Paper SAS2095-2018

How to Build a Recommendation Engine Using SAS® Viya®

Jared Dean, SAS Institute Inc., Cary, NC

ABSTRACT

Helping users find items of interest is useful and positive in nearly all situations. It increases employee productivity, product sales, customer loyalty, and so on. This capability is available and easy to use for SAS® Viya® customers. This paper describes each step of the process: 1) loading data into SAS Viya; 2) building a collaborative filtering recommendation model using factorization machines; 3) deploying the model for production use; and 4) integrating the model so that users can get on-demand results through a REST web service call. These steps are illustrated using the SAS Research and Development Library as an example. The library recommends titles to patrons using implicit feedback from their check-out history.

INTRODUCTION

Factorization machines are a common technique for creating user item recommendations, there is evidence they generate double digit increases in engagement and sales. SAS has had recommendation methods for many years including market basket analysis, K-nearest neighbors (KNN), and link analysis, along with other techniques for creating a next best offer. This paper focuses on creating recommendations using factorization machines and SAS® Viya® 3.3. The outcome of the paper is a recommendation engine that can be called from a RESTful API that returns the top five recommended books to library patrons. This process requires three main tasks be completed. (See Figure 1. Workflow for Recommendation Engine.) The tasks can be completed in any order, but all three must be completed before the service can be called through a RESTful service call.

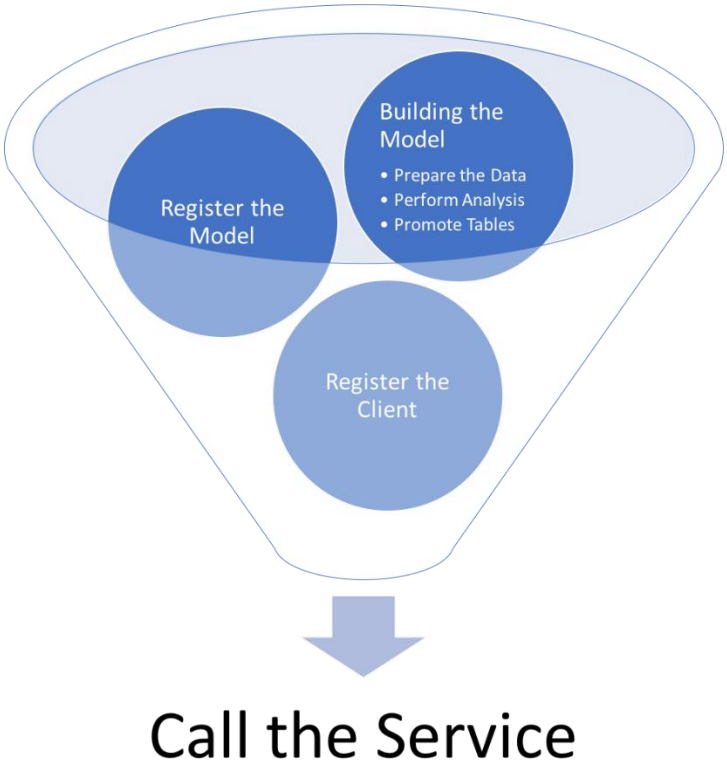

Figure 1. Workflow for Recommendation Engine

- Build the model. This is the analytical modeling section, which includes preparing the data, performing the factorization machine analysis, and creating the artifacts needed for providing recommendation requests on demand.
- Register the model. This is built in section two so that it is available to requestors on demand through a RESTful interface.
- Register the client within SAS Viya. This task is typically performed by a SAS administrator for the system, and the information is provided to the application developers.

The final section describes how the service can be called through a simple URL. This URL can then be embedded in an application, allowing SAS® Analytics to be part of your application in a simple and consistent manner.

GETTING STARTED

This paper uses SAS Viya 3.3, released in December 2017, to create a recommendation engine for the SAS R&D library. This application is meant to demonstrate the utility and ease of creating recommendations for your internal and external audiences using SAS Analytics. The technique used is a factorization machine. This example assumes that the FACTMAC is licensed.

Through the multiple language clients available for SAS Viya, several parts of this project can be accomplished using one of many programming languages. Examples are provided for you in SAS and Python. The Python code uses the SWAT package, which is available on GitHub at https://github.com/sassoftware/python-swat.

