Sử dụng Spark để xây dựng một hệ thống đề xuất âm nhạc

Hệ thống đề xuất là công cụ giúp dự đoán những gì mà người dùng có thể thích hoặc không thích trong danh sách những mục đã cho. Đây này là cách để thay thế cho việc tìm kiếm nội dung, nó sẽ giúp người dùng khám phá sản phẩm hoặc nội dung mà họ có thể chưa bắt gặp. Ví dụ: Facebook gợi ý bạn bè, các page cho người dùng, Youtube gợi ý video mà người dùng có thể quan tâm, Tiki, Lazada gợi ý các sản phẩm, mặt hàng mà người dùng có thể cần... Công cụ này sẽ thu hút người dùng đến các dịch vụ từ đó có thể tối ưu hóa doanh thu cho các nhà cung cấp và duy trì sự quan tâm đến dịch vụ.

Trong notebook này, chúng tôi sẽ tập trung vào việc sử dụng Spark để xây dựng một hệ thống đề xuất đơn giản và sử dụng một thuật toán để dự đoán những mục mà người dùng có thể thích được gọi là *ALS (Alternating least squares)*.

1. Data

Trong notebook này, chúng tôi sử dụng bổ dữ liệu được xuất bản bởi Audioscrobbler - một hệ thống đề xuất âm nhạc cho last.fm.

1.1 Data schema

Không giống như tập dữ liệu xếp hạng chứa thông tin về sở thích của người dùng đối với sản phẩm (một sao, năm sao,...)(explicit ratings), bộ dữ liệu từ Audioscrobbler chỉ có thông tin về số lần một người dùng phát bài hát của một nghệ sĩ và tên của nghệ sĩ (implicit ratings).

Dữ liệu chúng tôi sử dụng trong Notebook này có 3 tệp:

- **user_artist_data.txt**: Nó chứa khoảng 141000 người dùng, 1.6 triệu nghệ sĩ và khoảng 24.2 triệu lượt phát của người dùng được ghi lại. Nó có 3 cột cách nhau bởi dấu cách.

userID	artistID	playCount

- **artist_data.txt**: Nó có 2 cột được tách nhau bởi dấu tab (\t). Cột thứ nhất chứa ID của nghệ sĩ, cột thứ hai chứa tên tương ứng với ID.

artistID	name

- **artist_alias.txt**: Có trường hợp tên của nghệ sĩ có thể bị sai chính tả hoặc không chuẩn. Ví dụ: "The Smiths", "Smiths, The" và "the smiths" có thể xuất hiện với ID riêng biệt trong tập dữ liệu mặc dù chúng cùng là một nghệ sĩ. artist_alias.txt ánh xạ ID nghệ sĩ lỗi chính tả hoặc không chuẩn thành ID chuẩn của nghệ sĩ đó. Dữ liệu trong tệp này có 2 cột được phân tách bằng dấu tab (\t).

misspelledID	standardID

1.2 Một số thống kê mô tả về dữ liệu

Để lựa chọn hoặc thiết kế một thuật toán phù hợp cho bài toán, trước tiên chúng ta tìm hiểu các đặc điểm của dữ liệu. Trước tiên, ta cần import một số thư viện cần thiết. Đồng thời, do Notebook này được thực thi trên Google Colab nên ta cần cài đặt một số các gói và thiết lập biến môi trường.

In [0]:
```
# cài đặt java 8, apache spark, findspark
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q http://mirror.downloadvn.com/apache/spark/spark-2.4.4/spark-2.4.4-bin
-hadoop2.7.tgz
!tar xvf spark-2.4.4-bin-hadoop2.7.tgz
!pip install -q findspark
```

In [0]:
```
# cài đặt biến môi trường
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.4-bin-hadoop2.7"
```

In [0]:
```
# liên kết tới drive
from __future__ import print_function

from google.colab import drive
drive.mount('/content/drive')
```

In [0]:
```
import findspark
findspark.init()
```

In [0]:
```
import pyspark
```

In [0]:
```
import sys
import re
import random
from pyspark import SparkContext, SparkConf
from pyspark.sql import SparkSession, Row, SQLContext
from pyspark.sql.types import *
from pyspark.sql.functions import *

%matplotlib inline
import pandas as pd
import numpy as np
from time import time
```

In [0]:
```
try:
    sc.stop()
except:
    pass
sc=SparkContext()
spark = SparkSession(sparkContext=sc)
sqlContext = SQLContext(sc)
base = "/content/drive/My Drive/Dataset/"
```

In [8]:
```
# tạo RDD bằng textFile từ user_artist_data.txt
rawUserArtistData = sc.textFile(base + "user_artist_data.txt")

# đưa ra 5 dòng đầu tiên của rawUserArtistData
rawUserArtistData.take(5)
```

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

`RDD` gốc là tập hợp của các dòng trong dữ liệu. Để tiện cho việc xử lý sau này ta sẽ đưa `RDD` gốc này về `RDD` của các `tuple` hoặc `list`

In [9]:
```
# chia mỗi dòng bởi dấu "khoảng trắng"
# đưa mỗi dòng về dạng tuple(int, int, int)
userArtistData = rawUserArtistData \
    .map(lambda line: line.split()) \
    .map(lambda p: (int(p[0]), int(p[1]), int(p[2])))

userArtistData.take(5)
```

Out[9]:
```
[(1000002, 1, 55),
 (1000002, 1000006, 33),
 (1000002, 1000007, 8),
 (1000002, 1000009, 144),
 (1000002, 1000010, 314)]
```

Ta sẽ sử dụng một số Action, Transformation trong Spark RDD để đưa ra một vài các thống kê cơ bản.

