University of Virginia

DS 5559: Big Data Analytics

Music Recommendation

Last updated: Feb 29, 2020

Instructions

In this assignment, you will work with a recommendation algorithm based on user listening data from Autoscrobbler.

The code is outlined below. Make the requested modifications, run the code, and copy all answers to the **ANSWER SECTION** at the bottom of the notebook. Note the *None* variable is a placeholder for code.

NOTE: For a given userID, some/many recommendation might come back as $None$.
These should be filtered out using a list comprehension as follows:

In [1]:
```
#print([x for x in recommendationsForUser if x is not None])
```

TOTAL POINTS: 10

About the Alternating Least Squares Parameters

`rank`
The number of latent factors in the model, or equivalently, the number of columns k in the user-feature and product-feature matrices. In nontrivial cases, this is also their rank.

`iterations`
The number of iterations that the factorization runs. More iterations take more time but may produce a better factorization.

`lambda`
A standard overfitting parameter. Higher values resist overfitting, but values that are too high hurt the factorization’s accuracy.

In [2]:

```
# import modules
import os

from pyspark import SparkContext
from pyspark import SparkConf
from pyspark.mllib import recommendation
from pyspark.mllib.recommendation import *
import pandas as pd

from pyspark.mllib.recommendation import *
import random
from operator import *
```

In [3]:

```
from pyspark import SparkContext, SparkConf
spark = SparkContext.getOrCreate()
spark.stop()
spark = SparkContext('local','Recommender')
```

In [4]:

```
# set configurations
conf = SparkConf().setMaster("local").setAppName("autoscrobbler")
```

In [5]:

```
# set context
sc = SparkContext.getOrCreate(conf=conf)
```

In [6]:

```
# pathing and params
user_artist_data_file = 'user_artist_data.txt'
artist_data_file = 'artist_data.txt'
artist_alias_data_file  = 'artist_alias.txt'

numPartitions = 2
topk = 10
```

In [7]:
```
# read user_artist_data_file into RDD (417MB file, 24MM records of users' plays of artists, along with count)
# specifically, each row holds: userID, artistID, count
rawDataRDD = sc.textFile(user_artist_data_file, numPartitions)
rawDataRDD.cache()
```

Out[7]:
```
user_artist_data.txt MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java:0
```

In [8]:
```
# userid, artistid, playcount
rawDataRDD.take(5)
```

Out[8]:
```
['1000002 1 55',
 '1000002 1000006 33',
 '1000002 1000007 8',
 '1000002 1000009 144',
 '1000002 1000010 314']
```

In [9]:
```
# read artist_data_file using *textFile*
# Import test files from location into RDD variables
# YOUR CODE GOES HERE
#import os
#os.getcwd()
artistData = sc.textFile('artist_data.txt').map(lambda s:(int(s.split("\t")[0]),s.split("\t")[1]))
```

In [10]:
```
# inspect some records
#artistid, artist_name
artistData.take(5)
```

Out[10]:
```
[(1134999, '06Crazy Life'),
 (6821360, 'Pang Nakarin'),
 (10113088, 'Terfel, Bartoli- Mozart: Don'),
 (10151459, 'The Flaming Sidebur'),
 (6826647, 'Bodenstandig 3000')]
```

In [11]:
```
# read artist_alias_data_file using *textFile*
artist_alias= sc.textFile('artist_alias.txt')
```

In [12]:

```python
# inspect some records
# id, id
artist_alias.take(5)
```

Out[12]:

```
['1092764\t1000311',
 '1095122\t1000557',
 '6708070\t1007267',
 '10088054\t1042317',
 '1195917\t1042317']
```

In [13]:

```python
from pyspark.mllib.recommendation import *
import random
from operator import *
def parser(s, delimeters=" ", to_int=None):
    s = s.split(delimeters)
    if to_int:
        return tuple([int(s[i]) if i in to_int else s[i] for i in range(len(s))])
    return tuple(s)
artistData = sc.textFile("artist_data.txt").map(lambda x: parser(x,'\t',[0]))
artistAlias = sc.textFile("artist_alias.txt").map(lambda x: parser(x,'\t', [0,1]))
userArtistData = sc.textFile("user_artist_data.txt").map(lambda x: parser(x,' ',[0,1,2]))
```

In [14]:

```python
# 1) (1 PT) Print the first 10 records from rawDataRDD
rawDataRDD
rawDataRDD.top(topk)
```