Three columns are required in the simplest application of a recommendation engine: User, Item, and Rating. More columns (attributes) can be used in creating recommendations, which is often referred to as tensor factorization. This factorization can add accuracy to your recommendations. All the columns used in factorization machine analysis must have values, and all the columns (except for the ratings) are treated as nominal variables.

The patron check-out data has various fields, but the fields that map to our application are the name of the patron (user) and the title (item) of the media the patron checked out. For an example, see Table 1. Example of Check-out History.

Table 1. Example of Check-out History

Name	Title
Dean Jared	Steve Jobs
Dean Jared	How Google works
Dean Jared	R for everyone advanced analytics and graphics
Dean Jared	Beautiful data the stories behind elegant data solutions
Dean Jared	Adapt why success always starts with failure
Dean Jared	Connectography mapping the future of global civilization
Dean Jared	Python in a nutshell
Dean Jared	Programming Python
Dean Jared	Practical statistics for data scientists 50 essential concepts

In the beginning of this project, I worked with a static copy of the data for development and validation. In production, the static copy of the data was replaced by a RESTful API call to get the latest library circulation data upon request.

Recommendations typically use a train/score model pattern. Here is the basic pattern: A model is trained on the most recent data available. After training is completed, the scoring tables are replaced with

updated versions. If the frequency for providing recommendations is very high (lots of users or users requesting recommendations often), you could have a continuous train/score cycle where as soon as the training ends it immediately gathers the latest data and begins the process again. For lower demand recommendation engines, you can schedule the training on a regular interval (hourly, daily, and so on). The time to train the model depends on the number of distinct user and item combinations (plus any additional attributes you include) and the number of transactions involved in the training. Factorization machines in SAS Viya can take advantage of parallel computing so that the elapsed time can be greatly reduced by using multiple CPUs.

BUILDING THE MODEL (TRAINING)

SETUP

The first step is to establish a connection to a CAS server. SAS® Cloud Analytic Services, the CAS server, is the next step for SAS in the evolution of SAS Analytics high-performance distributed processing on single or multiple machines.

Here is example SAS code:

```
options cashost="myserver.sas.com" casport=31004 casuser='Jared';
cas mysession;
```

Here is example Python code:

```
import swat
conn = swat.CAS('myserver.sas.com', 31004)
# Load the needed action sets
actionsets = ['astore', 'factmac', 'dataStep', 'fedSql']
[conn.builtins.loadactionset(i) for i in actionsets]
```

Notice the Python code has a few extra lines because the action sets must be loaded explicitly.

CREATE RATINGS

In a traditional recommendation setting, the items have ratings given by users (explicit feedback). In this example, ratings are not available, so a model is built using implicit feedback. For more information about creating implicit feedback, see the References and Recommended Reading sections.

To create quality recommendations without ratings, implicit feedback is used. Implicit feedback supplements our check-out history by randomly adding a book the user has not checked out for each book the user has checked out. This supplement creates a ratings data set that is twice the size of the actual check-out history. All of the books actually checked out by patrons receive a rating of 1, and all of the randomly selected books receive a rating of 0.

Here is a SAS macro, rate0, to generate implicit feedback:

```
%macro rate0(user);
    proc sql;
      create table user as
      select distinct(title), (1) format=1. as rating,
                  (&user.) as user
      from d.bhist
```

```
  where name = "&user.";
quit;
```

The preceding SQL procedure creates a distinct list of books for a specific user:

```
%let DSID = %sysfunc(open(user, IS));
%let n = %sysfunc(attrn(&DSID, NLOBS));
%let DSID=%sysfunc(close(&DSID));
proc sql outobs=&n.;
    create table rate0 as
        select title, (0) format=1. as rating, (&user.) as user
        from item
        except all
        select *
        from user
        order by ranuni(-1);
quit;
```

The preceding SQL procedure merges the user's books with all the books in the library, keeping only a random selection of the books the specific user did not check out and equal to the number they did check out.

```
proc append base=rate0_base data=rate0; run;
%mend rate0;
```

The remainder of the code partitions the check-out history and runs the rate0 macro for each user until there is a data set with all the actual check-out items that have a rating of 1 and all the randomly selected items that have a rating of 0. The data set has exactly twice as many records as the check-out history.

```
data item;
    set d.bhist;
    by title;
    if first.title;
    keep title;
run;
proc fedsql;
    create table user_cnt as
    select distinct(name) as "user"
    from d.bhist
    group by name
    order by name;
quit;
filename file1 temp;
data _null_;
    set user_cnt;
    file file1;
    put '%rate0(' name ');';
run;
proc delete data=rate0_base; run;
%include file1;
```

This macro takes a data set of the check-out history and returns a data set with the implicit feedback performed.