In [10]:
```
# Đưa ra 5 nghệ sĩ có nhiều lượt nghe nhất:
# map để lấy ra artistID và playCount
# reduceByKey để tính tổng playCount cho mỗi artistID
# sortBy để sắp xếp theo playCount đã tổng hợp

listenCountSorted = userArtistData \
    .map(lambda p: (p[1], p[2])) \
    .reduceByKey(lambda v1,v2: v1 + v2) \
    .sortBy(lambda x: x[1], ascending = False)

listenCountSorted.take(5)
```

Out[10]:
```
[(979, 2502130),
 (1000113, 2259185),
 (4267, 1930592),
 (1000024, 1542806),
 (4468, 1425942)]
```

Ta cũng có thể dùng hàm `combineByKey(creatCombiner, mergeValue, mergeCombiners)` thay cho hàm `reduceByKey` trong trường hợp trên như đoạn mã dưới đây:

In [11]:
```
def createCombiner(d):
    return d

def mergeValue(a,b):
    return a + b

def mergeCombiners(a,b):
    return a + b

listenCountSorted = userArtistData \
    .map(lambda p: (p[1], p[2])) \
    .combineByKey(createCombiner, mergeValue, mergeCombiners) \
    .sortBy(lambda x: x[1], ascending = False)

listenCountSorted.take(5)
```

Out[11]:
```
[(979, 2502130),
 (1000113, 2259185),
 (4267, 1930592),
 (1000024, 1542806),
 (4468, 1425942)]
```

In [12]:
```
## Đối với mỗi user, đưa ra tổng số artist mà user đó đã nghe

# map để lấy ra userID và artistID
# groupByKey để nhóm các artistID thành một bộ theo userID
# mapValues(len) để đếm số phần tử trong bộ đó.
numArtistListened = userArtistData \
    .map(lambda p: (p[0], p[1])) \
    .groupByKey() \
    .mapValues(len)

numArtistListened.take(5)
```

Out[12]:
```
[(1000025, 233), (1000038, 59), (1000051, 50), (1000077, 116), (1000090, 15
0)]
```

Đối với cùng một `userID`, 1 `artistID` chỉ xuất hiện 1 lần nên ta có thể sử dụng hàm `aggregateByKey` như bên dưới để đưa ra tổng số playCount và tổng số artist mà user đó đã nghe.

In [13]:
```
## Đối với mỗi user, đưa ra tổng số playCount và tổng số artist mà user đó đã
 nghe

seqFunc = (lambda x, y: (x[0] + y, x[1] + 1))
combFunc = (lambda x, y: (x[0] + y[0], x[1] + y[1]))

# map để lấy ra userID và playCount
numPlayCountListened = userArtistData \
    .map(lambda p: (p[0], p[2])) \
    .aggregateByKey((0,0), seqFunc, combFunc)

numPlayCountListened.take(5)
```

Out[13]:
```
[(1000025, (3589, 233)),
 (1000038, (129, 59)),
 (1000051, (131, 50)),
 (1000077, (2657, 116)),
 (1000090, (904, 150))]
```

In [14]:
```
# sử dụng spark.creatDataFrame để tạo DataFrame gán tên cho các cột
userArtistDF = spark.createDataFrame(userArtistData, ("userID", "artistID", "p
layCount"))

userArtistDF.show(10)
```

```
+-------+--------+---------+
| userID|artistID|playCount|
+-------+--------+---------+
|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|
+-------+--------+---------+
only showing top 10 rows
```

In [15]:
```
# hoặc có thể dùng toDF để tạo DataFrame
userArtistDF = userArtistData.toDF(("userID", "artistID", "playCount"))

userArtistDF.show(10)
```

```
+-------+--------+---------+
| userID|artistID|playCount|
+-------+--------+---------+
|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|
+-------+--------+---------+
only showing top 10 rows
```

In [16]:
```
# Lưu trữ DataFrame vào bố nhớ cache
userArtistDF.cache()
```

Out[16]:
```
DataFrame[userID: bigint, artistID: bigint, playCount: bigint]
```

In [17]:
```
# Đếm số người dùng khác nhau trong dữ liệu
uniqueUsers = userArtistDF.select("userID").distinct().count()
print("Tổng số người dùng: ", uniqueUsers)
```

```
Tổng số người dùng:  148111
```

In [18]:
```
# Đếm số nghệ sĩ khác nhau trong dữ liệu
uniqueArtists = userArtistDF.select("artistID").distinct().count()
print("Tổng số nghệ sĩ: ", uniqueArtists)
```

```
Tổng số nghệ sĩ:  1631028
```

Một hạn chế khi sử dụng ALS trong Spark MLlib mà chúng ta sẽ sử dụng sau này là nó yêu cầu những IDs phải là những số nguyên 32-bit không âm tức là 0 <= IDs <= 2^31 - 1 (2147483647). Do đó, ta phải kiểm tra bộ dữ liệu này có phù hợp không.

In [19]:
```
# đưa ra các mô tả của DataFrame
userArtistDF.describe().show()
```

```
+-------+------------------+------------------+------------------+
|summary|            userID|          artistID|         playCount|
+-------+------------------+------------------+------------------+
|  count|          24296858|          24296858|          24296858|
|   mean|1947573.2653533637|1718704.0937568964| 15.29576248089362|
| stddev| 496000.5551818977|2539389.0924283406|153.91532446980173|
|    min|                90|                 1|                 1|
|    max|           2443548|          10794401|            439771|
+-------+------------------+------------------+------------------+
```

Nhận thấy trong tập dữ liệu chỉ có 148111 người dùng, 1631028 nghệ sĩ, giá trị lớn nhất của `userID` và `artistID` vẫn nhỏ hơn số lớn nhất của kiểu integer. Do đó, ta có thể sử dụng được luôn những ID này.

