ROC Curve Plotting in SAS 9.2

ROC curve capabilities incorporated in the LOGISTIC procedure

With version 9.2, SAS introduces more graphics capabilities integrated with statistical procedures than were previously available. Most statistical procedure have certain graphical outputs which are frequently if not routinely employed to evaluate results. When a logistic regression is fit, ROC curves are routinely employed to summarize the model fit. The following ROC curves can be generated:

- The fitted model employing data from the estimation data set
- The fitted model employing data from an evaluation data set
- The fitted model at each step of a stepwise selection algorithm overlaid in a single plot
- ROC plots for multiple (continuous) markers overlaid in a single plot

Note that in order to take advantage of the graphical elements which are associated with a statistical procedure, one must have the SAS/GRAPH module licensed and installed. It is also necessary to inform the statistical procedure *before the procedure is invoked* that graphical results need to be generated and where the graphics are to be produced. The SAS statements below outline code that is needed.

ods graphics on;
ods <device specification>;
[*SAS/STAT procedure code*]
ods <device specification> **close;**
ods graphics off;

Frequently employed device specifications are **html, pdf, ps,** and **rtf.** The SAS/STAT procedure code will generally include some sort of plot specification. A simple example requesting an ROC curve would be:

```
ods graphics on;
ods html;
proc logistic data=mydata plots(only)=(roc);
   model Y=marker;
run;
ods html close;
ods graphics off;
```

Observe that in addition to the ODS statements requesting the availability of ODS graphics and naming an output device, the PROC LOGISTIC statement includes a

PLOTS specification that requests an ROC plot of the fitted model. Other plot specifications are available but not discussed here since our attention is limited to producing ROC curves.

If a stepwise selection process is invoked and the PROC LOGISTIC statement includes a request to produce an ROC curve, then two ROC curve plots are generated. The first plot displays the ROC curve for the final model while the second plot displays the ROC curve at each step of the estimation process. Note that step 0 has no predictors in the model. The step 0 ROC curve is simply the (uninformed model) curve where SENS=1-SPEC. In addition to displaying the ROC curves, the AUC for each ROC curve is written in a plot legend. Apart from the options which are required to obtain the stepwise selection model, the code for requesting the ROC curves is identical to previously shown code.

In order to apply the classifier obtained from some estimation data to a test data set, you must specify a SCORE statement. On the SCORE statement, you must name the data set which contains the test data and you must request that sensitivity and specificity be written to an output data set. Variables named on the MODEL statement must be found in the data named for scoring. Note that it is not necessary to invoke a plotting procedure (GPLOT) to display the plot of sensitivity vs 1-specificity. The LOGISTIC procedure will display the ROC curve in the test data set (and provide AUC in the test data) directly. The code below provides an example. Note that data for this example can be found at http://labs.fhcrc.org/pepe/dabs/datasets.html.

```
ods graphics on;
ods html;
proc logistic data=Remission_train plots(only)=(roc);
  model remiss(event='1')=li;
  score data=Remission_test outroc=roc_score;
run;
ods html close;
ods graphics off;
```

Finally, suppose that you have several candidate markers all in a single data set and you want to produce ROC curves for all of the markers in a single plot. For this, we no longer need the PLOTS option on the LOGISTIC statement (although the PLOTS option will still produce an ROC curve for each candidate marker separately). In version 9.2, there is an ROC statement which can be employed to construct a plot in which ROC curves for all markers are overlaid. All of the markers for which an ROC curve is to be generated must be named on the MODEL statement, and the NOFIT option must be specified on the MODEL statement. Each marker requires its own ROC statement. The code below illustrates:

```
ods graphics on;
ods html;
proc logistic data= plots=roc;
   model popind(event='0') = alb tp totscore / nofit;
   roc 'Albumin' alb;
   roc 'K-G Score' totscore;
```

```
   roc 'Total Protein' tp;
run;
ods html close;
ods graphics off;
```

Data for In addition to producing an overlay plot, the ROC statements will produce an asymptotic standard error for the area under the curve as well as confidence limits for the AUC of the named marker(s).

With the addition of an ROCCONTRAST statement, a test can be obtained examining whether the ROC curves for the various markers are different statistically. The ROCCONTRAST statement names a reference marker against which the other markers are tested. Each of the markers is tested against the marker named as the reference in a single df test. In addition, if there are ROC statements for k markers, a test with k-1 degrees of freedom will also be produced. Again, data for this example are found at the same web page as for the example showing ROC curve construction for a test data set.

```
ods graphics on;
ods html;
proc logistic data=MarkersCompare plots=roc;
   model popind(event='0') = alb tp totscore / nofit;
   roc 'Albumin' alb;
   roc 'K-G Score' totscore;
   roc 'Total Protein' tp;
   roccontrast reference('K-G Score') / estimate e;
run;
ods html close;
ods graphics off;
```

Accessing ROC curve functionality available in R

A suite of tools for constructing ROC curves which can be generated using the open source R program mentioned at http://labs.fhcrc.org/pepe/dabs/software.html can be executed directly from SAS version 9.2 if you run SAS on a Windows platform. In fact, any R code can be submitted directly from SAS as long as you have licensed the software which allows this functionality. When the R code produces a result which normally would be directed to the R Console window, those results are returned back to SAS for display. Any R graphics will be displayed or routed according to directives of the R code.