Out[14]:

```
['9875 9973009 2',
 '9875 979 41',
 '9875 976 3',
 '9875 949 29',
 '9875 930 1',
 '9875 929 1',
 '9875 92 1',
 '9875 910 1',
 '9875 891 32',
 '9875 868 12']
```

In [15]:

```
def parseArtistIdNamePair(singlePair):
   splitPair = singlePair.rsplit('\t')
   # we should have two items in the list - id and name of the artist.
   if len(splitPair) != 2:
       #print singlePair
       return []
   else:
       try:
           return [(int(splitPair[0]), splitPair[1])]
       except:
           return []
```

In [16]:

```
rawArtistRDD = sc.textFile(artist_data_file)
```

In [17]:

```
artistByID = dict(rawArtistRDD.flatMap(lambda x: parseArtistIdNamePair(x)).collect())
```

In [22]:

```
# 2) (1 PT) Print 10 values from artistByID, using topk variable
from collections import Counter
import collections
topk = 10
# Hint: the most_common() function may help
```

In [23]:

```
c = Counter(artist_vals)
c.most_common(topk)
```

Out[23]:

```
[('06Crazy Life', 1),
 ('Pang Nakarin', 1),
 ('Terfel, Bartoli- Mozart: Don', 1),
 ('The Flaming Sidebur', 1),
 ('Bodenstandig 3000', 1),
 ('Jota Quest e Ivete Sangalo', 1),
 ('Toto_XX (1977', 1),
 ('U.S Bombs -', 1),
 ('artist formaly know as Mat', 1),
 ('Kassierer - Musik für beide Ohren', 1)]
```

In [24]:
```python
def parseArtistAlias(alias):
    splitPair = alias.rsplit('\t')
    # we should have two ids in the list.
    if len(splitPair) != 2:
        #print singlePair
        return []
    else:
        try:
            return [(int(splitPair[0]), int(splitPair[1]))]
        except:
            return []
```

In [25]:
```python
rawAliasRDD = sc.textFile(artist_alias_data_file)
```

In [26]:
```python
artistAlias = rawAliasRDD.flatMap(lambda x: parseArtistAlias(x)).collectAsMap()
```

In [27]:
```python
# Create a dictionary of artist id's
# artist
```

In [28]:
```python
# turn the artistAlias into a broadcast variable.
# This will distribute it to worker nodes efficiently, so we save bandwidth.
artistAliasBroadcast = sc.broadcast( artistAlias )
```

In [29]:
```python
artistAliasBroadcast.value.get(2097174)
```

Out[29]:
```
1007797
```

In [30]:
```python
# Print the number of records from the largest RDD, rawDataRDD
print( rawDataRDD.count() )
```

```
24296858
```

In [31]:
```python
# Sample 10% of rawDataRDD using seed 314, to reduce runtime. Call it sample.
seed = 314
weights = [.9, .1]
sample, _ = rawDataRDD.randomSplit(weights, seed)
sample.cache()
```

Out[31]:
```
PythonRDD[26] at RDD at PythonRDD.scala:53
```

In [32]:
```python
# take the first 5 records from the sample. each row represents userID, artistID, count.
sample.take(5)
```

Out[32]:
```
['1000002 1 55',
 '1000002 1000006 33',
 '1000002 1000007 8',
 '1000002 1000009 144',
 '1000002 1000010 314']
```

In [33]:
```python
artistByIDBroadCast = sc.broadcast(artistByID)
```

In [34]:
```python
# Based on sampled data, build the matrix for model training
def mapSingleObservation(x):
    # Returns Rating object represented as (user, product, rating) tuple.
    # [add line of code here to split each record into userID, artistID, count]
    userID, artistID, count = map(lambda lineItem: int(lineItem), x.split())
    # given possible aliasing, get finalArtistID
    finalArtistID = artistAliasBroadcast.value.get(artistID)
    if finalArtistID is None:
        finalArtistID = artistID
    return Rating(userID, finalArtistID, count)
```

In [35]:
```python
trainData = sample.map(lambda x: mapSingleObservation(x))
trainData.cache()
```

Out[35]:
```
PythonRDD[28] at RDD at PythonRDD.scala:53
```

In [36]:

```
# 3) (1 PT) Print the first 5 records from trainData
trainData.take(5)
```

