



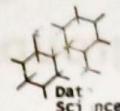
## BIG DATA IN MACHINE LEARNING

Bài 9: PySpark ML – Clustering, Recommender System

Phòng LT & Mạng

<https://cse.edu.vn/fsp/kinh-tai-csd/Big-Data-va-Machine-Learning-195>

2020



### Nội dung



1. Clustering K-Means
2. Recommender System – ALS
3. Association rules - FP-Growth

## Clustering K-Means

### □ Clustering

- Input: Unlabeled data
- Apply: Unsupervised learning Algorithm
- Output: Các cluster có thể của data

□ Nghĩa là, chúng ta có data chỉ chứa các feature và muốn tìm ra các pattern trong dữ liệu giúp có thể tạo ra các group/cluster.



## Clustering K-Means

### □ Bản chất

- Về mặt bản chất: vẫn đề là khó có thể đánh giá các group/cluster theo "mức độ chính xác" ("correctness").
- Phần lớn công việc diễn giải các cluster này dựa vào "domain knowledge".
- Ngoài ra, tùy thuộc vào thuật toán phân cụm (clustering algorithm), chúng ta có thể quyết định trước số lượng cụm muốn tạo.
- Rất nhiều vấn đề phân cụm không có cách tiếp cận hoặc trả lời đúng 100%, đó là bản chất của việc học tập không giám sát!



## Clustering K-Means

- Ví dụ:

- Nếu chúng ta có dữ liệu khách hàng, sau đó được yêu cầu phân cụm thành các nhóm riêng biệt.
- Việc phân nhóm này sẽ tùy thuộc vào chúng ta quyết định những gì sẽ đại diện cho các nhóm. Công việc này đôi khi dễ dàng, đôi khi rất khó khăn.

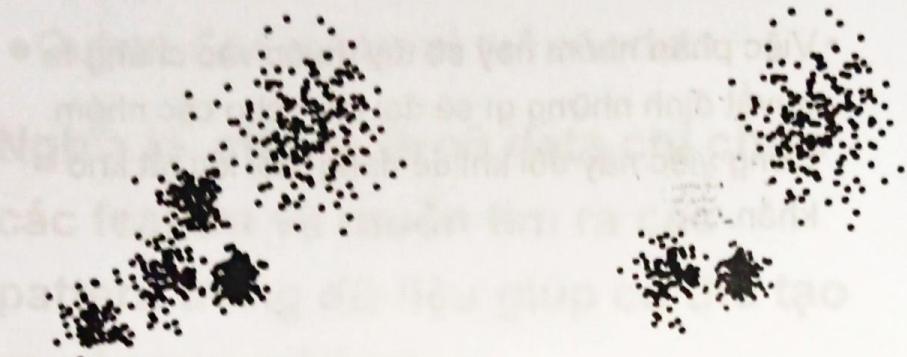
## Clustering K-Means

- K-Means

- K Means Clustering là một thuật toán thuộc nhóm unsupervised learning, cố gắng nhóm các cụm tương tự lại với nhau dựa trên dữ liệu.
- Vì vậy, một vấn đề phân cụm điển hình là:
  - Cụm các tài liệu tương tự
  - Cụm khách hàng dựa trên các thuộc tính
  - Phân khúc thị trường
  - Xác định các nhóm (vật lý) tương tự

## Clustering K-Means

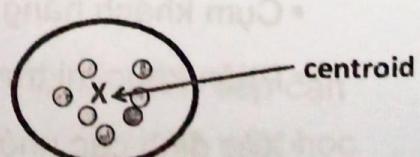
- Mục tiêu tổng thể là phân chia dữ liệu thành các nhóm riêng biệt sao cho các đối tượng trong mỗi nhóm tương tự nhau.



## Clustering K-Means

### ❑ K-Means Algorithm

- Lựa chọn k centroid ban đầu (cluster center)
- Lặp:
  - Gán từng mẫu với centroid gần nhất
  - Tính toán trung bình của các cluster để xác định centroid mới
- Cho đến khi:
  - Đạt được tiêu chí dừng



# Clustering K-Means

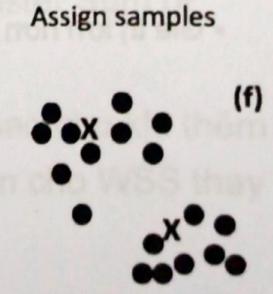
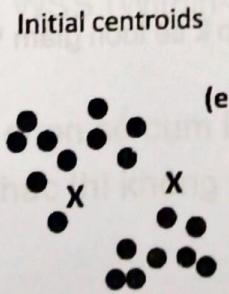
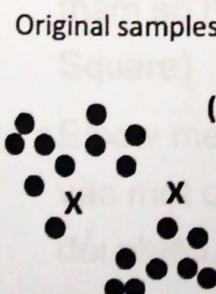
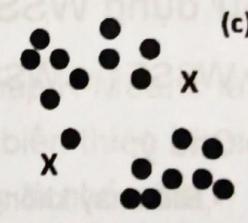
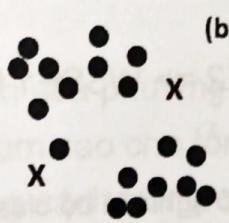
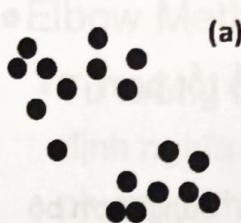


## • Tiêu chí dừng

- Việc lặp sẽ dừng khi không có sự thay đổi các centroid
- Số lượng các mẫu thay đổi cluster dưới ngưỡng (threshold)

Big Data in Machine Learning

# Clustering K-Means



Big Data in Machine Learning

## Clustering K-Means

- Tính toán kết quả cluster

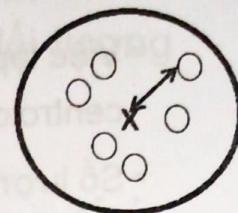
- error = khoảng cách giữa sample và centroid

- squared error =  $\text{error}^2$

- Tính tổng các squared error giữa tất cả các sample và centroid

=> Tính tổng trên tất cả các cluster => WSSE

Within-Cluster Sum of Squared Error



## Clustering K-Means

- Sử dụng WSSE

- WSSE1 < WSSE2 => WSSE1 là số tốt hơn

- Chú ý:

- Điều này không có nghĩa là bộ cluster 1 “đúng” hơn bộ cluster 2

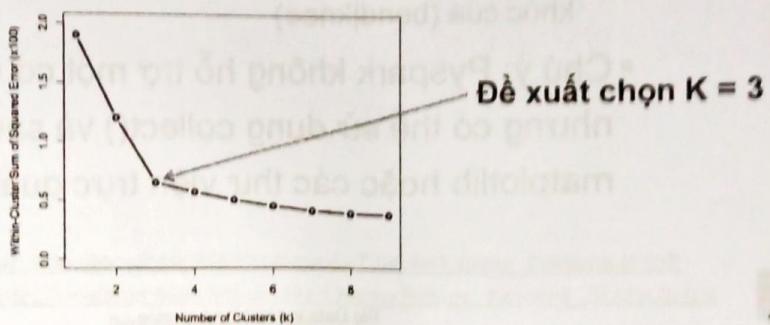
- Giá trị lớn hơn cho k sẽ luôn giảm WSS



# Clustering K-Means

## • Chọn K

- Không có câu trả lời dễ dàng cho việc chọn giá trị tốt nhất của K
- Một cách là dùng phương pháp khuỷu tay (elbow method)



13

# Clustering K-Means

## • Elbow Method

- Tư tưởng chính của phương pháp K-Means là định nghĩa 1 cụm sao cho tổng biến thiên bình phương khoảng cách trong cụm là nhỏ nhất, tham số này là WSS (Within-cluster Sum of Square)
- Elbow method chọn số cụm K sao cho khi thêm vào một cụm khác thì không làm cho WSS thay đổi nhiều.



## Clustering K-Means

### • Elbow Method

- Qui trình triển khai Elbow method như sau:
  - Triển khai thuật toán K-Means với các số cụm K thay đổi (ví dụ từ 2 đến 10)
  - Với mỗi giá trị K, tính giá trị WSS
  - Vẽ Elbow curve theo các giá trị K.
  - Dựa vào Elbow curve chọn số K thích hợp, là vị trí ở khúc cua (bend/knee)
- Chú ý: PySpark không hỗ trợ một cơ chế vẽ, nhưng có thể sử dụng collect() và sau đó vẽ bằng matplotlib hoặc các thư viện trực quan khác.



## Clustering K-Means

- Nhưng không nên coi đây là một quy tắc nghiêm ngặt khi chọn giá trị K. Việc phân nhóm còn phụ thuộc vào bối cảnh của tình huống chính xác (domain knowledge)



## Clustering K-Means

### ☐ Triển khai K-Means

- pyspark.ml.classification.clustering. KMeans

Without Scaler: [http://localhost:8888/notebooks/Chapter9/demo\\_Kmeans.ipynb](http://localhost:8888/notebooks/Chapter9/demo_Kmeans.ipynb)  
With Scaler: [http://localhost:8888/notebooks/Chapter9/demo\\_Kmeans\\_Scaler.ipynb](http://localhost:8888/notebooks/Chapter9/demo_Kmeans_Scaler.ipynb)

Big Data in Machine Learning

17

## Clustering K-Means

- Ví dụ: Thực hiện bài toán phân cụm với dữ liệu 5000\_points.txt. Với số cụm dao động từ 2 – 10.

- Đọc dữ liệu

```
# Loads data.
data = spark.read.csv("5000_points.txt", header=False,
inferSchema=True, sep="\t")
```

```
data.show(3)
```

	_c0	_c1
	664159	550946
	665845	557965
	597173	575538

```
+-----+-----+
| _c0| _c1|
+-----+
| 664159|550946|
| 665845|557965|
| 597173|575538|
+-----+
```

```
only showing top 3 rows
```

Big Data in Machine Learning

18

## Clustering K-Means

- Chuẩn hóa dữ liệu (nếu cần)

```
from pyspark.sql.functions import col  
data = data.select(col("_c0").alias("x"), col("_c1").alias("y"))  
  
data.show(3)  
  
+---+---+  
| x | y |  
+---+---+  
| 664159 | 550946 |  
| 665845 | 557965 |  
| 597173 | 575538 |  
+---+---+  
only showing top 3 rows
```



## Clustering K-Means

- Chuyển đổi dữ liệu

```
from pyspark.ml.linalg import Vectors  
from pyspark.ml.feature import VectorAssembler  
  
data.columns  
['x', 'y']  
  
vecAssembler = VectorAssembler(inputCols = data.columns,  
                                outputCol='features')  
  
final_data = vecAssembler.transform(data)
```



# Clustering K-Means



## ▪ Scale dữ liệu (nêu cần)

```
from pyspark.ml.feature import StandardScaler

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures",
                        withStd=True, withMean=False)

# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(final_data)

# Normalize each feature to have unit standard deviation.
final_data = scalerModel.transform(final_data)

final_data.show(3, False)
```

x	y	features	scaledFeatures
664159	550946	[664159.0,550946.0]	[2.716775618700365, 2.336092297255214]
665845	557965	[665845.0,557965.0]	[2.7236722780340017, 2.3658538924649704]
597173	575538	[597173.0,575538.0]	[2.4427660270639544, 2.440366004250274]

only showing top 3 rows

Big Data in Machine Learning

21

# Clustering K-Means



## ▪ Xây dựng model

- Chọn k thích hợp (trong khoảng từ 2-10)

```
from pyspark.ml.clustering import KMeans
```

```
# Trains a k-means model.
k_list = []
wssse_list = []
for k in range(2,11):
    # kmeans = KMeans(featuresCol='features',k=k)
    kmeans = KMeans(featuresCol='scaledFeatures',k=k)
    model = kmeans.fit(final_data)
    wssse = model.computeCost(final_data)
    k_list.append(k)
    wssse_list.append(wssse)
print("With k =", k, "Set Sum of Squared Errors = " + str(wssse))
```

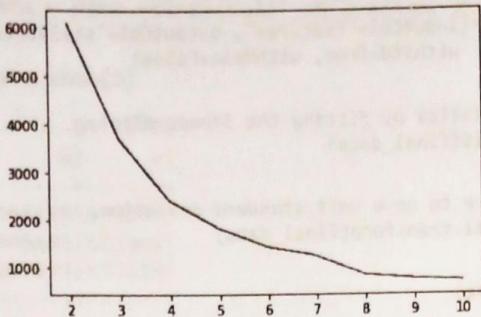
Big Data in Machine Learning

22

# Clustering K-Means

## ▪ Xây dựng model (tt)

```
plt.plot(k_list, wssse_list)
plt.show()
```



```
With k = 2 Set Sum of Squared Errors = 6086.71908212722
With k = 3 Set Sum of Squared Errors = 3706.746994293558
With k = 4 Set Sum of Squared Errors = 2406.331723769621
With k = 5 Set Sum of Squared Errors = 1817.0452245537108
With k = 6 Set Sum of Squared Errors = 1431.651801147981
With k = 7 Set Sum of Squared Errors = 1239.5344779608115
With k = 8 Set Sum of Squared Errors = 844.1587813075814
With k = 9 Set Sum of Squared Errors = 758.2190969540461
With k = 10 Set Sum of Squared Errors = 716.2779344197979
```

Big Data in Machine Learning

23



# Clustering K-Means

## ▪ Xây dựng model (tt)

```
# Trains a k-means model.
```

```
kmeans = KMeans(featuresCol='scaledFeatures', k=10)
model = kmeans.fit(final_data)
```

```
# Evaluate clustering by computing Within Set Sum of Squared Errors.
```

```
wssse = model.computeCost(final_data)
print("Within Set Sum of Squared Errors = " + str(wssse))
```

```
Within Set Sum of Squared Errors = 716.2779344197979
```

```
# Shows the result.
```

```
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Cluster Centers:

```
[3.48538772 0.66860987]
[0.96278248 2.3742748 ]
[2.47672914 2.43665775]
[1.70360373 0.71250277]
[3.28321464 1.37765613]
[2.48029822 1.6930302 ]
[1.33679274 3.4704405 ]
[2.77092183 3.63555263]
[3.45308435 2.676229 ]
[1.14099447 1.58997597]
```



Big Data in Machine Learning

24

# Clustering K-Means

## ▪ Dự đoán kết quả

```
predictions = model.transform(final_data)
```

```
predictions.select("prediction").show(5)
```

```
+---+-----+
| prediction |
+---+-----+
| 2 |
| 2 |
| 2 |
| 2 |
| 2 |
+---+-----+
only showing top 5 rows
```

predictions.groupBy('prediction').count().show()	
prediction	count
1	640
6	358
3	667
5	341
9	349
4	349
8	650
7	397
2	320
0	929

Big Data in Machine Learning

25

# Clustering K-Means

## ▪ Dự đoán kết quả (tt)

```
final_data.show(3, False)
```

```
+---+-----+
|x |y |features |scaledFeatures |
+---+-----+
|664159|550946|[664159.0,550946.0]|[[2.716775610700365,2.336092297255214]|
|665845|557965|[665845.0,557965.0]|[[2.7236722780340017,2.3658538924649704]|
|597173|575538|[597173.0,575538.0]|[[2.4427660270639544,2.440366004250274]|
+---+-----+
only showing top 3 rows
```

```
temp = final_data.select("scaledFeatures").rdd.map(lambda x: x[0].toArray().tolist()).toDF()
```

```
temp.show(3)
```

```
+---+-----+
| _1 | _2 |
+---+-----+
| 2.716775610700365| 2.336092297255214|
| 2.7236722780340017| 2.3658538924649704|
| 2.4427660270639544| 2.440366004250274|
+---+-----+
only showing top 3 rows
```

Big Data in Machine Learning

26

## Clustering K-Means

### ▪ Dự đoán kết quả (tt)

