

Open Source Integration Using the Base SAS® Java Object

Patrick Hall, Radhikha Myneni, Ruiwen Zhang, and Tim Haley
SAS Institute Inc.

ABSTRACT

This paper illustrates a new technique that uses the Base SAS® Java Object to integrate SAS software and popular open source packages. The communication medium between SAS and the open source packages is a pair of freely available Java classes and data sets in standard comma-separated-value (CSV) format. Examples in this paper pass CSV data sets to support vector machine (SVM) models that are implemented with scikit-learn in Python and kernlab in R, and return the modeling results to SAS for assessment.

INTRODUCTION

The capability to run third-party packages from SAS enables statisticians and data scientists to use open source tools alongside the statistical, data mining, and machine learning methodologies that are available in SAS. This paper describes a methodology that uses the Java Object DATA step component to execute Python and R scripts from Base SAS. First introduced in 2004 with SAS 9.1.3, the Base SAS Java Object provides a mechanism for instantiating and accessing Java objects from within a DATA step. This interface serves as a gateway within SAS to call Java programs, enabling users to create hybrid solutions that bring together the capabilities of SAS and Java. Applications for various purposes such as transferring data streams, reading archived data, data visualization, and data persistence have been implemented using the Base SAS Java Object (DeVenezia 2004; Mengelbier and Skowronski 2008; Hall and Snyder 2012).

CONFIGURATION AND EXAMPLE DATA

INITIAL SETUP

To begin, you download files from the following GitHub repository:

https://github.com/sassoftware/enlighten-integration/tree/master/SAS_Base_OpenSrcIntegration

The downloaded directory is referred to as WORK_DIR throughout the paper and is assumed to be C:\SGF2015\OpenSrcIntegration in the examples.

The project directory (WORK_DIR) contains the following:

- Training and test data sets: digitsdata_17_train.csv and digitsdata_17_test.csv
- A Java class: SASJavaExec.java
- A Python script and an R script: digitsdata_svm.py and digitsdata_svm.R
- A SAS program: main_caller.sas

Next, you open main_caller.sas, find the WORK_DIR, PYTHON_EXEC_COMMAND, and R_EXEC_COMMAND variables located at the top of the file, and update them appropriately for your system. The WORK_DIR variable specifies the working directory of the project. The PYTHON_EXEC_COMMAND and R_EXEC_COMMAND variables should specify the location of the Python and R executables (as shown in the following example); these locations can vary depending on where these packages are installed.

```

*** WORKING DIRECTORY      (----- USER UPDATE NEEDED -----);
%let WORK_DIR = C:\SGF2015\OpenSrcIntegration;
*** SYSTEM PYTHON LOCATION (----- USER UPDATE NEEDED -----);
%let PYTHON_EXEC_COMMAND = C:\Anaconda\python.exe;
*** SYSTEM R LOCATION      (----- USER UPDATE NEEDED -----);
%let R_EXEC_COMMAND = C:\Program Files\R\R-3.1.2\bin\x64\Rscript.exe;

```

The example code was developed and tested using SAS 9.4, Anaconda Python 2.7, R 3.1.2, and Oracle Java Development Kit (JDK) 1.7.0_25 on a Windows 7 Professional workstation with 16G RAM. Having suitable installed versions of Java, Python, and R is a prerequisite to run the example code, but their installation is beyond the scope of this paper.

DATA

The example data are a subset of the MNIST database, which is a set of labeled handwritten digit images that are commonly used for experimenting with new approaches in the field of machine learning and image processing (Wikipedia 2015). The label (dependent variable) is the actual digit that the image represents, which is located in the first column of the example data. The 784 predictor columns that follow contain the associated pixel intensity values. The example data include only the digits 1 and 7. The digits are equally balanced within the training and test data in the provided `digitsdata_17_train.csv` and `digitsdata_17_test.csv` files, respectively. CSV format is used for both the input data files and the output files from R and Python.