Here is a Python function to generate implicit feedback:

```
def sampleTitles(transhist: 'pd.DataFrame' = None,
                 user = 'name',
                 item = 'title') -> 'pd.DataFrame':
    nonco = pd.DataFrame()
    users = transhist[user].unique()
    for i in users:
        # get list of titles checked out
        titles = transhist.loc[transhist[user] == i]

        # get list of non-titles checked out
        nct = transhist.loc[~transhist[item].isin(titles[item].unique())]

        # randomly select non-checked out titles equal to the number of
checkouts.
        samp = nct.sample(n=titles[item].count())[['bib', 'processed']]
        samp['rating'] = 0
        samp[user] = i

        nonco = nonco.append(samp)
    return nonco
```

The function takes a pandas dataframe, user, and item. The dataframe is of the borrowing history. User and name represent the columns in the dataframe that correspond to user and item. For this example, the patron is the user, and the book is the item.

Regardless of the programming language (SAS, Python, R, and so on), here is the procedure for generating implicit feedback:

1. Create a unique list of all the patrons.
2. Create a unique list of the books each patron has checked out and the total number of checkouts. A random book is selected each time a book is checked out.
3. Create a list of all titles offered by the library. If there are multiple copies or media (audiobook, e-book, hardback, and so on), they are treated as a single title.
4. Sample without replacement from the universe of titles that the user has not checked out. This is represented by the blue area in **Error! Reference source not found.**.

   The circle represents all the titles. The white area is books the patron has checked out. The shaded area is books that have not been checked out by the patron.

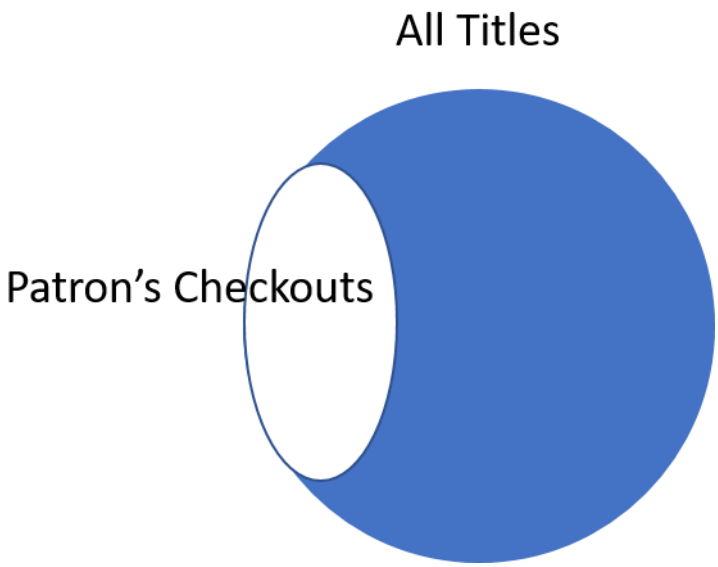

Figure 2. Illustration of Sampling Design

5. Add the sampled books to the check-out history.

This procedure is then repeated for each patron in the library. For more information about implicit feedback, see the References and Recommended Reading sections.

With the implicit feedback completed, a sample of our data now looks like Table 2. Example Data after Implicit Feedback. The books with rating 1 are books I have checked out from the SAS library. The books with rating 0, I have not checked out. The complete table would include the check-out history for each library patron. The books checked out by each patron have a rating of 1, and all the randomly selected books that were not checked out have a rating of 0.