```
temp=temp.withColumn('row_index', f.monotonically_increasing_id())
data_result=data_result.withColumn('row_index', f.monotonically_increasing_id())
temp = temp.join(data_result, on=["row_index"]).sort("row_index").drop("row_index")

temp.show(3)

+-----+-----+-----+
|      _1|      _2|prediction|
+-----+-----+-----+
| 2.716775610700365| 2.336092297255214|      2|
| 2.7236722780340017| 2.3658538924649704|      2|
| 2.4427660270639544| 2.440366004250274|      2|
+-----+-----+-----+
only showing top 3 rows

temp = temp.select(col("_1").alias("x_scale"), col("_2").alias("y_scale"), "prediction")

df = temp.toPandas()
```



## Clustering K-Means

### ▪ Trực quan hóa dữ liệu

```
df = temp.toPandas()
```

```
centers_df = pd.DataFrame(centers)
centers_df.head()
```

	0	1
0	3.485388	0.668610
1	0.962782	2.374275
2	2.476729	2.436658
3	1.703604	0.712503
4	3.283215	1.377656



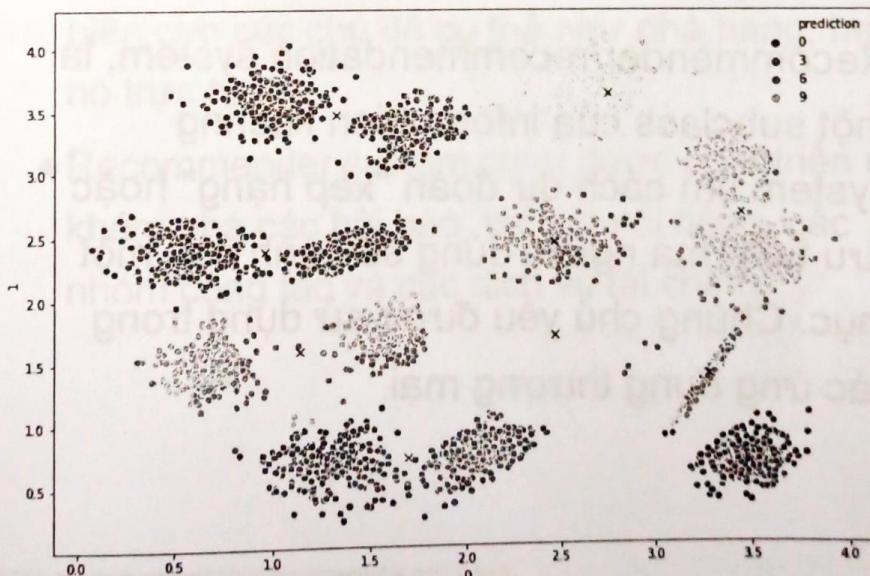
# Clustering K-Means

- Trực quan hóa dữ liệu (tt)

```
plt.figure(figsize=(12,8))
sns.scatterplot(x="x_scale", y="y_scale", data = df,
                 hue="prediction",
                 # size = "prediction",
                 palette="Set1")
sns.scatterplot(data = centers_df, x=0, y=1, color="black", marker="x")
plt.show()
```

# Clustering K-Means

- Trực quan hóa dữ liệu (tt)





## Nội dung

1. Clustering K-Means
2. Recommender System – ALS
3. Association rules - FP-Growth



## Recommender System

### □ Giới thiệu

- Recommender/recommendation system, là một subclass của information filtering system tìm cách dự đoán "xếp hạng" hoặc "ưu tiên" mà người dùng sẽ dành cho một mục. Chúng chủ yếu được sử dụng trong các ứng dụng thương mại.



## Recommender System

- Recommender systems được sử dụng trong nhiều lĩnh vực: tạo danh sách phát nhạc/video cho các dịch vụ như Netflix, YouTube & Spotify, đề xuất sản phẩm cho các dịch vụ như Amazon, đề xuất nội dung cho các nền tảng truyền thông xã hội (social media platform) Facebook & Twitter. Những system có thể hoạt động bằng cách sử dụng một single input (như music), hay multiple input trong và trên các nền tảng như news, books,... và truy vấn tìm kiếm (search query).

Big Data in Machine Learning

33



## Recommender System

- Ngoài ra, còn có các recommender system phổ biến cho các chủ đề cụ thể như nhà hàng, hẹn hò trực tuyến...
- Recommender system cũng được phát triển để khám phá các bài báo, tác giả nổi tiếng, các nhóm cộng tác và các dịch vụ tài chính.

## Recommender System

### ☐ Một cách tổng quát

- Recommender system là các thuật toán nhằm đề xuất các item có liên quan cho người dùng (Item có thể là phim để xem, văn bản để đọc, sản phẩm cần mua hoặc bất kỳ thứ gì khác tùy thuộc vào ngành).
- Recommender system thực sự quan trọng trong một số lĩnh vực vì chúng có thể tạo ra một khoản thu nhập khổng lồ hoặc cũng là một cách để nổi bật đáng kể so với các đối thủ cạnh tranh.

Big Data in Machine Learning

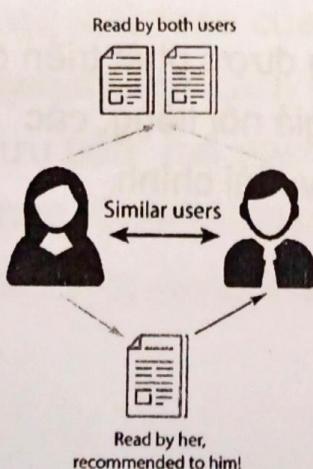
35

## Recommender System

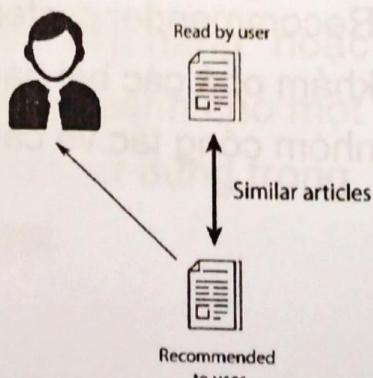


### ☐ Có hai recommender system phổ biến nhất là Collaborative Filtering (CF) và Content-Based

COLLABORATIVE FILTERING



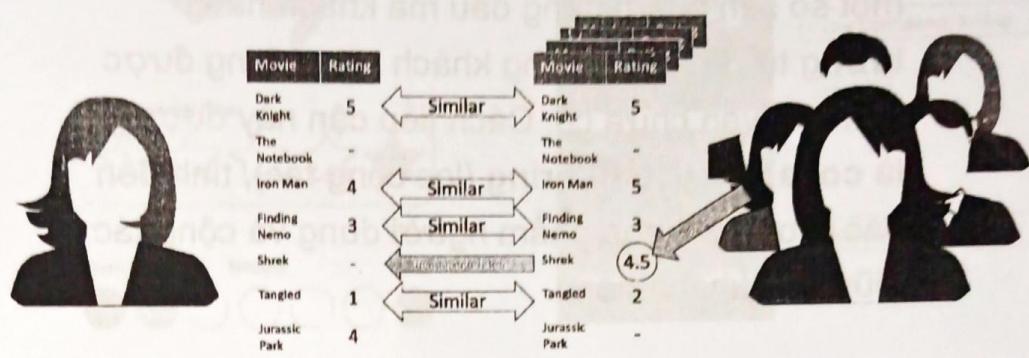
CONTENT-BASED FILTERING



# Recommender System



- Collaborative filtering tạo ra các đề xuất dựa trên kiến thức của người dùng về thái độ đối với các item, nó sử dụng kiến thức của số đông ("wisdom of the crowd") để đề xuất các item.



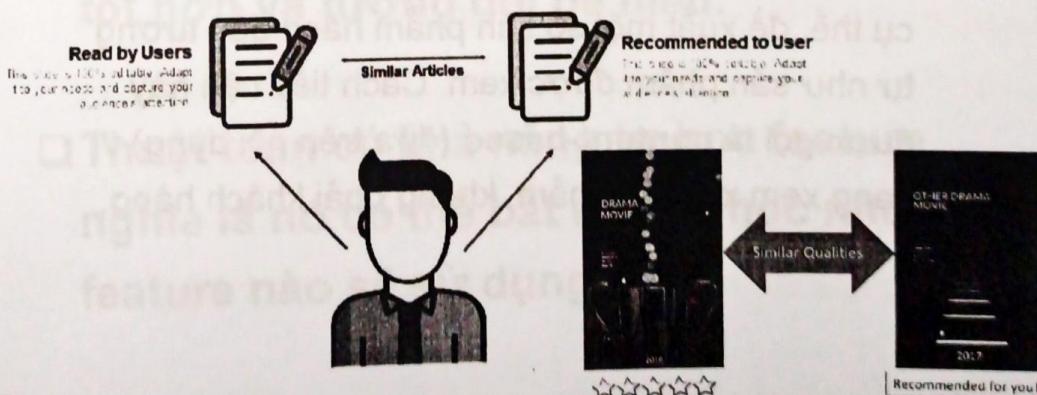
Big Data in Machine Learning

37

# Recommender System



- Content-based recommender system tập trung vào các thuộc tính của item và cung cấp cho người dùng các đề xuất dựa trên sự tương tự giữa chúng.



Big Data in Machine Learning

38

## Recommender System

### • Ví dụ:

- Xem xét lịch sử giao dịch của khách hàng. Đối với một khách hàng cụ thể, hãy tìm những khách hàng có lịch sử giao dịch tương tự và giới thiệu một số sản phẩm hàng đầu mà khách hàng tương tự đã mua nhưng khách hàng đang được xem xét vẫn chưa có. Cách tiếp cận này được gọi là **collaborative filtering** (lọc cộng tác), tính đến các tương tác sản phẩm người dùng và cộng tác giữa các khách hàng.



## Recommender System

### • Ví dụ:

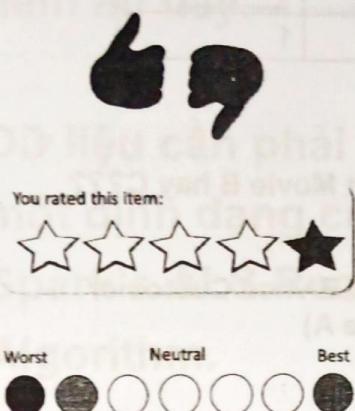
- Xem xét bộ sản phẩm và sử dụng thông tin sản phẩm để tìm sản phẩm tương tự. Do đó, khi người dùng mở một liên kết đến một sản phẩm cụ thể, đề xuất một số sản phẩm hàng đầu tương tự như sản phẩm được xem. Cách tiếp cận này được gọi là **content-based** (dựa trên nội dung) vì đang xem xét sản phẩm, không phải khách hàng.



## Recommender System

- Rating: có 2 loại là explicit và implicit

### EXPLICIT RATINGS



### IMPLICIT RATINGS



## Recommender System

- Nói chung, Collaborative filtering (CF) được sử dụng nhiều hơn các content-based system vì nó thường cho kết quả tốt hơn và tương đối dễ hiểu.

- Thuật toán có khả năng tự học feature, nghĩa là nó có thể bắt đầu tự học những feature nào sẽ sử dụng.

## Recommender System

- Ví dụ:

	Movie A	Movie B	Movie C
Customer 1	5	5	1
Customer 2	5	5	1

	Movie A
Customer 3	5

Đề xuất Movie B hay C???

=> Đề xuất Customer 3 chọn Movie B vì cả Customer 1 và 2 đều chọn movie B (và movie A)



## Recommender System

- ❑ Những kỹ thuật này nhằm mục đích điền vào các mục còn thiếu của ma trận kết hợp user-item (user-item association matrix).
- ❑ spark.ml hiện hỗ trợ model cho collaborative filtering, trong đó người dùng và sản phẩm được mô tả bằng một nhóm nhỏ các yếu tố tiềm ẩn (latent factor) có thể được sử dụng để dự đoán các mục bị thiếu.



## Recommender System

- ❑ spark.ml sử dụng thuật toán alternating least squares (ALS) để học các yếu tố tiềm ẩn này.
- ❑ Dữ liệu cần phải được chuyển thành một định dạng cụ thể làm việc với Spark's ALS Recommendation Algorithm.

## Recommender System

- ❑ ALS về cơ bản là một cách tiếp cận Matrix Factorization để thực hiện recommendation algorithm mà chúng ta phân tách ma trận user/item lớn thành các yếu tố user và yếu tố item có chiều thấp hơn.

# Recommender System

## ❑ ALS

- Diễn giải với bài toán Users-Products

- Matrix  $R_{m \times n}$

- m: số lượng user
- n: số lượng product

- Phân rã thành  $P_{m \times k}$  và  $Q_{k \times n}$  sao cho

- Với  $i$ \_th product và  $u$ \_th user có thể dự đoán product rating  $r_{iu} = q_i^T p_u$  => đưa ra đề xuất.
- Có nghĩa là mỗi user và mỗi product được mô tả bằng vectors của các latent variable là  $p$  và  $q$ , mỗi biến có chiều dài là  $k$ . Và ma trận  $P$  và  $Q$  lưu trữ các latent variable (là unobservable variable, ví dụ: một số rating đã biết và một số rating không biết)



<https://datascience.stackexchange.com/questions/22417/what-is-the-nature-in-collaborative-filtering>

Big Data in Machine Learning

47



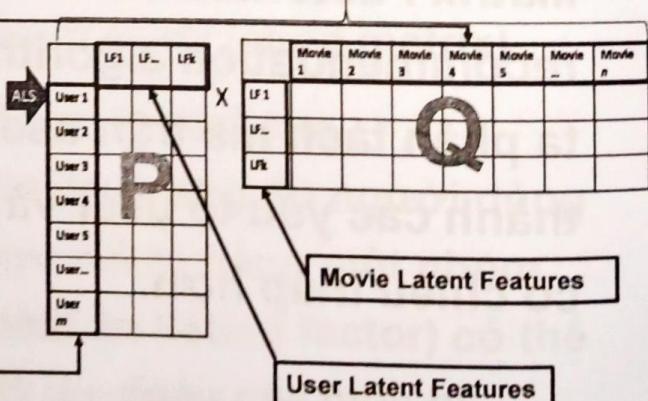
# Recommender System

## ❑ ALS

Original Ratings Matrix

	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5	Movie -	Movie n
User 1							
User 2							
User 3							
User 4							
User 5							
User -							
User n							

Factor Matrices



Big Data in Machine Learning

48

# Recommender System



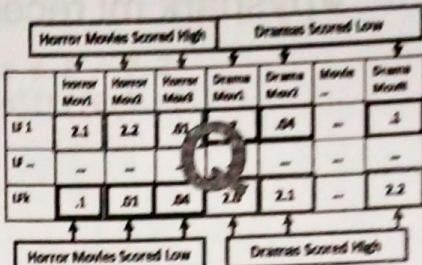
## ALS

Original Ratings Matrix

	Horror Mov1	Horror Mov2	Horror Mov3	Drama Mov1	Drama Mov2	Movi... ...	Drama MovN
User 1	5	5	1	1	1	—	1
User 2	4	4	1	1	1	—	1
User 3	4	5	1	1	1	—	1
User 4	2	2	1	1	1	—	1
User 5	1	1	1	5	5	—	5
User...	...	...	...	...	...	—	—
User m	1	1	1	4	5	—	5

ALS

	LF 1	LF ...	LF k
User 1	2.2	—	0.0
User 2	1.9	—	0.0
User 3	2.1	—	0.1
User 4	—	—	0.1
User 5	0.1	—	2.3
User...	—	—	—
User m	0.0	—	2.2



Big Data in Machine Learning

49

# Recommender System



## ALS

Original Matrix

	Movie 1	Movie 2	Movie 3	Movie ...	Movie n
User 1	5	4.8	4.9	...	4.9
User 2	4.5	4.3	4.4	...	4.3
User 3	4.8	4.6	4.7	...	4.6
User 4	2.1	1.98	2	...	2
User 5	4.1	4	4	...	4
User...	...	...	...	...	—
User m	4.6	4.4	4.5	...	4.4
Avg Rating	4.2	4.0	4.1	...	4

ALS

	LF 1	LF ...	LF k		Movie 1	Movie 2	Movie 3	Movie ...	Movie n
LF 1	2.3	—	0.0		2.3	2.2	2.3	—	2.2
LF ...	—	—	—		—	—	—	—	—
LF k	0.1	0.01	0.02		—	—	—	—	0.02
User 1	2.2	—	0.0		—	—	—	—	—
User 2	1.95	—	0.0		—	—	—	—	—
User 3	2.1	—	0.01		—	—	—	—	—
User 4	0.90	—	0.01		—	—	—	—	—
User 5	1.8	—	0.03		—	—	—	—	—
User...	—	—	—		—	—	—	—	—
User m	2.0	—	0.02		—	—	—	—	—

Big Data in Machine Learning

50

# Recommender System

## ☐ Triển khai ALS

- pyspark.ml.recommendation.ALS

[http://localhost:8888/notebooks/Chapter3/demo\\_Recommender.ipynb](http://localhost:8888/notebooks/Chapter3/demo_Recommender.ipynb)

Big Data in Machine Learning

51



# Recommender System

- Ví dụ: Movies Rating

- Đọc dữ liệu