Out[36]:

```
[Rating(user=1000002, product=1, rating=55.0),
 Rating(user=1000002, product=1000006, rating=33.0),
 Rating(user=1000002, product=1000007, rating=8.0),
 Rating(user=1000002, product=1000009, rating=144.0),
 Rating(user=1000002, product=1000010, rating=314.0)]
```

In [41]:

```
# Train the ALS model, using seed 314, rank 10, iterations 5, lambda_ 0.01. Call it model.
from pyspark.mllib.recommendation import *
model = ALS.trainImplicit(trainData, rank=10, iterations = 5, alpha = 0.01)
```

In [43]:

```
# Model Evaluation

# fetch artists for a test user
testUserID = 1000002

# broadcast artistByID for speed
artistByIDBroadcast = sc.broadcast( artistByID )

# from trainData, collect the artists for the test user. Call the object artistsForUser.
# hint: you will need to apply .value.get(x.product) to the broadcast artistByID, where x is the Rating RDD.
# if you don't do this, you may see artistIDs. you want artist names.
artistsForUser = (trainData
                  .filter(lambda observation: observation.user == testUserID)
                  .map(lambda observation: artistByIDBroadcast.value.get(observation.product))
                  .collect())
```

In [48]:

```
res = [i for i in artistsForUser if i]
print(res)
```

```
['Mallrats', 'Kerrang', 'Brian Hughes', 'Joshua Redman', 'The Mystick Krewe of Clearlight', 'Benny Goodman Orchestra',
'YMC', 'Brant Bjork and The Operators', 'Firebird', 'Elvis Costello', 'Café Del Mar', 'Eric Clapton', 'Enigma', 'Eurythm
ics', 'Armand Van Helden', 'Echo & the Bunnymen', 'George Duke']
```

In [49]:
```python
# 4) (1 PT) Print the artist listens for testUserID = 1000002
c = Counter(artist_vals)
c.most_common(topk)
```

Out[49]:
```
[('06Crazy Life', 1),
 ('Pang Nakarin', 1),
 ('Terfel, Bartoli- Mozart: Don', 1),
 ('The Flaming Sidebur', 1),
 ('Bodenstandig 3000', 1),
 ('Jota Quest e Ivete Sangalo', 1),
 ('Toto_XX (1977', 1),
 ('U.S Bombs -', 1),
 ('artist formaly know as Mat', 1),
 ('Kassierer - Musik für beide Ohren', 1)]
```

In [59]:
```python
# 5) (2 PTS) Make 10 recommendations for testUserID = 1000002
num_recomm = 500
recommendationsForUser_rank10 = map(lambda observation: artistByID.get(observation.product), model.call("recommendPro
print([x for x in recommendationsForUser_rank10 if x is not None])
```

```
['Eric Clapton', 'Elvis Costello', 'Eurythmics', 'Scorpions', 'Enigma', 'Gary Jules', '植松伸夫', 'Nena', 'Joss Stone']
```

In [56]:
```python
# Train a second ALS model with rank 20, iterations 5, lambda 0.01.
model_2 = ALS.trainImplicit(trainData, rank= 20, iterations = 5, alpha = 0.01)
```

In [60]:
```python
# 6) (2 PTS) Using the rank 20 model, make 10 recommendations for the same test user
num_recomm = 500
recommendationsForUser_rank20 = map(lambda observation: artistByID.get(observation.product), model_2.call("recommendP
print([x for x in recommendationsForUser_rank20 if x is not None])
```

```
['Eric Clapton', 'Eurythmics', 'Scorpions', 'Elvis Costello', 'Enigma', 'Gary Jules', 'Nena', 'Joss Stone']
```

ANSWER SECTION (COPY ALL ANSWERS HERE)

In [220]:

```
# ANSWER 1 (1 PT)
# Print the first 10 records from rawDataRDD
rawDataRDD.top(topk)
```

Out[220]:

```
['9875 9973009 2',
 '9875 979 41',
 '9875 976 3',
 '9875 949 29',
 '9875 930 1',
 '9875 929 1',
 '9875 92 1',
 '9875 910 1',
 '9875 891 32',
 '9875 868 12']
```

In [86]:

```
# ANSWER 2 (1 PT)
# Print topk values from artistByID
c = Counter(artist_vals)
c.most_common(topk)
```