Table 2. Example Data after Implicit Feedback

Name	Title	Rating
Dean Jared	Steve Jobs	1
Dean Jared	How Google works	1
Dean Jared	R for everyone advanced analytics and graphics	1
Dean Jared	Beautiful data the stories behind elegant data solutions	1
Dean Jared	Connectography mapping the future of global civilization	1
Dean Jared	Adapt why success always starts with failure	1
Dean Jared	Python in a nutshell	1
Dean Jared	Programming Python	1
Dean Jared	Practical statistics for data scientists 50 essential concepts	1
Dean Jared	HTML5 up and running	0
Dean Jared	Wordpress for dummies	0
Dean Jared	PHP and MySQL by example	0
Dean Jared	Adobe Photoshop CS5 classroom in a book	0

Dean Jared	Exploratory factor analysis	0
Dean Jared	Spatial statistics	0
Dean Jared	SAS certification prep guide base programming for SAS 9	0
Dean Jared	Beginning Lua programming	0
Dean Jared	Head First Excel	0

CREATE RECOMMENDATIONS

Now that we have a variety of ratings in the data, the data can be loaded into CAS and a factorization machine analysis performed.

Here is the SAS code to load the data and run the FACTMAC procedure:

```
libname mycas cas;
data mycas.checkout;
    set final_rating;
run;

proc factmac data=mycas.checkout outmodel=mycas.factors_out;
   autotune;
   input Name Title /level=nominal;
   target rating /level=interval;
   savestate;
   output out=mycas.score_out1 copyvars=(rating);
run;
```

Here is the Python code to load the data and run the FACTMAC action:

```
conn.upload(casout={'name':'checkout', 'replace':True},
data=final_rating.dropna())
rec1 = conn.factmac(table='checkout',
                    inputs   = ['Name', 'Title'],
                    nominals = ['Name', 'Title'],
                    id       = ['Name', 'Title'],
                    target   = 'rating',
                    nfactors = 10,
                    maxiter  = 100,
                    learnstep= 0.15,
                    seed=9878,
                    output=  {'casout':{'name':'score_out1',
                                        'replace':'TRUE'},
                              'copyvars':['rating']},
                    outModel={'name':'factors_out', 'replace':'TRUE'},
                    saveState={'name':'state'},
)
```

Regardless of which interface we use to run the analysis, there are several details that need to be specified.

The INPUTS, ID, and TARGET statements must be specified. The number of factors (nfactors), maximum iterations (maxiter), and the learning rate (learnstep) variables have defaults but can be specified by the user or optimal settings can be found using autotuning. I have explicitly listed the options here for clarity. The quality of a factorization machine is based on the root mean squared error (RMSE). For more information about the FACTMAC syntax, see the Recommended Reading section.

To facilitate making recommendations (scoring) on demand for users, we need to save the model in an ASTORE object. An ASTORE is a compressed binary representation of the model. Saving the model is accomplished in the OUTMODEL statement. For more information about ASTORE, see the Recommended Reading section.

PROMOTE TABLES

By default, CAS tables are available only in the session that created them. To make them available globally for requests on demand, we must promote the tables.

Three tables must be promoted for this application:

1. the ASTORE from the SAVESTATE statement. This is used for recommending books to returning patrons.
2. the factors table from the OUTMODEL statement. This is used for recommending books to new patrons.
3. the borrower history from the DATA statement. This is used for creating a list of distinct books at the time a recommendation is requested.

Here is the SAS code to promote the needed tables:

```
proc casutil;
   promote casdata="checkout" casout='libraryrec_latest';
   promote casdata="state" casout='libAstore_latest';
   promote casdata="factors_out" casout='factors_latest';
quit;
```

Here is the Python code to promote the needed tables:

```
conn.droptable(name='libraryrec_latest', quiet=True)
conn.droptable(name='libAstore_latest', quiet=True)
conn.droptable(name='factors_latest', quiet=True)
conn.promote(name='checkout', target='libraryrec_latest')
conn.promote(name='state', target='libAstore_latest')
conn.promote(name='factors_out', target='factors_latest')
```

With these tables promoted, the training portion is complete. The next sections demonstrate how to register the model as a service in SAS Viya, how to register the client so that it can be called as a service, and how to make a RESTful API call to provide recommendations on demand.

REGISTERING THE MODEL

The SAS Viya infrastructure has many micro services. For this application, I used the SAS Job Execution service because I found it the simplest to work with. There is a user interface specifically designed to help you register SAS jobs, which is experimental in SAS Viya 3.3 (released in December 2017).