```
data = spark.read.csv('ml-latest-small/ratings.csv', inferSchema=True, header=True)

data.show(5, False)

+-----+-----+-----+
|movieId|rating|userId|
+-----+-----+-----+
|2      |3.0   |0     |
|3      |1.0   |0     |
|5      |2.0   |0     |
|9      |4.0   |0     |
|11     |1.0   |0     |
+-----+-----+-----+
only showing top 5 rows

# Distinct users and movies
users = data.select("userId").distinct().count()
movies = data.select("movieId").distinct().count()
numerator = data.count()

display(numerator, users, movies)
1501
30
100
```

## Recommender System

- Chuẩn hóa dữ liệu, chuyển đổi dữ liệu (nếu cần)
- Chia dữ liệu train-test

```
# Smaller dataset so we will use 0.8 / 0.2
(training, test) = data.randomSplit([0.8, 0.2])
```

## Recommender System

- Xây dựng model

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS

als = ALS(maxIter=5, regParam=0.01,
           userCol="userId",
           itemCol="movieId",
           ratingCol="rating")
model = als.fit(training)
```

# Recommender System

## Đánh giá kết quả

# Evaluate the model by computing the RMSE on the test data  
 predictions = model.transform(test)

```
predictions.show(5)
```

movieId	rating	userId	prediction
31	1.0	5	2.424184
31	3.0	7	1.7328327
31	2.0	25	1.9922988
31	3.0	14	-0.49735457
85	1.0	23	5.652081

```
only showing top 5 rows
evaluator = RegressionEvaluator(metricName="rmse",
                                 labelCol="rating",
                                 predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
Root-mean-square error = 1.6706362678979425
```

#On average, this model is ~ 1.7 from perfect recommendations.



# Recommender System

## Dự đoán

```
userId = 27
single_user = test.filter(test['userId']==userId).select(['movieId','userId'])

reccomendations = model.transform(single_user)
single_user.show()

# prediction
reccomendations.orderBy('prediction',ascending=False).show()

+-----+-----+
|movieId|userId|
+-----+-----+
|      |      | |
| 90| 27| 4.205997|
| 94| 27| 3.3630939|
| 98| 27| 1.9856262|
| 55| 27| 1.8030219|
| 40| 27| 1.730906|
| 66| 27| 1.7132561|
| 9| 27| 1.7111404|
| 92| 27| 1.6712735|
| 24| 27| 1.5636977|
| 78| 27| 1.5062279|
| 64| 27| 1.1982841|
| 59| 27| 0.87414837|
| 72| 27| 0.18111625|
+-----+-----+
```

movieId	userId
9	27
24	27
40	27
55	27
59	27
64	27
66	27
70	27
72	27
90	27
92	27
94	27
98	27

# Recommender System

## ▪ Đưa ra đề xuất cho tất cả user

```
# Get 20 recommendations which have highest rating.
user_recs = model.recommendForAllUsers(20)
```

```
for user in user_recs.head(2):
    print(user)
    print("\n")
Row(userId=28, recommendations=[Row(movieId=92, rating=5.118626117706299), Row(movieId=81,
rating=4.832043170928955), Row(movieId=12, rating=4.752755165100098), Row(movieId=53,
rating=4.616611003875732), Row(movieId=89, rating=4.304506301879883), Row(movieId=74,
rating=4.064364910125732), Row(movieId=2, rating=3.830737352371216), Row(movieId=49,
rating=3.7942352294921875), Row(movieId=82, rating=3.71313214302063), Row(movieId=30,
rating=3.5869648456573486), Row(movieId=4, rating=3.494983196258545), Row(movieId=68,
rating=3.4522364139556885), Row(movieId=24, rating=3.1765313148498535), Row(movieId=23,
rating=3.1444807052612305), Row(movieId=51, rating=3.142695426940918), Row(movieId=77,
rating=2.927616596221924), Row(movieId=62, rating=2.917660713195801), Row(movieId=22,
rating=2.882833957672119), Row(movieId=0, rating=2.7797999382019043), Row(movieId=9,
rating=2.740954637527466)], Row(userId=26, recommendations=[Row(movieId=32,
rating=6.711236953735352), Row(movieId=30, rating=5.319496154785156), Row(movieId=94,
rating=5.201180458068848), Row(movieId=98, rating=5.186388969421387), Row(movieId=22,
rating=5.1540913581848145), Row(movieId=74, rating=5.077207088470459), Row(movieId=7,
rating=4.8612518310546875), Row(movieId=75, rating=4.7022175788879395), Row(movieId=88,
rating=4.6973724365234375), Row(movieId=23, rating=4.655683517456055), Row(movieId=90,
rating=4.458372116088867), Row(movieId=54, rating=4.276267051696777), Row(movieId=68,
rating=4.1668806076049805), Row(movieId=73, rating=3.908524751663208), Row(movieId=18,
rating=3.8087682723999023), Row(movieId=77, rating=3.8087258338928223), Row(movieId=53,
rating=3.7034876346588135), Row(movieId=10, rating=3.54668927192688), Row(movieId=87,
rating=3.53151798248291), Row(movieId=69, rating=3.2649927139282227)])]
```

57

# Recommender System

## ▪ Đưa ra đề xuất cho một user cụ thể

```
userId = 27
user_recs.filter(user_recs["userId"] == 27).show(truncate=False)
```

userId	recommendations
27	[[[38, 3.841758], [18, 3.8377333], [80, 3.7730618], [65, 3.2105079], [66, 3.184661], [19, 3.1673741], [51, 3.0379815], [27, 3.0274985], [55, 2.5979254], [81, 2.5484018], [11, 2.390921], [85, 2.389595], [32, 2.3870957], [33, 2.3860118], [94, 2.3158324], [83, 2.2594254], [30, 2.255559], [35, 1.9348334], [87, 1.9159526], [2, 1.8787222]]]

• Thuộc nhóm Association rule learning



## Nội dung

1. Clustering K-Means
2. Recommender System – ALS
3. Association rules - FP-Growth



## Association rules

### ❑ Association rule learning

- Là một phương pháp Machine Learning dựa trên quy tắc để khám phá mối quan hệ thú vị giữa các biến trong cơ sở dữ liệu lớn. Nó dùng để xác định các quy tắc mạnh được phát hiện trong cơ sở dữ liệu bằng cách sử dụng một số thuật toán thú vị.



❑ Market Basket Analysis, association rules mining, là một phương pháp rất hữu ích để hiểu rõ hơn về các bộ dữ liệu giao dịch (transactional data set) và đề xuất sản phẩm.

- Ví dụ: Dữ liệu giao dịch trong một siêu thị. Đối với mỗi khách hàng, chúng ta biết những sản phẩm riêng lẻ mà khách hàng đã mua. Với association rules mining, chúng ta có thể xác định các mặt hàng thường được mua cùng nhau.

## Association rules

❑ Frequent Pattern Mining (FPM)

- Khai thác các item, itemset, subsequence, hoặc substructure thường xuyên xuất hiện là một trong những bước đầu tiên để phân tích một tập dữ liệu quy mô lớn, vốn là một chủ đề nghiên cứu tích cực trong khai thác dữ liệu.

- Thuộc nhóm Association rule learning

## Association rules

### ❑ FP-growth algorithm (FP: frequent pattern)

- Là thuật toán trong nhóm FPM dùng để phân tích luật kết hợp.
- Thuật toán
  - Cung cấp dữ liệu transactions
  - Bước đầu tiên là tính toán tần số xuất hiện của các item và xác định các frequent item.
  - Bước thứ hai sử dụng suffix tree (FP-tree) để mã hóa các giao dịch mà không tạo ra các bộ ứng cử viên rõ ràng.
  - Sau bước thứ hai, các frequent itemset có thể được trích xuất từ FP-tree

Big Data in Machine Learning

63

## Association rules

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

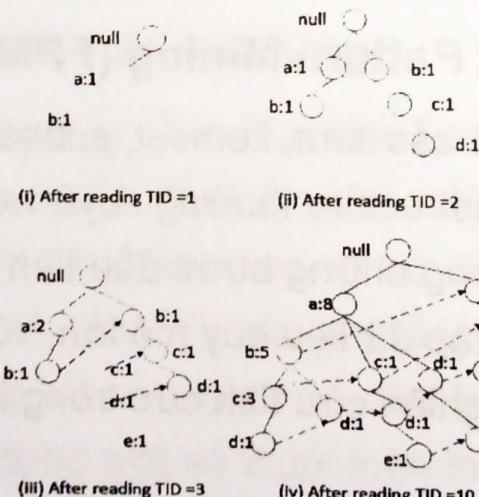


Fig. 7. Construction of an FP-tree - Based on [28]

<http://liberians.com/2014/07/fpgrowth.html>

Big Data in Machine Learning

64

# Association rules

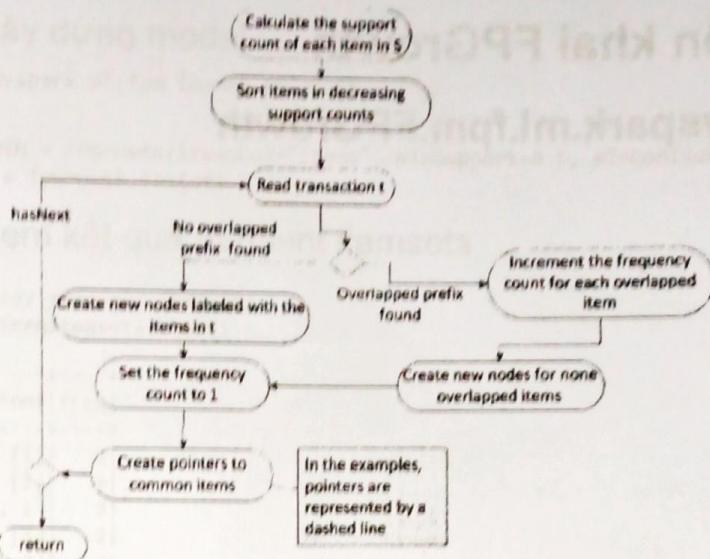


Fig. 8. Activity diagram - Construction of the FP-Tree - Notation: [27]

Big Data in Machine Learning

65

# Association rules

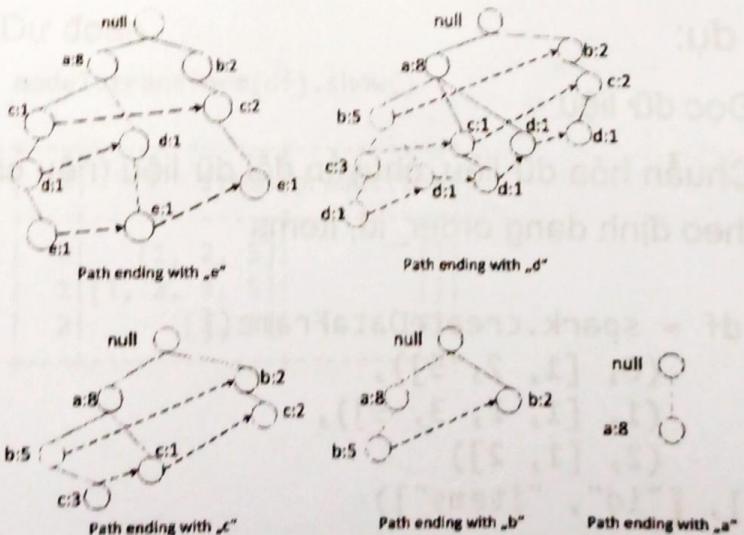


Fig. 9. Subproblems - Based on [28]

Big Data in Machine Learning

66

## Association rules

### ☐ Triển khai FP-Growth

- pyspark.ml.fpm.FPGrowth



[http://localhost:4226/notebooks/Chapter04/demo\\_FPM.ipynb](http://localhost:4226/notebooks/Chapter04/demo_FPM.ipynb)

Big Data in Machine Learning

67

## Association rules

- Ví dụ:

- Đọc dữ liệu
- Chuẩn hóa dữ liệu, chuyển đổi dữ liệu (nếu cần) theo định dạng order\_id, items

```
df = spark.createDataFrame([
    (0, [1, 2, 5]),
    (1, [1, 2, 3, 5]),
    (2, [1, 2])
], ["id", "items"])
```



Big Data in Machine Learning

68

# Association rules

## ▪ Xây dựng model

```
from pyspark.ml.fpm import FPGrowth
```

```
fpGrowth = FPGrowth(itemsCol="Items", minSupport=0.5, minConfidence=0.6)
model = fpGrowth.fit(df)
```

## ▪ Xem kết quả frequent itemsets

```
# Display frequent itemsets.
model.freqItemsets.show()
```

items	freq
[1]	3
[2]	3
[2, 1]	3
[5]	2
[5, 2]	2
[5, 2, 1]	2
[5, 1]	2

Big Data in Machine Learning

69

# Association rules

## ▪ Dự đoán

```
model.transform(df).show()
```

id	items	prediction
0	[1, 2, 5]	[]
1	[1, 2, 3, 5]	[]
2	[1, 2]	[5]

Big Data in Machine Learning

70



## Chapter 9: K-Means Clustering

### Ex1: 3D points

**Dataset:** data3D.csv.

#### Requirement:

- Read dataset
- Pre-process data
- Use K-means clustering algorithm to cluster 3D points in data3D.csv.

```
In [8]: import findspark
findspark.init()
```

```
In [9]: import pyspark
```

```
[10]: from pyspark.sql import SparkSession
```

```
[11]: spark = SparkSession.builder.appName('kmeans_3D_point').getOrCreate()
```

```
In [12]: # Loads data.
data = spark.read.csv("data3D.csv", header=True,
inferSchema=True)
```

```
In [13]: data.show(3)
```

id	x	y	z
point0	5.647627534046943	-6.356222340123802	-7.240816026826695
point1	4.414367138680041	-10.32624175635328	8.963324308916228
point2	5.005396944639823	-9.301070062115645	10.35473056351597

only showing top 3 rows

```
In [14]: from pyspark.sql.functions import col
```

```
In [17]: data = data.select(['x', 'y', 'z'])
```



```
In [18]: data.show(3)
```

	x	y	z
5.647627534046943	-6.356222340123802	-7.240816026826695	
4.414367138680041	-10.32624175635328	8.963324308916228	
5.005396944639823	-9.301070062115645	10.35473056351597	

only showing top 3 rows

## Format from data

```
In [19]: from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

```
In [20]: data.columns
```

```
Out[20]: ['x', 'y', 'z']
```

```
In [21]: vecAssembler = VectorAssembler(inputCols = data.columns,
                                         outputCol='features')
```

