Paper AD-133

New Game in Town: SAS® Proc Lua with Applications

Jiangtang Hu
d-Wise, Morrisville, NC

ABSTRACT

Lua (pronounced as 'lu-aa') is scripting language like Python. It is very popular in gaming industry. In recent years it's also moved to machine learning world with a widely used library *Torch 7*, which is one of the core libraries used in the AlphaGo system. SAS introduced Lua to its Base module since version 9.4 which will definitely extend SAS programming functionality.

In this paper, several examples (dynamic programming as replacement of SAS macro, reading external files like JSON, XML, HDF5, and implementing a machine learning algorithms like Naïve Bayes) will be presented to showcase how Proc Lua can add to SAS programmers' toolbox. All codes will be available in Github:

https://github.com/Jiangtang/SESUG/tree/master/2016

INTRODUCTION

Lua (pronounced as 'lu-aa') is scripting language like Python. It is very popular in gaming industry. In recent years it also moved to machine learning world with a widely used library *Torch 7*, which is one of the core libraries used in the AlphaGo system which beat world champion Lee Sedol. SAS introduced Lua to its Base module since version 9.4 which will definitely extend SAS programming functionality.

We definitely will not use Lua in SAS to build games like Angry Bird or create machine learning system like AlphaGo. In this paper, several examples will be presented to showcase how Proc Lua can add to SAS programmers' toolbox:

Conditional and dynamic programming as replacement or alternative of SAS macro,

reading external files like JSON, XML, HDF5, and

implementing a machine learning algorithms like Naïve Bayes

All codes will be available in Github:

https://github.com/Jiangtang/SESUG/tree/master/2016

Note that this paper is not a Lua 101 but rather showcase some Lua enrichment to SAS programming. The good news is the Lua scripting language is pretty self-explanatory and you won't have trouble at least to READ it.

A. MACRO ALTERNATIVES

A.1 WRAPER

In conditional programming settings, we need some wrapper programs to trigger out conditions like:

1. If a dataset exists
2. If a variable exists
3. Or any other conditions in and out of SAS system

One valid method is to put it to a Data Step by using SAS external file functions like **fexist**:

```
data _null_;
    fname="tmp";
```

```
    rc=filename(fname,"class.sas7dat");
    if fexist(fname) then do;
       put "haha";
    end;
    else put "wow";
    rc=filename(fname);
run;
```

The limitation of programming behavior under a Data Step is the drawback of above approach. Another is the macro wrapper. Since the %if is not a valid statement in open code in SAS, usually people will wrap it in a macro call:

```
%macro check_it(ds);
      %if %sysfunc(exist(&ds)) %then %do;
            %put &ds exists. Good!;
      %end;
      %else %do;
          %put ERROR: &ds does not exist. Bad!;
          %abort cancel;
      %end;
%mend check_it;
%check_it(sashelp.class)
```

The macro wrapper is flexible enough to perform almost any SAS programming tasks. The drawback, I might argue, is aesthetic consideration. It's verbose, even for a simple conditional evaluation.

```
proc lua ;
      submit;

            if sas.exists("sashelp.class") then
                  print "haha"
                  sas.submit [[
                    data class;
                        set sashelp.class;
                        a = 1;
                    run;

                    proc print data=class;
                    run
                   ]]
               else print "clash.."
             end

      endsubmit;
run;
```

The benefits of Proc Lua wrapper at least:

1. It looks elegant. So it's much preferable from aesthetic point of view.
2. The programming elements are rich enough inside the Proc Lua wrapper: you can run Lua scripts, and also SAS Data Steps and Procedures.
3. Easy to debug.

A.2 DYNAMIC PROGRAMMING

SAS offers rich techniques on dynamic programming (or data driven programming), for example:

1. Macro array
2. Call execute

For a quick demo, I will start with a simple scenario in which the data set **sashelp.zipcode** should be spitted to pieces of datasets by states (in real projects, the codes would be more complicated but share the simple atom structure). For example, the dataset for North Carolina:

```
data zipcode_NC;
    set sashelp.zipcode(where=(statecode="NC"));
run;
```

There are 50 states plus territories like Puerto Rico in the source data, so you won't just use a simple string replacement by macro variable.

```
proc sort data=sashelp.zipcode (keep=statecode) nodupkey out=statecode;
    by statecode;
run;

data _null_;
    set statecode end=eof;
    i+1;
    II=left(put(i,2.));
    call symputx('statecode'||II,statecode);
    if eof then call symputx('n',II);
run;

%macro doit;
%do i=1 %to &n;
    data class_&&statecode&i;
        set sashelp.zipcode(where=(statecode="&&statecode&i"));
    run;
%end;
%mend;
%doit
```

And the Proc Lua version:

```
proc lua ;
      submit;

      local dsid =sas.open("sashelp.zipcode")

        local state = {}
        local hash = {}
        local res = {}

        for row in sas.rows(dsid) do
            state[#state+1] = row.statecode
            for _,v in ipairs(state) do
                  if (not hash[v]) then
                        res[#res+1] = v
                        hash[v] = true
                  end
            end
        end

        sas.close(dsid)
```

```
      for i,state in ipairs(res) do
             sas.submit[[
                    data class@state@;
                        set
sashelp.zipcode(where=(statecode="@state@"));
                    run;
              ]]
      end

      endsubmit;
run;
```

