCA to handle nominal data by treating each category of every nominal variable as a separate binary variable. Let x_{ij} be a binary indicator that represents whether the observation i belongs to the category j. LPCA assumes that x_{ij} follows a Bernoulli distribution with parameter p_{ij}, and its natural parameter is $\theta_{ij} = \text{logit}(p_{ij})$. For all observations $i = 1, 2, \ldots, I$ and all categories $j = 1, 2, \ldots, J$, the loss function is defined as

$$\text{loss} = -\frac{2}{IJ}\sum_{i=1}^{I}\sum_{j=1}^{J}\log P\left(x_{ij}\middle|\theta_{ij}\right)$$

where $P\left(x_{ij}\middle|\theta_{ij}\right)$ denotes the probability of observing x_{ij} under a Bernoulli distribution with the natural parameter θ_{ij}. LPCA minimizes this loss function; this is equivalent to maximizing the Bernoulli log likelihood.

LPCA solves the optimization problem subject to a low-rank structure of the $I \times J$ natural parameter matrix Θ, where the (i,j)th element is θ_{ij}. Different formulations of this low-rank constraint have been proposed, including those in Schein, Saul, and Ungar (2003), De Leeuw (2006), and Landgraf and Lee (2020). The LPCA method in PROC NOMINALDR enforces the low-rank structure as in Landgraf and Lee (2020):

$$\Theta = 1_n\mu^\top + \left(\tilde{\Theta} - 1_n\mu^\top\right)UU^\top, \quad U^\top U = I_k$$

Here $\tilde{\Theta}$ is the I $\times J$ matrix whose (i,j)th element $\tilde{\theta}_{ij}$ is the saturated natural parameter of x_{ij}, 1_n is an I-dimensional vectors of ones, I_k is the $k \times k$ identity matrix, and μ (a J-dimensional vector) and U (a $J \times k$ orthonormal matrix) are estimated during the optimization. The scalar k is the dimension of the reduced variables.

For each binary indicator x_{ij} that is assumed to follow a Bernoulli distribution, the saturated distribution occurs when $\tilde{p}_{ij} = x_{ij}$, which implies $\tilde{\theta}_{ij} = \text{logit}(0) = -\infty$ if $x_{ij} = 0$, and $\tilde{\theta}_{ij} = \text{logit}(1) = \infty$ if $x_{ij} = 1$. To make computation feasible, these infinite limits are approximated by using a finite positive constant m: $\text{logit}(1)$ is approximated by m, and $\text{logit}(0)$ is approximated by $-m$. In practice, m does not need to be very large, because the inverse logit function is 0.9933 at 5 and 0.9999 at 10. In PROC NOMINALDR, m has the range (0, 10] with the default value 4. The choice of m can be guided by the nature of the data: if a category is nearly deterministic (with probabilities close to 0 or 1), a larger m might be appropriate; if a category is more stochastic (with probabilities closer to 0.5), a smaller m is preferred. Validation that uses subsequent analytical model performance is also recommended to select the best value of m.

In LPCA, the constrained optimization problem is solved iteratively by using the majorization-minimization (MM) algorithm. The initial value $\mu^0 = \text{logit}(\bar{X})$ if you use the mean of X over the observations, and the initial U is set to the first k right singular vectors of $\tilde{\Theta}$. At iteration t, the loss is majorized by $\|\Theta - Z^t\|^2$, where $Z^t = \Theta^{t-1} + 4(X - \sigma(\Theta^{t-1}))$, Θ^{t-1} is constructed from the previous estimates μ^{t-1} and U^{t-1}, and $\sigma(\cdot)$ is the sigmoid (inverse logit) function that is applied elementwise. The estimates μ^t and U^t are solved to minimize the majorization function $\|\Theta - Z^t\|^2$ with the constraints $\Theta = 1_n\mu^\top + \left(\tilde{\Theta} - 1_n\mu^\top\right)UU^\top$ and $U^\top U = I_k$. For more information, see Landgraf and Lee (2020). In the MM algorithm, the loss is expected to decrease at each iteration. The algorithm stops when either the loss converges (that is, the decrease between consecutive iterations is less than a specified criterion, or the loss increases) or the maximum number of iterations is reached. In PROC NOMINALDR, if the MM algorithm stops before converging (that is, it reaches the maximum number of iterations), a warning is displayed. If the subsequent model performance is unsatisfactory, increasing the maximum number of iterations can improve results.

When the optimization is completed, LPCA computes the reduced variables as $\left(\tilde{\Theta} - 1_n\mu^\top\right)U$.

Although the preceding LPCA formulas are written for data without observation frequencies, the LPCA method in PROC NOMINALDR supports frequencies that you specify in the FREQ statement.

Examples: Using PROC NOMINALDR for Data Preprocessing

This section presents three examples of how to use PROC NOMINALDR as a preprocessing step for subsequent modeling. In each example, PROC NOMINALDR reduces the dimension of nominal variables and outputs the reduced variables along with the target variable and any addition covariates that are specified in the COPYVAR= option. These outputs are then used as inputs for downstream procedures. The first example applies logistic regression (PROC LOGISTIC; SAS Institute Inc. 2025b) to the Soybean (Large) data set (Michalski and Chilausky, UCI Machine Learning Repository 1980). The second example applies a multilayer perceptron neural network (PROC NNET; SAS Institute Inc. 2025c) to the Molecular Biology (Splice-Junction Gene Sequences) data set (UCI Machine Learning Repository 1991). In both cases, models are trained on the original nominal data as well as on dimension-reduced data that are obtained using MCA and LPCA, and their performance is compared. The third example applies Gaussian process classification (PROC GPCLASS; SAS Institute Inc. 2025a) to the Mushroom data set (UCI Machine Learning Repository 1981). Because PROC GPCLASS accepts only interval variables, preprocessing with MCA or LPCA enables the analysis of this nominal data set. All code in this paper is available in the associated GitHub repo, which is available at https://github.com/sassoftware/iotaa/tree/main/Nominal%20Variables'%20Dimension%20Reduction%20Using%20SAS.

Example 1: PROC NOMINALDR with Logistic Regression on Soybean Data

This example uses the Soybean (Large) data set and the logistic regression model. The data set is from Michalski and Chilausky, UCI Machine Learning Repository (1980), and is available at https://archive.ics.uci.edu/dataset/90/soybean+large. The data describe soybean plants that are affected by various diseases, and each observation is described by nominal attributes such as leaf conditions, stem condition, and seed appearance. The downloaded training and testing files (`soybean-large.data` and `soybean-large.test`) include these nominal attributes encoded numerically (first category = 0, second category = 1, and so on). Both files have no header row, and missing values are indicated by a question mark ("?"). For convenience, column headers are added, and missing values are replaced with a "`.`" character. The data are then imported into SAS as `Train` and `Test` by using the following two PROC IMPORT statements:

```
proc import datafile="soybean-large.data" /*or other user-defined location*/
    out=Train dbms=csv replace; getnames=yes;
run;

proc import datafile="soybean-large.test" /*or other user-defined location*/
    out=Test dbms=csv replace; getnames=yes;
run;
```

The imported data tables Train and Test contain 307 and 376 observations, respectively. Of these, 41 training and 80 testing observations include missing values. Observations that have missing values are ignored during training and testing in SAS procedures, including PROC LOGISTIC and PROC NOMINALDR, resulting in missing values for their corresponding outputs. Each observation contains 36 variables: the target variable `class` (19 disease categories reduced to 15 after excluding the missing observations); `date`, which records the month of observation (April–October are encoded as 0–6) and is treated as an interval variable in the analysis; and 34 nominal variables, which serve as categorical descriptors and are stored in the following macro variable, `nominal_vars`, for

convenience of use in later steps. Further details about the variables and their categories can be found in the UCI Machine Learning Repository documentation.

```
%let nominal_vars = plant_stand precip temp hail crop_hist area_damaged
severity seed_tmt germination plant_growth leaves leafspots_halo leafspots_marg
leafspot_size leaf_shread leaf_malf leaf_mild stem lodging stem_cankers
canker_lesion fruiting_bodies external_decay mycelium int_discolor sclerotia
fruit_pods fruit_spots seed mold_growth seed_discolor seed_size shriveling
roots;
```

Logistic regression can classify data sets that have multiple nominal target labels, such as the Soybean data set in this example. The LOGISTIC procedure (SAS Institute Inc. 2025b) is used to train the logistic regression model. In the following three subsections, logistic regression is applied to the original-dimension data as well as to the MCA- and LPCA-reduced data. Training and scoring run times and classification performance are recorded and compared.