```
In [25]: final_data = vecAssembler.transform(data)
```

## Scale the Data

```
In [22]: from pyspark.ml.feature import StandardScaler
```

```
In [23]: scaler = StandardScaler(inputCol="features",
                               outputCol="scaledFeatures",
                               withStd=True,
                               withMean=False)
```

```
In [26]: # Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(final_data)
```

```
In [27]: # Normalize each feature to have unit standard deviation.
final_data = scalerModel.transform(final_data)
```



In [28]: `final_data.show(3, False)`

```
+-----+-----+-----+-----+
|x          |y          |z          |features
+-----+-----+-----+-----+
|5.647627534046943|-6.356222340123802|-7.240816026826695|[5.647627534046943,-6.
356222340123802,-7.240816026826695]|[[1.0159673512169811,-0.81335799160424,-1.13
00631023636807] |
|4.414367138680041|-10.32624175635328|8.963324308916228|[4.414367138680041,-1
0.32624175635328,8.963324308916228]|[[0.7941127247055396,-1.3213715327025133,1.
3988923401033817] |
|5.005396944639823|-9.301070062115645|10.35473056351597|[5.005396944639823,-9.
301070062115645,10.35473056351597]|[[0.9004347126254774,-1.1901880174546189,1.6
160469899240197] |
+-----+-----+-----+-----+
only showing top 3 rows
```

## Train the Model and Evaluate

### Select k with minimum WSSSE: k between 2 - 10

In [29]: `from pyspark.ml.clustering import KMeans`

```
In [31]: # Trains a k-means model.
k_list = []
wssse_list = []
for k in range(2,11):
    kmeans = KMeans(featuresCol='scaledFeatures',k=k)
    model = kmeans.fit(final_data)
    wssse = model.computeCost(final_data)
    k_list.append(k)
    wssse_list.append(wssse)
    print("With k =", k, "Set Sum of Squared Errors = " + str(wssse))
```

With k = 2 Set Sum of Squared Errors = 1155067.2563008904

With k = 3 Set Sum of Squared Errors = 297656.40920043044

With k = 4 Set Sum of Squared Errors = 146718.50451770602

With k = 5 Set Sum of Squared Errors = 72720.18504204818

With k = 6 Set Sum of Squared Errors = 68583.68208541229

With k = 7 Set Sum of Squared Errors = 64483.499921190276

With k = 8 Set Sum of Squared Errors = 60180.384358530224

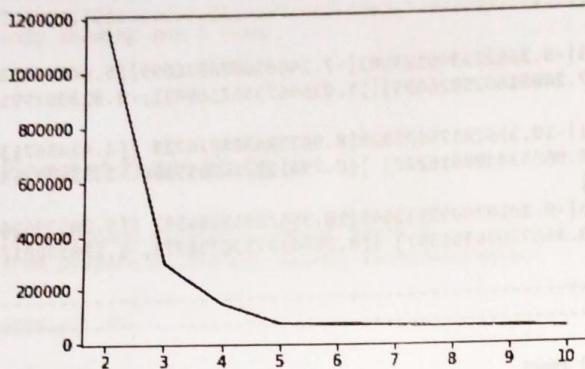
With k = 9 Set Sum of Squared Errors = 58364.71279295459

With k = 10 Set Sum of Squared Errors = 52251.866281251554



```
In [32]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [33]: plt.plot(k_list, wssse_list)
plt.show()
```



- According to Elbow Method, we choose  $k = 5$ . Look like there is very little gain after  $k=5$ , so we stick to that choice when processing the full data set.

### Select $k = 5$

```
In [34]: # Trains a k-means model.
kmeans = KMeans(featuresCol='scaledFeatures', k=5)
model = kmeans.fit(final_data)
```

```
In [35]: # Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(final_data)
print("Within Set Sum of Squared Errors = " + str(wssse))
```

Within Set Sum of Squared Errors = 72720.18504204816

```
In [36]: # Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)

Cluster Centers:
[-1.59030525  0.93719373  0.31581855]
[ 1.19780483 -0.7365009 -0.99420908]
[ 0.74896823 -1.2269015  1.46702482]
[-0.45182036  1.15367577  0.72369935]
[ 0.35333753 -0.87989287 -1.0735622 ]
```

3/6/2020

Ex1\_Kmeans\_Scaler\_3D\_points - Jupyter Notebook



In [37]: predictions = model.transform(final\_data)

In [38]: predictions.select("prediction").show(5)

```
+-----+  
|prediction|  
+-----+  
| 1|  
| 2|  
| 2|  
| 2|  
| 0|  
+-----+  
only showing top 5 rows
```

In [ ]: # Check number points of each cluster

In [60]: predictions.groupBy('prediction').count().show()

```
+-----+-----+  
|prediction| count|  
+-----+-----+  
| 1|199987|  
| 3|200017|  
| 4|200013|  
| 2|200000|  
| 0|199983|  
+-----+-----+
```

In [ ]: # Our clustering algorithm created 5 equally sized clusters with K=5

In [39]: data\_result = predictions.select("prediction")  
data\_result.columns

Out[39]: ['prediction']

In [40]: type(data\_result)

Out[40]: pyspark.sql.dataframe.DataFrame



In [41]: final\_data.show(3, False)

```
+-----+-----+-----+
| x | y | z | features |
+-----+-----+-----+
| scaledFeatures |
+-----+-----+-----+
| 5.647627534046943 | -6.356222340123802 | -7.240816026826695 | [5.647627534046943, -6.356222340123802, -7.240816026826695] | [1.0159673512169811, -0.81335799160424, -1.1300631023636807] |
| 4.414367138680041 | -10.32624175635328 | 8.963324308916228 | [4.414367138680041, -10.32624175635328, 8.963324308916228] | [0.7941127247055396, -1.3213715327025133, 1.3988923401033817] |
| 5.005396944639823 | -9.301070062115645 | 10.35473056351597 | [5.005396944639823, -9.301070062115645, 10.35473056351597] | [0.9004347126254774, -1.1901880174546189, 1.6160469899240197] |
+-----+-----+-----+
only showing top 3 rows
```

In [42]: temp = final\_data.select("scaledFeatures").rdd.map(lambda x: \x[0].toArray().tolist()).toDF()

In [43]: temp.show(3)

```
+-----+-----+-----+
| _1 | _2 | _3 |
+-----+-----+-----+
| 1.0159673512169811 | -0.81335799160424 | -1.1300631023636807 |
| 0.7941127247055396 | -1.3213715327025133 | 1.3988923401033817 |
| 0.9004347126254774 | -1.1901880174546189 | 1.6160469899240197 |
+-----+-----+-----+
only showing top 3 rows
```

In [44]: import pyspark.sql.functions as f

In [45]: # since there is no common column between these two dataframes add row\_index so
temp=temp.withColumn('row\_index', f.monotonically\_increasing\_id())
data\_result=data\_result.withColumn('row\_index',
f.monotonically\_increasing\_id())
temp = temp.join(data\_result,
on=['row\_index']).sort("row\_index").drop("row\_index")

3/16/2020

Ex1\_Kmeans\_Scaler\_3D\_points - Jupyter Notebook



In [46]: temp.show(3)

	_1	_2	_3 prediction
1	1.0159673512169811	-0.81335799160424	-1.1300631023636807
0	0.7941127247055396	-1.3213715327025133	1.3988923401033817
0	0.9004347126254774	-1.1901880174546189	1.6160469899240197

only showing top 3 rows

In [48]: temp = temp.select(col("\_1").alias("x\_scale"),  
 col("\_2").alias("y\_scale"),  
 col("\_3").alias("z\_scale"),  
 "prediction")

In [49]: df = temp.toPandas()

In [54]: df.head(3)

Out[54]:

	x_scale	y_scale	z_scale	prediction
0	1.015967	-0.813358	-1.130063	1
1	0.794113	-1.321372	1.398892	2
2	0.900435	-1.190188	1.616047	2

In [50]: centers\_df = pd.DataFrame(centers)  
centers\_df.head()

Out[50]:

	0	1	2
0	-1.590305	0.937194	0.315819
1	1.197805	-0.736501	-0.994209
2	0.748968	-1.226902	1.467025
3	-0.451820	1.153676	0.723699
4	0.353338	-0.879893	-1.073562

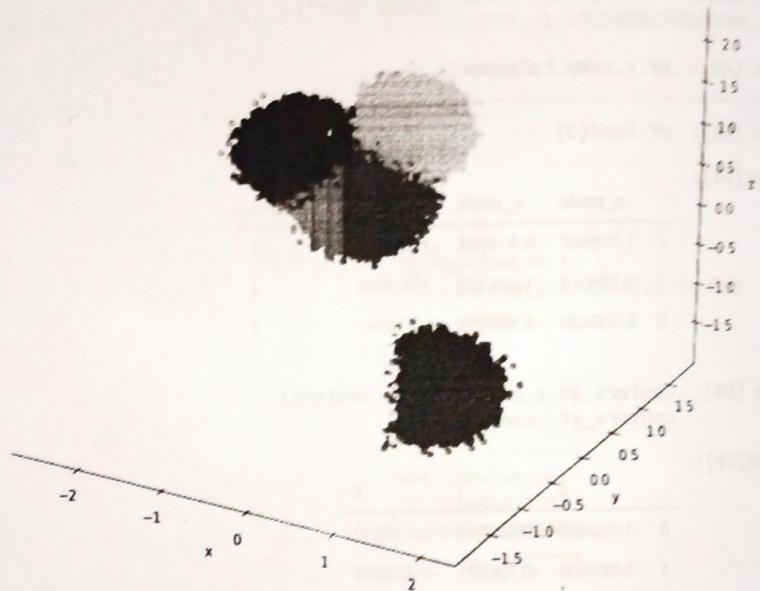
In [55]: import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D

3/16/2020

Ex1\_Kmeans\_Scaler\_3D\_points - Jupyter Notebook

```
In [58]: threedee = plt.figure(figsize=(12,10)).gca(projection='3d')
threedee.scatter(df.x_scale, df.y_scale,
                  df.z_scale,
                  c=df.prediction)
threedee.set_xlabel('x')
threedee.set_ylabel('y')
threedee.set_zlabel('z')

plt.show()
```



## Combine results



```
In [51]: # since there is no common column between these two dataframes add row_index
final_data=final_data.withColumn('row_index',
                                  f.monotonically_increasing_id())
temp=temp.withColumn('row_index',
                      f.monotonically_increasing_id())
final_data = final_data.join(temp,
                             on=["row_index"]).sort("row_index").drop("row_index")
```

```
In [52]: final_data.show(3, False)
```

x	y	z	features
scaledFeatures			x_scale
y_scale	z_scale	prediction	
[5.647627534046943]	-6.356222340123802	-7.240816026826695	[5.647627534046943, -6.
356222340123802, -7.240816026826695]	][1.0159673512169811, -0.81335799160424, -1.13	00631023636807]	00631023636807]  1.0159673512169811 -0.81335799160424  -1.1300631023636807 1
4.414367138680041]	-10.32624175635328	8.963324308916228	[4.414367138680041, -1
0.32624175635328, 8.963324308916228]	][0.7941127247055396, -1.3213715327025133, 1.	3988923401033817]	3988923401033817]  0.7941127247055396 -1.3213715327025133 1.3988923401033817  2
5.005396944639823]	-9.301070062115645	10.35473056351597	[5.005396944639823, -9.
301070062115645, 10.35473056351597]	][0.9004347126254774, -1.1901880174546189, 1.6	160469899240197]	160469899240197]  0.9004347126254774 -1.1901880174546189 1.6160469899240197  2

only showing top 3 rows

```
In [53]: from pyspark.ml import SparkSession
spark = SparkSession.builder.appName('k-means').getOrCreate()
```

```
In [54]: from pyspark.ml.clustering import KMeans
kmeans = KMeans(k=3).setSeed(12345)
kmeans.setFeaturesCol("features")
```



## Chapter 9: K-Means Clustering

### Ex2: Clustering Consulting Project - Hack\_data

A large technology firm needs your help, they've been hacked! Luckily their forensic engineers have grabbed valuable data about the hacks, including information like session time, locations, wpm typing speed, etc. The forensic engineer relates to you what she has been able to figure out so far, she has been able to grab meta data of each session that the hackers used to connect to their servers. These are the features of the data:

- 'Session\_Connection\_Time': How long the session lasted in minutes
- 'Bytes Transferred': Number of MB transferred during session
- 'Kali\_Trace\_Used': Indicates if the hacker was using Kali Linux
- 'Servers\_Corrupted': Number of server corrupted during the attack
- 'Pages\_Corrupted': Number of pages illegally accessed
- 'Location': Location attack came from (Probably useless because the hackers used VPNs)
- 'WPM\_Typing\_Speed': Their estimated typing speed based on session logs.

The technology firm has 3 potential hackers that perpetrated the attack. They're certain of the first two hackers but they aren't very sure if the third hacker was involved or not. They have requested your help! Can you help figure out whether or not the third suspect had anything to do with the attacks, or was it just two hackers? It's probably not possible to know for sure, but maybe what you've just learned about Clustering can help!

One last key fact, the forensic engineer knows that the hackers trade off attacks. Meaning they should each have roughly the same amount of attacks. For example if there were 100 total attacks, then in a 2 hacker situation each should have about 50 hacks, in a three hacker situation each would have about 33 hacks. The engineer believes this is the key element to solving this, but doesn't know how to distinguish this unlabeled data into groups of hackers.



## Chapter 9: K-means Clustering

### Ex 2: Clustering Consulting Project - Hack\_data - Solutions

A large technology firm needs your help, they've been hacked! Luckily their forensic engineers have grabbed valuable data about the hacks, including information like session time, locations, wpm typing speed, etc. The forensic engineer relates to you what she has been able to figure out so far, she has been able to grab meta data of each session that the hackers used to connect to their servers. These are the features of the data:

- 'Session\_Connection\_Time': How long the session lasted in minutes
- 'Bytes Transferred': Number of MB transferred during session
- 'Kali\_Trace\_Used': Indicates if the hacker was using Kali Linux
- 'Servers\_Corrupted': Number of server corrupted during the attack
- 'Pages\_Corrupted': Number of pages illegally accessed
- 'Location': Location attack came from (Probably useless because the hackers used VPNs)
- 'WPM\_Typing\_Speed': Their estimated typing speed based on session logs.

The technology firm has 3 potential hackers that perpetrated the attack. They're certain of the first two hackers but they aren't very sure if the third hacker was involved or not. They have requested your help! Can you help figure out whether or not the third suspect had anything to do with the attacks, or was it just two hackers? It's probably not possible to know for sure, but maybe what you've just learned about Clustering can help!

**One last key fact, the forensic engineer knows that the hackers trade off attacks. Meaning they should each have roughly the same amount of attacks. For example if there were 100 total attacks, then in a 2 hacker situation each should have about 50 hacks, in a three hacker situation each would have about 33 hacks. The engineer believes this is the key element to solving this, but doesn't know how to distinguish this unlabeled data into groups of hackers.**

```
In [1]: import findspark
findspark.init()
```

```
In [2]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('hack_find').getOrCreate()
```