Your SAS administrator should provide you with a URL. For this paper, assume it is http://myviya.sas.com.

When you open that link in your browser, you are prompted to sign in.

Figure 3. Sign-in Screen

After a successful sign-in, you will likely be redirected to http://myviya.sas.com/SASHome and your dashboard will look like Figure 4. SASHome Dashboard.

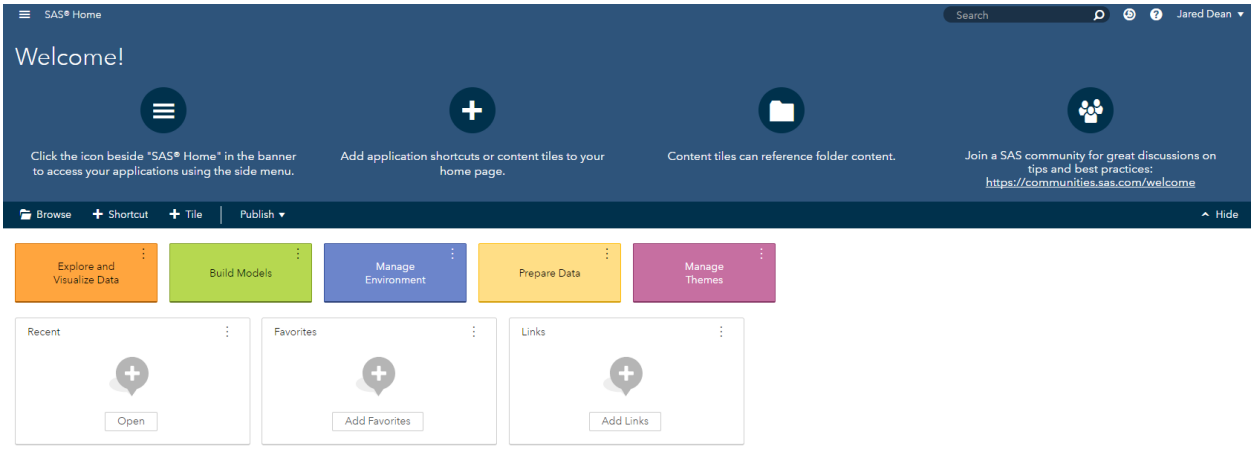

Figure 4. SASHome Dashboard

Next, navigate to http://myviya.sas.com/SASJobExecution/admin.

Some important items to note:

- The first part of the URL will be different for your organization.
- The URL is case sensitive.
- You must have administrative rights in SAS Viya to register a job with the SAS Job Execution service.

Your browser should now display the SAS Job Execution client. (See Figure 5. SAS Job Execution Client.)