Logistic Regression with Original Soybean Data

As shown in the following statements, the first logistic regression model is trained on the original `Train` data table to predict the nominal target variable `class` by using the nominal variables in `nominal_vars` and the interval variable `date`. The OUTMODEL= option specifies the name of the trained model. The CLASS statement specifies the nominal predictors. The MODEL statement defines the response variable and the predictors. Model fit statistics are generated by the SCORE statement, captured in the ODS table ScoreFitStat, and stored as `ScoreFitStatTrain`. The %LET macro statements record the training time in `logisticTimeOrigTrn`.

```
%let t0=%sysfunc(datetime());
proc logistic data=Train outmodel=LOGISTICMODELOriginal;
   class &nominal_vars;
   model class=date &nominal_vars / LINK=GLOGIT;
   score fitstat;
   ods output ScoreFitStat=ScoreFitStatTrain;
run;
%let logisticTimeOrigTrn=%sysevalf(%sysfunc(datetime())-&t0);
```

Table 1 shows the ODS table Nobs, which includes the number of observations that are read and used. All 307 training observations are read, but only 266 of them are used in the model. The 41 training observations that have missing values are ignored in the model training. The same ODS table is produced for PROC NOMINALDR and PROC LOGISTIC with the MCA- and LPCA-reduced data and is not shown in the paper again.

Table 1. *Number of Observations from PROC LOGISTIC for Original Soybean Data*

Number of Observations Read	307
Number of Observations Used	266

Table 2 shows the ODS table ClassLevelInfo, where each nominal variable is expanded into design variables. For each nominal variable, the number of design variables equals the number of categories minus one. For example, the nominal variable `precip` has three different categories (0, 1, and 2) and is expanded into two design variables: category 0 is represented as (1, 0), category 1 as (0, 1), and category 2 as (–1, –1). Across the 34 nominal variables in the Soybean data set, there are 90 categories in total, which are encoded into 56 designed variables.

The expansion leads to a high-dimensional representation, which increases memory usage, computation time, and the risk of overfitting.

Table 2. *Class Level Information from PROC LOGISTIC for Original Soybean Data*

Class Level Information				
Class	**Value**	**Design Variables**		
plant_stand	0	1		
	1	-1		
precip	0	1	0	
	1	0	1	
	2	-1	-1	
temp	0	1	0	
	1	0	1	
	2	-1	-1	
hail	0	1		
	1	-1		
crop_hist	0	1	0	0
	1	0	1	0
	2	0	0	1
	3	-1	-1	-1
…	…	…	…	…
roots	0	1	0	
	1	0	1	
	2	-1	-1	

The trained model `LOGISTICMODELOriginal` is then applied to the `Test` data table by using the INMODEL=LOGISTICMODELORIGINAL option in the following PROC LOGISTIC statements. Scoring statistics are saved in `ScoreFitStatTest`, and the scoring time is recorded in `logisticTimeOrigTst`.

```
%let t0=%sysfunc(datetime());
proc logistic inmodel=LOGISTICMODELOriginal;
   score data=Test fitstat;
   ods output ScoreFitStat=ScoreFitStatTest;
run;
%let logisticTimeOrigTst=%sysevalf(%sysfunc(datetime())-&t0);
```

The fit statistics for the training and testing data are saved in `ScoreFitStatTrain` and `ScoreFitStatTest`, respectively. Table 3 summarizes the run times and scoring statistics that are extracted from these tables. The training log likelihood (–84.0647) and accuracy (0.92105) are both higher than the testing log likelihood (–486.166) and accuracy (0.81081), indicating some degree of overfitting.

Table 3. *Run Times and Scoring Statistics of PROC LOGISTIC for Original Soybean Data*

Data	TimeTrain	TimeTest	LogLikeTrn	AccTrn	LogLikeTst	AccTst
Original	2.95754	0.18460	-84.0647	0.92105	-486.166	0.81081

Nominal Dimension Reduction by MCA and Then Logistic Regression

In this section, PROC NOMINALDR with the MCA method is first applied to reduce the dimension of the nominal variables. The resulting reduced variables, together with the interval variable `date`, are used as inputs to a logistic regression model to predict the target variable `class`. The reduced dimension is selected by setting the DIMENSION= option to values from 5 to 20 and evaluating the classification accuracy on the testing data `Test`. The highest test accuracy is achieved at DIMENSION=8; the code that follows uses this value.

The following PROC NOMINALDR statements are applied to the `Train` data table to reduce the 34 nominal input variables to 8 by using the MCA method. The reduced dimension is specified by the DIMENSION= option, and the method is specified by the METHOD= option. The PREFIX= option specifies that reduced variables will be named with the prefix `mca_rv`.

```
proc NOMINALDR data=Train dimension=8 method=MCA prefix=mca_rv;
    input  &nominal_vars / level=nominal;
    output out=mcaTrain copyVars=(class date);
    savestate RSTORE=mcaSTORE;
run;
```

When the training is completed, the trained MCA model is saved in `mcaSTORE`, as specified by the RSTORE= option in the SAVESTATE statement, and the reduced variables are saved in `mcaTrain`, as specified by the OUT= option in the OUTPUT statement. Besides the eight reduced variables, `mcaTrain` also contains the target variable `class` and the interval variable `date`, as specified by the COPYVARS= option in the OUTPUT statement. Table 4 shows the first five observations of the `mcaTrain` data table.

Table 4. *First Five Observations of MCA-Reduced Soybean Data*

Obs	class	date	mca_rv1	mca_rv2	mca_rv3	mca_rv4	mca_rv5	mca_rv6	mca_rv7	mca_rv8
1	diaporthe-stem-canker	6	0.68133	-0.004956	-0.17655	0.12225	0.11265	-0.41228	0.39044	-0.18566
2	diaporthe-stem-canker	4	0.64916	0.040440	-0.20292	-0.02757	0.16094	-0.33590	0.40171	-0.21890
3	diaporthe-stem-canker	3	0.56052	-0.045225	-0.17125	-0.02012	0.05086	-0.36402	0.20283	-0.08166
4	diaporthe-stem-canker	3	0.56173	-0.012085	-0.13644	-0.02665	0.07119	-0.35539	0.33473	-0.12557

Obs	class	date	mca_rv1	mca_rv2	mca_rv3	mca_rv4	mca_rv5	mca_rv6	mca_rv7	mca_rv8
5	diaporthe-stem-canker	6	0.55713	0.060842	-0.17624	0.06348	0.12687	-0.47853	0.47351	-0.19949

The trained model in `mcaSTORE` is then applied to score the `Test` table by using the ASTORE procedure as follows:

```
proc astore;
    score data=Test rstore=mcaSTORE out=mcaTest copyVars=(class date);
quit;
```

After scoring, the output data table `mcaTest` is generated. It contains the eight reduced variables for the `Test` data table along with the target variable `class` and the interval variable `date`, as specified by the COPYVARS= option in the SCORE statement. The structure of `mcaTest` is the same as that of `mcaTrain`, as shown in Table 4.

In the following statements, a logistic regression model is trained and evaluated on the MCA-reduced data table. For MCA-reduced data, predictor variables include `date` and `mca_rv1` through `mca_rv8`.

```
%let t0=%sysfunc(datetime());
proc logistic data=mcaTrain outmodel=LOGISTICMODELMCA;
   model class=date mca_rv1-mca_rv8 / LINK=GLOGIT;
   score fitstat;
   ods output ScoreFitStat=ScoreFitStatRVMCATrain;
run;
%let logisticTimeMcaTrn=%sysevalf(%sysfunc(datetime())-&t0);

%let t0=%sysfunc(datetime());
proc logistic inmodel=LOGISTICMODELMCA;
   score data=mcaTest fitstat;
   ods output ScoreFitStat=ScoreFitStatRVMCATest;
run;
%let logisticTimeMcaTst=%sysevalf(%sysfunc(datetime())-&t0);
```

The run times and scoring statistics are summarized in Table 5. Compared with the logistic regression model on the original data (Table 3), the MCA-reduced data require substantially less time while achieving higher log likelihood and accuracy on both the training and testing sets. Specifically, the training and testing run times decrease from 2.95754s and 0.18460s to 0.22892s and 0.019160s, respectively, representing approximately a 90% reduction. The accuracy for training and testing increases from 0.92105 and 0.81081 to 0.96241 and 0.91216, respectively, with the testing accuracy improving by approximately 10%. Additionally, the gap in log likelihood and accuracy between training and testing is smaller than that observed with the original data, indicating reduced overfitting.

Table 5. *Run Times and Scoring Statistics of PROC LOGISTIC for MCA-Reduced Soybean Data*

Data	TimeTrain	TimeTest	LogLikeTrn	AccTrn	LogLikeTst	AccTst
MCA-reduced	0.22892	0.019160	-26.2834	0.96241	-94.4749	0.91216

Nominal Dimension Reduction by LPCA and Then Logistic Regression

In this section, PROC NOMINALDR with the LPCA method is applied to reduce the dimension of the nominal variables. The resulting reduced variables, together with the interval variable `date`, are used as inputs to a logistic regression model to predict the target variable `class`. The reduced dimension and the finite approximation of the logit function's infinite limits are selected by setting the DIMENSION= option to values from 5 to 20 and the M= option to values from 1 to 10, and then evaluating the classification accuracy on the testing data `Test`. The highest test accuracy is achieved at the option values DIMENSION=8 and M=10.