```
In [3]: from pyspark.ml.clustering import KMeans

# Loads data.
dataset = spark.read.csv("hack_data.csv", header=True,
                         inferSchema=True)
```



In [4]: `dataset.head()`

Out[4]: Row(Session\_Connection\_Time=8.0, Bytes\_Transferred=391.09, Kali\_Trace\_Used=1, Servers\_Corrupted=2.96, Pages\_Corrupted=7.0, Location='Slovenia', WPM\_Typing\_Speed=72.37)

In [5]: `dataset.describe().show()`

	Session_Connection_Time	Bytes_Transferred	Kali_Trace_Used	Servers_Corrupted	Pages_Corrupted	Location	WPM_Typing_Speed
count	334	334	334	334	334	334	334
mean	30.008982035928145	607.2452694610777	0.5119760479041916	5.258502994011977	10.838323353293413	null	57.342395209580864
stddev	14.088200614636158	286.33593163576757	0.5006065264451406	2.30190693339697	3.06352633036022	null	13.41106336843464
min	1.0	1.0	10.0	0	6.0	Afghanistan	40.0
max	10.0	15.0	1330.5	1	60.0	Zimbabwe	75.0

In [6]: `dataset.columns`

Out[6]: ['Session\_Connection\_Time',  
'Bytes\_Transferred',  
'Kali\_Trace\_Used',  
'Servers\_Corrupted',  
'Pages\_Corrupted',  
'Location',  
'WPM\_Typing\_Speed']

In [7]: `from pyspark.ml.linalg import Vectors  
from pyspark.ml.feature import VectorAssembler`

In [8]: `feat_cols = ['Session_Connection_Time',  
'Bytes_Transferred',  
'Kali_Trace_Used',  
'Servers_Corrupted',  
'Pages_Corrupted', 'WPM_Typing_Speed']`

In [9]: `vecAssembler = VectorAssembler(inputCols = feat_cols,  
outputCol='features')`

In [10]: `final_data = vecAssembler.transform(dataset)`

In [11]: `from pyspark.ml.feature import StandardScaler`



```
In [12]: scaler = StandardScaler(inputCol="features",
                                outputCol="scaledFeatures",
                                withStd=True,
                                withMean=False)
```

```
In [13]: # Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(final_data)
```

```
In [14]: # Normalize each feature to have unit standard deviation.
cluster_final_data = scalerModel.transform(final_data)
```

\*\* Time to find out whether its 2 or 3! \*\*

```
In [15]: kmeans3 = KMeans(featuresCol='scaledFeatures',k=3)
kmeans2 = KMeans(featuresCol='scaledFeatures',k=2)
```

```
In [16]: model_k3 = kmeans3.fit(cluster_final_data)
model_k2 = kmeans2.fit(cluster_final_data)
```

```
In [17]: wssse_k3 = model_k3.computeCost(cluster_final_data)
wssse_k2 = model_k2.computeCost(cluster_final_data)
```

```
[18]: print("With K=3")
print("Within Set Sum of Squared Errors = " + str(wssse_k3))
print('--'*30)
print("With K=2")
print("Within Set Sum of Squared Errors = " + str(wssse_k2))
```

With K=3  
Within Set Sum of Squared Errors = 434.1492898715845

-----  
With K=2  
Within Set Sum of Squared Errors = 601.7707512676716

Not much to be gained from the WSSSE, after all, we would expect that as K increases, the WSSSE decreases. We could however continue the analysis by seeing the drop from K=3 to K=4 to check if the clustering favors even or odd numbers. This won't be substantial, but it's worth a look:



```
In [19]: for k in range(2,9):
    kmeans = KMeans(featuresCol='scaledFeatures',k=k)
    model = kmeans.fit(cluster_final_data)
    wssse = model.computeCost(cluster_final_data)
    print("With K={}".format(k))
    print("Within Set Sum of Squared Errors = " + str(wssse))
    print('---'*30)

With K=2
Within Set Sum of Squared Errors = 601.7707512676716
-----
With K=3
Within Set Sum of Squared Errors = 434.1492898715845
-----
With K=4
Within Set Sum of Squared Errors = 414.8171173373783
-----
With K=5
Within Set Sum of Squared Errors = 247.80143915458297
-----
With K=6
Within Set Sum of Squared Errors = 232.65169349742308
-----
With K=7
Within Set Sum of Squared Errors = 219.57683951525962
-----
With K=8
Within Set Sum of Squared Errors = 206.34236284345502
-----
```

\*\* Nothing definitive can be said with the above, but wait! The last key fact that the engineer mentioned was that the attacks should be evenly numbered between the hackers! Let's check with the transform and prediction columns that result form this! Congratulations if you made this connection, it was quite tricky given what we've covered!\*\*

```
In [20]: model_k3.transform(cluster_final_data).groupBy('prediction').count().show()

+-----+-----+
|prediction|count|
+-----+-----+
|      1|   167|
|      2|    83|
|      0|    84|
+-----+-----+
```



```
In [21]: model_k2.transform(cluster_final_data).groupBy('prediction').count().show()
```

prediction	count
1	167
0	167

**It was 2 hackers, in fact, our clustering algorithm created two equally sized clusters with K=2, no way that is a coincidence!**

In [ ]:



## Chapter 9: Recommender

### Ex3: Musical Instruments

#### Dataset: Musical\_Instruments\_5.json

Read more about dataset: <http://jmcauley.ucsd.edu/data/amazon/>  
<http://jmcauley.ucsd.edu/data/amazon/>

#### Requirement:

- Read dataset
- Pre-process data
- Use "asin" (ProductID), "reviewerID" and overall (User's reviews for each product - rating) to build model to predict overalls => Give recommendation for users.

```
In [1]: import findspark
findspark.init()
```

```
In [2]: from pyspark.sql import SparkSession
```

```
In [3]: spark = SparkSession.builder.appName('Recommendation_system').getOrCreate()
```

```
In [4]: data = spark.read.json("Musical_Instruments_5.json")
```

```
In [5]: data.show(5,truncate=True)
```

asin	helpful	overall	reviewText	reviewTime	reviewerID
reviewerName			summary	unixReviewTime	
[1384719342]	[0, 0]	5.0	Not much to write...	02 28, 2014	A2IBP120UZIR0U ca
sandra tu "Yea...			good	1393545600	
[1384719342]	[13, 14]	5.0	The product does ...	03 16, 2013	A14VAT5EAX3D95
Jake	Jake	1363392000			
[1384719342]	[1, 1]	5.0	The primary job o ...	08 28, 2013	A195EZSQDW3E21 Ri
ck Bennette "Ri...			It Does The Job Well	1377648000	
[1384719342]	[0, 0]	5.0	Nice windscreen p ...	02 14, 2014	A2C00NNNG1ZQQG2 Ru
styBill "Sunday... GOOD WINDSCREEN F...					
[1384719342]	[0, 0]	5.0	This pop filter i ...	02 21, 2014	A94QU4C90B1AX
SEAN MASLANKA No more pops when...		1392940800			

only showing top 5 rows

## Ex3\_Recommendation\_Project\_Musical - Jupyter Notebook



```
In [6]: data_sub = data.select(['asin', 'overall', 'reviewerID'])
```

```
In [7]: data_sub.count()
```

```
Out[7]: 10261
```

```
In [8]: from pyspark.sql.functions import col, udf
from pyspark.sql.functions import isnan, when, count, col
```

```
In [10]: data_sub.show(5, truncate=True)
```

	asin overall  reviewerID
1384719342	5.0 A2IBPI20UZIR0U
1384719342	5.0 A14VAT5EAX3D9S
1384719342	5.0 A195EZSQDW3E21
1384719342	5.0 A2C00NNNG1ZQQG2
1384719342	5.0  A94QU4C90B1AX

only showing top 5 rows

```
In [11]: data_sub.select([count(when(col(c).isNull(), c)).alias(c) for c in
data_sub.columns]).toPandas().T
```

```
Out[11]:
```

	0
asin	0
overall	0
reviewerID	0

```
In [12]: # Distinct users and movies
users = data_sub.select("reviewerID").distinct().count()
products = data_sub.select("asin").distinct().count()
numerator = data_sub.count()
```

```
In [13]: display(numerator, users, products)
```

```
10261
```

```
1429
```

```
900
```

```
In [14]: # Number of ratings matrix could contain if no empty cells
denominator = users * products
denominator
```

```
Out[14]: 1286100
```



```
In [15]: #Calculating sparsity
sparsity = 1 - (numerator*i.0 / denominator)
print ("Sparsity: "), sparsity
```

Sparsity:

```
Out[15]: (None, 0.992021615737501)
```

```
In [16]: from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
```

```
In [17]: # Converting String to index
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.sql.functions import col
```

```
In [18]: # Create an indexer
indexer = StringIndexer(inputCol='asin',
                        outputCol='asin_idx')
```

# Indexer identifies categories in the data  
indexer\_model = indexer.fit(data\_sub)

# Indexer creates a new column with numeric index values  
data\_indexed = indexer\_model.transform(data\_sub)

# Repeat the process for the other categorical feature  
indexer1 = StringIndexer(inputCol='reviewerID',
 outputCol='reviewerID\_idx')  
indexer1\_model = indexer1.fit(data\_indexed)  
data\_indexed = indexer1\_model.transform(data\_indexed)

```
In [19]: data_indexed.show(5, truncate=True)
```

	asin overall	reviewerID asin_idx reviewerID_idx	
1384719342	5.0 A2IBPI20UZIR0U	781.0	72.0
1384719342	5.0 A14VAT5EAX3D9S	781.0	359.0
1384719342	5.0 A195EZ5QDW3E21	781.0	436.0
1384719342	5.0 A2C00NNG1ZQQG2	781.0	1216.0
1384719342	5.0 A94QU4C90B1AX	781.0	1137.0

only showing top 5 rows

3/16/2020

Ex3\_Recommendation\_Project\_Musical - Jupyter Notebook



```
In [20]: data_indexed.select([count(when(col(c).isNull(), c)).alias(c) for c in data_indexed.columns]).toPandas().T
```

```
out[20]:
```

	0
asin	0
overall	0
reviewerID	0
asin_idx	0
reviewerID_idx	0

```
In [21]: # Smaller dataset so we will use 0.8 / 0.2  
(training, test) = data_indexed.randomSplit([0.8, 0.2])
```

```
In [22]: # Creating ALS model and fitting data  
from pyspark.ml.evaluation import RegressionEvaluator  
from pyspark.ml.recommendation import ALS
```

```
In [23]: als = ALS(maxIter=5,  
                 regParam=0.09,  
                 rank = 25,  
                 userCol="reviewerID_idx",  
                 itemCol="asin_idx",  
                 ratingCol="overall",  
                 coldStartStrategy="drop",  
                 nonnegative=True)  
model = als.fit(training)
```

```
In [24]: # Evaluate the model by computing the RMSE on the test data  
predictions = model.transform(test)
```

```
In [25]: predictions.select(["asin_idx", "reviewerID_idx",  
                           "overall", "prediction"]).show(5)
```

asin_idx	reviewerID_idx	overall	prediction
148.0	34.0	4.0	3.732204
148.0	502.0	5.0	3.8744607
148.0	1377.0	5.0	2.8333657
148.0	30.0	3.0	4.517724
463.0	671.0	5.0	4.361149

only showing top 5 rows



```
In [29]: evaluator = RegressionEvaluator(metricName="rmse",
                                         labelCol="overall",
                                         predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

Root-mean-square error = 1.1928346794514073
```

In [ ]: # On average, this model is ~ 1.2 from perfect recommendations.

## Providing Recommendations: for all users

```
In [33]: # get 20 recommendations which have highest rating.
user_recs = model.recommendForAllUsers(20)
```

User ID	Product ID	Rating
1	1	4.0
1	2	3.0
1	3	4.0
1	4	3.0
1	5	4.0
1	6	3.0
1	7	4.0
1	8	3.0
1	9	4.0
1	10	3.0
1	11	4.0
1	12	3.0
1	13	4.0
1	14	3.0
1	15	4.0
1	16	3.0
1	17	4.0
1	18	3.0
1	19	4.0
1	20	3.0
2	1	4.0
2	2	3.0
2	3	4.0
2	4	3.0
2	5	4.0
2	6	3.0
2	7	4.0
2	8	3.0
2	9	4.0
2	10	3.0
2	11	4.0
2	12	3.0
2	13	4.0
2	14	3.0
2	15	4.0
2	16	3.0
2	17	4.0
2	18	3.0
2	19	4.0
2	20	3.0
3	1	4.0
3	2	3.0
3	3	4.0
3	4	3.0
3	5	4.0
3	6	3.0
3	7	4.0
3	8	3.0
3	9	4.0
3	10	3.0
3	11	4.0
3	12	3.0
3	13	4.0
3	14	3.0
3	15	4.0
3	16	3.0
3	17	4.0
3	18	3.0
3	19	4.0
3	20	3.0
4	1	4.0
4	2	3.0
4	3	4.0
4	4	3.0
4	5	4.0
4	6	3.0
4	7	4.0
4	8	3.0
4	9	4.0
4	10	3.0
4	11	4.0
4	12	3.0
4	13	4.0
4	14	3.0
4	15	4.0
4	16	3.0
4	17	4.0
4	18	3.0
4	19	4.0
4	20	3.0
5	1	4.0
5	2	3.0
5	3	4.0
5	4	3.0
5	5	4.0
5	6	3.0
5	7	4.0
5	8	3.0
5	9	4.0
5	10	3.0
5	11	4.0
5	12	3.0
5	13	4.0
5	14	3.0
5	15	4.0
5	16	3.0
5	17	4.0
5	18	3.0
5	19	4.0
5	20	3.0
6	1	4.0
6	2	3.0
6	3	4.0
6	4	3.0
6	5	4.0
6	6	3.0
6	7	4.0
6	8	3.0
6	9	4.0
6	10	3.0
6	11	4.0
6	12	3.0
6	13	4.0
6	14	3.0
6	15	4.0
6	16	3.0
6	17	4.0
6	18	3.0
6	19	4.0
6	20	3.0
7	1	4.0
7	2	3.0
7	3	4.0
7	4	3.0
7	5	4.0
7	6	3.0
7	7	4.0
7	8	3.0
7	9	4.0
7	10	3.0
7	11	4.0
7	12	3.0
7	13	4.0
7	14	3.0
7	15	4.0
7	16	3.0
7	17	4.0
7	18	3.0
7	19	4.0
7	20	3.0
8	1	4.0
8	2	3.0
8	3	4.0
8	4	3.0
8	5	4.0
8	6	3.0
8	7	4.0
8	8	3.0
8	9	4.0
8	10	3.0
8	11	4.0
8	12	3.0
8	13	4.0
8	14	3.0
8	15	4.0
8	16	3.0
8	17	4.0
8	18	3.0
8	19	4.0
8	20	3.0
9	1	4.0
9	2	3.0
9	3	4.0
9	4	3.0
9	5	4.0
9	6	3.0
9	7	4.0
9	8	3.0
9	9	4.0
9	10	3.0
9	11	4.0
9	12	3.0
9	13	4.0
9	14	3.0
9	15	4.0
9	16	3.0
9	17	4.0
9	18	3.0
9	19	4.0
9	20	3.0
10	1	4.0
10	2	3.0
10	3	4.0
10	4	3.0
10	5	4.0
10	6	3.0
10	7	4.0
10	8	3.0
10	9	4.0
10	10	3.0
10	11	4.0
10	12	3.0
10	13	4.0
10	14	3.0
10	15	4.0
10	16	3.0
10	17	4.0
10	18	3.0
10	19	4.0
10	20	3.0
11	1	4.0
11	2	3.0
11	3	4.0
11	4	3.0
11	5	4.0
11	6	3.0
11	7	4.0
11	8	3.0
11	9	4.0
11	10	3.0
11	11	4.0
11	12	3.0
11	13	4.0
11	14	3.0
11	15	4.0
11	16	3.0
11	17	4.0
11	18	3.0
11	19	4.0
11	20	3.0
12	1	4.0
12	2	3.0
12	3	4.0
12	4	3.0
12	5	4.0
12	6	3.0
12	7	4.0
12	8	3.0
12	9	4.0
12	10	3.0
12	11	4.0
12	12	3.0
12	13	4.0
12	14	3.0
12	15	4.0
12	16	3.0
12	17	4.0
12	18	3.0
12	19	4.0
12	20	3.0
13	1	4.0
13	2	3.0
13	3	4.0
13	4	3.0
13	5	4.0
13	6	3.0
13	7	4.0
13	8	3.0
13	9	4.0
13	10	3.0
13	11	4.0
13	12	3.0
13	13	4.0
13	14	3.0
13	15	4.0
13	16	3.0
13	17	4.0
13	18	3.0
13	19	4.0
13	20	3.0
14	1	4.0
14	2	3.0
14	3	4.0
14	4	3.0
14	5	4.0
14	6	3.0
14	7	4.0
14	8	3.0
14	9	4.0
14	10	3.0
14	11	4.0
14	12	3.0
14	13	4.0
14	14	3.0
14	15	4.0
14	16	3.0
14	17	4.0
14	18	3.0
14	19	4.0
14	20	3.0
15	1	4.0
15	2	3.0
15	3	4.0
15	4	3.0
15	5	4.0
15	6	3.0
15	7	4.0
15	8	3.0
15	9	4.0
15	10	3.0
15	11	4.0
15	12	3.0
15	13	4.0
15	14	3.0
15	15	4.0
15	16	3.0
15	17	4.0
15	18	3.0
15	19	4.0
15	20	3.0
16	1	4.0
16	2	3.0
16	3	4.0
16	4	3.0
16	5	4.0
16	6	3.0
16	7	4.0
16	8	3.0
16	9	4.0
16	10	3.0
16	11	4.0
16	12	3.0
16	13	4.0
16	14	3.0
16	15	4.0
16	16	3.0
16	17	4.0
16	18	3.0
16	19	4.0
16	20	3.0
17	1	4.0
17	2	3.0
17	3	4.0
17	4	3.0
17	5	4.0
17	6	3.0
17	7	4.0
17	8	3.0
17	9	4.0
17	10	3.0
17	11	4.0
17	12	3.0
17	13	4.0
17	14	3.0
17	15	4.0
17	16	3.0
17	17	4.0
17	18	3.0
17	19	4.0
17	20	3.0
18	1	4.0
18	2	3.0
18	3	4.0
18	4	3.0
18	5	4.0
18	6	3.0
18	7	4.0
18	8	3.0
18	9	4.0
18	10	3.0
18	11	4.0
18	12	3.0
18	13	4.0
18	14	3.0
18	15	4.0
18	16	3.0
18	17	4.0
18	18	3.0
18	19	4.0
18	20	3.0
19	1	4.0
19	2	3.0
19	3	4.0
19	4	3.0
19	5	4.0
19	6	3.0
19	7	4.0
19	8	3.0
19	9	4.0
19	10	3.0
19	11	4.0
19	12	3.0
19	13	4.0
19	14	3.0
19	15	4.0
19	16	3.0
19	17	4.0
19	18	3.0
19	19	4.0
19	20	3.0
20	1	4.0
20	2	3.0
20	3	4.0
20	4	3.0
20	5	4.0
20	6	3.0
20	7	4.0
20	8	3.0
20	9	4.0
20	10	3.0
20	11	4.0
20	12	3.0
20	13	4.0
20	14	3.0
20	15	4.0
20	16	3.0
20	17	4.0
20	18	3.0
20	19	4.0
20	20	3.0