Tiếp theo, ta sẽ đi phân tích file dữ liệu `artist_data.txt`

In [20]:
```
# tạo RDD từ artist_data.txt
rawArtistData = sc.textFile(base + "artist_data.txt")

# đưa ra 5 dòng đầu tiên của rawArtistData
rawArtistData.take(5)
```

Out[20]:
```
['1134999\t06Crazy Life',
 '6821360\tPang Nakarin',
 '10113088\tTerfel, Bartoli- Mozart: Don',
 '10151459\tThe Flaming Sidebur',
 '6826647\tBodenstandig 3000']
```

Như đã nói ở trên, 2 cột của `artist_data.txt` được tách nhau bởi dấu tab (\t). Ta sẽ thêm một vài xử lý cho dữ liệu này để đưa nó về DataFrame để có thể dễ dàng quan sát và xử lý.

Trước tiên, do trong `artist_data.txt` sẽ có trường hợp không có ID của nghệ sĩ hoặc ID không ở kiểu integer nên ta sẽ đi xây dựng hàm `isInt()` để kiểm tra ID có là kiểu int không.

In [0]:
```
# hàm isInt() kiểm tra 1 đầu vào có phải kiểu Int không
def isInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False
```

In [22]:
```
# chia mỗi dòng bởi dấu "\t"
# loại bỏ những dòng không có ID hoặc ID không phải kiểu Int
# đưa mỗi dòng về dạng tuple(int, str)
artistData = rawArtistData \
    .map(lambda line: line.split('\t')) \
    .filter(lambda line: line[0] and isInt(line[0])) \
    .map(lambda p: (int(p[0]),p[1].strip()))

artistData.take(5)
```

Out[22]:
```
[(1134999, '06Crazy Life'),
 (6821360, 'Pang Nakarin'),
 (10113088, 'Terfel, Bartoli- Mozart: Don'),
 (10151459, 'The Flaming Sidebur'),
 (6826647, 'Bodenstandig 3000')]
```

In [23]:
```
# chuyển về DataFrame và gán tên cho các cột
artistDF = artistData.toDF(("artistID", "name"))

# Lưu trữ DataFrame vào bộ nhớ cache
artistDF.cache()

artistDF.show(5)
```

```
+--------+--------------------+
|artistID|                name|
+--------+--------------------+
| 1134999|        06Crazy Life|
| 6821360|        Pang Nakarin|
|10113088|Terfel, Bartoli- ...|
|10151459| The Flaming Sidebur|
| 6826647|   Bodenstandig 3000|
+--------+--------------------+
only showing top 5 rows
```

In [24]:
```
# đưa ra những nghệ sĩ trong tên có chứa "Aerosmith"
artistDF[locate("Aerosmith", artistDF.name) > 0].show(20,False)

# đưa ra tên của nghệ sĩ có ID là 1000010 và 2082323
artistDF[artistDF.artistID==1000010].show()
artistDF[artistDF.artistID==2082323].show()
```

```
+--------+---------------------------------------------+
|artistID|name                                         |
+--------+---------------------------------------------+
|10586006|Dusty Springfield/Aerosmith                  |
|6946007 |Aerosmith/RunDMC                             |
|10475683|Aerosmith: Just Push Play                    |
|1083031 |Aerosmith/ G n R                             |
|6872848 |Britney, Nsync, Nelly, Aerosmith,Mary J Blige.|
|10586963|Green Day - Oasis - Eminem - Aerosmith       |
|10028830|The Aerosmith Antology2                      |
|10300357|Run-DMC + Aerosmith                          |
|2027746 |Aerosmith by MusicInter.com                  |
|1140418 |[rap]Run DMC and Aerosmith                   |
|10237208|Aerosmith + Run DMC                          |
|10588537|Aerosmith, Kid Rock, & Run DMC               |
|9934757 |Aerosmith - Big Ones                         |
|10437510|Green Day ft. Oasis & Aerosmith              |
|6936680 |RUN DNC & Aerosmith                          |
|10479781|Aerosmith Hits                               |
|10114147|Charlies Angels - Aerosmith                  |
|1262439 |Kid Rock, Run DMC & Aerosmith                |
|7032554 |Aerosmith & Run-D.M.C.                       |
|10033592|Aerosmith?                                   |
+--------+---------------------------------------------+
only showing top 20 rows

+--------+---------+
|artistID|     name|
+--------+---------+
| 1000010|Aerosmith|
+--------+---------+

+--------+------------+
|artistID|        name|
+--------+------------+
| 2082323|01 Aerosmith|
+--------+------------+
```

Nhận thấy hai nghệ sĩ có ID `1000010` và `2082323` thực chất là một. Đáng lẽ chúng phải cùng trỏ đến một ID duy nhất. Do đó, ta cần sử dụng tập dữ liệu `artist_alias.txt` chứa ID của các nghệ sĩ viết sai và ID của các nghệ sĩ chuẩn.