For this particular example, Proc Lua seems not a better choice since at least it needs more lines of codes and not necessary simpler than the macro array version above. But from the Proc Lua codes, they are lots of good potentials for applications:

1. It can iterate with SAS datasets
2. Advanced data structure like hash can be applied to SAS programming
3. Advance (and REAL) For Loop can be used

B. SAS OPTIONS MANAGEMENT

SAS options are global. It's designed in this way for good purpose. But sometimes you just want to evoke a SAS option but are too lazy to close it at the end. Proc Lua is good to have in this case:

```
proc lua;
    submit;
        sas.submit[[
            options obs=1;
            proc print data=sashelp.class(keep=sex age);
            run;
        ]]
    endsubmit;
run;


proc print data=sashelp.class(keep=sex age);
run;
```

The trick is the **options** inside the **SAS.SUBMIT** blocks are pretty local; it will not be inherited by the outside programs.

C. EXTERNAL FILES

SAS offers a full range of external files support, but there is at least one I can tell, that there is no native JSON support! This looks odd since JSON is much the standard file format for internet exchange. Another is HDF5.

Lua itself has several packages to read JSON, HDF5 and other data formats. In this example, I take a JSON library from

http://regex.info/blog/lua/json

by Jeffrey Friedl. This is a way to load it to SAS:

```
proc lua ;
      submit;

    JSON = (loadfile "JSON.lua")() -- one-time load of the routines

    local raw_json_text = '{"Hello":["lunajson",1.5]}'
    local lua_value = JSON:decode(raw_json_text) -- decode example
    local pretty_json_text = JSON:encode_pretty(lua_value) -- "pretty
printed" version

    print(lua_value)

      endsubmit;
run;
```

The output looks like:

```
table: 000000000A967800 {
  [Hello] => table: 000000000A967800 {
               [1] => "lunajson"
               [2] => 1.5
             }
}
```

Note that:

1. I put the external Lua package JSON.lua in the "current working directory" of SAS, then use Lua statement **LOADFILE** to invoke it.
2. Due to limitation of Proc Lua, the external Lua package should be written by pure Lua.

D. MACHINE LEARNING

I have no intension to use pure Lua machine learning algorithms to beat the existing SAS Enterprise Miner. The example here is rather for educational purpose. The package is from:

http://lonelydeveloper.org/workspace/projects/misc/nbc.lua

This is one way to wrap it up in SAS:

```
proc lua infile="nbc" restart;
    submit;

    nbc = require('nbc')

    training_set = {
  { "sunny", "hot", "high", "weak", 0 },
  { "sunny", "hot", "high", "strong", 0 },
  { "cloudy", "hot", "high", "weak", 1 },
  { "rainy", "temperate", "high", "weak", 1 },
  { "rainy", "cold", "normal", "weak", 1 },
  { "rainy", "cold", "normal", "strong", 0 },
  { "cloudy", "cold", "normal", "strong", 1 },
  { "sunny", "temperate", "high", "weak", 0 },
  { "sunny", "cold", "normal", "weak", 1 },
  { "rainy", "temperate", "normal", "weak", 1 },
  { "sunny", "temperate", "normal", "strong", 1 },
```

```
  { "cloudy", "temperate", "high", "strong", 1 },
  { "cloudy", "hot", "normal", "weak", 1 },
  { "rainy", "temperate", "high", "strong", 0 }
    }

     domains = {
       { "sunny", "cloudy", "rainy" },
       { "hot", "temperate", "cold" },
       { "high", "normal" },
       { "weak", "strong" },
       { 0, 1 }
     }

   print( nbc.NaiveBayesClassifier(training_set, domains, {"sunny",
"hot", "high", "weak"}) )

   print('--')

   print( nbc.NaiveBayesClassifier(training_set, domains, {"rainy",
"temperate", "high", "weak"}) )

   endsubmit;
run;
```

The output:

```
NOTE: Lua initialized.
0    0.02152423469387
1    0.00903925619834

--
0    0.0161431760204
1    0.02008723599632
```

.

CONCLUSION

SAS is not “A” programming language. Instead, it’s a collection of multiple programming languages, which includes Data Step, DS2, Macro, SQL, IML, Groovy and such. The beauty is that such programming techniques can be interacted by each other. I’m glad Proc Lua joins SAS programming family; it offers a totally different programming style and flavor like a data structure, for loop, external packages support and other modern scripting language features. I believe taking even a little bite of Lua can make a more vesetile SAS programmer.

REFERENCES

[1] http://regex.info/blog/lua/json

[2] http://lonelydeveloper.org/workspace/projects/misc/nbc.lua

[3] http://support.sas.com/kb/57/993.html

[4] http://blogs.sas.com/content/sasdummy/2015/08/03/using-lua-within-your-sas-programs/

[5] http://support.sas.com/resources/papers/proceedings15/SAS1561-2015.pdf

CONTACT INFORMATION

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

Jiangtang('JT') Hu
Life Sciences Consultant
d-Wise Technologies, Inc.
1500 Perimeter Park Dr., Suite 150, Morrisville, NC, 27560
www.d-Wise.com

(O) 919-334-6096
(C) 919-801-9659
(F): 888-563-0931
(O) Jiangtang.Hu@d-Wise.com
(P) jiangtanghu@gmail.com
http://jiangtanghu.com/