```
In [35]: for user in user_recs.head(5):
    print(user)
    print("\n")
```

```
RowReviewerID_idx=471, recommendations=[Row(asin_idx=357, rating=5.99492979049
6826), Row(asin_idx=402, rating=5.971096515655518), Row(asin_idx=328, rating=5.
91753625869751), Row(asin_idx=544, rating=5.729032039642334), Row(asin_idx=476,
rating=5.716101169586182), Row(asin_idx=232, rating=5.712975978851318), Row(asin_
idx=852, rating=5.712711811065674), Row(asin_idx=888, rating=5.70264530181884
8), Row(asin_idx=360, rating=5.70063591003418), Row(asin_idx=427, rating=5.6923
59924316406), Row(asin_idx=346, rating=5.683685302734375), Row(asin_idx=746, ra
ting=5.678509712219238), Row(asin_idx=829, rating=5.665043830871582), Row(asin_
idx=368, rating=5.654289245605469), Row(asin_idx=805, rating=5.63354778289794
9), Row(asin_idx=350, rating=5.57622766494751), Row(asin_idx=723, rating=5.5738
38233947754), Row(asin_idx=396, rating=5.543375492095947), Row(asin_idx=632, ra
ting=5.540017127990723), Row(asin_idx=734, rating=5.527731895446777)])
```

```
RowReviewerID_idx=1342, recommendations=[Row(asin_idx=731, rating=6.3991994857
78809), Row(asin_idx=427, rating=6.240170001983643), Row(asin_idx=746, rating=
6.126894950866699), Row(asin_idx=328, rating=6.101090431213379), Row(asin_idx=2
32, rating=6.0971221923828125), Row(asin_idx=852, rating=6.052910327911377),
Row(asin_idx=476, rating=6.034964561462402), Row(asin_idx=645, rating=6.031756877
89917), Row(asin_idx=544, rating=6.0315022468566895), Row(asin_idx=829, rating=
6.031264781951904), Row(asin_idx=682, rating=6.015640735626221), Row(asin_idx=8
53, rating=6.006399154663086), Row(asin_idx=734, rating=6.0043182373046875), Ro
w(asin_idx=346, rating=5.98906135559082), Row(asin_idx=723, rating=5.9865040779
11377), Row(asin_idx=766, rating=5.969595432281494), Row(asin_idx=888, rating=
5.9575724601745605), Row(asin_idx=791, rating=5.956635475158691), Row(asin_idx=
402, rating=5.906132698059082), Row(asin_idx=809, rating=5.890671253204346)])
```

```
RowReviewerID_idx=463, recommendations=[Row(asin_idx=746, rating=5.70749330520
6299), Row(asin_idx=476, rating=5.547768592834473), Row(asin_idx=544, rating=5.
54088973990234), Row(asin_idx=368, rating=5.476436614990234), Row(asin_idx=80
5, rating=5.46685266494751), Row(asin_idx=731, rating=5.438949108123779), Row(a
sin_idx=465, rating=5.411510944366455), Row(asin_idx=427, rating=5.398251056671
143), Row(asin_idx=698, rating=5.384541988372803), Row(asin_idx=853, rating=5.3
731842041015625), Row(asin_idx=306, rating=5.3723273277282715), Row(asin_idx=18
3, rating=5.372140407562256), Row(asin_idx=664, rating=5.357931613922119), Row
(asin_idx=232, rating=5.35223913192749), Row(asin_idx=328, rating=5.33652114868
1641), Row(asin_idx=678, rating=5.331408977508545), Row(asin_idx=829, rating=5.
323209285736084), Row(asin_idx=518, rating=5.315012454986572), Row(asin_idx=90,
rating=5.2987165451049805), Row(asin_idx=791, rating=5.295729637145996)])
```

```
RowReviewerID_idx=833, recommendations=[Row(asin_idx=734, rating=6.09094333648
6816), Row(asin_idx=476, rating=5.975735187530518), Row(asin_idx=465, rating=5.
958075523376465), Row(asin_idx=746, rating=5.939692974090576), Row(asin_idx=36
8, rating=5.900303840637207), Row(asin_idx=437, rating=5.891534328460693), Row
(asin_idx=396, rating=5.8877973556518555), Row(asin_idx=853, rating=5.885824680
328369), Row(asin_idx=427, rating=5.86740779876709), Row(asin_idx=402, rating=
5.852911949157715), Row(asin_idx=888, rating=5.833059310913086), Row(asin_idx=6
84, rating=5.827024936676025), Row(asin_idx=791, rating=5.814032554626465), Row
(asin_idx=705, rating=5.810422420501709), Row(asin_idx=645, rating=5.8077998161
31592), Row(asin_idx=346, rating=5.799132347106934), Row(asin_idx=632, rating=
5.797298908233643), Row(asin_idx=829, rating=5.776956081390381), Row(asin_idx=8
```

```
09, rating=5.772419452667236), Row(asin_idx=662, rating=5.757423400878906)])
```



```
RowReviewerID_idx=496, recommendations=[Row(asin_idx=232, rating=5.53647279739
3799), Row(asin_idx=731, rating=5.501907825469971), Row(asin_idx=427, rating=5.
453770160675049), Row(asin_idx=791, rating=5.439143657684326), Row(asin_idx=74
6, rating=5.427332878112793), Row(asin_idx=853, rating=5.397946834564209), Row
(asin_idx=476, rating=5.383556842803955), Row(asin_idx=829, rating=5.3825254440
30762), Row(asin_idx=368, rating=5.379194736480713), Row(asin_idx=886, rating=
5.36549186706543), Row(asin_idx=632, rating=5.29641056060791), Row(asin_idx=46
5, rating=5.287215709686279), Row(asin_idx=698, rating=5.279950141986738), Row
(asin_idx=734, rating=5.271482467651367), Row(asin_idx=293, rating=5.2693777084
35059), Row(asin_idx=274, rating=5.261102199554443), Row(asin_idx=678, rating=
5.249166488647461), Row(asin_idx=402, rating=5.2356858253479), Row(asin_idx=54
4, rating=5.224824905395508), Row(asin_idx=805, rating=5.2206926345825195)])
```

## Converting back to string form

```
In [38]: import pandas as pd
recs=model.recommendForAllUsers(10).toPandas()
nrecs=recs.recommendations.apply(pd.Series) \
    .merge(recs, right_index = True, left_index = True) \
    .drop(['recommendations'], axis = 1) \
    .melt(id_vars = ['reviewerID_idx'], value_name = "recommendation") \
    .drop("variable", axis = 1) \
    .dropna()
nrecs=nrecs.sort_values('reviewerID_idx')
nrecs=pd.concat([nrecs['recommendation'].apply(pd.Series),
                 nrecs['reviewerID_idx']], axis = 1)
nrecs.columns = [
    'ProductID_index',
    'Rating',
    'UserID_index'
]
```

3/18/2020

## Ex3\_Recommendation\_Project\_Musical - Jupyter Notebook



```
In [41]: md=data_indexed.select(['reviewerID', 'reviewerID_idx',
                               'asin', 'asin_idx'])
md=md.toPandas()
dict1 =dict(zip(md['reviewerID_idx'],md['reviewerID']))
dict2=dict(zip(md['asin_idx'],md['asin']))
nrecs['reviewerID']=nrecs['UserID_index'].map(dict1)
nrecs['asin']=nrecs['ProductID_index'].map(dict2)
nrecs=nrecs.sort_values('reviewerID')
nrecs.reset_index(drop=True, inplace=True)
new=nrecs[['reviewerID','asin','Rating']]
new['recommendations'] = list(zip(new.asin, new.Rating))
res=new[['reviewerID','recommendations']]
res_new=res['recommendations'].groupby([res.reviewerID])\
    .apply(list).reset_index()

c:\program files\python36\lib\site-packages\ipykernel_launcher.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
y)
# Remove the CWD from sys.path while we load stuff.
```

Out[42]: res\_new

Out[42]:

	reviewerID	recommendations
0	A00625243Bi8W1SSZNLMD	[(B0002GXRF2, 5.8296637535095215), (B001CJ2QZU...]
1	A10044ECXDUVKS	[(B001L8KE06, 5.229329586029053), (B004PFWZHM,...]
2	A102MU6ZC9H1N6	[(B0002GXRF2, 5.819663047790527), (B000MWWT6E,...]
3	A109JTUZXO61UY	[(B001IM5KFY, 5.880716323852539), (B0002BACB4,...]
4	A109ME7C09HM2M	[(B000RYE5Y6, 6.396268844604492), (B000MWWT6E...]
...	...	...
1424	AZJPNK73JF3XP	[(B001CJ2QZU, 5.473949432373047), (B003AYEAHC,...]
1425	AZMHABTPXVLG3	[(B004T6M7DE, 3.672528028488159), (B0002CZVI2,...]
1426	AZMIKIG4BB6BZ	[(B000RYE5Y6, 5.908489227294922), (B003AYEAHC,...]
1427	AZPDO6FLSMLFP	[(B000RYE5Y6, 5.2437310218811035), (B0000AQRST...]
1428	AZVME8JMPD3F4	[(B000WGJ71U, 4.245903015136719), (B000RKL8R2,...]

1429 rows × 2 columns



## Chapter 9: Recommender

### Ex4: Beauty

#### Dataset: Beauty\_5.json.json

Read more about dataset: <http://jmcauley.ucsd.edu/data/amazon/>  
<http://jmcauley.ucsd.edu/data/amazon/>

#### Requirement:

- Read dataset
- Pre-process data
- Use "asin" (ProductID), "reviewerID" and overall (User's reviews for each product - rating) to build model to predict overalls => Give recommendation for users.

```
In [1]: import findspark
findspark.init()
```

```
In [2]: from pyspark.sql import SparkSession
```

```
In [3]: spark = SparkSession.builder.appName('Recommendation_Beauty').getOrCreate()
```

```
In [4]: data = spark.read.json("Beauty_5.json")
```

```
In [5]: data.show(5, truncate=True)
```

asin	helpful	overall	reviewText	reviewTime	reviewerID	reviewerName	summary	unixReviewTime
[7806397051]	[3, 4]	1.0	Very oily and cre...	[01 30, 2014]	A1YJFY40YUW4SE!	Andrea	Don't waste your ...	1391040000
[7806397051]	[1, 1]	3.0	This palette was ...	[04 18, 2014]	A60XNB876KYML	Jessica H.	OK Palette!!	1397779200
[7806397051]	[0, 1]	4.0	The texture of th...	[09 6, 2013]	A3G6XNM240RMWA	Karen	great quality	1378425600
[7806397051]	[2, 2]	2.0	I really can't te...	[12 8, 2013]	A1PQFP6SAJ6D80	Norah	Do not work on my...	1386460800
[7806397051]	[0, 0]	3.0	It was a little s...	[10 19, 2013]	A38FVHZTNQ271F	Nova Amor	It's okay.	1382140800

only showing top 5 rows

## Ex4\_Recommendation\_Project\_Beauty - Jupyter Notebook

```
In [6]: data_sub = data.select(['asin', 'overall', 'reviewerID'])

In [7]: data_sub.count()

Out[7]: 198502

In [8]: from pyspark.sql.functions import col, udf
        from pyspark.sql.functions import isnan, when, count, col

In [9]: data_sub.show(5, truncate=True)

+-----+-----+-----+
|    asin|overall| reviewerID|
+-----+-----+-----+
|7806397051| 1.0|A1YJEY40YUW4SE|
|7806397051| 3.0| A60XNB876KYM|
|7806397051| 4.0|A3G6XNM240RMWA|
|7806397051| 2.0|A1PQFP6SAJ6D80|
|7806397051| 3.0|A38FVHZTNQ271F|
+-----+-----+-----+
only showing top 5 rows

In [10]: data_sub.select([count(when(col(c).isNull(), c)).alias(c) for c in
                     data_sub.columns]).toPandas().T

Out[10]:
      0
  asin 0
  overall 0
  reviewerID 0

In [11]: # Distinct users and movies
users = data_sub.select("reviewerID").distinct().count()
products = data_sub.select("asin").distinct().count()
numerator = data_sub.count()