The following statements use PROC NOMINALDR with the LPCA method to reduce the 34 nominal variables to 8 and also use the trained LPCA model to score the `Test` data table:

```
proc NOMINALDR data=Train dimension=8 method=LPCA m=10 prefix=lpca_rv;
    input &nominal_vars / level=nominal;
    output out=lpcaTrain copyVars=(class date);
    savestate RSTORE=lpcaSTORE;
run;

proc astore;
    score data=Test rstore=lpcaSTORE out=lpcaTest copyVars=(class date);
quit;
```

In the PROC NOMINALDR statement, the METHOD= option specifies the LPCA method, the DIMENSION= option specifies the number of reduced variables as 8, and the M= option specifies the finite approximation of the logit function's infinite limits as 10. Table 6 shows the first five observations of the output data table `lpcaTrain`.

Table 6. *First Five Observations of LPCA-Reduced Soybean Data*

Obs	class	date	lpca_rv1	lpca_rv2	lpca_rv3	lpca_rv4	lpca_rv5	lpca_rv6	lpca_rv7	lpca_rv8
1	diaporthe-stem-canker	6	51.0414	-54.8113	1.08445	13.3686	3.47473	20.5249	18.6978	-15.5012
2	diaporthe-stem-canker	4	53.5319	-50.9658	2.51053	12.0968	7.21089	5.2250	13.4244	-20.1618
3	diaporthe-stem-canker	3	55.4662	-45.6174	-3.57772	9.0042	8.69237	7.4439	6.0313	-18.8381
4	diaporthe-stem-canker	3	56.8829	-46.1366	-2.02320	12.2841	0.69289	16.2692	20.5383	-11.8037
5	diaporthe-stem-canker	6	56.5229	-51.3719	0.72864	14.9717	0.44224	23.7864	16.2555	-3.3424

Similarly, another logistic regression model is trained on the LPCA-reduced data table as follows, by using `date` and `lpca_rv1` through `lpca_rv8` as predictors. The fit statistics and run times are summarized in Table 7.

```
%let t0=%sysfunc(datetime());
proc logistic data=lpcaTrain Outmodel=LOGISTICMODELLPCA;
```

```
   model class=date lpca_rv1-lpca_rv8 / LINK=GLOGIT;
   score fitstat;
   ods output ScoreFitStat=ScoreFitStatRVLPCATrain;
run;
%let logisticTimeLpcaTrn=%sysevalf(%sysfunc(datetime())-&t0);

%let t0=%sysfunc(datetime());
proc logistic inmodel=LOGISTICMODELLPCA;
   score data=lpcaTest fitstat;
   ods output ScoreFitStat=ScoreFitStatRVLPCATest;
run;
%let logisticTimeLpcaTst=%sysevalf(%sysfunc(datetime())-&t0);
```

Compared with the original data (Table 3), the LPCA-reduced data require substantially less time while achieving higher log likelihood and accuracy on both the training and testing sets. Training and testing run times decrease from 2.95754s and 0.18460s to 0.23162s and 0.01720s, respectively, representing approximately a 90% reduction, while accuracy increases from 0.92105 and 0.81081 to 0.95865 and 0.90878, respectively, representing an improvement in testing accuracy of approximately 10%. The smaller gap in log likelihood and accuracy between training and testing indicates reduced overfitting.

Table 7. *Run Times and Scoring Statistics of PROC LOGISTIC for LPCA-Reduced Soybean Data*

Data	TimeTrain	TimeTest	LogLikeTrn	AccTrn	LogLikeTst	AccTst
LPCA-reduced	0.23162	0.017200	-34.5566	0.95865	-96.1139	0.90878

From Tables 3, 5, and 7, you can see that the logistic regression model that is trained on the MCA- and LPCA-reduced data requires only about 10% of the run time of the model that is trained on original data. This is because the original data contain high-dimensional (56-dimensional) encodings of 34 nominal variables, whereas the reduced tables that are produced by MCA and LPCA contain only 8 continuous variables. Applying PROC NOMINALDR with either MCA or LPCA in a preprocessing step to transform the nominal data yields a significant computational efficiency gain for logistic regression compared to using the original nominal data.

Performance differences are also notable. Models that are trained on the MCA- and LPCA-reduced data achieve higher log-likelihood and accuracy values on both the training and testing sets. The testing accuracy for the reduced data could reach around 0.91, compared with only 0.81 for the original data, suggesting that removing unnecessary information from the original nominal variables can improve classification. In addition, smaller gaps between training and testing metrics indicate that the dimension reduction helps mitigate overfitting.

Example 2: PROC NOMINALDR with Neural Network on Molecular Biology Data

This example uses the Molecular Biology (Splice-Junction Gene Sequences) data set and multilayer perceptron neural networks to illustrate the benefits of using PROC NOMINALDR as a preprocessing step. The Molecular Biology (Splice-Junction Gene Sequences) data set is derived from molecular biology research and is available from the UCI Machine Learning Repository (1991) at https://archive.ics.uci.edu/dataset/69/molecular+biology+splice+junction+gene+sequences. The data set is split into 80% training and 20% testing sets, which are stored as the comma-separated-value (CSV) files `molecularBiologyTrain.csv` and `molecularBiologyTest.csv`, respectively. They are imported into SAS as `Train` and `Test` by using the following two PROC IMPORT statements:

```
proc import
datafile="molecularBiologyTrain.csv" /*or other user-defined location*/
    out=Train dbms=csv replace; getnames=yes;
run;

proc import
datafile="molecularBiologyTest.csv" /*or other user-defined location*/
    out=Test dbms=csv replace; getnames=yes;
run;
```

The loaded data tables `Train` and `Test` include 2,552 and 638 observations, respectively. Both tables consist of 60 nominal input variables (`Base1`, `Base2`, . . . , `Base60`) and one nominal target variable (`class`) with three categories ('EI', 'IE', and 'N'). These 60 nominal input variables have a total of 253 categories, resulting in 253 dummy variables if directly expanded. Table 8 displays the first five observations of the `Train` data table.

Table 8. *First Five Observations of Molecular Biology Training Data*

Obs	Base1	Base2	Base3	Base4	Base5	Base6	Base7	…	Base60	class
1	C	T	G	T	C	C	T	…	G	N
2	C	T	G	A	A	A	T	…	A	IE
3	C	A	G	C	A	A	A	…	G	EI
4	A	C	T	T	C	A	G	…	C	EI
5	C	T	C	A	A	A	T	…	T	N

The multilayer perceptron neural network is a supervised learning method that is designed to model the complex, nonlinear relationship between the predictors and the target. The NNET procedure (SAS Institute Inc. 2025c) is used to train the network for classification on the Molecular Biology data set. In the following three subsections, three networks are trained using the original data as well as the MCA- and LPCA-reduced data. Each network has a single hidden layer, and the number of nodes in the hidden layer is selected using the AUTOTUNE statement for each case. The run times and classification performance are recorded and compared. The reduced dimension is set to 10 for both MCA and LPCA.

Multilayer Perceptron Neural Network with Original Molecular Biology Data

The following statements train the first network to predict the nominal target variable `class` by using the nominal variables `Base1` through `Base60` from the original data table `Train`. The INPUT statement with the /LEVEL=NOMINAL option specifies that `Base1` through `Base60` are input variables and are nominal. The TARGET statement with the /LEVEL=NOMINAL option specifies `class` as the target variable, and it is also nominal. The network architecture is specified in the AUTOTUNE statement. It sets the number of hidden layers (NHIDDEN) to 1 and allows the number of nodes in the hidden layer (NUNITS1) to vary from 1 to 10. The AUTOTUNE statement also defines tuning options: the OBJECTIVE= option uses the misclassification rate (MCE) as the tuning metric, and the SEARCHMETHOD= option specifies the GA tuning method, which uses an initial Latin hypercube sample to seed a genetic algorithm that generates a new population of alternative configurations at each iteration. The OUTPUT statement specifies the output table for the prediction results, with the table name `nnetTrain` given by the OUT= option. The COPYVARS= option includes the true target label `class` in the output table. The OUTMODEL= option in the TRAIN statement specifies the name of the trained neural network. The training run time is recorded in `nnet_time_original_train`.

```
%let t0=%sysfunc(datetime());
proc nnet data=Train;
    input  Base1-Base60 / level=nominal;
    target class / level=nominal;
    autotune useparameters=custom objective=MCE searchmethod=GA
            tuningparameters=(nhidden(VALUES=(1) INIT=1)
                              nunits1(LB=1 UB=10 INIT=1)
                              );
    OUTPUT out=nnetTrain copyVars=class;
    TRAIN OUTMODEL=nnetModel seed=12345;
run;
%let nnet_time_original_train=%sysevalf(%sysfunc(datetime())-&t0);
```

Table 9 presents the ODS table ModelInfo, which include the model information. This table lists the number of input nodes as 253, which is from the expanded encoding of the 60 nominal variables. The number of nodes in the hidden layer is 9, and the number of weight parameters is 2,304.