Out[86]:

```
[('06Crazy Life', 1),
 ('Pang Nakarin', 1),
 ('Terfel, Bartoli- Mozart: Don', 1),
 ('The Flaming Sidebur', 1),
 ('Bodenstandig 3000', 1),
 ('Jota Quest e Ivete Sangalo', 1),
 ('Toto_XX (1977', 1),
 ('U.S Bombs -', 1),
 ('artist formaly know as Mat', 1),
 ('Kassierer - Musik für beide Ohren', 1)]
```

In [62]:

```
# ANSWER 3 (1 PT)
# Print the first 5 records from trainData
trainData.take(5)
```

Out[62]:

```
[Rating(user=1000002, product=1, rating=55.0),
 Rating(user=1000002, product=1000006, rating=33.0),
 Rating(user=1000002, product=1000007, rating=8.0),
 Rating(user=1000002, product=1000009, rating=144.0),
 Rating(user=1000002, product=1000010, rating=314.0)]
```

In [219]:
```
# ANSWER 4 (1 PT)
# Print the artist listens for testUserID = 1000002
res = [i for i in artistsForUser if i]
print(res)
```

['Mallrats', 'Kerrang', 'Brian Hughes', 'Joshua Redman', 'The Mystick Krewe of Clearlight', 'Benny Goodman Orchestra', 'YMC', 'Brant Bjork and The Operators', 'Firebird', 'Elvis Costello', 'Café Del Mar', 'Eric Clapton', 'Enigma', 'Eurythmics', 'Armand Van Helden', 'Echo & the Bunnymen', 'George Duke']

In [63]:
```
# ANSWER 5 (2 PTS)
# Make 10 recommendations for testUserID = 1000002
num_recomm = 500
recommendationsForUser_rank10 = map(lambda observation: artistByID.get(observation.product), model.call("recommendPro
print([x for x in recommendationsForUser_rank10 if x is not None])
```

['Eric Clapton', 'Elvis Costello', 'Eurythmics', 'Scorpions', 'Enigma', 'Gary Jules', '植松伸夫', 'Nena', 'Joss Stone']

In [64]:
```
# ANSWER 6 (2 PTS)
# Using the rank 20 model, make 10 recommendations for testUserID = 1000002
num_recomm = 500
recommendationsForUser_rank20 = map(lambda observation: artistByID.get(observation.product), model_2.call("recommendP
print([x for x in recommendationsForUser_rank20 if x is not None])
```

['Eric Clapton', 'Eurythmics', 'Scorpions', 'Elvis Costello', 'Enigma', 'Gary Jules', 'Nena', 'Joss Stone']

In [ ]:
```
# ANSWER 7 (2 PTS)
# How does the rank 10 model seem to perform versus the rank 20 model?
# The contents of artistsForUser may help answer the question.
```

In [70]:

```
list1 = ['Mallrats', 'Kerrang', 'Brian Hughes', 'Joshua Redman', 'The Mystick Krewe of Clearlight', 'Benny Goodman Or
list2_r10 = ['Eric Clapton', 'Elvis Costello', 'Eurythmics', 'Scorpions', 'Enigma', 'Gary Jules', '植松伸夫', 'Nena',
list3_r20 = ['Eric Clapton', 'Eurythmics', 'Scorpions', 'Elvis Costello', 'Enigma', 'Gary Jules', 'Nena', 'Joss Stone
comment_elements_r10 = []
comment_elements_r20 = []
set1=set(list1)
common_elements_r10= set1.intersection(list2_r10)
common_elements_r20 = set1.intersection(list3_r20)
```

In [71]:

```
print(common_elements_r10)
```

```
{'Eric Clapton', 'Enigma', 'Eurythmics', 'Elvis Costello'}
```

In [73]:

```
print(common_elements_r20)
```

```
{'Eric Clapton', 'Enigma', 'Eurythmics', 'Elvis Costello'}
```

```
The rank controls the number of internal parameters that must be fit from the data, too many and you get overgitting your trainning set.
Since we might not know the underlying factor. The more you use, the better the results up to a point, but the more memory and computation time you will need. I compare the rank 10 to artistForusers to see the common element. I also compare the rank 20 to artistForusers to see the common element. I found the common users are  {'Eric Clapton', 'Enigma', 'Eurythmics', 'Elvis Costello'}.  Again, since we might need to guess to see the underlying factors, the chosen rank higher should be better since the ranks refers to the presumed latent or hidden factors. However, we also need to avoid the overfitting issue.
```