As previously mentioned, this seamless ability to execute R code directly from SAS requires license of appropriate software in SAS. Two products, one from SAS and one from an external vendor, are readily available for this purpose. If you have the SAS/IML procedure licensed, then you can execute R code from a SAS IML Studio 3.2 session. SAS IML Studio 3.2 is a separate (free) download from SAS. IML Studio 3.2 was not

shipped with SAS 9.2 as it was developed subsequent to the release of version 9.2. IML Studio 3.2 can be obtained from:

http://www.sas.com/apps/demosdownloads/setupcat.jsp;jsessionid=A0DD98A33508E6D8EE83E123928FAF1B.tomcat4?cat=SAS%2FIML+Software

It should be noted (and will be clarified below) that an IML Studio session is different in many regards from a regular SAS session. For readers who are familiar with the IML procedure in SAS and who understand that procedure boundaries in SAS prevent executing data step or other procedures while in an IML session, please note that with IML Studio these restrictions no longer hold. You can execute data step code and any other licensed SAS procedure code while in an IML Studio session. The IML Studio product is intentionally designed as an interactive type application. With some syntax which is totally unfamiliar to people who have previously used SAS and SAS/IML, IML Studio requires some time investment to become familiar with new ways of doing things. But with the ability to run any data step or other procedure code and with the ability to execute R code directly from an IML Studio session, there are significant payoffs to learning IML Studio. Those are but two of the capabilities available with this product. Other benefits include dynamic linking such that when plots are generated from an opened data source, then observations selected in one window are selected in all windows. This enables interactive graphic techniques like graphics brushing as introduced by Becker and Cleveland.

Bridge to R is third party software which has essentially the same functionality as the IML Studio product from SAS. With **Bridge to R**, you can submit code to an R session directly from a normal SAS session. That is, you do not need to invoke the IML Studio product from SAS. There is no need to learn a totally new product as is necessary with IML Studio. However, **Bridge to R** requires buying a license for a product that is not distributed with SAS and never will be distributed with SAS. It may or may not be worth the investment.

In order to submit R code from the IML Studio application, you must place the R code in a submit block. Submit blocks have structure

```
submit;
<code to be executed>
endsubmit;
```

Note that submit blocks are employed not only for submitting code to R, but also to submit data step and SAS procedure code to a SAS server. The SUBMIT statement takes options, and it is the option R on the SUBMIT statement which indicates that code is to be directed to R. Thus, a clearer indication of the usage of submit block code would be:

```
submit;
<SAS data step or procedure code to be executed>
endsubmit;

submit / R;
```

```
<R code to be executed>
endsubmit;
```

Now, it is likely that R code is being submitted from a SAS session because the user is performing data manipulation (and perhaps some analyses) in SAS, but R has some functions for data analysis which are not available in SAS. Thus, data exist in the SAS session, and must be passed to the R session. The submit block directs code to an R session, but the user also needs to exchange data between SAS and R – often in both directions.

Now, R has many data types. Data types available in R include matrices and data frames. Matrices must contain all character data values or all numeric data values. A data frame can contain numeric and character data. The data frame is much like a SAS data set in that it can have a mix of data types. Moreover, columns of a data frame can be named and the names used to reference the columns. That is, column names can be employed like SAS data set variable names. Users of the older IML procedure will immediately note that IML allows character and numeric matrices, but not a mix of character and numeric data. But IML Studio introduces to SAS the concept of data objects which can contain both character and numeric data. IML Studio still has matrices which are either character or numeric. The methods for passing data to and from R differ according to whether one is passing a data object or SAS data set, both of which become an R data frame, or a matrix.

In order to pass a matrix from IML Studio to R, one uses the ExportMatrixToR function. Basic syntax of the ExportMatrixToR function is

```
run ExportMatrixToR( SAS_matrix_name, “R.matrix.name”);
```

Note that the R matrix name must be enclosed in quotation marks while the SAS matrix name is not quoted. If you have produced a matrix in R which you wish to return to IML Studio, then there is a function ImportMatrixFromR which has syntax exactly like that of the matrix export function. The IML Studio matrix name is indicated first without quotation and then the R matrix name is indicated with quotation.