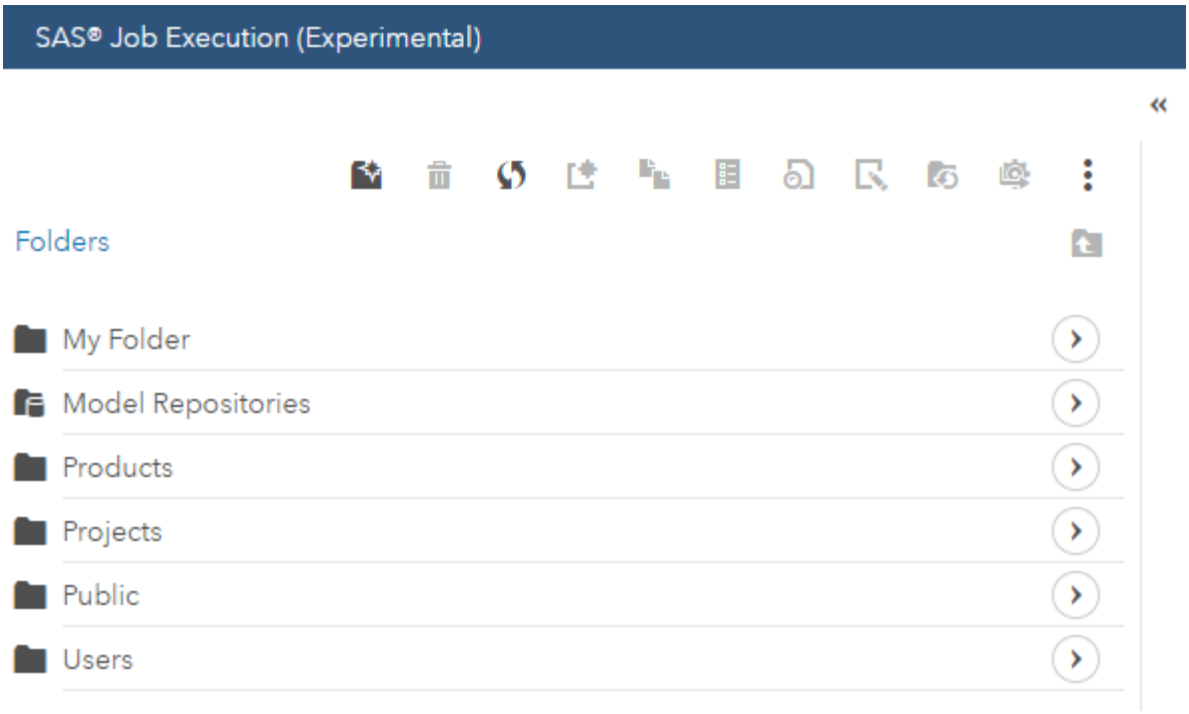

Figure 5. SAS Job Execution Client

This step (Job Execution service) does not currently support code from other languages such as R or Python. Therefore, you need to write exclusively SAS code.

You need to decide where to store your SAS code. From the toolbar at the top of the screen, you can navigate the folders and create new folders and programs.

Here is the SAS code that runs each time the RESTful API is called. Each code block is explained following the code. I have left commented parts of the code so that if your application has different requirements, you can use it as a template.

```
/* Close all ods destinations */
ods _all_ close;
/* for debug - print all the macro variables */
*%put _global_;
/* To create HTML output */
*filename _webout sasfsvam  parenturi="&SYS_JES_JOB_URI"  name='_webout.htm';
*ods html5 file=_webout style=HTMLBlue;

filename _webout sasfsvam parenturi="&SYS_JES_JOB_URI" name='_webout.json';
```

The preceding code closes all the output destinations and establishes a filename that will be in a JSON file to return to the requestor.

```
options cashost="ip.or.url.com" casport=<<port>> ;
/* establish a CAS session */
cas mysession;
```

```
/* create a libname to your CAS session */
libname mycas cas;
```

The preceding block of code creates the CAS session and creates a library reference between the SAS session and the CAS server. You need to specify these items:

- cashost (using IP address, DNS name, or localhost)
- casport (provided by your administrator)

Note: If you are authenticating using OAuth do not specify the casuser in the options this will over ride the OAuth authentication.

Next, we use several statements within the CAS procedure to prepare the data, create ratings, and determine which books to recommend.

These are the action sets that are needed:

```
proc cas;
          loadactionset "dataStep";
          loadactionset "fedSql";
          loadactionset "astore";
      run;
```

The ASTORE object we produced earlier in the Promote Tables section of the paper takes a table with patrons and titles and returns a predicted rating. In this DATA step code, we need to prepare a data set for scoring. We create two columns—one of the user, and one for each title in the library collection. The variable bib is an identifier for the title of the book.

```
      /* Drop and rename */
      dataStep.runCode code = "
          data user_rec;
              set libraryrec_latest;
              by bib;
              if first.bib;
              empno= lowcase(""&score_user"");
              keep empno bib processed;
          run;";
      run;
```

In the SCORE statement, we pass the CAS table we just created and create an output table named ranked_books.

```
      /* Score with Astore */
          astore.score /
              table = 'user_rec'
              rstore='libAstore_latest'
              out = {name='ranked_books' replace=True}
              ;
      run;
```

In the following SUMMARY action, we find the max rating. The FACTMAC ASTORE returns a rating of missing for any row where the patron or book is missing. A missing value is less than any other number in SAS, so if the max is missing that means all the values are missing.

```
      /* Find max rating. If all ratings are missing then new user. */
      simple.summary result=m /
            table='ranked_books'
            subset={'max'}
            inputs={'P_rating'};

      /* drop the table in preparation to replace it */
      table.droptable /
            name='book_recs'
            quiet=True;
```