In [25]:
```
# tạo RDD từ artist_alias.data
rawArtistAlias = sc.textFile(base + "artist_alias.txt")

rawArtistAlias.take(5)
```

Out[25]:
```
['1092764\t1000311',
 '1095122\t1000557',
 '6708070\t1007267',
 '10088054\t1042317',
 '1195917\t1042317']
```

In [26]:
```
# chia mỗi dòng bởi dấu "\t"
# loại bỏ những dòng không có ID hoặc ID không phải kiểu Int
# đưa mỗi dòng về dạng tuple(int, int)
artistAlias = rawArtistAlias \
    .map(lambda line: line.split("\t")) \
    .filter(lambda line: line[0] and isInt(line[0])) \
    .map(lambda p: (int(p[0]), int(p[1])))

artistAlias.take(5)
```

Out[26]:
```
[(1092764, 1000311),
 (1095122, 1000557),
 (6708070, 1007267),
 (10088054, 1042317),
 (1195917, 1042317)]
```

In [27]:
```
# chuyển về DataFrame và gán tên cho cột
artistAliasDF = artistAlias.toDF(("misspelledID", "standardID"))

# Lưu trữ DataFrame vào bộ nhớ cache
artistAliasDF.cache()

artistAliasDF.show(5)
```

```
+------------+----------+
|misspelledID|standardID|
+------------+----------+
|     1092764|   1000311|
|     1095122|   1000557|
|     6708070|   1007267|
|    10088054|   1042317|
|     1195917|   1042317|
+------------+----------+
only showing top 5 rows
```

In [28]:
```
# hiển thị ra những dòng có misspelledID = 100010, 2082323
artistAliasDF[artistAliasDF.misspelledID==1000010].show()
artistAliasDF[artistAliasDF.misspelledID==2082323].show()
```

```
+------------+----------+
|misspelledID|standardID|
+------------+----------+
+------------+----------+

+------------+----------+
|misspelledID|standardID|
+------------+----------+
|     2082323|   1000010|
+------------+----------+
```

Thấy rằng `1000010` là ID chuẩn còn `2082323` là ID không chính xác. Do đó nghệ sĩ có tên "01 Aerosmith" sẽ ánh xạ đến tên "Aerosmith".

Ta có thể xây dựng một hàm `isStandard()` để kiểm tra xem một ID nghệ sĩ có phải là ID chuẩn hay không.

In [0]:
```
def isStandard(id):
    # trả về True nếu id không nằm trong cột misspellID của artistAliasDF
    if artistAliasDF[artistAliasDF.misspelledID == id].collect() == []:
        return True
    else:
        return False
```

In [30]:
```
# kiểm tra xem id ứng với nghệ sĩ "Green Day - Oasis - Eminem - Aerosmith" có
 phải ID chuẩn không
name1 = "Green Day - Oasis - Eminem - Aerosmith"
id1 = artistDF[artistDF.name == name1].first().artistID
isStandard(id1)
```

Out[30]:
```
True
```

In [31]:
```
# In ra xem có bao nhiêu user nghe nghệ sĩ có tên là "Green Day - Oasis - Emin
em - Aerosmith"
print(userArtistDF[userArtistDF.artistID==id1].count(), " user đã nghe ", name
1)
```

```
1  user đã nghe  Green Day - Oasis - Eminem - Aerosmith
```

Việc có những ID hay là tên của nghệ sĩ không chuẩn sẽ dẫn đến những truy vấn không thực sự chính xác. Để khắc phục điều này thì ta sẽ thay thế những ID không chuẩn bằng ID chuẩn và tính toán lại các thống kê mô tả cơ bản.

Đầu tiên ta xây dựng một từ điển mà nó ánh xạ từng ID không chuẩn sang ID chuẩn.

In [0]:
```
# chuyển DataFrame về RDD bằng <dataframe>.rdd.map
# thu thập kết quả cho driver như một dictionary
artistAlias = artistAliasDF \
    .rdd.map(lambda row: (row.misspelledID, row.standardID)) \
    .collectAsMap()
```

In [33]:
```
# hàm thay thế ID sai trong userArtistDF
def replaceMisspelledIDs(fields):
    # input: (userID, artistID, playCount)
    # nếu ID có trong trong từ điển (misspelledID) thì ánh xạ nó về standardI
D, nếu không thì giữ nguyên.
    finalArtistID = artistAlias.get(fields[1], fields[1])
    return (fields[0], finalArtistID, fields[2])

# đặt mốc thời gian để xác định thời gian xử lý
t0 = time()

# tạo DataFrame mới từ DataFrame cũ đã thay thế misspelledID
newUserArtistDF = userArtistDF \
    .rdd.map(replaceMisspelledIDs) \
    .toDF(("userID", "artistID", "playCount"))

newUserArtistDF.show(5)

t1 = time()

print('thời gian thực thi là %f giây' %(t1 - t0))
```

```
+-------+--------+---------+
| userID|artistID|playCount|
+-------+--------+---------+
|1000002|       1|       55|
|1000002| 1000006|       33|
|1000002| 1000007|        8|
|1000002| 1000009|      144|
|1000002| 1000010|      314|
+-------+--------+---------+
only showing top 5 rows

thời gian thực thi là 1.989177 giây
```

Ta có thể cải thiện thời gian thực thi này bằng cách sử dụng *broadcast variable* (https://spark.apache.org/docs/latest/rdd-programming-guide.html#broadcast-variables). Broadcast variable cho phép lập trình viên giữ một biến chỉ đọc lưu trữ trong bộ nhớ cache trên **mỗi máy** thay vì gửi một bản sao của nó cùng với các tác vụ (tasks).