Unlike SAS which has an underlying C code base, IML Studio has a JAVA base. Many features of IML Studio require use of some JAVA syntax. Instead of passing matrices employing the ExportMatrixToR and ImportMatrixFromR functions, you can use what are referred to as JAVA methods to pass matrices. JAVA methods R.SetMatrix and R.GetMatrix pass a matrix to R and return a matrix from R. Syntax of R.SetMatrix and R.GetMatrix mirror the syntax of the ExportMatrixToR and ImportMatrixFromR functions in that the R matrix is named first (and quoted) while the SAS matrix is named second (without quotes). Thus, instead of the ExportMatrixToR function shown above, one could use

```
R.SetMatrix(“R.matrix.name”, SAS_matrix_name );
```

R data frames can be constructed from a SAS data set or from an IML Studio object. A function is employed to construct a data frame from a SAS data set while a JAVA method is employed to construct a data frame from an IML Studio object. To create a data frame from a SAS data set, one uses the ExportDataSetToR() function. To return a SAS data set from an R data frame, use the ImportDataSetFromR() function. Like the functions for passing matrices, the functions for passing data sets/data frames name the SAS data set first and the R data frame second. When we passed a matrix, the IML Studio matrix name was not quoted. The functions for passing SAS data sets require quoting of both the SAS data set name and the R data frame name. Also note that the SAS data set name must be the fully qualified (two-level) name, even if the data set is in a WORK directory. Thus, syntax to pass a SAS data set to R would be

```
run ExportDataSetToR( "dir.name", "R.name");
```

Syntax to return an R data frame as a SAS data set follows identical construction.

In order to pass an IML Studio data object, JAVA methods ExportToR and CreateFromR are employed. We prefix to these methods an IML Studio data object name. The data object prefixed to the method is passed to R or returned from R. Assuming that you already have a data object named DataObject, code to create an R data frame is

```
DataObject.ExportToR( "R.name" );
```

Note that the JAVA method is case-sensitive. However, the data object name is not case-sensitive. The R object name is also case sensitive. Code submitted to R through a submit block must reference the R object exactly as it is named when exported.

The following code creates a data object MyDOBJ from a SAS server data set and then passes the data object to R.

```
declare DataObject MyDOBJ;
MyDobj = DataObjectCreateFromServerDataSet("libname.filename");
MyDobj.ExportToR( "MyDOBJ" );
```

We now demonstrate submitting R code from an IML Studio session which accesses the ROC curve tools in the pcvsuite available at http://labs.fhcrc.org/pepe/dabs/rocbasic.html. The code shown in Appendix A assumes that you have installed the pcvsuite in R. After downloading the pcvsuite, begin an R session and click on

Packages → Install package(s) from local zip…

and follow the prompts from there.

After downloading and installing the IML Studio product, then launch an IML Studio session by clicking through the hierarchy Start → Programs → SAS → IML Studio 3.2. IML Studio should open up with a window that asks whether you want to start one of

four various activities: 1) Open a program or client data set; 2) Open a server data set; 3) Run a program; or 4) Create a new program. For this example, we will assume that you need to create a new program. If you are not presented with the window asking which of the various activities you would like to perform, then simply click on

File → New → Workspace

or click Control-N. Then enter the code found in Appendix A.

The program that we construct will perform the following tasks:

1) Execute data step code to read a CSV file from an internet source creating a SAS data set. This uses a submit block.
2) Create an IML Studio server data object.
3) Pass the IML Studio server data object to R where the server data object becomes a data frame.
4) Execute another submit block to pass code to R for execution. The R code will make available the pvcsuite tools with a library() function call and then construct an ROC plot using the roccurve() function from pvcsuite. The R code also queries whether the server data object passed to R is an R data frame and then lists the names of the elements in the data frame.
5) Control is passed back to IML Studio where a statement block separator is encountered. The statement block separator makes it possible to run only a section of code at a time. This is elaborated below.
6) After the statement block separator, there are additional invocations of the R function roccurve().

The purpose of the statement block separator for this example is two-fold. When the first block of R code is submitted, an R graphics window is produced. The R graphics window will remain open until it is closed by the user. After the first block of code has been executed, you may want to take time to turn on history recording of the R graphics so that when subsequent ROC curves are generated, you can review the results for a previous ROC curve request. The second purpose of the statement block separator is to demonstrate that the R session remains active throughout your entire IML Studio session. When control is returned to IML Studio, objects which were created by the first block of code passed to R remain available for later use. Also, you do not need to execute the library() function call for the pcvsuite functions to remain available.