In [12]: display(numerator, users, products)

198502
22363
12101

In [13]: # Number of ratings matrix could contain if no empty cells
denominator = users * products
denominator

Out[13]: 270614663
```



```
In [14]: #Calculating sparsity
sparsity = 1 - (numerator*1.0 / denominator)
print ("Sparsity: "), sparsity

Sparsity:

Out[14]: (None, 0.9992664772935825)

In [15]: from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS

In [16]: # Converting String to index
from pyspark.ml.feature import StringIndexer
from pyspark.ml import Pipeline
from pyspark.sql.functions import col

In [17]: # Create an indexer
indexer = StringIndexer(inputCol='asin',
                        outputCol='asin_idx')

# Indexer identifies categories in the data
indexer_model = indexer.fit(data_sub)

# Indexer creates a new column with numeric index values
data_indexed = indexer_model.transform(data_sub)

# Repeat the process for the other categorical feature
indexer1 = StringIndexer(inputCol='reviewerID',
                         outputCol='reviewerID_idx')
indexer1_model = indexer1.fit(data_indexed)
data_indexed = indexer1_model.transform(data_indexed)

In [18]: data_indexed.show(5, truncate=True)
```

asin	overall	reviewerID	asin_idx	reviewerID_idx
7806397051	1.0	A1YJEY40YUW4SE	6959.0	18008.0
7806397051	3.0	A60XNB876KYML	6959.0	10825.0
7806397051	4.0	A3G6XNM240RMWA	6959.0	5924.0
7806397051	2.0	A1PQFP6SAJ6D80	6959.0	12357.0
7806397051	3.0	A38FVHZTNQ271F	6959.0	6087.0

only showing top 5 rows



## Ex4\_Recommendation\_Project\_Beauty - Jupyter Notebook

```
In [19]: data_indexed.select([count(when(col(c).isNull(), c)).alias(c) for c in  
data_indexed.columns]).toPandas().T
```

```
Out[19]:
```

	0
asin	0
overall	0
reviewerID	0
asin_idx	0
reviewerID_idx	0

```
In [20]: # Smaller dataset so we will use 0.8 / 0.2  
(training, test) = data_indexed.randomSplit([0.8, 0.2])
```

```
In [21]: # Creating ALS model and fitting data  
from pyspark.ml.evaluation import RegressionEvaluator  
from pyspark.ml.recommendation import ALS
```

```
In [22]: als = ALS(maxIter=5,  
                 regParam=0.09,  
                 rank = 25,  
                 userCol="reviewerID_idx",  
                 itemCol="asin_idx",  
                 ratingCol="overall",  
                 coldStartStrategy="drop",  
                 nonnegative=True)  
model = als.fit(training)
```

```
In [23]: # Evaluate the model by computing the RMSE on the test data  
predictions = model.transform(test)
```

```
In [24]: predictions.select(["asin_idx", "reviewerID_idx",  
                           "overall", "prediction"]).show(5)
```

asin_idx	reviewerID_idx	overall	prediction
148.0	9492.0	5.0	4.2765565
148.0	5258.0	3.0	3.6554668
148.0	5909.0	4.0	3.1009164
148.0	14415.0	5.0	3.8484008
148.0	19062.0	5.0	3.4705784

only showing top 5 rows



```
In [25]: evaluator = RegressionEvaluator(metricName="rmse",
                                         labelCol="overall",
                                         predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))

Root-mean-square error = 1.3504136630485968
```

```
In [26]: # On average, this model is ~ 1.35 from perfect recommendations.
```

## Providing Recommendations: for all users

```
In [27]: # get 20 recommendations which have highest rating.
user_recs = model.recommendForAllUsers(20)
```



## Ex4\_Recommendation\_Project\_Beauty - jupyter Notebook

In [28]: for user in user\_recs.head(5):  
 print(user)  
 print("\n")

```
RowReviewerID_idx=1586, recommendations=[Row(asin_idx=9900, rating=7.452762603  
759766), Row(asin_idx=7885, rating=7.445200443267822), Row(asin_idx=8386, rating  
=7.283047199249268), Row(asin_idx=7300, rating=7.233673572540283), Row(asin_id  
x=9998, rating=7.199503421783447), Row(asin_idx=9432, rating=7.078475952148437  
5), Row(asin_idx=8888, rating=7.067144870758057), Row(asin_idx=8747, rating=7.0  
50375461578369), Row(asin_idx=7840, rating=7.04985237121582), Row(asin_idx=314  
3, rating=7.045359134674072), Row(asin_idx=12100, rating=7.037707328796387), Ro  
w(asin_idx=11372, rating=7.035341739654541), Row(asin_idx=6793, rating=7.007732  
391357422), Row(asin_idx=6539, rating=6.991481781005859), Row(asin_idx=8643, ra  
ting=6.974045753479004), Row(asin_idx=8691, rating=6.971042156219482), Row(asin  
_idx=10728, rating=6.964417457580566), Row(asin_idx=10466, rating=6.94252109527  
5879), Row(asin_idx=10039, rating=6.9183573722839355), Row(asin_idx=7585, rating  
=6.905022621154785)])
```

```
RowReviewerID_idx=4900, recommendations=[Row(asin_idx=10071, rating=6.86618375  
7781982), Row(asin_idx=10450, rating=6.866183757781982), Row(asin_idx=12002, ra  
ting=6.856912612915039), Row(asin_idx=10013, rating=6.856446743011475), Row(asin  
_idx=8386, rating=6.838607311248779), Row(asin_idx=10958, rating=6.82022857666  
0156), Row(asin_idx=10279, rating=6.657992362976074), Row(asin_idx=7885, rating  
=6.617455005645752), Row(asin_idx=11977, rating=6.580188274383545), Row(asin_id  
x=9703, rating=6.570758819580078), Row(asin_idx=9665, rating=6.53776121139526  
4), Row(asin_idx=8603, rating=6.454494953155518), Row(asin_idx=7300, rating=6.4  
27910327911377), Row(asin_idx=9549, rating=6.423874855041504), Row(asin_idx=112  
27, rating=6.382607936859131), Row(asin_idx=9802, rating=6.360374927520752), Ro  
w(asin_idx=3422, rating=6.358455181121826), Row(asin_idx=6136, rating=6.3500509  
26208496), Row(asin_idx=6396, rating=6.349891662597656), Row(asin_idx=8972, rat  
ing=6.346033573150635)])
```

```
RowReviewerID_idx=5300, recommendations=[Row(asin_idx=10450, rating=7.88202285  
7666016), Row(asin_idx=10071, rating=7.882022857666016), Row(asin_idx=10279, ra  
ting=7.857653617858887), Row(asin_idx=12002, rating=7.836369514465332), Row(asin  
_idx=10013, rating=7.834311485290527), Row(asin_idx=10958, rating=7.7301292419  
43359), Row(asin_idx=7885, rating=7.546260833740234), Row(asin_idx=7300, rating  
=7.035318374633789), Row(asin_idx=4299, rating=7.0198140144348145), Row(asin_id  
x=8386, rating=6.878007888793945), Row(asin_idx=9545, rating=6.74942064285278  
3), Row(asin_idx=10466, rating=6.746596813201904), Row(asin_idx=7082, rating=6.  
730844497680664), Row(asin_idx=11673, rating=6.722889423370361), Row(asin_idx=9  
409, rating=6.704495906829834), Row(asin_idx=8347, rating=6.698277950286865), R  
ow(asin_idx=7361, rating=6.693828582763672), Row(asin_idx=7920, rating=6.614742  
279052734), Row(asin_idx=7900, rating=6.586149215698242), Row(asin_idx=4710, ra  
ting=6.5770463943481445)])
```

```
RowReviewerID_idx=6620, recommendations=[Row(asin_idx=10071, rating=7.87718248  
3673096), Row(asin_idx=10450, rating=7.877182483673096), Row(asin_idx=12002, ra  
ting=7.858290672302246), Row(asin_idx=10013, rating=7.858236789703369), Row(asin  
_idx=10958, rating=7.803875923156738), Row(asin_idx=7885, rating=7.77376794815  
0635), Row(asin_idx=10279, rating=7.713746547698975), Row(asin_idx=7300, rating  
=7.360595703125), Row(asin_idx=8386, rating=7.12551736831665), Row(asin_idx=943  
2, rating=7.098012924194336), Row(asin_idx=7591, rating=7.04342222137451), Row  
(asin_idx=9665, rating=7.017157554626465), Row(asin_idx=9784, rating=6.98062896
```



```
7285156), Row(asin_idx=6396, rating=6.968845367431641), Row(asin_idx=8347
ng=6.954199314117432), Row(asin_idx=9900, rating=6.791838645935059), Row(
dx=5677, rating=6.754838943481445), Row(asin_idx=7421, rating=6.73906898498535
2), Row(asin_idx=6098, rating=6.733091831207275), Row(asin_idx=8407, rating=6.7
07345008850098)])
```

```
RowReviewerID_idx=7240, recommendations=[Row(asin_idx=8386, rating=6.249364852
905273), Row(asin_idx=7300, rating=6.18107795715332), Row(asin_idx=7885, rating
=6.09645414352417), Row(asin_idx=9900, rating=6.0728325843811035), Row(asin_idx
=8888, rating=5.873400688171387), Row(asin_idx=10450, rating=5.83579730987548
8), Row(asin_idx=10071, rating=5.835797309875488), Row(asin_idx=10013, rating=
5.817392349243164), Row(asin_idx=12002, rating=5.816351413726807), Row(asin_idx
=6396, rating=5.768041610717773), Row(asin_idx=10728, rating=5.76656913757324
2), Row(asin_idx=10958, rating=5.7658257484436035), Row(asin_idx=9784, rating=
5.753474235534668), Row(asin_idx=7082, rating=5.733229637145996), Row(asin_idx=
5.7152099609375), Ro
8406, rating=5.722962379455566), Row(asin_idx=6098, rating=5.7128973
w(asin_idx=9220, rating=5.712944984436035), Row(asin_idx=8681, rating=5.7128973
00720215), Row(asin_idx=11521, rating=5.710545539855957), Row(asin_idx=10279,
rating=5.710353374481201)])
```

## Converting back to string form

```
In [29]: import pandas as pd
recs=model.recommendForAllUsers(10).toPandas()
nrecs=recs.recommendations.apply(pd.Series) \
    .merge(recs, right_index = True, left_index = True) \
    .drop(["recommendations"], axis = 1) \
    .melt(id_vars = ['reviewerID_idx'], value_name = "recommendation") \
    .drop("variable", axis = 1) \
    .dropna()
nrecs=nrecs.sort_values('reviewerID_idx')
nrecs=pd.concat([nrecs['recommendation']]\ 
    .apply(pd.Series), nrecs[['reviewerID_idx']], axis = 1)
nrecs.columns = [
    'ProductID_index',
    'Rating',
    'UserID_index'
]
```



## Ex4\_Recommendation\_Project\_Beauty - Jupyter Notebook

```
In [30]: md=data_indexed.select(['reviewerID', 'reviewerID_idx', 'asin', 'asin_idx'])
md=md.toPandas()
dict1 =dict(zip(md['reviewerID_idx'],md['reviewerID']))
dict2=dict(zip(md['asin_idx'],md['asin']))
nrecs['reviewerID']=nrecs['UserID_index'].map(dict1)
nrecs['asin']=nrecs['ProductID_index'].map(dict2)
nrecs=nrecs.sort_values('reviewerID')
nrecs.reset_index(drop=True, inplace=True)
new=nrecs[['reviewerID','asin','Rating']]
new['recommendations'] = list(zip(new.asin, new.Rating))
res=new[['reviewerID','recommendations']]
res_new=res['recommendations'].groupby([res.reviewерID])\
    .apply(list).reset_index()
```

c:\program files\python36\lib\site-packages\ipykernel\_launcher.py:10: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [http://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

# Remove the CWD from sys.path while we load stuff.

```
In [31]: res_new
```

```
Out[31]:
```

	reviewerID	recommendations
0	A00414041RD0BXM6WK0GX	[(B00161IKD6, 4.968151092529297), (B000P8559S,...
1	A00473363TJ8YSZ3YAGG9	[(B001FO2GW0, 4.339757919311523), (B006J6R23M,...
2	A00700212KB3K0MVE SPIY	[(B000ALBJ40, 6.009731292724609), (B00H8JPMX6,...
3	A0078719IR14X3NNUGQF	[(B000PHP8L4, 7.588184833526611), (B000ALBJ40,...
4	A01198201H0E3GHV2Z17I	[(B00HHECHLC, 6.440869331359863), (B000VOHH56,...
...	...	...
22356	AZZNK89PXD006	[(B0009OAHQY, 3.877105236053467), (B000052YMG,...
22357	AZZQXL8VDCFTV	[(B001CB2OQQ, 6.011270523071289), (B0013YYNDM,...
22358	AZZT1ERHBSNQ8	[(B000C1ZFBG, 6.335065841674805), (B0013YYNDM,...
22359	AZZU6NXB8YJN9	[(B0042PE8LQ, 4.932450771331787), (B00161IKD6,...
22360	AZZZLM1E5JJ8C	[(B00HB831SM, 5.787951946258545), (B00HAPQT7Q,...

22361 rows × 2 columns

```
In [32]: res_new.to_csv("beauty.csv")
```



## Chapter 9: Frequent Pattern Mining

### Ex5: instacart\_online\_grocery\_shopping\_2017\_05\_01

Dataset: link download <https://www.instacart.com/datasets/grocery-shopping-2017>  
(<https://www.instacart.com/datasets/grocery-shopping-2017>)

#### Requirement:

- Read data
- Pre-process data
- Apply FP-Growth algorithm to find association rules from this dataset. Find the most popular items in a basket.