In [34]:
```
# tạo broadcast variable cho artistAlias
bArtistAlias = sc.broadcast(artistAlias)

def replaceMisspelledIDs(fields):
    finalArtistID = bArtistAlias.value.get(fields[1], fields[1]) # giá trị của
broadcast variable được truy cập bằng phương thức value
    return (fields[0], finalArtistID, fields[2])

t0 = time()

newUserArtistDF = userArtistDF \
    .rdd.map(replaceMisspelledIDs) \
    .toDF(("userID", "artistID", "playCount"))

newUserArtistDF.show(5)

t1 = time()

print('thời gian thực thi là %f giây' %(t1 - t0))

# Lưu trữ DataFrame vào bộ nhớ cache
newUserArtistDF.cache()
```

```
+-------+--------+---------+
| userID|artistID|playCount|
+-------+--------+---------+
|1000002|       1|       55|
|1000002| 1000006|       33|
|1000002| 1000007|        8|
|1000002| 1000009|      144|
|1000002| 1000010|      314|
+-------+--------+---------+
only showing top 5 rows

thời gian thực thi là 0.629630 giây
```

Out[34]:
```
DataFrame[userID: bigint, artistID: bigint, playCount: bigint]
```

Kết quả được cải thiện đáng kể, rõ ràng việc tạo broadcast variable chỉ hữu ích khi các tác vụ (tasks) qua nhiều giai đoạn cần dùng cùng một dữ liệu.

Bây giờ, DataFrame đã chứa dữ liệu chuẩn. Ta có thể sử dụng nó để thực hiện lại những truy vấn thống kê trước đó

In [35]:
```
# Đếm số nghệ sĩ khác nhau trong dữ liệu mới
uniqueArtists = newUserArtistDF.select("artistID").distinct().count()
print("Tổng số nghệ sĩ: ", uniqueArtists)
```

```
Tổng số nghệ sĩ:  1568126
```

Nhận thấy kết quả này ít hơn so với kết quả ban đầu từ tập dữ liệu cũ (1631028). Điều này là hoàn toàn hợp lý.

Giờ chúng ta đã có một vài thông tin cơ bản về dữ liệu. Trong phần tiếp theo, chúng tôi sẽ nghiên cứu để xây dựng mô hình thống kê.

2. Xây dựng mô hình để đưa ra các đề xuất

2.1 Giới thiệu về hệ thống đề xuất

Trong một ứng dụng hệ thống đề xuất, có hai loại thực thể là người dùng (`users`) và mặt hàng (`items`). Người dùng có những ưu tiên cho một số mặt hàng, và những ưu tiên này phải có từ dữ liệu. Bản thân dữ liệu được biểu biểu dưới dạng một ma trận ưu tiên (`preference matrix`) A, với mỗi cặp người dùng-mặt hàng, có một giá trị thể hiện mức độ ưu tiên của người dùng với mặt hàng đó. Ví dụ ma trận ưu tiên 5 người dùng và k mặt hàng:

	IT1	IT2	IT3	...	ITk
U1	1		5	...	3
U2		2	2	...	2
U3	5		3	...	
U4	3	3		...	4
U5		1		...	

Giá trị của hàng i, cột j biểu thị sự yêu thích của người dùng i đối với mặt hàng j, lấy ví dụ giá trị 1 (dislike), giá trị 5 (love). Những ô trống tức là không có thông tin về sở thích của người dùng với mặt hàng đó. Mục tiêu của hệ thống đề xuất là dự đoán những ô trống này. Có 2 cách tiếp cận:

- Xem xét sự giống nhau dựa trên các thuộc tính của mặt hàng: nhãn hiệu, giá cả, danh mục. Ví tụ IT2 khá giống IT3, U5 không thích IT2 nên có thể họ cũng không thích IT3.
- Quan sát xu hướng đánh giá của các người dùng cho các sản phẩm. Ví dụ những người dùng không thích IT2 có xu hướng không thích IT3. Do đó, có thể dự đoán U5 cũng không thích IT3.

2.2 Collaborative-Filtering và ví dụ của Matrix Factorization

`Collaborative-Filtering` là một kỹ thuật được sử dụng trong hệ thống đề xuất. Nó tập trung vào mối quan hệ giữa người dùng và mặt hàng, trong bài toán này, các nghệ sĩ giữ vai trò là mặt hàng. Độ tương tự của các mặt hàng được xác định bởi sự giống nhau về xếp hạng của các mặt hàng đó bởi những người dùng đã xếp hạng cả hai mặt hàng.

Ta sẽ nghiên cứu một thành viên trong lớp các thuật toán được gọi là mô hình `latent-factor` .

Ban đầu ta có n người dùng và m mặt hàng (trong bài này là nghệ sĩ) và `playCount` tương ứng (xếp hạng ngầm - implicit rating). Ta sẽ xây dựng được ma trận R có n hàng m cột. Mỗi ô (u, i) trong ma trận có giá trị là xếp hạng (rating) của người dùng u cho mặt hàng i.

Ma trận R có nhiều mục bị thiếu. Chúng ta sẽ sử dụng mô hình `matrix factorization` để điền những mục này.

Ví dụ: cho ma trận 5x5 như bên dưới. Ta muốn xấp xỉ ma trận thành hai ma trận nhỏ hơn X và Y.

$$M = \begin{bmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{bmatrix} \approx M' = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ x_{41} & x_{42} \\ x_{51} & x_{52} \end{bmatrix} \times \begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{14} & y_{15} \\ y_{21} & y_{22} & y_{23} & y_{24} & y_{25} \end{bmatrix}$$

M' là một xấp xỉ càng gần M càng tốt. Sai số M và M' được tính bằng tổng bình phương sai số của các phần tử không trống trong M và các phần tử tương ứng trong M'. Trong M' không có phần tử trống, do đó để xem mức độ người dùng i ưa thích mặt hàng j ta chỉ cần lấy ra giá trị $M'_{i,j}$.