In order for the statement block separator to be honored, a feature of IML Studio called Statement Mode must be turned on. Before running the code in Appendix A, you can turn on Statement Mode by clicking on Program → Statement Mode or simply by hitting the F4 key. There is also an icon on the toolbar for turning Statement Mode on and off With Statement Mode activated, copy the code shown in Appendix A to your IML Studio session. Submit the code in the top portion of the program by placing your cursor above the statement block separator and clicking on Program → Run or by hitting the F5 key.

Once again, there is an icon on the toolbar (a green "Play" button) which would cause the IML Studio code to be executed.

After executing the code above the statement block separator, an R graphics window should be displayed. Information which would normally be printed to the R Console is also displayed in the IML Studio Output window. Examine both the R graphics window (where the requested ROC curve is displayed) as well as the IML Studio Output window. Also, begin recording of the R graphics session by clicking on History → Recording.

After reviewing the ROC curve, submit the code below the statement block separator. You must place your cursor in the second statement block in order to submit the code which is in the second statement block. With your cursor in the proper location, use any of the methods described above in order to execute the code belonging to the second statement block.

Documentation of the roccurve() function can be found with the pcvsuite. Briefly, invocation of roccurve() in the first statement block produces an ROC curve for the response D (diseased) for a single marker Y1. Invocations of roccurve() in the second statement block result in: 1) a plot of ROC curves for two markers, Y1 and Y2, on the same plot, and 2) a plot of the ROC curve for marker Y1 with a 95% confidence interval for sensitivity at 90% specificity (1-specificity=0.10).

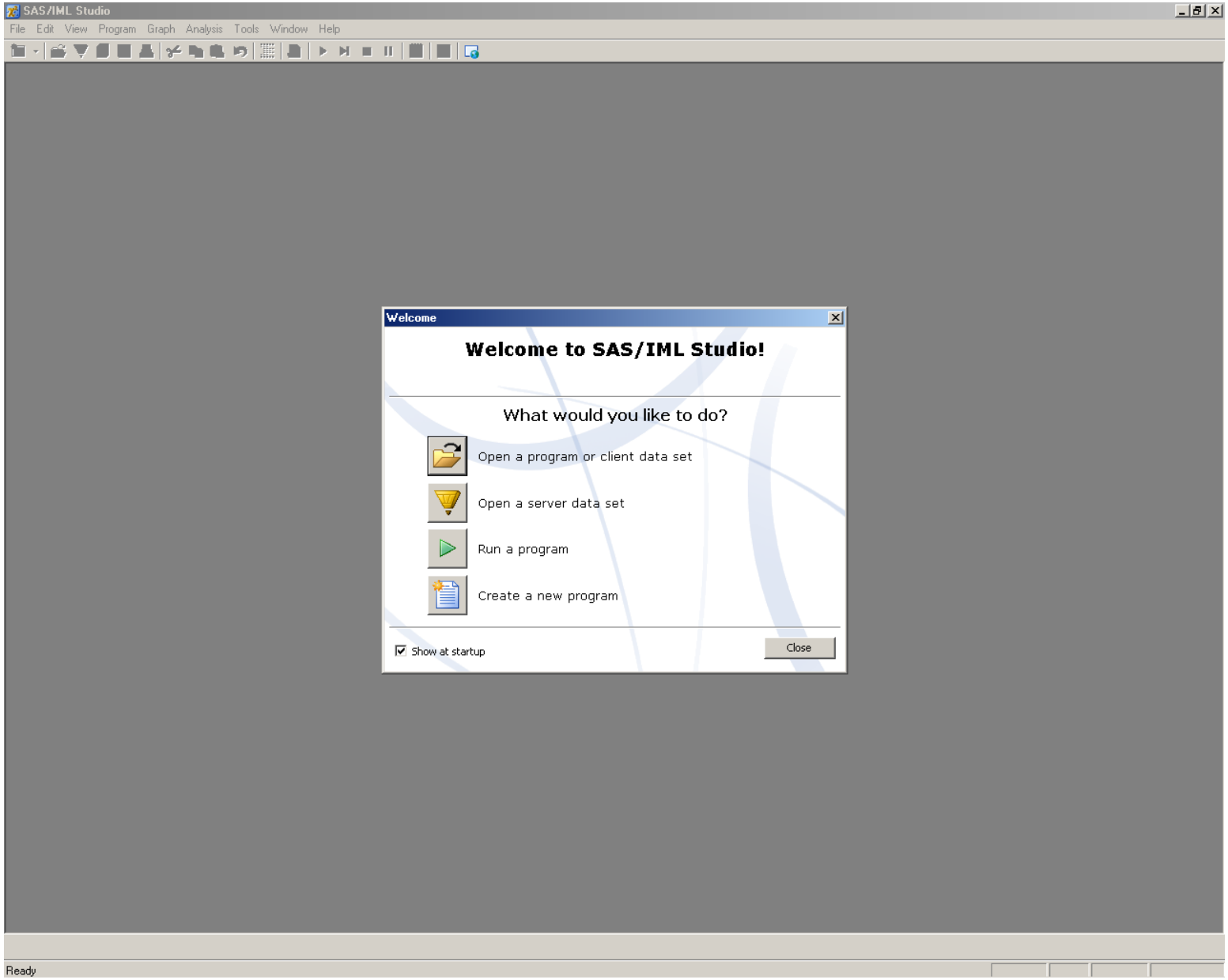
Screen shots of the IML studio session at various points along the way are shown in Appendix B.