COMPILING THE PROVIDED JAVA CLASSES

The source code for the Java classes is located in `WORK_DIR\src\dev`; it should be compiled and placed in `WORK_DIR\bin\dev` directory. To compile the Java classes, change directories to the `WORK_DIR` directory and issue the following `javac` command. This operation assumes that an appropriate JDK is part of the `PATH` environment variable.

```

cd C:\SGF2015\OpenSrcIntegration
javac src/dev/* -d bin

```

SETTING THE JAVA CLASSPATH

The Base SAS Java virtual machine (JVM), on which the Base SAS Java Object depends, needs the location of the compiled Java classes to access the additional functionality that is implemented in `SASJavaExec.java` Java class. This location is contained in the `CLASSPATH` environment variable. There are multiple ways to provide the `CLASSPATH` to SAS (Snyder and Hall 2012). For this example, it is provided by adding the following line of text to the bottom of the `sasv9.cfg` configuration file for the local SAS installation:

```

-SET CLASSPATH "C:\SGF2015\OpenSrcIntegration\bin"

```

The `sasv9.cfg` file is often located in a directory similar to `C:\Program Files\SASHome\SASFoundation\9.4\pls\en`. If necessary, you can refer to multiple directories in the `CLASSPATH` by concatenating them using a semicolon (;) in a Windows environment or a colon (:) in UNIX and Linux environments. The following code snippet from `main_caller.sas` performs a quick test to check whether the `CLASSPATH` variable is set correctly. When the `JAVA_BIN_DIR` variable points to

the directory where the compiled Java classes are located, these code statements should not return an error. If there is an error in the SAS log after running this code snippet, do not proceed further without correcting the problem.

```
*** JAVA LIBRARIES/CLASS FILES LOCATION;

%let JAVA_BIN_DIR = &WORK_DIR.\bin;

*** VALIDATE JAVA CLASSPATH;

data _null_;
    length _x1 $32767;
    _x1 = sysget('CLASSPATH');
    _x2 = index(upcase(trim(_x1)), %upcase("&JAVA_BIN_DIR"));
    if _x2 = 0 then put "ERROR: Invalid Java Classpath.";
run;
```

PROGRAM FLOW: MAIN_CALLER.SAS

The SAS program main_caller.sas contains the primary execution flow. After you successfully complete all the preceding configuration steps, you can run this program in a SAS session. The following sections sequentially explain the work done by this program.

JAVA: SASJAVAEXEC.JAVA

The Base SAS Java Object is used to call the Java class SASJavaExec. This class provides a universal mechanism for calling Python, R, or executables in other languages from SAS. In addition to the constructors, SASJavaExec has several public methods including: executeProcess, executeProcess with a timeout argument, and setWorkingDirectory.

The SASJavaExec constructors take two or three string arguments. The first argument is intended to be the executable program that will process the script file. The path to the script and any command-line parameters are specified by the second argument. If a third argument is provided, it is passed as standard input to the running process.

In this paper, the constructor arguments are calls to Python and R scripts, but they can be any valid executable command and their necessary command-line arguments and interactive input.

The executeProcess method creates a new Process object using the constructor arguments, redirects STDERR stream to STDOUT, and begins executing the process. The STDOUT stream carries log and error output messages that are printed to the SAS log. When an error occurs (say, in a Python or R script), process execution is halted and SASJavaExec returns the exit code from the script. But when the process completes without any errors, the return code is 0.

Although executeProcess with the timeout argument is never used in this example, it is an alternative method that provides a timeout option to force an external process to quit after a certain predefined period of time in minutes. This method can be used to prevent an errant process from hanging. An additional method, setWorkingDirectory, is provided to set the directory in which the process will execute.

