autoscrobbler

March 7, 2021

0.0.1 Music Recommendation

0.0.2 University of California, Santa Barbara

0.0.3 PSTAT 135/235: Big Data Analytics

0.0.4 Last Updated: May 30, 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:

[ ]:

```python
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.

`alpha`
Controls the relative weight of observed versus unobserved user-product interactions in the factorization.

[3]:

```python
# 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
```

```
[4]: # set configurations
     conf = SparkConf().setMaster("local").setAppName("autoscrobbler")
```

```
[8]: # set context
     sc = SparkContext.getOrCreate(conf=conf)
```

```
[9]: # 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
```

```
[10]: # 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()
```

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

```
[11]: # inspect some records
      rawDataRDD.take(2)
```

```
[11]: ['1000002 1 55', '1000002 1000006 33']
```

```
[12]: # read artist_data_file using *textFile*
      rawArtistRDD = sc.textFile(artist_data_file, numPartitions)
      rawArtistRDD.cache()
```

```
[12]: artist_data.txt MapPartitionsRDD[4] at textFile at
      NativeMethodAccessorImpl.java:0
```

```
[13]: # inspect some records
      rawArtistRDD.take(5)
```

```
[13]: ['1134999\t06Crazy Life',
       '6821360\tPang Nakarin',
       '10113088\tTerfel, Bartoli- Mozart: Don',
       '10151459\tThe Flaming Sidebur',
 '6826647\tBodenstandig 3000']
```

[14]:

```
# read artist_alias_data_file using *textFile*
rawAliasRDD = sc.textFile(artist_alias_data_file, numPartitions)
rawAliasRDD.cache()
```

[14]:

```
artist_alias.txt MapPartitionsRDD[7] at textFile at
NativeMethodAccessorImpl.java:0
```

[15]:

```
# inspect some records
rawAliasRDD.take(5)
```

[15]:

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

[16]:

```
# 1) (1 PT) Print the first 10 records from rawDataRDD
rawDataRDD.take(10)
```

[16]:

```
['1000002 1 55',
 '1000002 1000006 33',
 '1000002 1000007 8',
 '1000002 1000009 144',
 '1000002 1000010 314',
 '1000002 1000013 8',
 '1000002 1000014 42',
 '1000002 1000017 69',
 '1000002 1000024 329',
 '1000002 1000025 1']
```

[17]:

```
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 []
```

[18]:

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

[19]:

```
artist_vals = artistByID.values()
```

[20]:
```
# 2) (1 PT) Print 10 values from artistByID, using topk variable
from collections import Counter
# Hint: the most_common() function may help

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

[20]:
```
[('"', 3),
 ('Rio Natsuki', 2),
 ('•', 2),
 ('Einojuhani Rautavaara', 2),
 ('Ò', 2),
 ('  ', 2),
 ('The Beatnuts', 2),
 ('Stone Temple Pilots', 2),
 ('{', 2),
 ('º', 2)]
```

[21]:
```
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 []
```

[22]:
```
artistAlias = rawAliasRDD.flatMap(lambda x: parseArtistAlias(x)).collectAsMap()
```

[23]:
```
# turn the artistAlias into a broadcast variable.
# This will distribute it to worker nodes efficiently, so we save bandwidth.
artistAliasBroadcast = sc.broadcast( artistAlias )
```

[24]:
```
artistAliasBroadcast.value.get(2097174)
```

[24]:
```
1007797
```

[25]:
```
# Print the number of records from the largest RDD, rawDataRDD
print( rawDataRDD.count() )
```

```
2269468
```

[26]:
```
# Sample 10% of rawDataRDD using seed 314, to reduce runtime. Call it sample.
seed = 314
weights = [0.1,0.9]
```

```
sample, _ = rawDataRDD.randomSplit(weights, seed)
sample.cache()
```

[26]:

```
PythonRDD[13] at RDD at PythonRDD.scala:53
```

[27]:

```
# take the first 5 records from the sample. each row represents userID,␣
 ↪artistID, count.
sample.take(5)
```

[27]:

```
['1000002 1000014 42',
 '1000002 1000088 157',
 '1000002 1000139 56',
 '1000002 1000140 95',
 '1000002 1000210 23']
```

[28]:

```
# 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)
```

[29]:

```
trainData = sample.map(lambda x: mapSingleObservation(x))
trainData.cache()
```

[29]:

```
PythonRDD[15] at RDD at PythonRDD.scala:53
```

[30]:

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

[30]:

```
[Rating(user=1000002, product=1000014, rating=42.0),
 Rating(user=1000002, product=1000088, rating=157.0),
 Rating(user=1000002, product=1000139, rating=56.0),
 Rating(user=1000002, product=1000140, rating=95.0),
 Rating(user=1000002, product=1000210, rating=23.0)]
```