Appendix A

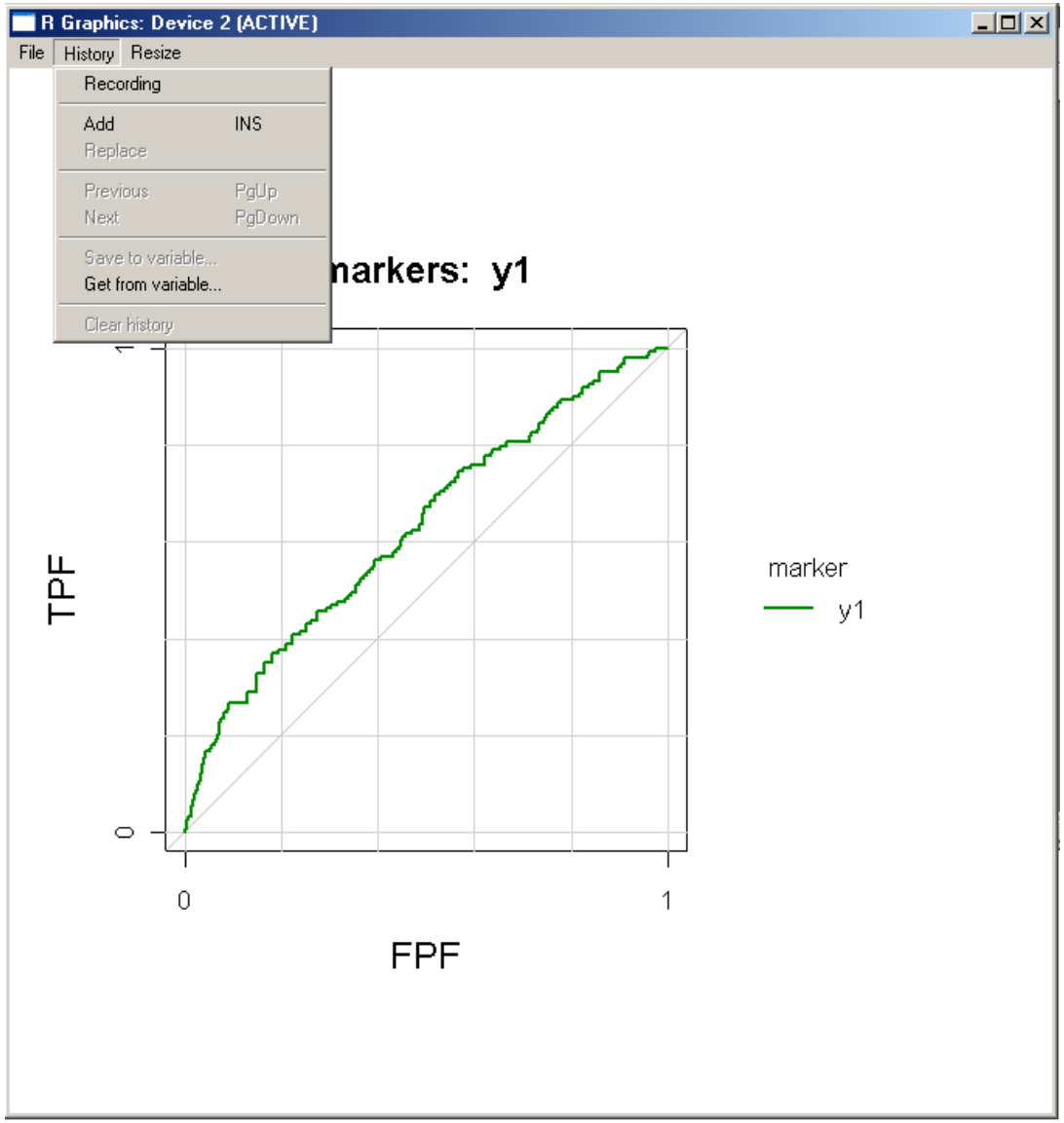
```
/* SUBMIT block statement to submit data step and SAS procedure code */
/* Data step reads a CSV file from an internet source.  The first 5  */
/* observations of the data set are then printed using PROC PRINT.   */
/* PROC PRINT results are displayed in the IML Studio Output window. */
submit;
  filename nnhs2 url “http://labs.fhcrc.org/pepe/book/data/nnhs2.csv”;

  data nnhs2;
    length id $ 5;
    infile nnhs2 firstobs=2 delimiter=”,” dsd;
    input id ear sitenum currage gender d y1 y2 y3;
  run;

  proc print data=nnhs2(obs=5);
  run;
endsubmit;


/* Create IML Studio data object and pass the data object to R */
declare DataObject nnhs2;
nnhs2 = DataObject.CreateFromServerDataSet(“work.nnhs2”);
nnhs2.ExportToR( “nnhs2” );


/* Now execute the first roccurve() function in R */
submit / R;
  library(pcvsuite)
  dataframe.test <- is.data.frame(nnhs2)
  print(dataframe.test)
  names(nnhs2)
  roccurve(dataset=”nnhs2”, d=”d”, markers=”y1”)
endsubmit;


/*** statement block separator ***/

submit / R;
  roccurve(d=”nnhs2$d”, markers=c(“nnhs2$y1”, “nnhs2$y2”))
  roccurve(dataset=”nnhs2”, d=”d”, markers=”y1”, roc=0.10,
noccsamp=TRUE, nsamp=100)
endsubmit;


/*** statement block separator ***/
```

