Demo Steps: Modeling Sparse Data Using the Python API

The data set **movie_ratings** contains information about ratings given by users to movies. Both users and movies are identified by an anonymous numeric ID. A rating is a numeric value between 1 (very bad) and 5 (very good). The data source contains 10,000,000 rows and three columns. There are 10,000 users and 10,000 items, so 10,000,000 reviews are available from a possible 100,000,000 rows. The original matrix is very sparse, with 90% missing values.

Name	Model Role	Measurement Level	Description
USER_ID	Input	Nominal	ID of movie watcher
MOVIE_ID	Input	Nominal	ID of movie
RATING	Target	Nominal	Rating given by **user_id** for **movie_id**

1. From Jupyter Lab, select **File Browser** > **Home** > **Courses** > **EVMLOPRC** > **Notebooks** and select the **Python_Factorization_Machine_Demo.ipynb** notebook.

2. Load the os, sys, SWAT, numpy, pandas, and matplotlib packages.
```
import os
import sys
import swat
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
swat.options.cas.print_messages = True
```
3. Connect to CAS.
```
conn = swat.CAS(os.environ.get("CASHOST"), os.environ.get("CASPORT"), None, os.environ.get("SAS_VIYA_TOKEN"))
```
4. Load the **movie_ratings.sas7bdat** file on the server and name the data table **ratings**.
```
castbl =
conn.read_sas(os.environ.get("HOME")+"/Courses/EVMLOPRC/DATA/movie_ratings.sas7bdat",
              casout = dict(name="ratings", replace=True))

indata = "ratings"
```
5. Print the dimension of the table and the first few observations. Also, find the mean of the **rating** variable.
```
display(castbl.shape)
display(castbl.head())

mean_rating = castbl['rating'].mean()
print('Average overall rating = ' + str(round(mean_rating, 2)))
```
6. Use the distinct action from the simple action set to find the number of levels for all three data set variables.
```
conn.loadActionSet('simple')
actions = conn.builtins.help(actionSet='simple')

conn.simple.distinct(
    table = indata,
    inputs = list(castbl)
)
```
7. Use the freq action to find the number of ratings that are given for each level. Then create a data frame with the proportion of ratings at each level. Finally, plot the rating level by its proportion.
```
tbl = conn.simple.freq(
    table = indata,
    inputs = 'rating'
)['Frequency']

tbl['Proportion'] = tbl['Frequency']/tbl['Frequency'].sum(axis=0)
tbl

plt.figure(figsize=(8,8))
plt.plot(tbl['Level'], tbl['Proportion'], color='blue',
         linestyle='-', marker='o', markersize=12)
plt.title('Proportion of Ratings Per Level', fontsize=20)
plt.xlabel('Level', fontsize=15)
plt.ylabel('Proportion', fontsize=15)
plt.show()
```
Bias occurs because users unknowingly rate on different scales. For example, a four-star rating does not mean the same thing for two different users. Factorization machines explain these innate biases when they make predictions, and they can estimate the pairwise interactions between specific users and movies in sparse data.

The factorization machine accounts for the following biases:
   - a *global bias* (the average rating over all users and movies)
   - a *per-user bias* (the average of the ratings given by the user)
   - a *per-item bias* (the average of the ratings given to that movie)
   - a *pairwise interaction term* between the user and that particular movie

8. Use SQL to find each user's bias compared to the overall average rating.

```
conn.loadActionSet('fedSql')
actions = conn.builtins.help(actionSet='fedSql')

user_bias = conn.fedSql.execDirect(query =
    '''
    SELECT user_id,
        COUNT(rating) as Frequency,
        AVG(rating) AS AVG_Rating
    FROM ratings
    GROUP BY user_id
    ORDER BY user_id ASC;
    '''
)['Result Set']

user_bias['user_bias'] = user_bias['AVG_RATING']-mean_rating
user_bias.head()
```

9. Partition the data into 90% for training and 10% for validation.

```
conn.loadActionSet('sampling')
actions = conn.builtins.help(actionSet='sampling')

conn.sampling.srs(
    table = indata,
    samppct = 90,
    seed = 649,
    partind = True,
    output  = dict(casOut = dict(name = indata, replace = True),
                   copyVars = 'ALL')
)
```