In the following block of code, we check the max value from the ranked_books table. If the max value is missing, it means this is a new user. We do not have any history with new users, so we will recommend the most popular books in the library. In both cases, we create a table named book_recs with the top five recommendations.

```
      if missing(m.summary[1,2]) then do;
            fedsql.execdirect result=top5rec /
              query="create table book_recs as
              select a.Level as bib, b.processed
              from factors_latest a, user_rec b
              where Variable='bib' and a.level=b.bib
              order by Bias desc limit 5;";
      end;
      else do;
          fedsql.execdirect result=top5rec /
              query="create table book_recs as
              select bib, processed
              from ranked_books
              where P_rating^=.
              order by P_rating
              desc limit 5;";
      end;
      run;
quit;
```

In the following block of code, the recommendation table is written as JSON output, which is returned to the application that called the RESTful API. JSON is the standard return format for REST API calls.

```
proc json out=_webout;
      export mycas.book_recs(keep=title);
run;

/* code to use for HTML results or debug */
/*
proc print data=mycas.ranked_books(obs=5);
run;
proc print data=mycas.book_recs;
run;
ods html5 close;
*/
```

REGISTERING THE CLIENT

Before we can call our recommendation scoring service, we must register the client with SAS Logon. This task is typically performed by the SAS administrator, not the application developer, but it must be completed before anything will work. For more information, see “Obtain an ID Token to Register a New Client ID” in the References section. Registering the client is needed to ensure that the application is authorized. The process involves generating a token as an authorized user, and then using that token to authorize this application.

The referenced documentation goes into more detail, but here is the high-level process:

1. Get a consul token from the system files:
    a. cd /opt/sas/viya/config/etc/SASSecurityCertificateFramework/tokens/consul/default
    b. sudo export CONSUL_TOKEN=`cat client.token`
2. Get a client access token to register the client:
    a. curl -X POST "http://localhost/SASLogon/oauth/clients/consul?callback=false&serviceId=horizonapp" -H "X-Consul-Token: <*****-****-****-*****>"
3. Save the client access token for future commands:
    a. export TOKEN=eyJhbGc...PDKgg
4. Register the new client. Give it a name and assign it a secret password:
    a. curl -X POST "http://localhost/SASLogon/oauth/clients" -H "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -d '{"client_id": "mysuperapp", "client_secret": "<SECRET_PASSWORD>", "scope": ["openid", "openstackusers"],"authorized_grant_types": ["client_credentials"]}'

Note: In step 4, use “client_credentials” as the authorized grant type instead of a password for improved security.

CALLING THE SERVICE (SCORING)

Because of the work we did to register the model, calling the service is very simple. An authorized user can make a simple REST call to the SASJobExecution endpoint. There are two ways to call programs that are registered for the SAS Job Execution service. You can reference the program by the job definition ID as shown here:

http://myviya.sas.com/SASJobExecution/?_job=/jobDefinitions/definitions/1404f786-2358-48bb-a41f-f82b2a6a0791&score_user=’John Doe’

Or, you can use the path to the program as shown here:

http://myviya.sas.com/SASJobExecution/?_program=/Public/libraryRecScore &score_user=’Jane Doe’

Both calls yield the same results. It is personal preference which one you would like to call. After completing the steps in this paper, you should be able to paste a URL similar to either preceding call and get JSON results displayed in your browser.

When either call is made, JSON is returned. The JSON response is not intended to be read by users, but it will be processed. Here is an example of the code that is returned:

```
{"SASJSONExport":"1.0","SASTableData+BOOK_RECS":[{"BIB":"77949","processed":"
High performance habits how extraordinary people become that
way"},{"BIB":"7860","processed":"Proceedings of the fifth annual SAS Users
Group International SUGI Conference San Antonio Texas February 18 20
```

```
1980"},{"BIB":"8159","processed":"SEUGI 93 proceedings of the eleventh SAS
European Users Group International Conference Jersey U K June 22 25
1993"},{"BIB":"7873","processed":"Proceedings of the tenth annual SAS users
group international conference SUGI 10"},{"BIB":"9399","processed":"Step by
step programming with base SAS software"}]}
```


As a SAS programmer, you might not have any experience calling a RESTful service and using the JSON response, but the web developers in your organization use these tools all the time. You can now quickly and efficiently provide easy access to SAS Analytics in the applications that your organization is building.