Appendix B

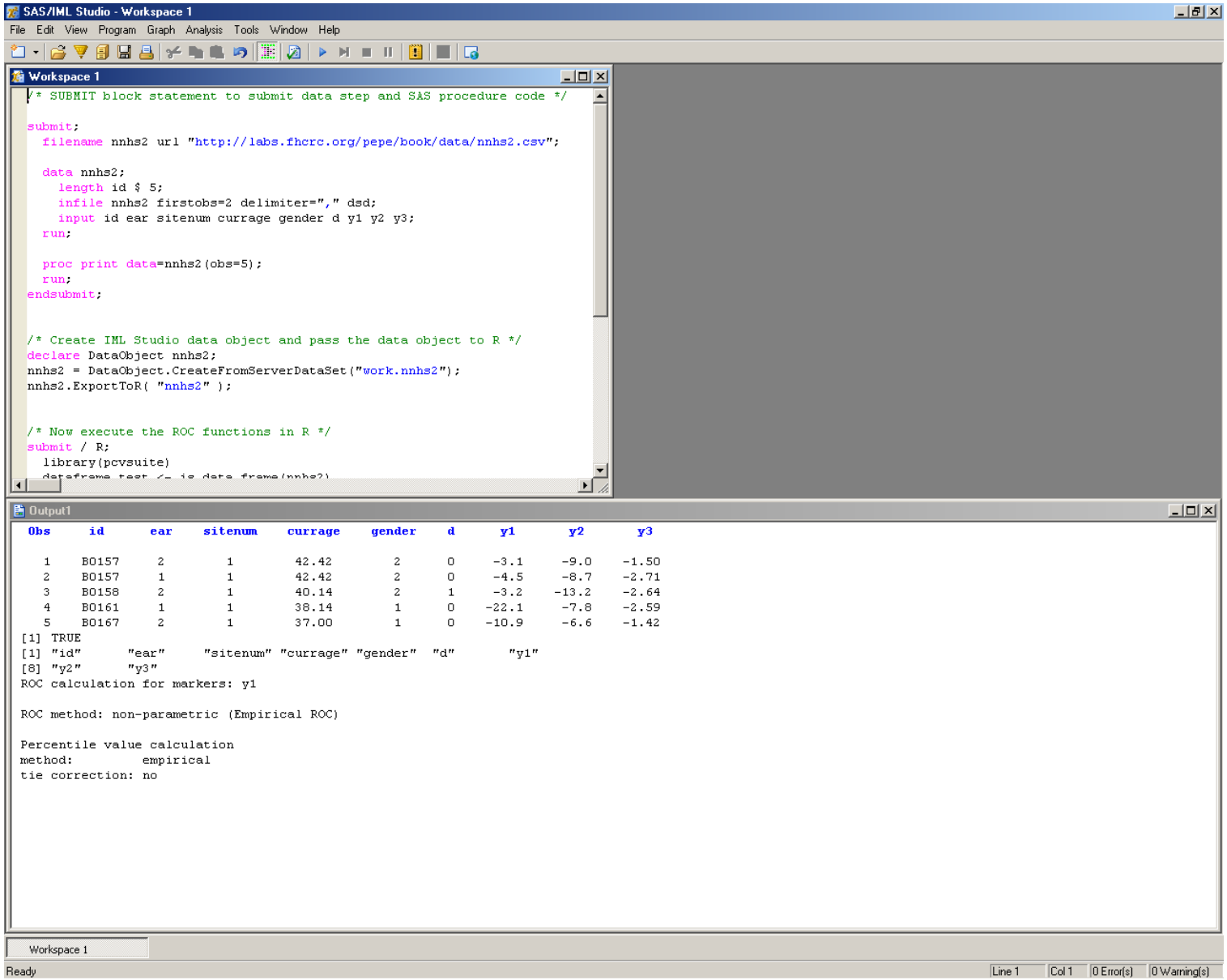
Screen shot of IML Studio on startup. Select “Create a new program” to get started with the code in Appendix A.