Table 9. *Model Information from PROC NNET with Original Molecular Biology Data*

Model Information	
Model	Neural Net
Number of Observations Used	2552
Number of Observations Read	2552
Target/Response Variable	class
Number of Nodes	265
Number of Input Nodes	253
Number of Output Nodes	3
Number of Hidden Nodes	9
Number of Hidden Layers	1
Number of Weight Parameters	2304
Number of Bias Parameters	12
Architecture	MLP
Seed for Initial Weight	12345
Optimization Technique	LBFGS
Number of Neural Nets	1
Objective Value	0.017312321
Misclassification Rate for Validation	0.0008

Table 10 shows the first five observations in the PROC NNET prediction table `nnetTrain`, including the true target `class` and the predicted target `I_class`, and the predicted probabilities for the three target categories.

Table 10. *First Five Observations of Prediction from PROC NNET on Original Molecular Biology Data*

Obs	class	I_class	P_classEI	P_classIE	P_classN
1	N	N	6.417324E-16	8.631261E-20	1
2	IE	IE	2.6492672E-9	0.7099698145	0.2900301829
3	EI	EI	0.9999999994	4.270355E-11	5.464299E-10
4	EI	EI	0.9999999977	1.080012E-10	2.233105E-9
5	N	N	1.087821E-16	1.812936E-20	1

The trained model is applied to the `Test` data set by using the INMODEL= option in the following PROC NNET statement. The OUTPUT statement specifies the output table for the prediction results, and the table name `NNetTest` is given by the OUT= option. The COPYVARS= option includes the true target label `class` in the output table. The table `NNetTest` contains the same variables as the `NNetTrain` table shown in Table 10. The scoring run time is recorded in `nnet_time_original_test`.

```
%let t0=%sysfunc(datetime());
proc nnet data=Test inmodel=NNetModel;
    OUTPUT out=NNetTest copyVars=class;
run;
%let nnet_time_original_test=%sysevalf(%sysfunc(datetime())-&t0);
```

The following two ASSESS statements evaluate the classification results from the network. The fit statistics are saved in `fitstat_original_train` and `fitstat_original_test`; they include the average square error (ASE), multiclass log loss (MCLL, corresponding to the average log likelihood), and mean consequential error (MCE, representing the misclassification rate). The receiver operating characteristic (ROC) information table for the testing data set, which is computed using event "N" as the positive class, is saved in `ROCInfo_original_test`.

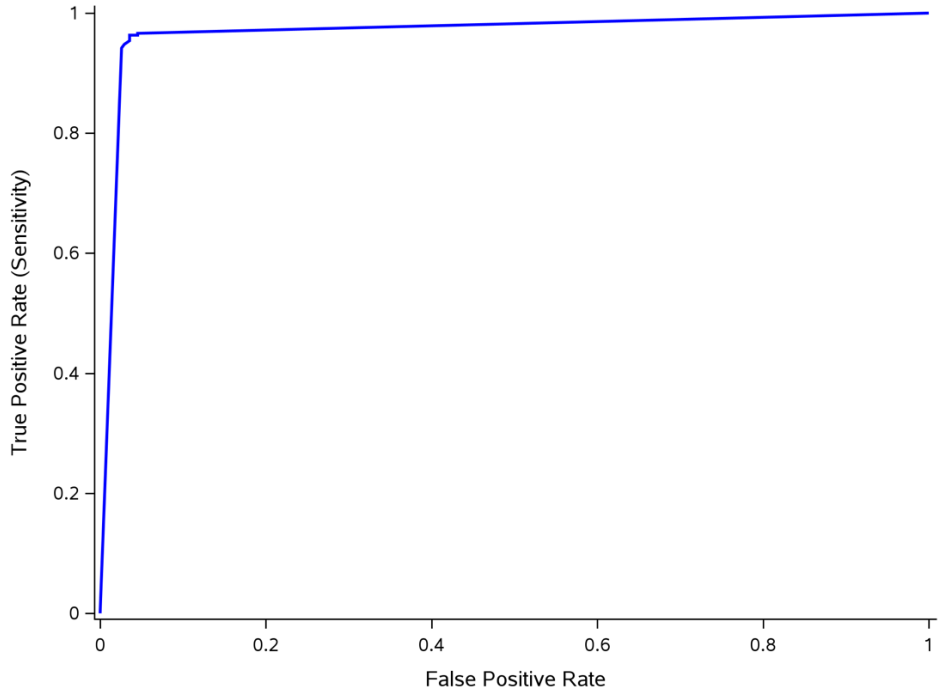
```
proc assess data=nnetTrain ncuts=20 nbins=2;
   var P_classN;
   target class / event="N" level=nominal;
   fitstat pvar=P_classEI P_classIE / pevent="EI IE" delimiter=" ";
   ods output FitStat=fitstat_original_train;
run;

proc assess data=nnetTest ncuts=20 nbins=2;
   var P_classN;
   target class / event="N" level=nominal;
   fitstat pvar=P_classEI P_classIE / pevent="EI IE" delimiter=" ";
   ods output FitStat=fitstat_original_test ROCInfo=ROCInfo_original_test;
run;
```

The PROC NNET run times and network fit statistics are summarized in Table 11. The ROC curve (EVENT="N") is plotted in Figure 1.

Table 11. *Run Times and Scoring Statistics of PROC NNET for Original Molecular Biology Data*

Data	TimeTrain	TimeTest	ASETrn	LogLikeTrn	AccTrn	ASETst	LogLikeTst	AccTst
Original	9.02717	0.043670	0.000465668	0.005908532	0.99922	0.036727	0.54308	0.94016

Figure 1. *ROC Curve (EVENT="N") of PROC NNET for Original Molecular Biology Data*



Nominal Dimension Reduction by MCA and Then Multilayer Perceptron Neural Network

In this section, first PROC NOMINALDR with the MCA method is applied to reduce the dimension of nominal variables in the Molecular Biology data set, and then PROC NNET is applied to predict the nominal target variable `class` by using the MCA-reduced variables.

In the following statements, PROC NOMINALDR with the MCA method is applied to the `Train` data table to reduce the 60 nominal input variables, as specified by the INPUT statement with the /LEVEL=NOMINAL option. The reduced dimension is set to 10, as specified by the DIMENSION= option. The METHOD= option specifies that the MCA method is to be used. The PREFIX= option specifies that the names of reduced variables will have the prefix `mca_rv`.

```
proc NOMINALDR data=Train dimension=10 method=MCA prefix=mca_rv;
   input Base1-Base60 /LEVEL=NOMINAL;
   output out=mcaTrain copyVars=class;
   savestate RSTORE=mcaSTORE;
run;
```

When the training is completed, the trained MCA model is saved in `mcaSTORE`, and the reduced variables are saved in `mcaTrain`. In addition to the 10 reduced variables, `mcaTrain` includes the target variable `class`, as specified by the COPYVARS= option in the OUTPUT statement. Table 12 displays the first five observations of the `mcaTrain` data table.

Table 12. *First Five Observations of MCA-Reduced Molecular Biology Training Data*

Obs	class	mca_rv1	mca_rv2	mca_rv3	mca_rv4	mca_rv5	mca_rv6	…	mca_rv10
1	N	-0.03129	-0.06150	-0.018375	-0.17942	0.19145	0.10898	…	-0.10952
2	IE	0.12460	0.17181	0.012651	0.14780	-0.28620	0.10275	…	0.06498
3	EI	-0.22641	-0.34190	-0.026648	0.10561	0.27711	-0.10451	…	0.02902
4	EI	0.28139	-0.30324	-0.034884	0.19276	-0.15780	0.02846	…	0.00015
5	N	0.35981	-0.24276	-0.032522	-0.11938	-0.12375	-0.00030	…	-0.08306

The trained model is then applied to score the `Test` data table by using the ASTORE procedure as follows. The output data table `mcaTest` includes the MCA-reduced testing data and has the same form as the `mcaTrain` table shown in Table 11.

```
proc astore;
    score data=Test rstore=mcaSTORE out=mcaTest copyVars=class;
quit;
```

Using the MCA-reduced data, the second multilayer perceptron neural network is trained on `mcaTrain` and then applied to `mcaTest` as follows. The INPUT statement specifies `mca_rv1` through `mca_rv10` as interval inputs. The TARGET and AUTOTUNE statements are identical to those that are used to train the original data.

```
%let t0=%sysfunc(datetime());
proc nnet data=mcaTrain;
    input  mca_rv1-mca_rv10 / level=interval;
    target class / level=nominal;
    autotune useparameters=custom objective=MCE searchmethod=GA
            tuningparameters=(nhidden(VALUES=(1) INIT=1)
                              nunits1(LB=1 UB=10 INIT=1)
                              );
    OUTPUT out=mcaNNetTrain copyVars=class;
    TRAIN OUTMODEL=mcaNNetModel seed=12345;
run;
%let nnet_time_mca_train=%sysevalf(%sysfunc(datetime())-&t0);

%let t0=%sysfunc(datetime());
proc nnet data=mcaTest inmodel=mcaNNetModel;
        OUTPUT out=mcaNNetTest copyVars=class;
run;
%let nnet_time_mca_test=%sysevalf(%sysfunc(datetime())-&t0);
```

Table 13 presents the model information for the network that is trained on the MCA-reduced data. The number of input nodes is 10, matching the reduced dimension. The number of nodes in the hidden layer is 6. The number of input nodes and the number hidden nodes are both less than those in the network that is trained on the original data. The number of weight parameters, 78, is also substantially less than the 2,304 parameters in Table 9 for the network that is trained on the original data.