Vấn đề là làm sao tính được X và Y. Ta không thể tính cùng lúc X và Y đều tốt nhất nhưng nếu biết Y ta có thể tính X tốt nhất và ngược lại. Do đó, ta khởi tạo những giá trị ban đầu cho X và Y, ta tính X tốt nhất theo Y và sau đó tính Y tốt nhất theo X mới. Lặp lại quá trình này cho đến khi sai số $X \times Y$ tới M là hội tụ.

Với ví dụ trên, ta khởi tạo giá trị của X và Y như sau:

$$M' = X \times Y = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Trong lần lặp đầu tiên, tính trung bình bình phương sai số (RMSE - Root Mean Square Error) giữa XY và M (lưu ý: ô nào trống bỏ qua):

$$RMSE = \sqrt{\frac{(5-2)^2 + (2-2)^2 + \ldots + (5-2)^2 + (4-2)^2}{23}} = \sqrt{\frac{75}{23}} = 1.806$$

23 là số ô không trống trong M.

Tiếp theo, với Y đã cho, ta tính X bằng cách tìm giá trị tốt nhất cho X_{11}.

$$M' = X \times Y = \begin{bmatrix} x & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Bây giờ để giảm thiểu $RMSE$, ta sẽ giảm thiểu sai số ở hàng đầu tiên, bằng cách đạo hàm ta sẽ tìm được $x = 2.6$

$$(5-(x+1))^2 + (2-(x+1))^2 + (4-(x+1))^2 + (4-(x+1))^2 + (3-(x+1))^2$$

Với giá trị mới của X, ta có thể tính giá trị tốt nhất cho Y

$$M' = X \times Y = \begin{bmatrix} 2.6 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} y & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 3.6 & 3.6 & 3.6 & 3.6 & 3.6 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{bmatrix}$$

Thực hiện tương tự, ta sẽ tìm được $y = 1.617$. Sau đó, ta có thể kiểm tra nếu $RMSE$ không hội tụ, ta tiếp tục cập nhật X và Y và ngược lại. Trong ví dụ này, ta chỉ cập nhật một phần tử của mỗi ma trận trong mỗi lần lặp. Thực tế chúng ta có thể cập nhật một hàng đầy đủ hoặc một ma trận đầy đủ trong một lần.

2.3 Matrix Factorization và thuật toán ALS trên một máy

Nếu chúng ta chọn k latent features và mô tả mỗi người dùng u với một vector k-chiều x_u, và mỗi mặt hàng i với một vector k-chiều y_i.

Sau đó, để dự đoán xếp hạng của người dùng u cho mặt hàng i, ta tính: $r_{ui} \approx x_u^T y_i$

Có n người dùng, sẽ có n vector $x_1, \ldots, x_n$. Có m mặt hàng, sẽ có m vector $y_1, \ldots, y_m$. Ta định nghĩa ma trận X và Y như sau:

$$X = \begin{bmatrix} | & & | \\ x_1 & \dots & x_n \\ | & & | \end{bmatrix}$$
$$Y = \begin{bmatrix} | & & | \\ y_1 & \dots & y_n \\ | & & | \end{bmatrix}$$

Mục tiêu là phải ước tính ma trận $R \approx X^T Y$ hay bài toán đưa về tìm X và Y tốt nhất để tối ưu hàm mục tiêu dưới đây:

$$\min_{X,Y} \sum_{r_{ui} observed} \left(r_{ui} - x_u^T y_i\right)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

Cách làm là ta sửa Y và tối ưu X, sau đó sửa X và tối ưu Y, lặp lại cho đến khi hội tụ. Đây được gọi là thuật toán **ALS (Alternating Least Squares)**. Dưới đây là giả code của thuật toán:

Khởi tạo X, Y

while (không hội tụ) **do**

for $u = 1 ... n$ **do**

$x_u = \left(\sum_{r_u i \in r_{u*}} y_i y_i^T + \lambda I_k\right)^{-1} \sum_{r_u i \in r_{u*}} r_{ui} y_i$

end for

for $u = 1 ... n$ **do**

$y_i = \left(\sum_{r_u i \in r_{*i}} x_u x_u^T + \lambda I_k\right)^{-1} \sum_{r_u i \in r_{*i}} r_{ui} x_u$

end for

end while

2.4 Thuật toán ALS song song trên Spark

Ta có thể tính toán thuật toán ALS song song trên Spark bằng cách sau:

Dữ liệu đầu vào xếp hạng (ratings) và tham số (X và Y) được lưu trữ trong Spark RDD. Cụ thể, các xếp hạng được lưu trữ dưới dạng RDD của một bộ ba

Ratings: RDD$\left((u, i, r_{ui}), \ldots\right)$

Ma trận X và Y được lưu trữ dưới dạng RDD của vectors:

X: RDD$(x_1, \ldots, x_n)$

Y: RDD$(y_1, \ldots, y_m)$

Biểu thức tính x_u:

$x_u = \left(\sum_{r_u i \in r_{u*}} y_i y_i^T + \lambda I_k\right)^{-1} \sum_{r_u i \in r_{u*}} r_{ui} y_i$

Gọi phần tổng đầu tiên là *part A*, phần tổng thứ hai là *part B*. Để tính toán các part như vậy song song, ta có thể thực hiện với giả code sau:

- `join` RDD Ratings với RDD ma trận Y bằng key i (items)
- `map` để tính toán $y_i y_i^T$ và phát ra bằng key u (user)
- `reduceByKey` u (user) để tính toán $\sum_{r_u i \in r_{u*}} y_i y_i^T$
- Đảo ngược
- `reduceByKey` u (user) để tính toán $\sum_{r_u i \in r_{u*}} r_{ui} y_i$

Tương tự, ta cũng tính toán được y_i

3. Hệ thống đề xuất âm nhạc

Trong phần này, chúng ta sẽ xây dựng hệ thống gợi ý âm nhạc cho người dùng.