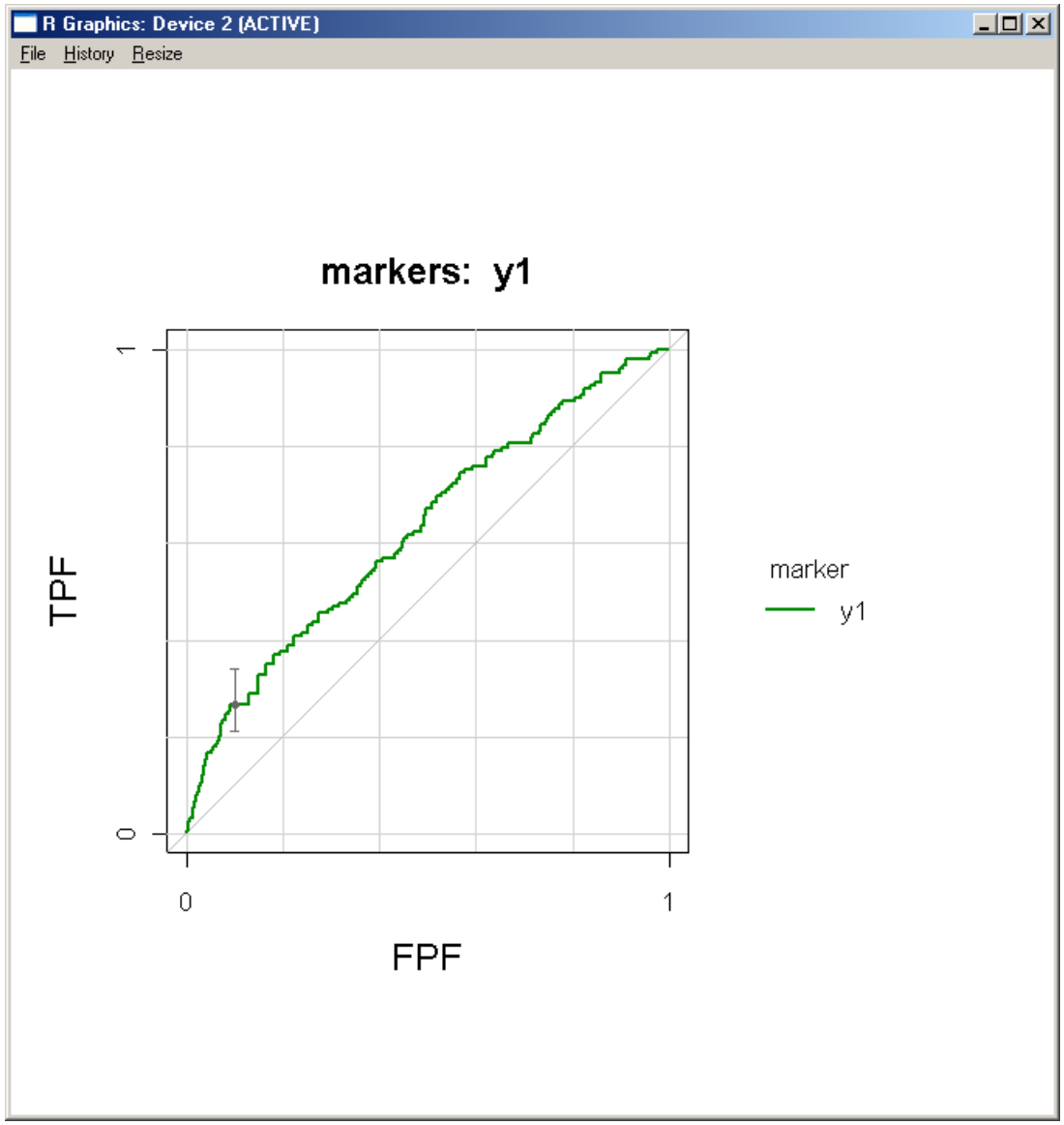
Screen shot displaying the ROC curve produced by the first R code statement block roccurve() request Selecting the History → Recording option in the R graphics window at this point will allow scrolling back to this plot for comparison after subsequent roccurve() requests in the next IML Studio statement block.