Table 13. *Model Information from PROC NNET for MCA-Reduced Molecular Biology Data*

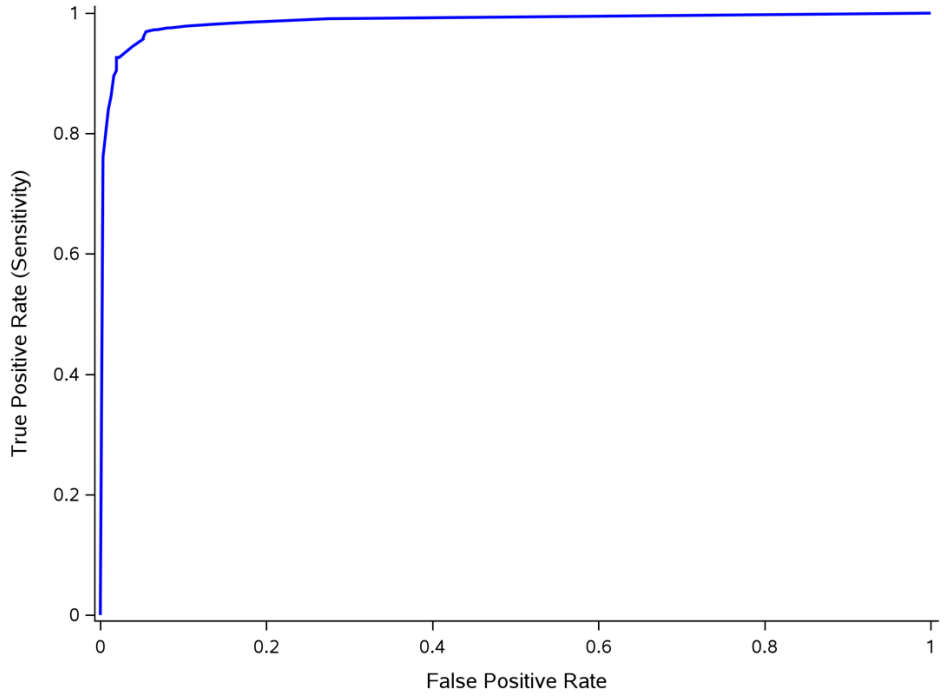
Model Information	
Model	Neural Net
Number of Observations Used	2552
Number of Observations Read	2552
Target/Response Variable	class
Number of Nodes	19
Number of Input Nodes	10
Number of Output Nodes	3
Number of Hidden Nodes	6
Number of Hidden Layers	1
Number of Weight Parameters	78
Number of Bias Parameters	9
Architecture	MLP
Seed for Initial Weight	12345
Optimization Technique	LBFGS
Number of Neural Nets	1
Objective Value	0.6860183323
Misclassification Rate for Validation	0.0654

The prediction results of the network that is trained on the MCA-reduced data are saved in `mcaNNetTrain` and `mcaNNetTest`, which contain the same variables as `NNetTrain` shown in Table 10. As with the original data, these prediction results tables are evaluated by PROC ASSESS. The run times and fit statistics are summarized in Table 14. The ROC curve (EVENT="N") is displayed in Figure 2. The training and testing run times (5.53069s and 0.026180s, respectively) with the MCA-reduced data are approximately 60% of those (9.02717s and 0.043670s, respectively) that are required for the original data (Table 11). Although the training average square error and log-likelihood error are higher and the training accuracy is lower than those of the model that is trained on the original data in Table 11, the performance on the testing data set is comparable.

Table 14. *Run Times and Scoring Statistics of PROC NNET for MCA-Reduced Molecular Biology Data*

Data	TimeTrain	TimeTest	ASETrn	LogLikeTrn	AccTrn	ASETst	LogLikeTst	AccTst
MCA-reduced	5.53069	0.026180	0.033680	0.18016	0.93456	0.031565	0.17633	0.93543

Figure 2. *ROC Curves (EVENT="N") of PROC NNET for MCA-Reduced Molecular Biology Data*



Nominal Dimension Reduction by LPCA and Then Multilayer Perceptron Neural Network

In this section, PROC NOMINALDR with the LPCA method is used to reduce the dimension of the nominal variables in the Molecular Biology data set. The LPCA-reduced variables are then used as input to a network to predict the nominal target variable `class`. The reduced dimension is set to 10. The finite approximation parameter M in LPCA is selected by setting the value from 1 to 10 and evaluating the classification accuracy on the testing data `Test`. The testing accuracies are generally similar for different values of M. In the following statements, M=3 is used because it yields the highest testing accuracy.

In the following statements, PROC NOMINALDR with the LPCA method is applied to the `Train` data table, and then the trained LPCA model is used to score the `Test` data table. The output data tables `lpcaTrain` and `lpcaTest` contain the variables that are reduced by the LPCA method, along with the target variable `class`. Table 15 shows the first five observations of the `lpcaTrain` data table.

```
proc NOMINALDR data=Train dimension=10 method=LPCA m=3 prefix=lpca_rv;
   input Base1-Base60 /LEVEL=NOMINAL;
   output out=lpcaTrain copyVars=class;
   savestate RSTORE=lpcaSTORE;
run;

proc astore;
   score data=Test rstore=lpcaSTORE out=lpcaTest copyVars=class;
quit;
```

Table 15. *First Five Observations of LPCA-Reduced Molecular Biology Training Data*

Obs	class	lpca_rv1	lpca_rv2	lpca_rv3	lpca_rv4	lpca_rv5	lpca_rv6	…	lpca_rv10
1	N	-25.8861	-1.11995	-2.16722	-4.85821	5.77432	-2.00371	…	8.44812
2	IE	-27.6836	-2.08846	4.84070	3.04223	-5.47340	-4.45473	…	6.15434
3	EI	-27.8914	4.90769	-9.93065	1.86612	4.99422	3.58632	…	-0.72344
4	EI	-26.6729	-5.46569	-5.77512	5.57466	-5.19736	-1.70421	…	-1.82657
5	N	-25.2688	-8.90014	-4.90600	-2.29119	-2.45748	-1.43642	…	5.51034

Using the LPCA-reduced data, the third network is trained as follows, with `lpca_rv1` through `lpca_rv10` as predictors. The trained model is then evaluated on `lpcaTest`.

```
%let t0=%sysfunc(datetime());
proc nnet data=lpcaTrain;
    input  lpca_rv1-lpca_rv10 / level=interval;
    target class / level=nominal;
    autotune useparameters=custom objective=MCE searchmethod=GA
            tuningparameters=(nhidden(VALUES=(1) INIT=1)
                              nunits1(LB=1 UB=10 INIT=1)
                              );
    OUTPUT out=lpcaNNetTrain copyVars=class;
    TRAIN OUTMODEL=lpcaNNetModel seed=12345;
run;
%let nnet_time_lpca_train=%sysevalf(%sysfunc(datetime())-&t0);
%let t0=%sysfunc(datetime());
proc nnet data=lpcaTest inmodel=lpcaNNetModel;
    OUTPUT out=lpcaNNetTest copyVars=class;
run;
%let nnet_time_lpca_test=%sysevalf(%sysfunc(datetime())-&t0);
```

Table 16 presents the model information for the network that is trained on LPCA-reduced data. The network has 10 input nodes, corresponding to the reduced dimension, and 3 nodes in the hidden layer, fewer than the networks that are trained on MCA-reduced data. The number of input nodes and the number of hidden nodes are both less than those in the network that is trained on the original data. The network has 39 weight parameters, substantially fewer than the 78 weight parameters for the network that is trained on MCA-reduced data in Table 12 and the 2,304 weight parameters for the network that is trained on the original data in Table 9.

Table 16. *Model Information from PROC NNET for LPCA-Reduced Molecular Biology Data*

Model Information	
Model	Neural Net
Number of Observations Used	2552
Number of Observations Read	2552
Target/Response Variable	class
Number of Nodes	16
Number of Input Nodes	10
Number of Output Nodes	3
Number of Hidden Nodes	3
Number of Hidden Layers	1
Number of Weight Parameters	39
Number of Bias Parameters	6
Architecture	MLP
Seed for Initial Weight	12345
Optimization Technique	LBFGS
Number of Neural Nets	1
Objective Value	0.5165777339
Misclassification Rate for Validation	0.0455

The PROC NNET classification results are evaluated by PROC ASSESS as with the original data. The run times and fit statistics are summarized in Table 17. The ROC curve is displayed in Figure 3. The training time of 3.20264s with the LPCA-reduced data as shown in Table 17 is only approximately 35% of the 9.02717s that is required for the original data (Table 11), and the testing time of 0.028910s is about 66% of the 0.043670s that is required for the original data. Although the training average square error and log-likelihood error are higher and the training accuracy is lower than those of the model that is trained on the original data in Table 11, the performance on the testing data set is better. This suggests that the dimension reduction preprocessing results in less overfitting.