[31]:

```
# Train the ALS model, using seed 314, rank 10, iterations 5, alpha 0.01. Call␣
 ↪it model.
model = ALS.trainImplicit(trainData, rank = 10, iterations = 5, alpha = 0.01,␣
 ↪seed = 314)
```

[32]:

```
# 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())
```

[33]:

```
# 4) (1 PT) Print the artist listens for testUserID = 1000002
print(artistsForUser)
```

```
['Pantera', 'Counting Crows', 'Muse', '(hed) Planet Earth', 'Eve 6', 'Meat
Loaf', 'Helmet', "Fun Lovin' Criminals", "Guns N' Roses", 'Pieces Of A Dream',
'Mike & the Mechanics', 'Clueless Soundtrack', 'Seal', 'Fugees', 'Chef', '60ft
Dolls', 'Kittie', 'Clam Abuse', 'Cliff Richard', 'Queen Adreena', 'Carl
Douglas', 'MC Hammer', 'Apollo 440', 'Powerman 5000', 'The Seahorses', 'Golden
Earring', 'Steve Miller', 'John Mayer', 'Frank Black', '4 Non Blondes', 'Frankie
Goes To Hollywood', 'Amy Grant', 'Living Colour', 'The Delfonics', 'T. Rex',
'Lao Tizer', 'Natalie Cole', 'Steve Cole', 'Take 6/Joe Sample', 'Braxton
Brothers', 'Warren Hill', 'Charlie Hunter', 'Steve Reid', 'Randy Scott', 'Patti
Austin', 'Tom Grant', 'Jeanie Bryson', 'Paul Hardcastle', 'Russ Freeman/Chris
Botti', 'Torcuato Mariano', 'k.d. lang', 'Ronan Hardiman', 'Special EFX',
'Mother Love Bone', 'Nelly Furtado', 'Sponge', 'Brandon Fields/Phil Perry',
'Pimps', 'Derek and the Dominos', 'Afro Blue', 'Congalegre', 'Haig and Haig',
'Travels', 'Ian Dury and The Blockheads', 'Jane Siberry', '[unknown]', 'Marty
Robbins', 'Romatt Project', 'Jeno Jando', 'Doves', 'Eva Cassidy', 'Faith No
More', 'Jimi Hendrix', 'Café Del Mar', 'The Electric Prunes', 'Duke Ellington
and Johnny Hodg', 'Eric Clapton', 'Tom Waits', 'Wayman Tisdale', 'Incognito',
'Simply Red', 'Pavement', 'Tindersticks', 'Butthole Surfers', 'New Order', 'Phil
Collins', 'David Bowie', 'Eurythmics', 'Dream Theater', 'Kid Rock', 'Rancid',
'Ella Fitzgerald', 'Mescalito', 'Eminem']
```

[34]:

```
# 5) (2 PTS) Make 10 recommendations for testUserID = 1000002
num_recomm = 10
recommendationsForUser = map(lambda observation: artistByID.get(observation.
 ↪product), model.call("recommendProducts", testUserID, num_recomm))
print(None)
```

```
None
```

```
[35]: # Train a second ALS model with rank 20, iterations 5, lambda 0.01.
      model2 = ALS.trainImplicit(trainData, rank = 20 ,iterations = 5, alpha = 0.01,␣
       ↪seed = 314)
```

```
[36]: # 6) (2 PTS) Using the rank 20 model, make 10 recommendations for the same test␣
       ↪user
      recommendationsForUser_rank20 = map(lambda observation: artistByID.
       ↪get(observation.product), model2.call("recommendProducts", testUserID,␣
       ↪num_recomm))
      print(None)
```

```
None
```

ANSWER SECTION (COPY ALL ANSWERS HERE)

```
[37]: # ANSWER 1 (1 PT)
      # Print the first 10 records from rawDataRDD
      rawDataRDD.take(10)
```

```
[37]: ['1000002 1 55',
       '1000002 1000006 33',
       '1000002 1000007 8',
       '1000002 1000009 144',
       '1000002 1000010 314',
       '1000002 1000013 8',
       '1000002 1000014 42',
       '1000002 1000017 69',
       '1000002 1000024 329',
       '1000002 1000025 1']
```

```
[38]: # ANSWER 2 (1 PT)
      # Print topk values from artistByID
      counted.most_common(topk)
```

```
[38]: [('"', 3),
       ('Rio Natsuki', 2),
       ('•', 2),
       ('Einojuhani Rautavaara', 2),
       ('Ò', 2),
       ('  ', 2),
       ('The Beatnuts', 2),
       ('Stone Temple Pilots', 2),
       ('{', 2),
       ('º', 2)]
```