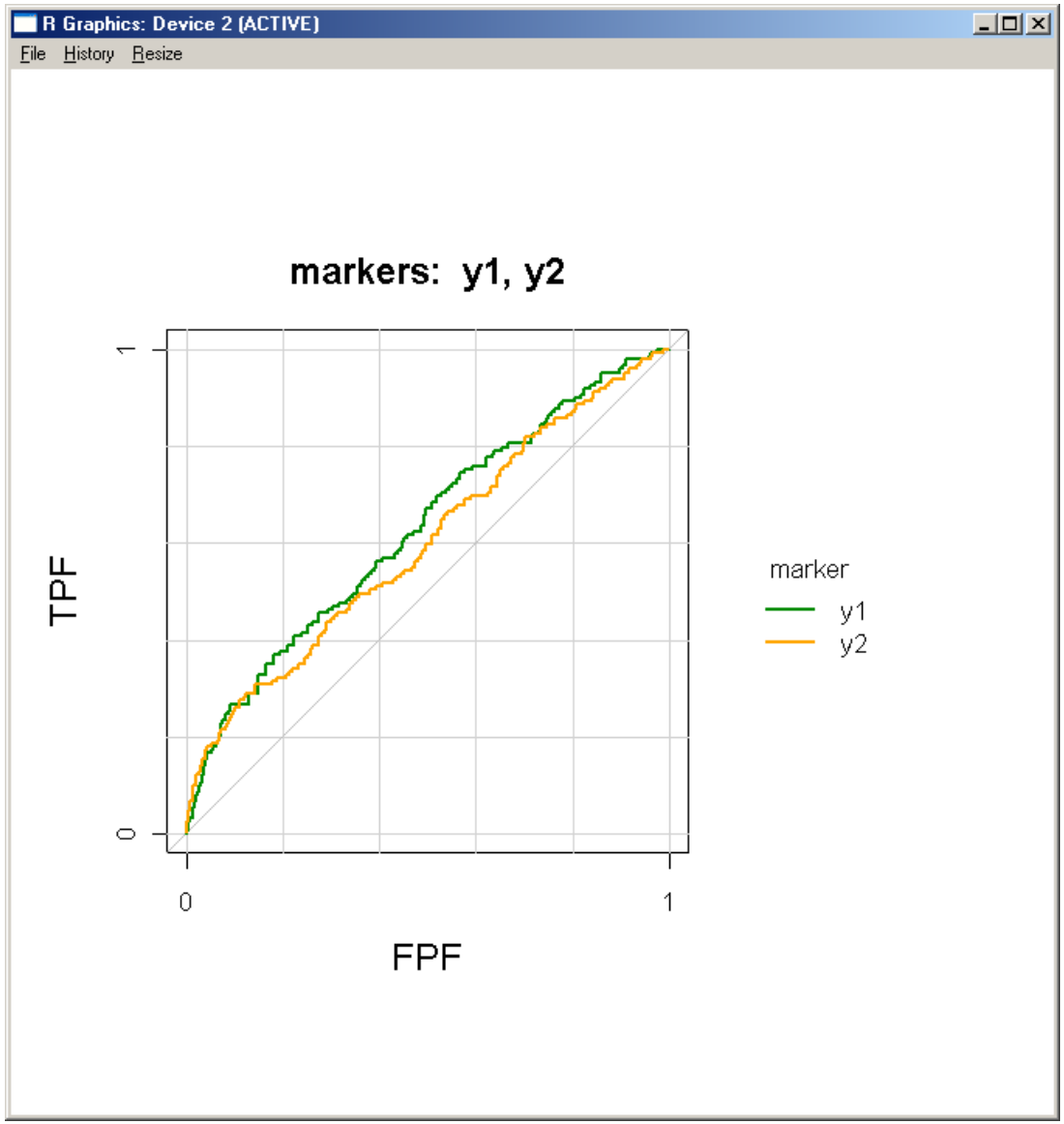
State of IML Studio upon completion of the first statement block. Note that the first submit block constructed the data set nnhsw as a SAS data set. The first 5 observations of nnhs2 were then printed. R code tested whether nnhs2 in R was a data frame. This is shown as TRUE. Names ("id", "ear", …) of the columns in the data frame are then shown. Finally, there is some output from roccurve() which would be directed to the R Console and which is captured in the IML Studio Output window.



R Graphics window after the second statement block is executed. Note that the ROC curve displayed here is identical to the first ROC curve. However, there is a 95% confidence interval for sensitivity at 90% specificity (1-specificity=0.10).



R Graphics window after scrolling back through the history to obtain the plot with ROC curves for markers Y1 and Y2.



State of IML Studio after execution of code in the second statement block. Invocations of roccurve() in the second statement block report information to the R Console which is captured and displayed in the IML Studio Output window.