3.1 Làm sạch dữ liệu

Làm tương tự phần 1, chỉ khác rằng đầu ra của dữ liệu sau khi làm sạch để ở RDD.

In [36]:
```
# tạo RDD từ artist_alias.txt
rawArtistAlias = sc.textFile(base + "artist_alias.txt")

rawArtistAlias.take(5)
```

Out[36]:
```
['1092764\t1000311',
 '1095122\t1000557',
 '6708070\t1007267',
 '10088054\t1042317',
 '1195917\t1042317']
```

In [0]:
```
# hàm isInt() kiểm tra 1 đầu vào có phải kiểu Int không
def isInt(s):
    try:
        int(s)
        return True
    except ValueError:
        return False
```

In [0]:
```
# chia từng dòng trong RDD gốc bởi dấu "\t"
# loại bỏ những dòng không có ID hoặc ID không phải kiểu Int
# đưa về tuple(int, int)
# lưu kết quả như một dictionary
artistAlias = rawArtistAlias \
    .map(lambda line: line.split("\t")) \
    .filter(lambda line: line[0] and isInt(line[0])) \
    .map(lambda p: (int(p[0]), int(p[1]))) \
    .collectAsMap()
```

In [39]:
```
# tạo RDD từ user_artist_data.txt
rawUserArtistData = sc.textFile(base + "user_artist_data.txt")

rawUserArtistData.take(5)
```

Out[39]:
```
['1000002 1 55',
 '1000002 1000006 33',
 '1000002 1000007 8',
 '1000002 1000009 144',
 '1000002 1000010 314']
```

In [40]:
```
# tạo broadcast variable cho artistAlias
bArtistAlias = sc.broadcast(artistAlias)

# hàm thay thế ID sai trong rawUserArtistData
def replaceMisspelledIDs(line):
    [userID, artistID, playCount] = map(lambda p: int(p), line.split(" "))
    finalArtistID = bArtistAlias.value.get(artistID, artistID)
    return (userID, finalArtistID, playCount)

# thay thế ID sai trong rawUserArtistData và lưu thành RDD mới
userArtistDataRDD = rawUserArtistData.map(replaceMisspelledIDs)

userArtistDataRDD.take(5)
```

Out[40]:
```
[(1000002, 1, 55),
 (1000002, 1000006, 33),
 (1000002, 1000007, 8),
 (1000002, 1000009, 144),
 (1000002, 1000010, 314)]
```

In [41]:
```
# tạo RDD từ artist_data.txt
rawArtistData = sc.textFile(base + "artist_data.txt")

rawArtistData.take(5)
```

Out[41]:
```
['1134999\t06Crazy Life',
 '6821360\tPang Nakarin',
 '10113088\tTerfel, Bartoli- Mozart: Don',
 '10151459\tThe Flaming Sidebur',
 '6826647\tBodenstandig 3000']
```

In [42]:
```
# chia từng dòng trong RDD gốc bởi dấu "\t"
# loại bỏ những dòng không có ID hoặc ID không phải kiểu Int
# đưa về tuple(int, str)
artistDataRDD = rawArtistData \
    .map(lambda line: line.split('\t')) \
    .filter(lambda line: line[0] and isInt(line[0])) \
    .map(lambda p: (int(p[0]),p[1].strip())) \

artistDataRDD.take(5)
```

Out[42]:
```
[(1134999, '06Crazy Life'),
 (6821360, 'Pang Nakarin'),
 (10113088, 'Terfel, Bartoli- Mozart: Don'),
 (10151459, 'The Flaming Sidebur'),
 (6826647, 'Bodenstandig 3000')]
```

3.2 Training mô hình

Chú ý: Các khái niệm sử dụng trong bài: user ⇔ người dùng: cụ thể là `userID`

item, product ⇔ mặt hàng, sản phẩm: cụ thể là `artistID` , nghệ sĩ

rating ⇔ xếp hạng: cụ thể là `playCount`

Để huấn luyện mô hình bằng `ALS` , ta phải sử dụng ma trận ưu tiên (preference matrix) như một đầu vào. MLlib sử dụng class `Rating` để hỗ trợ xây dựng ma trận ưu tiên phân tán.