PYTHON: EXECUTING DIGITSDATA_SVM.PY

In the following DATA step, the Java Object is instantiated by giving the class name (dev.SASJavaExec) and its two constructor arguments. The constructor arguments are the commands that include the location of a Python executable (C:\Anaconda\python.exe) and the Python script to be executed

(digitsdata_svm.py) along with any additional command-line arguments that need to be passed to that script. The executeProcess method in SASJavaExec then executes the concatenated string of the two constructor arguments.

In the following example code, WORK_DIR is passed in as an argument to the Python script in the python_call variable. This is necessary because the script needs access to the working directory, from which input data are read and to which output data are written.

```
/** Part I: Python */
data _null_;
    length rtn_val 8;
    /** Python program takes working directory as first argument;
python_pgm = "&WORK_DIR.\digitsdata_svm.py";
python_arg1 = "&WORK_DIR";
python_call = cat("'", trim(python_pgm), "' ", trim(python_arg1), "'");

    declare javaobj j("dev.SASJavaExec", "&PYTHON_EXEC_COMMAND",
                        python_call);
    j.callIntMethod("executeProcess", rtn_val);
run;
```

The digitsdata_svm.py script uses the pixel intensity values for each image in the training data to build an SVM model that assigns a label to the handwritten digits. The fitted model is then used to assign a predicted label to the images in the test data. The assigned labels are output to a CSV file named predict_test_python.csv in WORK_DIR.

The following example Python code from digitsdata_svm.py shows an SVM model fitting the training data and scoring the test data. Output 1 contains statements from the SAS log after successful execution of this step.

```
clf = svm.LinearSVC()
clf.fit(X, y)
pred = clf.predict(Xtest)
```

```
Apr 07, 2015 3:54:47 PM dev.SASJavaExec executeProcess
INFO: Executing [C:\Anaconda\python.exe,
C:\SGF2015\OpenSrcIntegration\digitsdata_svm.py, C:\SGF2015\OpenSrcIntegration] ...
Apr 07, 2015 3:54:47 PM dev.SASJavaExec executeProcess
INFO: Starting external process ...
Apr 07, 2015 3:54:47 PM dev.SASJavaExec executeProcess
INFO: External process exit value 0.
NOTE: DATA statement used (Total process time):
      real time           0.73 seconds
      cpu time            0.01 seconds
```

Output 1. Output from SAS Log after Executing the Python Script

R: EXECUTING DIGITSDATA_SVM.R

The execution of the R script is very similar to that of the Python script. The only difference is that instead of Python executable and script names, the R executable and script names are passed as the constructor arguments. Here, R_EXEC_COMMAND points to Rscript.exe because it is the command-line alternate to the R graphical user interface (GUI).

```
/** Part II: R */
data _null_;
    length rtn_val 8;
    *** R program takes working directory as first argument;
    r_pgm = "&WORK_DIR.\digitsdata_svm.R";
    r_arg1 = "&WORK_DIR";
    r_call = cat("'", trim(r_pgm), " ", trim(r_arg1), "'");

    declare javaobj j("dev.SASJavaExec", "&R_EXEC_COMMAND", r_call);
    j.callIntMethod("executeProcess", rtn_val);
run;
```

Much like the Python script, the R script digitsdata_svm.R uses the training data to build an SVM model, which is then used to make predictions of the digit label for the test data. The output is written to a CSV file named predict_test_R.csv in WORK_DIR.