CONCLUSION

Factorization machines are a modern recommendation technique using SAS Viya 3.3 that you can easily incorporate in your applications to give users suggestions and guidance. To create a recommendation engine takes four steps: training a model, registering the model, registering the client, and calling the RESTful service.

To train the model, you gather the data, create ratings if they do not already exist, perform a factorization machine analysis, and finally save the results to create on-demand recommendations.

Registering the model must be written in SAS code, and the user must have administrator rights in SAS Viya. This is the code that runs each time the API is called.

Registering the client is usually done by a SAS administrator.

Calling the RESTful service makes it simple to embed SAS Analytics in your application with JSON results being returned.

By following these steps, you can unleash the power of SAS in your applications in a straightforward way.

REFERENCES

Henry, Joseph. “Show Off Your OAuth.” *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings17/SAS0224-2017.pdf.

SAS Documentation 2017. “Obtain an ID Token to Register a New Client ID.” *Encryption in SAS Viya 3.2: Data in Motion*. Accessed December 4, 2017. http://go.documentation.sas.com/?cdcId=calcdc&cdcVersion=3.2&docsetId=secref&docsetTarget=n1xdqv1sezyrahn17erzcunxwix9.htm&locale=en#p1w1tzdisw4147n1hlc8cwhi9fwg

SAS Documentation 2017. “Obtain an Access Token Using Password Credentials.” *Encryption in SAS Viya 3.2: Data in Motion*. Accessed December 21, 2017. http://go.documentation.sas.com/?cdcId=calcdc&cdcVersion=3.3&docsetId=calauthmdl&docsetTarget=n1pkgyrtk8bp4zn1d0v1ln4869og.htm&locale=en#p0lxoq5bx2i6t8n13b3y3tcjwj9v

Hu, Y., Y. Koren, and C. Volinsky. 2008. “Collaborative Filtering for Implicit Feedback Datasets.” *Proceedings of the Eighth IEEE International Conference on Data Mining*. Pisa, Italy. pp. 263-272. doi: 10.1109/ICDM.2008.22

Grau, J., Personalized product recommendations: predicting shoppers’ needs. eMarketer, March 2009

Rendle, Steffen. 2010 "Factorization machines." *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE.

SAS Documentation 2017. "SAS Cloud Analytic Services (CAS)." *Differences in the SAS 9 and SAS Viya 3.2 Platforms*. Accessed January 1, 2018. http://go.documentation.sas.com/?docsetId=whatsdiff&docsetTarget=p1gfaswb875orbn1xn4ao8b8jbfq.htm&docsetVersion=3.2&locale=en

RECOMMENDED READING

- SAS Institute Inc. 2016. "The ASTORE Procedure." *SAS Visual Data Mining and Machine Learning 8.1: Data Mining and Machine Learning Procedures*. Cary, NC: SAS Institute Inc. Available http://go.documentation.sas.com/?docsetId=casml&docsetTarget=viyaml_astore_toc.htm&docsetVersion=8.1&locale=en
- SAS Institute Inc. 2016. "The FACTMAC Procedure." *SAS Visual Data Mining and Machine Learning 8.1: Data Mining and Machine Learning Procedures*. Cary, NC: SAS Institute Inc. Available http://go.documentation.sas.com/?docsetId=casml&docsetTarget=viyaml_factmac_toc.htm&docsetVersion=8.1&locale=en
- Hu, Y., Y. Koren, and C. Volinsky. "Collaborative Filtering for Implicit Feedback Datasets." 2008. *Proceedings of the Eighth IEEE International Conference on Data Mining*. Pisa, Italy. 2008. pp. 263-272. doi: 10.1109/ICDM.2008.22
- Silva, Jorge and Wright, Raymond E. "Factorization Machines: A New Tool for Sparse Data" *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings17/SAS0388-2017.pdf

ACKNOWLEDGMENTS

The author would like to thank Jorge Silva, Joseph Henry, Sath Sourisak, Mike Roda, Vince DelGobbo, Matt Bailey, and Brett Wujek for their contributions to the paper. He is also grateful to Tate Renner for her editorial contributions.

CONTACT INFORMATION

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

Jared Dean
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Jared.Dean@sas.com
http://www.sas.com