10. Load the factmac action set and then use the factmac action to train the factorization machine.

```
conn.loadActionSet('factmac')
actions = conn.builtins.help(actionSet='factmac')

target = 'rating'
inputs = ['user_id','movie_id']

conn.factmac.factmac(
    table     = dict(name = indata, where = '_PartInd_ = 1'),
    target    = target,
    inputs    = inputs,
    nominals  = inputs,
    maxIter   = 5,
    nFactors  = 10,
    learnStep = 0.1,
    seed      = 919,
    saveState = dict(name = 'factmac_model', replace = True),
    output    = dict(casout = dict(name = "training_scored",
                      replace = True), copyvars = 'ALL')
)
```

Selected arguments:

Argument	Description
maxIter	specifies the maximum number of iterations. The default is 30.
nFactors	specifies the number of factors to be estimated.
learnStep	specifies the learning step size for the optimization.
saveStep	specifies the output data table in which to save the state of the factorization machine for future scoring as an analytic store.
output	specifies the output data table in which to save the scored observations.

```
cas.table.fetch(conn, table='training_scored', to=5
```

11. Using the saved analytics store from the factmac action, use the aStore action set and score action to score the validation data.

```
conn.loadActionSet('aStore')
actions = conn.builtins.help(actionSet='aStore')

conn.aStore.score(
    table    = dict(name = indata, where = '_PartInd_ = 0'),
    rstore = "factmac_model",
    copyVars = list(castbl),
    out = dict(name="factmac_scored", replace=True)
)
```

Selected arguments:

Argument	Description
rstore	specifies a binary table that contains the analytic store.

12. Use a DATA step to find the error in the predictions and then SQL to find the mean square error.

```
conn.loadActionSet('dataStep')
actions = conn.builtins.help(actionSet='dataStep')

conn.dataStep.runCode(code=
    '''
    data factmac_scored;
        set factmac_scored;
```

```
        error = rating - P_rating;
    run;
    '''
)

conn.fedSql.execDirect(query =
    '''
    SELECT
        AVG(error**2) AS MSE,
        SQRT(AVG(error**2)) AS RMSE
    FROM factmac_scored
    '''
)
```

13. Find the mean square error again, but this time download the scored information to the client and use open source code.

```
factmac_scored = conn.CASTable(name='factmac_scored')

factmac_scored['err'] = factmac_scored['rating'] -
                        factmac_scored['P_rating']
factmac_scored['err_sq'] = (factmac_scored['rating'] -
                            factmac_scored['P_rating'])**2

factmac_scored.head()

factmac_scored['err'].summary()

MSE = factmac_scored['err_sq'].mean()
print(" MSE = " + str(round(MSE,4)))
print("RMSE = " + str(round(MSE**.5,4)))
```

14. For each rating level, use SQL to find the average prediction rating.

```
rating_pred = conn.fedSql.execDirect(query =
    '''
    SELECT rating,
        count(*) AS frequency,
        AVG(P_rating) AS avg_prediction
    FROM factmac_scored
    GROUP BY rating;
    '''
)

rating_pred['Result Set'].sort_values('rating')
```

15. Use a DATA step to round the predictions to the nearest rating.

```
conn.dataStep.runCode(code=
    '''
    data factmac_scored;
        set factmac_scored;
        P_rating_round = round(P_rating,1);
        if P_rating_round = 0 then P_rating_round = 1;
    run;
    '''
)

conn.table.fetch(table='factmac_scored', to=5)
```

16. Use the crossTab action from the simple action set to find the actual versus predicted ratings from the validation data. Then use the crosstabulation matrix to find the proportion of correct predictions and the conditional probabilities of the predicted rating given the actual rating.

```
crosstab = conn.simple.crossTab(
    table='factmac_scored',
    row='rating',
    col='P_rating_round'
)['Crosstab']

crosstab.columns = ['rating','P1','P2','P3','P4','P5']
crosstab.index = range(1,6)
crosstab

crosstab = crosstab.drop('rating', axis=1)
pd.DataFrame(np.diagonal(crosstab)).sum(axis=0)
             /crosstab.values.sum()

crosstab.divide(crosstab.sum(axis=1), axis=0)
```

17. Create a function that uses the execDirect action from the fedSql action set. Let the parameter be an individual user identification number. For the given user, enable the function to return the top five recommended movies.

```
def useri_top5(which_user):
    tmp = conn.fedSql.execDirect(query =
        '''
        SELECT user_id, movie_id, P_rating
        FROM factmac_scored
        WHERE user_id= ''' + str(which_user) + '''
        ORDER BY P_rating DESC;
        '''
        )['Result Set']
    return tmp.head()

useri_top5(1)
```

18. End the CAS session.

```
conn.session.endSession()
```

Copyright © 2019 SAS Institute Inc., Cary, NC, USA. All rights reserved.