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

```
[39]: [Rating(user=1000002, product=1000014, rating=42.0),
       Rating(user=1000002, product=1000088, rating=157.0),
       Rating(user=1000002, product=1000139, rating=56.0),
       Rating(user=1000002, product=1000140, rating=95.0),
       Rating(user=1000002, product=1000210, rating=23.0)]
```

```
[40]: # ANSWER 4 (1 PT)
      # Print the artist listens for testUserID = 1000002
      print(artistsForUser)
```

```
['Pantera', 'Counting Crows', 'Muse', '(hed) Planet Earth', 'Eve 6', 'Meat
Loaf', 'Helmet', "Fun Lovin' Criminals", "Guns N' Roses", 'Pieces Of A Dream',
'Mike & the Mechanics', 'Clueless Soundtrack', 'Seal', 'Fugees', 'Chef', '60ft
Dolls', 'Kittie', 'Clam Abuse', 'Cliff Richard', 'Queen Adreena', 'Carl
Douglas', 'MC Hammer', 'Apollo 440', 'Powerman 5000', 'The Seahorses', 'Golden
Earring', 'Steve Miller', 'John Mayer', 'Frank Black', '4 Non Blondes', 'Frankie
Goes To Hollywood', 'Amy Grant', 'Living Colour', 'The Delfonics', 'T. Rex',
'Lao Tizer', 'Natalie Cole', 'Steve Cole', 'Take 6/Joe Sample', 'Braxton
Brothers', 'Warren Hill', 'Charlie Hunter', 'Steve Reid', 'Randy Scott', 'Patti
Austin', 'Tom Grant', 'Jeanie Bryson', 'Paul Hardcastle', 'Russ Freeman/Chris
Botti', 'Torcuato Mariano', 'k.d. lang', 'Ronan Hardiman', 'Special EFX',
'Mother Love Bone', 'Nelly Furtado', 'Sponge', 'Brandon Fields/Phil Perry',
'Pimps', 'Derek and the Dominos', 'Afro Blue', 'Congalegre', 'Haig and Haig',
'Travels', 'Ian Dury and The Blockheads', 'Jane Siberry', '[unknown]', 'Marty
Robbins', 'Romatt Project', 'Jeno Jando', 'Doves', 'Eva Cassidy', 'Faith No
More', 'Jimi Hendrix', 'Café Del Mar', 'The Electric Prunes', 'Duke Ellington
and Johnny Hodg', 'Eric Clapton', 'Tom Waits', 'Wayman Tisdale', 'Incognito',
'Simply Red', 'Pavement', 'Tindersticks', 'Butthole Surfers', 'New Order', 'Phil
Collins', 'David Bowie', 'Eurythmics', 'Dream Theater', 'Kid Rock', 'Rancid',
'Ella Fitzgerald', 'Mescalito', 'Eminem']
```

```
[41]: # ANSWER 5 (2 PTS)
      # Make 10 recommendations for testUserID = 1000002
      num_recomm = 10
      recommendationsForUser = map(lambda observation: artistByID.get(observation.
       ↪product), model.call("recommendProducts", testUserID, num_recomm))
      print(list(recommendationsForUser))
```

```
['[unknown]', 'Radiohead', 'The White Stripes', 'Dream Theater', 'Nine Inch
Nails', 'Led Zeppelin', 'Metallica', 'Incubus', 'Queens of the Stone Age', 'Rage
Against the Machine']
```

```
[42]: # ANSWER 6 (2 PTS)
      # Using the rank 20 model, make 10 recommendations for testUserID = 1000002
      num_recomm = 10
      recommendationsForUser_rank20 = map(lambda observation: artistByID.
       ↪get(observation.product), model2.call("recommendProducts", testUserID,␣
       ↪num_recomm))
```

```
print(list(recommendationsForUser_rank20))
```

```
['Eminem', '[unknown]', "Guns N' Roses", 'Radiohead', 'Depeche Mode', 'The White
Stripes', 'Dream Theater', 'The Velvet Underground', 'Bruce Springsteen', 'David
Bowie']
```

[43]:

```
# 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.
num_recomm = 10
recommendationsForUser = map(lambda observation: artistByID.get(observation.
 ↪product), model.call("recommendProducts", testUserID, num_recomm))

recommendationsForUser_rank20 = map(lambda observation: artistByID.
 ↪get(observation.product), model2.call("recommendProducts", testUserID,␣
 ↪num_recomm))
print(list(recommendationsForUser_rank20))

x = list(recommendationsForUser)
y = list(recommendationsForUser_rank20)
a = []
b = []
for item in x:
    if item in artistsForUser:
        a.append(True)
    else:
        a.append(False)
for item in y:
    if item in artistsForUser:
        b.append(True)
    else:
        b.append(False)
print("This is the result for our rank 10 model: ")
```

```
['Eminem', '[unknown]', "Guns N' Roses", 'Radiohead', 'Depeche Mode', 'The White
Stripes', 'Dream Theater', 'The Velvet Underground', 'Bruce Springsteen', 'David
Bowie']
This is the result for our rank 10 model:
```

[ ]:

```
# Save notebook as PDF document
!jupyter nbconvert --to pdf `pwd`/*.ipynb
```

[ ]:

```
```