In [0]:
```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
```

In [44]:
```
# xây dựng RDD mới để training bằng cách biến đổi từng mục về đối tượng Rating
# đối tượng Rating đại diện cho một tuple(user, product, rating)
allData = userArtistDataRDD.map(lambda r: Rating(r[0], r[1], r[2])).cache()

allData.take(5)
```

Out[44]:
```
[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)]
```

Mô hình có thể được train bằng cách sử dụng `ALS.trainImplicit(<trainingData>, <rank>, <interation>, <lambda>, <alpha>)` trong đó:

- `trainingData` : RDD của Rating hoặc tuple(userID, productID, rating)
- `rank` : số feature
- `interation` : số lần lặp của ALS. Mặc định: 5
- `lambda` : tham số regularization. Mặc định: 0.01
- `alpha` : tham số được sử dụng trong computing confidence. Mặc định: 0.01

Xem thêm ALS.trainImplicit (https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html#pyspark.mllib.recommendation.ALS.trainImplic

In [45]:
```
t0 = time()
model = ALS.trainImplicit(allData, 10, 5, 0.01, alpha=1.0)
t1 = time()
print("Hoàn thành training mô hình trong %f giây" % (t1 - t0))
```

```
Hoàn thành training mô hình trong 323.555163 giây
```

Việc training mô hình tiêu tốn khá nhiều thời gian do đó sau khi traning xong ta nên lưu trữ mô hình này lại.

In [0]:
```
# Lưu trữ mô hình để có thể sử dụng lại
model.save(sc, "music_model.spark")
```

Mô hình đã lưu có thể được load từ file bằng cách `MatrixFactorizationModel.load(sc, <file_name>)` ,

In [47]:
```
t0 = time()
model = MatrixFactorizationModel.load(sc, "music_model.spark")
t1 = time()
print("Hoàn thành load mô hình trong %f giây" % (t1 - t0))
```

```
Hoàn thành load mô hình trong 1.158806 giây
```

In [48]:
```
# In ra hàng đầu tiên của người dùng đặc trưng (user features) trong mô hình
# userFeatures() trả về một RDD, phần tử đầu tiên là user, phần tử thứ hai là
 một mảng các đặc trưng (features) tương ứng của user đó
first_row = model.userFeatures().first()
first_row
```

Out[48]:
```
(90,
 array('d', [0.2216065675020218, 0.31353285908699036, 0.5235552787780762, 0.1
0361422598361969, 0.40287503600120544, 0.04182986542582512, -0.92033332586288
45, 1.1663964986801147, -0.3672795593738556, -0.39548608660697937]))
```

In [49]:
```
# số lượng phần tử trong mảng các đặc trưng của user
len(first_row[1])
```

Out[49]:
```
10
```

In [50]:
```
# Đưa ra 5 đề xuất cho user 2093760
recommendations = (model.recommendProducts(2093760, 5))
recommendations
```

Out[50]:
```
[Rating(user=2093760, product=2814, rating=0.02920381773900963),
 Rating(user=2093760, product=1037970, rating=0.02874108157876482),
 Rating(user=2093760, product=1300642, rating=0.028686512592812546),
 Rating(user=2093760, product=1001819, rating=0.028609727738179675),
 Rating(user=2093760, product=4605, rating=0.028416943788400956)]
```

Ta sẽ so sánh kết quả trên với việc sử dụng hàm `ALS.train()`

In [51]:
```
model_2 = ALS.train(allData, 10, 5, 0.01)

recommendations_2 = (model_2.recommendProducts(2093760, 5))
recommendations_2
```

Out[51]:
```
[Rating(user=2093760, product=6674945, rating=951.9245134532371),
 Rating(user=2093760, product=10659711, rating=794.1949213008279),
 Rating(user=2093760, product=1282813, rating=557.5058268452832),
 Rating(user=2093760, product=2106722, rating=553.5065873552369),
 Rating(user=2093760, product=6703558, rating=532.6767992907326)]
```

Ta có thể thấy sự khác nhau giữa `train` và `trainImplicit` là kết quả `rating` của `trainImplicit` được scale về [0:1] bằng MinMaxScaler. Điều này sẽ mang nhiều ý nghĩa hơn đối với đầu vào là tập dữ liệu implicit. Giá trị `rating` càng gần 1 thì `user` càng có nhiều khả năng thích `product` đó.

Những đề xuất này chỉ là ID của nghệ sĩ. Ta sẽ ánh xạ chúng thành tên nghệ sĩ bằng cách sử dụng dữ liệu trong `artist_data.txt`

In [52]:
```
def artistNames(line):
    # input: (artistID, name)
    # trả về True nếu artistID có trong list các ID được đề xuất, ngược lại tr
ả về False
    if line[0] in [recommendations[i][1] for i in range(len(recommendations
))]:
        return True
    else:
        return False

# lọc ra các nghệ sĩ được đề xuất, lấy tên, in ra
recList = artistDataRDD.filter(lambda line: artistNames(line)).values().collec
t()

print(recList)
```

```
['50 Cent', 'Snoop Dogg', 'Kanye West', '2Pac', 'The Game']
```

In [0]:
```
def unpersist(model):
    model.userFeatures().unpersist()
    model.productFeatures().unpersist()

# giải phóng dữ liệu và model khỏi bộ nhớ cache khi không sử dụng
unpersist(model)
```

Tổng kết

Trong notebook này, chúng ta đã học được cách xây dựng một hệ thống lớn trên môi trường SPARK, tìm hiểu về `matrix factorization` và sử dụng nó để đưa ra các đề xuất.

Tài liệu tham khảo

- Uri Laserson, Sandy Ryza, Sean Owen, and Josh Wills, Advanced Analytics with Spark, 2nd ed.: O'Reilly Media, Inc, 2017.
- https://github.com/novelari/advanced-analytics-spark/blob/master/ch3/ch3-recommender.py (https://github.com/novelari/advanced-analytics-spark/blob/master/ch3/ch3-recommender.py)
- http://infolab.stanford.edu/~ullman/mmds/ch9.pdf (http://infolab.stanford.edu/~ullman/mmds/ch9.pdf)