Table 17. *Run Times and Scoring Statistics of PROC NNET for LPCA-Reduced Molecular Biology Data*

Data	TimeTrain	TimeTest	ASETrn	LogLikeTrn	AccTrn	ASETst	LogLikeTst	AccTst
LPCA-reduced	3.20264	0.028910	0.023730	0.13573	0.95455	0.021856	0.12161	0.95748

Figure 3. *ROC Curve (EVENT="N") of PROC NNET for LPCA-Reduced Molecular Biology Data*

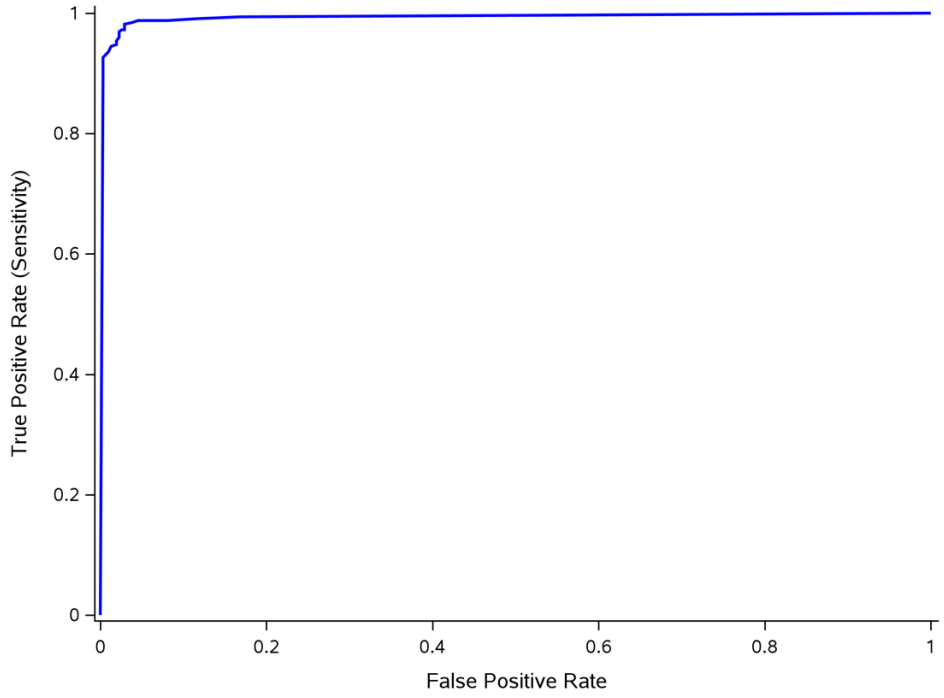
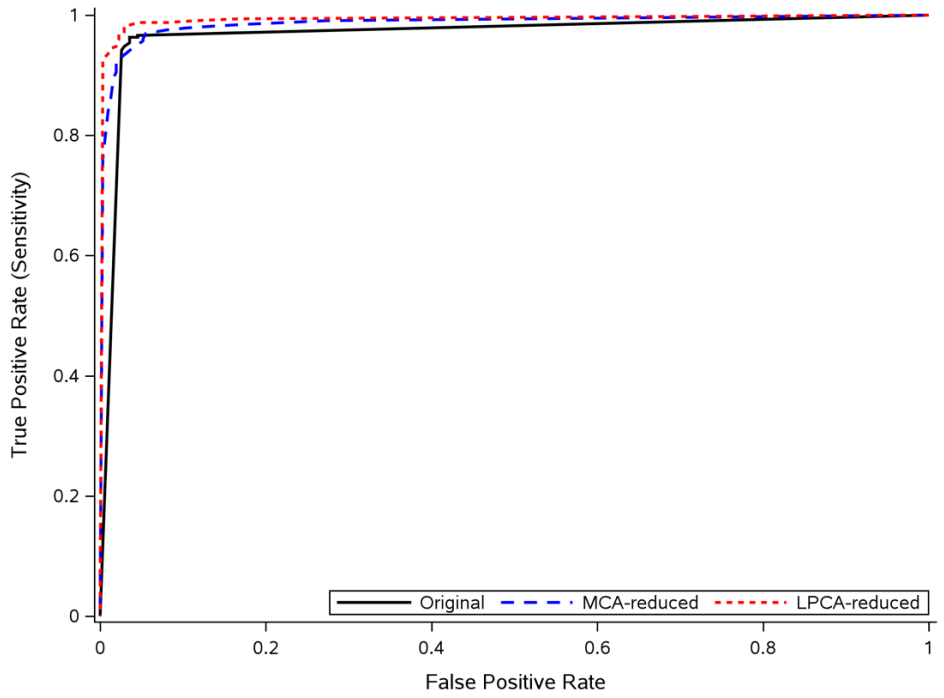


Figure 4 shows the ROC curves from PROC NNET for the original data, MCA-reduced data, and LPCA-reduced data together. All three curves are close to the curve of a perfect classifier. The curves from the original data and MCA-reduced data are similar, while the curve from the LPCA-reduced data is even closer to the perfect classifier curve than those from the original data and MCA-reduced data.

Figure 4. *Comparison of ROC Curves (EVENT="N") from PROC NNET for Molecular Biology Data*



The networks require fewer nodes and substantially fewer weight parameters when using the MCA- or LPCA-reduced data compared to the original data (compare Tables 13 and 16 to Table 9): 6 and 3 nodes versus 9 nodes in the hidden layer; 19 and 16 total nodes versus 265 total nodes; and 78 and 39 weight parameters versus 2,304 weight parameters. The smaller network size for MCA- and LPCA-reduced data results in much shorter run times while achieving performance comparable to the network that is trained on the original data. Between MCA and LPCA, the network that is trained on LPCA-reduced data uses a smaller network and achieves better prediction performance than the network that is trained on MCA-reduced data.

Example 3: PROC NOMINALDR with Gaussian Process Classification on Mushroom Data

This example shows that nominal data can be analyzed by the models that accept interval variables only after preprocessing with PROC NOMINALDR. The nominal Mushroom data set is used, and the Gaussian process classification model is applied as the downstream model that requires interval inputs. The Mushroom data set is available from the UCI Machine Learning Repository (1981) at https://archive.ics.uci.edu/dataset/73/mushroom. It describes hypothetical samples of 23 species of gilled mushrooms in the Agaricus and Lepiota families. The data are split into 80% training and 20% testing sets, which are saved as `mushroomTrain.csv` and `mushroomTest.csv`, respectively. The following PROC IMPORT statements load them into SAS as `Train` and `Test`:

```
proc import datafile="mushroomTrain.csv" /*or other user-defined location*/
    out=Train dbms=csv replace; getnames=yes;
run;

proc import datafile=" mushroomTest.csv" /*or other user-defined location*/
    out=Test dbms=csv replace; getnames=yes;
run;
```

The loaded `Train` and `Test` tables contain 6,511 and 1,613 observations, respectively. They both consist of 22 nominal input variables and one nominal target variable (`poisonous`). The target has two categories: ‘e’ (edible) and ‘p’ (poisonous). The 22 nominal variables include 117 categories in total. Details about their categories are documented in the UCI Machine Learning Repository. The following macro variable, `mushroom_nominal_vars`, stores these 22 nominal variables for later use:

```
%let mushroom_nominal_vars = cap_shape cap_surface cap_color bruises odor
gill_attachment gill_spacing gill_size gill_color stalk_shape stalk_root
stalk_surface_above_ring stalk_surface_below_ring stalk_color_above_ring
stalk_color_below_ring veil_type veil_color ring_number ring_type
spore_print_color population habitat;
```

The Gaussian process classification model is a nonparametric probabilistic model for classification. The GPCLASS procedure (SAS Institute Inc. 2025a) trains Gaussian process classification models for binary classification. PROC GPCLASS accepts only interval variables, so the nominal variables in the Mushroom data set cannot be analyzed directly. In the following two subsections, PROC NOMINALDR is applied using the MCA and LPCA methods to reduce the nominal variables. The resulting reduced variables are intervals, and thus they can be analyzed by PROC GPCLASS. Before PROC GPCLASS is run, the reduced variables are standardized by the STDIZE procedure (SAS Institute Inc. 2025e) to have zero mean and unit variance. The prediction results from PROC GPCLASS are reported.

Nominal Dimension Reduction by MCA Prior to Gaussian Process Classification

In this section, PROC NOMINALDR with the MCA method is first used to preprocess the nominal variables. The preprocessed variables are then standardized using PROC STDIZE. After that, PROC GPCLASS is applied to the standardized variables. The reduced dimension is set to 5 in PROC NOMINALDR for the Mushroom data set.