The following example code from digitsdata_svm.R shows the SVM model fitting the training data and scoring the test data. Output 2 shows statements from the SAS log after successful execution of this step.

```
svmmmodel = ksvm(label~., data=train, type='C-svc', scaled=c(FALSE))
pred = predict(svmmmodel, test[, -1])
```

```
Apr 07, 2015 3:56:01 PM dev.SASJavaExec executeProcess
INFO: Executing [C:\Program, Files\R\R-3.1.2\bin\x64\Rscript.exe,
C:\SGF2015\OpenSrcIntegration\digitsdata_svm.R, C:\SGF2015\OpenSrcIntegration] ...
Apr 07, 2015 3:56:01 PM dev.SASJavaExec executeProcess
INFO: Starting external process ...
Apr 07, 2015 3:56:01 PM dev.SASJavaExec handleProcessOutput
INFO: PROCESS OUTPUT >>> Loading required package: methods
Apr 07, 2015 3:56:02 PM dev.SASJavaExec handleProcessOutput
INFO: PROCESS OUTPUT >>> Using automatic sigma estimation (sigest) for RBF or
laplace kernel
Apr 07, 2015 3:56:02 PM dev.SASJavaExec executeProcess
INFO: External process exit value 0.
NOTE: DATA statement used (Total process time):
      real time           1.49 seconds
      cpu time           0.01 seconds
```

Output 2. Output from SAS Log after Executing the R Script

ASSESSING RESULTS IN SAS

The last step is to bring all the test data predictions together by using PROC IMPORT in Base SAS to convert CSV files that were produced by Python and R scripts into SAS data sets. Figures 1 and 2 are produced by using PROC FREQ, which prints confusion matrices that compare the actual and predicted label assignments.

Table of label by pred_py

label	pred_py		
Frequency Percent	1	7	Total
1	100 50.00	0 0.00	100 50.00
7	2 1.00	98 49.00	100 50.00
Total	102 51.00	98 49.00	200 100.00

Figure 1. Actual (label) versus Predicted (pred_py) Using SVM in Python

Table of label by pred_r

label	pred_r		
Frequency Percent	1	7	Total
1	99 49.50	1 0.50	100 50.00
7	0 0.00	100 50.00	100 50.00
Total	99 49.50	101 50.50	200 100.00

Figure 2. Actual (label) versus Predicted (pred_r) Using SVM in R

The misclassification rate, which is defined as the proportion of misclassified predictions out of the total is 2/200 (1%) for the SVM model in Python and 1/200 (0.5%) for the SVM model in R. (The goal of this paper is not to build the best or most comparable models between Python and R, but to show how these packages can easily be integrated with SAS.)

SUMMARY

This paper gives an example of how data can be manipulated, modeled, scored, and assessed using SAS, Python, R, and the Base SAS Java Object. However, the presented integration scheme is not limited to the Python and R scripts. The functionality in Base SAS to run Java programs by using the Base SAS Java Object is a flexible and powerful integration tool that enables users to mix and match techniques from many different types of third-party software.

REFERENCES

DeVenezia, Richard A. 2004. "Greetings from the Edge: Using javaobj in DATA Step." *Proceedings of the Twenty-Ninth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute Inc. Available <http://www2.sas.com/proceedings/sugi29/033-29.pdf>

Hall, Patrick, and Snyder, Ryan. 2012. "Let the Data Paint the Picture." *Proceedings of the SAS Global Forum 2012 Conference*. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/resources/papers/proceedings12/008-2012.pdf>

Mengelbier, Magnus, and Skowronski, Jan. 2008. "SAS Talking via the Java Object Interface." *Proceedings of the SAS Global Forum 2008 Conference*. Cary, NC: SAS Institute Inc. Available <http://www2.sas.com/proceedings/forum2008/027-2008.pdf>

"MNIST Database." 2015. Wikipedia. Accessed March 20, 2015. http://en.wikipedia.org/wiki/MNIST_database

Snyder, Ryan, and Hall, Patrick. 2012. "A Guide for Connecting Java to SAS Data Sets." *Proceedings of SAS Global Forum 2012 Conference*. Cary, NC: SAS Institute Inc. Available <http://support.sas.com/resources/papers/proceedings12/008-2012.pdf>

CONTACT INFORMATION

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

Patrick Hall
patrick.hall@sas.com

Radhikha Myneni
radhikha.myneni@sas.com

Ruiwen Zhang
ruiwen.zhang@sas.com

Tim Haley
tim.haley@sas.com

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

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

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