In the following statements, PROC NOMINALDR is applied to the `Train` data table to reduce the 22 nominal input variables to 5, as specified by the DIMENSION= option. The METHOD= option specifies that the MCA method is used, and the PREFIX= option names the reduced variables with the prefix `mca_rv`. The trained model, which is saved in `mcaSTORE`, is then applied to score the `Test` data table. The reduced variables, together with the target variable `poisonous`, are stored in the output tables `mcaTrain` and `mcaTest`.

```
proc NOMINALDR data=Train dimension=5 method=MCA prefix=mca_rv;
  input &mushroom_nominal_vars / level=nominal;
  output out=mcaTrain copyVars=poisonous;
  savestate RSTORE=mcaSTORE;
run;
proc astore;
   score data=Test rstore=mcaSTORE
       out=mcaTest copyVars=poisonous;
quit;
```

Before PROC GPCLASS is used for analysis, the reduced variables in the tables `mcaTrain` are standardized using PROC STDIZE as follows to implement the STD method, in which each variable is centered by its mean and scaled by its standard deviation. The standardization model is saved in `mcaStdSTAT` and then applied to standardize the data table `mcaTest`.

```
proc stdize data=mcaTrain out=mcaStdTrain OUTSTAT=mcaStdSTAT method=std;
      var mca_rv1-mca_rv5;
run;

proc stdize data=mcaTest out=mcaStdTest method=in(mcaStdSTAT);
      var mca_rv1-mca_rv5;
run;
```

Next, the reduced and standardized data table `mcaStdTrain` is analyzed using PROC GPCLASS as follows. Both the DATA= and TESTDATA= options are set to `mcaStdTrain`. The INPUT statement uses `mca_rv1` through `mca_rv5` as predictors, and the TARGET statement specifies `poisonous` as the nominal classification variable. A Gaussian kernel is specified in the KERNEL statement, and its bandwidth is controlled by the SIGMA= option; because `mcaStdTrain` is standardized to unit variance, SIGMA=1 is used. The INFERENCE statement uses the Laplace approximation (LA) algorithm for inference; the MAXITER= option sets the maximum number of Newton iterations, and the THRESHOLD= option sets the convergence criterion. The OUTPUT statement writes the classification results to `GPCLASS_mcaStd_train` and includes the true target label `poisonous` that is specified by the COPYVARS= option. The first five observations of the Gaussian process classification results table `GPCLASS_mcaStd_train` are shown in Table 18. The trained Gaussian process classification model is saved in `mcaStdModel`, as specified by the RSTORE= option in the SAVESTATE statement.

```
proc GPCLASS data=mcaStdTrain TESTDATA=mcaStdTrain seed=12345;
    input mca_rv1-mca_rv5;
    target poisonous /LEVEL=NOMINAL;
    kernel gaussian(sigma=1);
    inference LA(maxIter=10 threshold=0.001);
    output out=GPCLASS_mcaStd_train copyVars=poisonous;
    savestate rstore=mcaStdModel;
run;
```

Table 18. *First Five Observations from PROC GPCLASS for MCA-Reduced and Standardized Mushroom Data*

Obs	poisonous	P_poisonouse	P_poisonousp	I_poisonous	P_VAR_
1	p	0.27841	0.72159	p	0.04081
2	e	0.92986	0.07014	e	0.05577
3	e	0.95244	0.04756	e	0.11645
4	p	0.16910	0.83090	p	0.04042
5	e	0.89409	0.10591	e	0.04980

The following statements use PROC ASTORE to apply the saved model to the reduced and standardized test data table. The classification results together with the true target label are saved in `GPCLASS_mcaStd_test`.

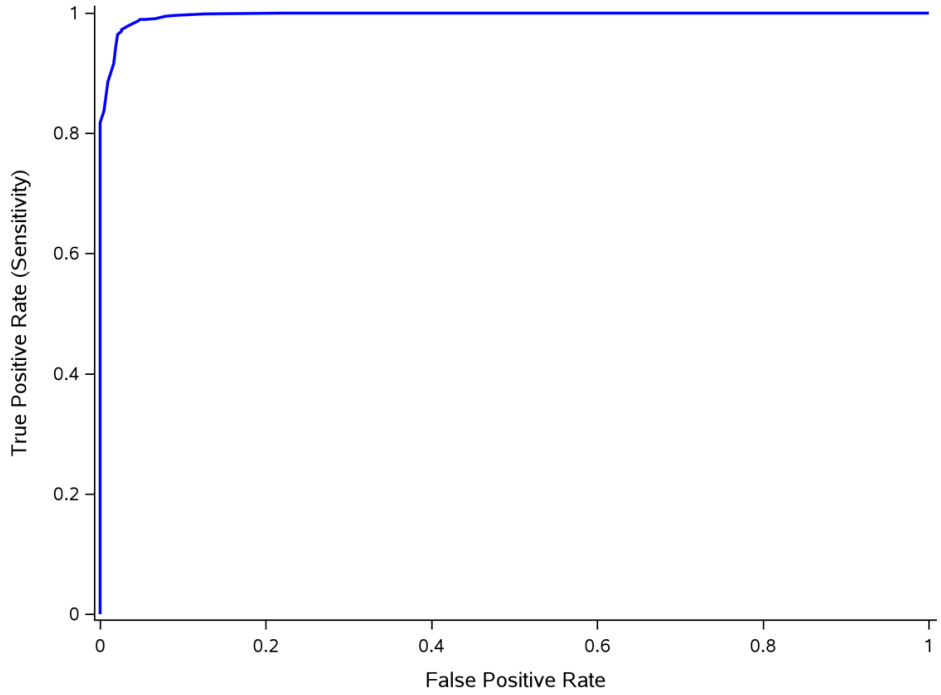
```
proc astore;
    score data=mcaStdTest rstore=mcaStdModel out=GPCLASS_mcaStd_test
copyVars=poisonous;
run;
```

The classification results `GPCLASS_mcaStd_train` and `GPCLASS_mcaStd_test` are evaluated using PROC ASSESS as in Example 2. The fit statistics are summarized in Table 19, and the ROC curve is displayed in Figure 5.

Table 19. *Scoring Statistics of PROC GPCLASS for MCA-Reduced and Standardized Mushroom Data*

Data	ASETrn	LogLikeTrn	AccTrn	ASETst	LogLikeTst	AccTst
MCA-reduced	0.024313	0.088944	0.97005	0.023795	0.088793	0.97334

Figure 5. *ROC Curve (EVENT="p") of PROC GPCLASS for MCA-Reduced and Standardized Mushroom Data*



The 22 nominal input variables in the Mushroom data set are reduced to five dimensions by using the MCA method. These reduced variables are then standardized and supplied as inputs to PROC GPCLASS. Even with only five variables, PROC GPCLASS achieves a high classification accuracy of 0.97334, as shown in Table 19, and the resulting ROC curve shown in Figure 5 is close to a perfect classifier curve.

Nominal Dimension Reduction by LPCA Prior to Gaussian Process Classification

Similarly, PROC NOMINALDR is applied to the `Train` data table with the LPCA method as follows to reduce the 22 nominal input variables to 5. The trained model is then used to score the `Test` data table. The reduced variables, along with the target variable `poisonous`, are stored in the tables `lpcaTrain` and `lpcaTest`.

```
proc NOMINALDR data=Train dimension=5 method=LPCA m=4 maxiter=200
prefix=lpca_rv;
    input &mushroom_nominal_vars / level=nominal;
    output  out=lpcaTrain copyVars=poisonous;
    savestate RSTORE=lpcaSTORE;
 run;
proc astore;
score data=Test rstore=lpcaSTORE
    out=lpcaTest copyVars=poisonous;
quit;
```

The reduced data tables are then standardized as follows by PROC STDIZE and analyzed by PROC GPCLASS, as with the MCA-reduced data. The LPCA-reduced variables (`lpca_rv1` through `lpca_rv5`) are specified as predictors in PROC GPCLASS.

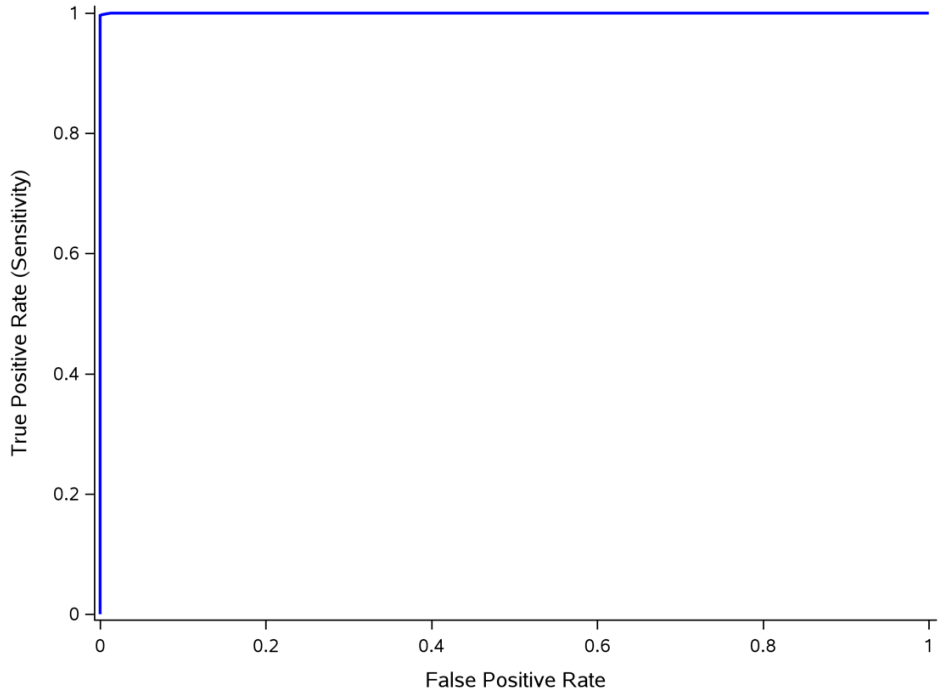
```
proc stdize data=lpcaTrain out=lpcaStdTrain OUTSTAT=lpcaStdSTAT method=std;
      var lpca_rv1-lpca_rv5;
run;
proc stdize data=lpcaTest out=lpcaStdTest method=in(lpcaStdSTAT);
      var lpca_rv1-lpca_rv5;
run;

proc GPCLASS data=lpcaStdTrain TESTDATA=lpcaStdTrain seed=12345;
    input lpca_rv1-lpca_rv5;
    target poisonous /LEVEL=NOMINAL;
    kernel gaussian(sigma=1);
    inference LA(maxIter=10 threshold=0.001);
    output out=GPCLASS_lpcaStd_train copyVars=poisonous;
    savestate rstore=lpcaStdModel;
run;
proc astore;
    score data=lpcaStdTest rstore=lpcaStdModel out=GPCLASS_lpcaStd_test
copyVars=poisonous;
run;
```

The classification results `GPCLASS_lpcaStd_train` and `GPCLASS_lpcaStd_test` contain the same variables as the `GPCLASS_mcaStd_train` table shown in Table 18. They are evaluated using PROC ASSESS as in Example 2. The fitting statistics are summarized in Table 20, and the ROC curve is plotted as shown in Figure 6.

Table 20. *Scoring Statistics of PROC GPCLASS for LPCA-Reduced and Standardized Mushroom Data*

Data	ASETrn	LogLikeTrn	AccTrn	ASETst	LogLikeTst	AccTst
LPCA-reduced	0.0060495	0.036008	0.99708	0.006757020	0.038659	0.99814

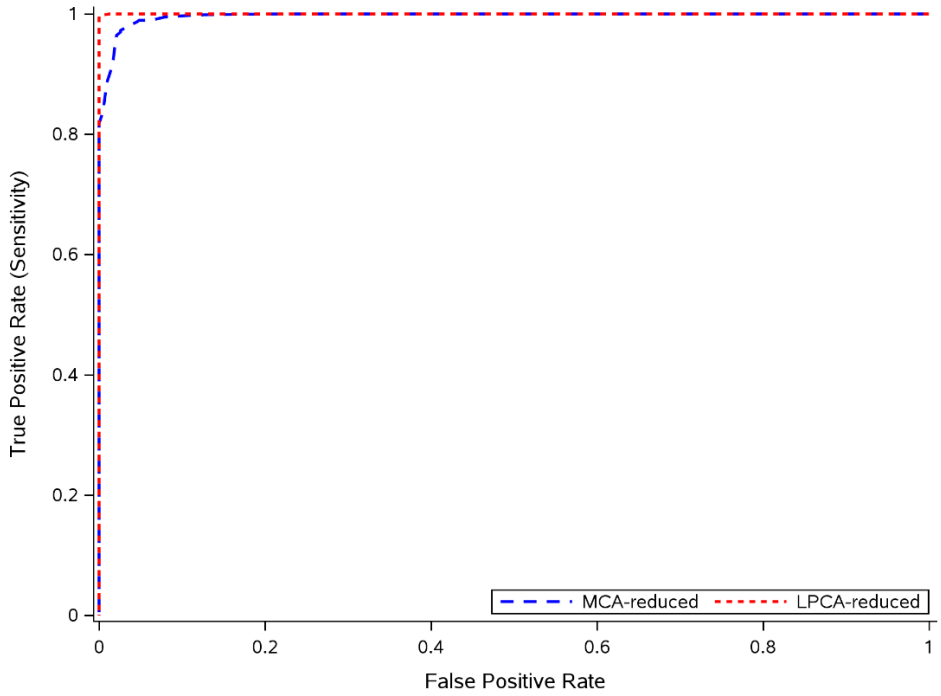
Figure 6. *ROC Curve (EVENT="p") of PROC GPCLASS for the LPCA-Reduced and Standardized Mushroom Data*



The 22 nominal input variables in the Mushroom data set are also reduced to five dimensions by using the LPCA method. These reduced variables are then standardized and input to PROC GPCLASS, achieving near-perfect accuracy (0.99814 in Table 20) and an almost perfect ROC curve as shown in Figure 6.

Figure 7 shows the ROC curves from PROC GPCLASS for the MCA-reduced and LPCA-reduced data together. Both curves are close to the curve of a perfect classifier. The curve that results from the LPCA-reduced data is closer to the perfect classifier curve than the curve that results from the MCA-reduced data.

Figure 7. *Comparison of ROC Curves (EVENT="p") from PROC GPCLASS for Mushroom Data*



Although PROC GPCLASS accepts only interval variables and therefore cannot directly analyze the nominal Mushroom data set, PROC NOMINALDR with either MCA or LPCA can transform the nominal data into lower-dimensional interval variables. Using these reduced interval variables, PROC GPCLASS achieves strong classification results, as shown in Tables 19 and 20 and Figure 7. In particular, the classification performance of PROC GPCLASS with the LPCA-reduced data is excellent, with greater than 0.99 accuracy and an almost perfect ROC curve.

Conclusion

In this paper, we have demonstrated how PROC NOMINALDR with the MCA and LPCA methods can reduce the dimensionality of nominal data as a preprocessing step. The first two examples show that both methods can substantially improve the efficiency of subsequent analysis while maintaining or even enhancing predictive performance on testing data. Although the first two examples illustrate single downstream analysis, using dimension-reduced data for multiple subsequent procedures would further save run time compared with using the original nominal data. The third example highlights that procedures that are restricted to interval variables can still be applied effectively to nominal data after preprocessing that uses PROC NOMINALDR.

References

Abdi, H., and Valentin, D. (2007). "Multiple Correspondence Analysis." In Salkind, N. J., ed., *Encyclopedia of Measurement and Statistics*, 1–13. Thousand Oaks, CA: Sage Publications. Available at https://personal.utdallas.edu/~herve/Abdi-MCA2007-pretty.pdf.

De Leeuw, J. (2006). “Principal Component Analysis of Binary Data by Iterated Singular Value Decomposition.” *Computational Statistics and Data Analysis* 50:21–39. Available at https://www.sciencedirect.com/science/article/pii/S0167947304002300.

Greenacre, M. (2017). *Correspondence Analysis in Practice*. 3rd ed. Boca Raton, FL: Chapman and Hall/CRC.

Khangar, N. V., and Kamalja, K. K. (2017). “Multiple Correspondence Analysis and Its Applications.” *Electronic Journal of Applied Statistical Analysis* 10(2), 432–462. Available at https://www.academia.edu/125560871/Multiple_Correspondence_Analysis_and_its_applications.

Landgraf, A. J., and Lee, Y. (2020). “Dimensionality Reduction for Binary Data through the Projection of Natural Parameters.” *Journal of Multivariate Analysis* 180:104668. Available at https://www.sciencedirect.com/science/article/pii/S0047259X20302499.

Michalski, R. S., and Chilausky, R. L. UCI Machine Learning Repository (1980). “Soybean (Large).” UCI MLR. Available at https://doi.org/10.24432/C5JG6Z.

SAS Institute Inc. (2025a). GPCLASS Procedure. Available at https://go.documentation.sas.com/api/collections/pgmsascdc/v_069/docsets/casml/content/casml.pdf?locale=it#nameddest=casml_gpclass_toc.

SAS Institute Inc. (2025b). LOGISTIC Procedure. Available at https://go.documentation.sas.com/doc/en/pgmsascdc/v_068/statug/statug_logistic_toc.htm.

SAS Institute Inc. (2025c). NNET Procedure. Available at https://go.documentation.sas.com/api/collections/pgmsascdc/v_069/docsets/casml/content/casml.pdf?locale=it#nameddest=casml_nnet_toc.

SAS Institute Inc. (2025d). NOMINALDR Procedure. Available at https://go.documentation.sas.com/api/collections/pgmsascdc/v_069/docsets/casml/content/casml.pdf?locale=it#nameddest=casml_nominaldr_toc.

SAS Institute Inc. (2025e). STDIZE Procedure. Available at https://go.documentation.sas.com/doc/en/pgmsascdc/v_068/statug/statug_stdize_toc.htm.

Schein, A. I., Saul, L. K., and Ungar, L. H. (2003). “A Generalized Linear Model for Principal Component Analysis of Binary Data.” In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 240–247. PMLR. Available at http://proceedings.mlr.press/r4/schein03a/schein03a.pdf.

UCI Machine Learning Repository (1991). “Molecular Biology (Splice-Junction Gene Sequences).” UCI MLR. Available at https://doi.org/10.24432/C5M888.

UCI Machine Learning Repository (1981). “Mushroom.” UCI MLR. Available at https://doi.org/10.24432/C5959T.

Release Information Content Version: 1.0 December 2025.

To contact your local SAS office, please visit: sas.com/offices