```
In [1]: import findspark  
findspark.init()
```

```
In [2]: import pyspark
```

```
In [3]: from pyspark import SparkContext  
from pyspark.conf import SparkConf  
from pyspark.sql import SparkSession
```

```
[4]: sc = SparkContext()
```

```
In [5]: spark = SparkSession.builder.appName('ex_demo').getOrCreate()
```

```
In [6]: from pyspark.ml.fpm import FP_Growth
```

```
In [7]: # Loads data.  
data = spark.read.csv('instacart_2017_05_01/order_products_train.csv',  
header=True,  
inferSchema=True)
```

```
In [8]: data.count()
```

```
Out[8]: 1384617
```



In [9]: `data.show()`

<code>order_id</code>	<code>product_id</code>	<code>add_to_cart_order</code>	<code>reordered</code>
1	49302	1	1
1	11109	2	1
1	10246	3	0
1	49683	4	0
1	43633	5	1
1	13176	6	0
1	47209	7	0
1	22035	8	1
36	39612	1	0
36	19660	2	1
36	49235	3	0
36	43086	4	1
36	46620	5	1
36	34497	6	1
36	48679	7	1
36	46979	8	1
38	11913	1	0
38	18159	2	0
38	4461	3	0
38	21616	4	1

only showing top 20 rows

In [10]: `# Pre-processing data`  
`from pyspark.sql.functions import collect_list, col, count, collect_set`

In [11]: `data.createOrReplaceTempView("order_products_train")`

In [12]: `products = spark.sql("select distinct product_id from order_products_train")`  
`products.count()`

Out[12]: 39123

In [13]: `rawData = spark.sql("select * from order_products_train")`  
`baskets = rawData.groupBy('order_id').agg(collect_set('product_id')\`  
`.alias('items'))`  
`baskets.createOrReplaceTempView('baskets')`



```
In [14]: baskets.show(5, truncate=False)
```

order_id	items
1342	[30827, 3798, 14966, 21137, 46129, 33081, 13176, 7862]
1591	[48246, 44116, 24852, 5194, 9130, 48823, 46473, 40310, 32520, 22105, 16900, 27681, 4103, 44008, 17758, 41671, 25316, 45061, 38805, 48205, 25237, 19604, 5384, 27344, 17203, 18792, 12986, 39758, 34358, 31215, 9387]
4519	[29270]
4935	[45190]
6357	[33731, 14669, 43789, 37524, 39408, 43129, 24852, 48745, 38772]

only showing top 5 rows

```
In [15]: type(baskets)
```

```
Out[15]: pyspark.sql.dataframe.DataFrame
```

```
In [16]: fpGrowth = FP_Growth(itemsCol="items", minSupport=0.003,
                           minConfidence=0.003)
model = fpGrowth.fit(baskets)
```



In [17]: # Display frequent itemsets.  
model.freqItemsets.show()

items	freq
[13629]	772
[5194]	475
[24852]	18726
[13176]	15480
[35921]	769
[20345]	473
[21137]	10894
[21137, 13176]	3074
[21137, 24852]	2174
[23165]	764
[13380]	473
[7969]	472
[21903]	9784
[21903, 21137]	1639
[21903, 21137, 13...]	587
[21903, 13176]	2236
[21903, 24852]	2000
[32478]	763
[47626]	8135
[47626, 21137]	1017

only showing top 20 rows

In [18]: # transform examines the input items against all the association rules and summarizes consequents as prediction  
mostPopularItemInABasket = model.transform(baskets)



3/16/2020

## Ex5\_FPM\_store - Jupyter Notebook

In [19]: `mostPopularItemInABasket.show()`

order_id	items	prediction
1342	[30827, 3798, 149...]	[21903, 47626, 47...]
1591	[48246, 44116, 24...]	[21137, 21903, 47...]
4519		[29270]
4935		[45190]
6357	[33731, 14669, 43...]	[21137, 21903, 47...]
10362	[28522, 43789, 12...]	[21137, 47626, 47...]
19204	[45255, 37285, 48...]	[2716, 48057, 219...]
29601		[21137, 21903, 47...]
31035	[40723, 8174, 131...]	[21137, 21903, 47...]
40011	[27292, 35213, 21...]	[27292, 35213, 21...]
46266	[38558, 48642, 13...]	[21137, 13176, 24...]
51607	[41390, 42752, 17...]	[47626, 47766, 47...]
58797	[30827, 8803, 326...]	[21137, 21903, 47...]
61793	[26348, 6184, 433...]	[21137, 16797, 39...]
67089	[47766, 29388, 21...]	[47766, 29388, 21...]
70863	[34791, 2618, 173...]	[47626, 21137, 47...]
88674	[25659, 16262, 22...]	[13176, 24852]
91937	[20708, 38200, 26...]	[20708, 38200, 26...]
92317	[18105, 34969, 17...]	[18105, 34969, 17...]
99621	[21616, 43789, 38...]	[21616, 43789, 38...]

only showing top 20 rows

## Use product\_name instead of product\_id

In [20]: `product_data = spark.read.csv('instacart_2017_05_01/products.csv', header=True, inferSchema=True)`



In [21]: `product_data.show(5, truncate=False)`

product_id	product_name	isle_id	department_id	a
1	Chocolate Sandwich Cookies	19	1	6
2	All-Seasons Salt	13	1	1
3	Robust Golden Unsweetened Oolong Tea	7	1	9
4	Smart Ones Classic Favorites Mini Rigatoni With Vodka Cream Sauce	11	3	3
5	Green Chile Anytime Sauce	1	5	5
13				

only showing top 5 rows

In [22]: `product_data.createOrReplaceTempView("products")`

```
In [23]: rawData_1 = spark.sql('''select p.product_name, o.order_id from products p
inner join order_products_train o
where o.product_id = p.product_id''')
baskets_1 = rawData_1.groupBy('order_id').agg(collect_set('product_name')\.
alias('items'))
baskets_1.createOrReplaceTempView('baskets')
```

In [24]: `baskets_1.head(3)`

```
Out[24]: [Row(order_id=1342, items=['Raw Shrimp', 'Seedless Cucumbers', 'Versatile Stain
Remover', 'Organic Strawberries', 'Organic Mandarins', 'Chicken Apple Sausage',
'Pink Lady Apples', 'Bag of Organic Bananas']),
Row(order_id=1591, items=['Cracked Wheat', 'Strawberry Rhubarb Yoghurt', 'Orga
nic Bunny Fruit Snacks Berry Patch', 'Goodness Grapeness Organic Juice Drink',
'Honey Graham Snacks', 'Spinach', 'Granny Smith Apples', 'Oven Roasted Turkey B
reast', 'Pure Vanilla Extract', 'Chewy 25% Low Sugar Chocolate Chip Granola',
'Banana', 'Original Turkey Burgers Smoke Flavor Added', 'Twisted Tropical Tango
Organic Juice Drink', 'Navel Oranges', 'Lower Sugar Instant Oatmeal Variety',
'Ultra Thin Sliced Provolone Cheese', 'Natural Vanilla Ice Cream', 'Cinnamon Mu
ltigrain Cereal', 'Garlic', 'Goldfish Pretzel Baked Snack Crackers', 'Original
Whole Grain Chips', 'Medium Scarlet Raspberries', 'Lemon Yogurt', 'Original Pat
ties (100965) 12 Oz Breakfast', 'Nutty Bars', 'Strawberry Banana Smoothie', 'Gr
een Machine Juice Smoothie', 'Coconut Dreams Cookies', 'Buttermilk Waffles', 'U
ncured Genoa Salami', 'Organic Greek Whole Milk Blended Vanilla Bean Yogurt']),
Row(order_id=4519, items=['Beet Apple Carrot Lemon Ginger Organic Cold Pressed
Juice Beverage'])]
```

## Ex5\_FPM\_store - Jupyter Notebook



```
In [25]: fpGrowth_1 = FP_Growth(itemsCol="items",
                           minSupport=0.003,
                           minConfidence=0.003)
model_1 = fpGrowth_1.fit(baskets_1)
```

```
In [26]: # Display frequent itemsets.
model_1.freqItemsets.show(truncate=False)
```

items	freq
[[Organic Tomato Basil Pasta Sauce]]	772
[[Organic Spinach Bunch]]	475
[[Banana]]	18726
[[Bag of Organic Bananas]]	15480
[[Organic Large Grade A Brown Eggs]]	769
[[Organic Blue Corn Tortilla Chips]]	473
[[Organic Strawberries]]	10894
[[Organic Strawberries, Bag of Organic Bananas]]	3074
[[Organic Strawberries, Banana]]	2174
[[Organic Leek]]	764
[[Thin Crust Pepperoni Pizza]]	473
[[Lime]]	472
[[Organic Baby Spinach]]	9784
[[Organic Baby Spinach, Organic Strawberries]]	1639
[[Organic Baby Spinach, Organic Strawberries, Bag of Organic Bananas]]	587
[[Organic Baby Spinach, Bag of Organic Bananas]]	2236
[[Organic Baby Spinach, Banana]]	2000
[[Reduced Fat 2% Milk]]	763
[[Large Lemon]]	8135
[[Large Lemon, Organic Strawberries]]	1017

only showing top 20 rows

```
In [27]: mostPopularItemInABasket_1 = model_1.transform(baskets_1)
```



In [28]: `mostPopularItemInABasket_1.head(3)`

```
Out[28]: [Row(order_id=1342, items=['Raw Shrimp', 'Seedless Cucumbers', 'Versatile Stain Remover', 'Organic Strawberries', 'Organic Mandarins', 'Chicken Apple Sausage', 'Pink Lady Apples', 'Bag of Organic Bananas'], prediction=['Organic Baby Spinac h', 'Large Lemon', 'Organic Avocado', 'Organic Hass Avocado', 'Strawberries', 'Limes', 'Organic Raspberries', 'Organic Cucumber', 'Organic Yellow Onion', 'Organic Garlic', 'Organic Zucchini', 'Organic Grape Tomatoes', 'Organic Red Onion', 'Seedless Red Grapes', 'Asparagus', 'Organic Cilantro', 'Organic Lemon', 'Organic Baby Carrots', 'Honeycrisp Apple', 'Organic Fuji Apple', 'Small Mon', 'Sparkling Water Grapefruit', 'Raspberries', 'Organic Fuji Apple', 'Small Hass Avocado', 'Organic Baby Arugula', 'Organic Large Extra Fancy Fuji Apple', 'Original Hummus', 'Organic Blackberries', 'Organic Gala Apples', 'Fresh Cauliflower', 'Organic Half & Half', 'Michigan Organic Kale', 'Organic Small Bunch Celery', 'Organic Garnet Sweet Potato (Yam)', 'Organic Tomato Cluster', 'Carrot s', 'Organic Peeled Whole Baby Carrots', 'Organic Italian Parsley Bunch', 'Organic Red Bell Pepper', 'Organic Granny Smith Apple', 'Hass Avocados', 'Apple Honeycrisp Organic', 'Spring Water', 'Organic Unsweetened Almond Milk', 'Unsweetened Almondmilk', 'Organic Ginger Root', 'Organic Whole String Cheese', 'Organic Navel Orange', 'Large Alfresco Eggs', 'Organic D'Anjou Pears', 'Organic Kiwi', 'Organic Grade A Free Range Large Brown Eggs', 'Organic Lacinato (Dinosaur) Kale', 'Organic Carrot Bunch', 'Organic Broccoli', 'Organic Black Beans', 'Banana', 'Broccoli Crown', 'Organic Banana']), Row(order_id=1591, items=['Cracked Wheat', 'Strawberry Rhubarb Yoghurt', 'Organic Bunny Fruit Snacks Berry Patch', 'Goodness Grapeness Organic Juice Drink', 'Honey Graham Snacks', 'Spinach', 'Granny Smith Apples', 'Oven Roasted Turkey Breast', 'Pure Vanilla Extract', 'Chewy 25% Low Sugar Chocolate Chip Granola', 'Banana', 'Original Turkey Burgers Smoke Flavor Added', 'Twisted Tropical Tango Organic Juice Drink', 'Navel Oranges', 'Lower Sugar Instant Oatmeal Variety', 'Ultra Thin Sliced Provolone Cheese', 'Natural Vanilla Ice Cream', 'Cinnamon Multigrain Cereal', 'Garlic', 'Goldfish Pretzel Baked Snack Crackers', 'Original Whole Grain Chips', 'Medium Scarlet Raspberries', 'Lemon Yogurt', 'Original Patties (100965) 12 Oz Breakfast', 'Nutty Bars', 'Strawberry Banana Smoothie', 'Green Machine Juice Smoothie', 'Coconut Dreams Cookies', 'Buttermilk Waffles', 'Uncured Genoa Salami', 'Organic Greek Whole Milk Blended Vanilla Bean Yogurt'], prediction=['Organic Strawberries', 'Organic Baby Spinach', 'Large Lemon', 'Organic Hass Avocado', 'Organic Hass Avocado', 'Strawberries', 'Limes', 'Organic Raspberries', 'Organic Blueberries', 'Organic Whole Milk', 'Organic Cucumber', 'Organic Zucchini', 'Organic Yellow Onion', 'Organic Garlic', 'Seedless Red Grapes', 'Asparagus', 'Organic Grape Tomatoes', 'Organic Red Onion', 'Yellow Onions', 'Organic Baby Carrots', 'Honeycrisp Apple', 'Organic Cilantro', 'Sparkling Water Grapefruit', 'Raspberries', 'Organic Fuji Apple', 'Small Hass Avocado', 'Broccoli Crown', 'Organic Baby Arugula', 'Red Peppers', 'Organic Large Extra Fancy Fuji Apple', 'Original Hummus', 'Organic Blackberries', 'Organic Gala Apples', 'Fresh Cauliflower', 'Organic Half & Half', 'Michigan Organic Kale', 'Organic Small Bunch Celery', 'Organic Garnet Sweet Potato (Yam)', 'Organic Tomato Cluster', 'Carrot s', 'Organic Peeled Whole Baby Carrots', 'Organic Italian Parsley Bunch', 'Organic Red Bell Pepper', 'Organic Granny Smith Apple', 'Hass Avocados', 'Apple Honeycrisp Organic', 'Spring Water', 'Organic Unsweetened Almond Milk', 'Unsweetened Almondmilk', 'Organic Ginger Root', 'Organic Whole String Cheese', 'Organic Navel Orange', 'Large Alfresco Eggs', 'Organic D'Anjou Pears', 'Organic Kiwi', 'Organic Grade A Free Range Large Brown Eggs', 'Organic Lacinato (Dinosaur) Kale', 'Organic Carrot Bunch', 'Organic Broccoli', 'Organic Black Beans', 'Banana', 'Broccoli Crown', 'Organic Banana']), Row(order_id=4519, items=['Beet Apple Carrot Lemon Ginger Organic Cold Pressed Juice Beverage'], prediction=[])]
```



3/6/2020 Ex5\_FPM\_store - Jupyter Notebook

In [29]: type(mostPopularItemInABasket\_1)  
Out[29]: pyspark.sql.dataframe.DataFrame

In [30]: # chuyển list array thành string  
from pyspark.sql.types import StringType

In [31]: mostPopularItemInABasket\_1.printSchema()

```
root
|-- order_id: integer (nullable = true)
|-- items: array (nullable = true)
|   |-- element: string (containsNull = true)
|-- prediction: array (nullable = true)
|   |-- element: string (containsNull = true)
```

In [33]: mostPopularItemInABasket\_1.createOrReplaceTempView("popular\_items")

In [34]: DF\_cast = mostPopularItemInABasket\_1.select('order\_id',
 mostPopularItemInABasket\_1.items.cast(StringType()),
 mostPopularItemInABasket\_1.prediction.cast(StringType()))

DF\_cast.printSchema()

```
root
|-- order_id: integer (nullable = true)
|-- items: string (nullable = true)
|-- prediction: string (nullable = true)
```

localhost:8888/notebooks/Chapter9/Ex5\_FPM\_store.ipynb 8/11



In [35]: DF\_cast.head(3)

```
Out[35]: [Row(order_id=1342, items='[Raw Shrimp, Seedless Cucumbers, Versatile Stain Remover, Organic Strawberries, Organic Mandarins, Chicken Apple Sausage, Pink Lady Apples, Bag of Organic Bananas]', prediction="Organic Baby Spinach, Large Lemon, Organic Avocado, Organic Hass Avocado, Strawberries, Limes, Organic Raspberries, Organic Blueberries, Organic Whole Milk, Organic Cucumber, Organic Zucchini, Organic Yellow Onion, Organic Garlic, Seedless Red Grapes, Asparagus, Organic Grape Tomatoes, Organic Red Onion, Organic Baby Carrots, Honeycrisp Apple, Organic Cilantro, Organic Lemon, Sparkling Water Grapefruit, Raspberries, Organic Fuji Apple, Small Hass Avocado, Organic Baby Arugula, Organic Large Extra Fancy Fuji Apple, Original Hummus, Organic Blackberries, Organic Gala Apples, Fresh Cauliflower, Organic Half & Half, Michigan Organic Kale, Organic Small Bunch Celery, Organic Garnet Sweet Potato (Yam), Organic Tomato Cluster, Carrots, Organic Peeled Whole Baby Carrots, Organic Italian Parsley Bunch, Organic Red Bell Pepper, Organic Granny Smith Apple, Hass Avocados, Apple Honeycrisp Organic, Spring Water, Organic Unsweetened Almond Milk, Unsweetened Almondmilk, Organic Ginger Root, Organic Whole String Cheese, Organic Navel Orange, Large Alfresco Eggs, Organic D'Anjou Pears, Organic Kiwi, Organic Grade A Free Range Large Brown Eggs, Organic Lacinato (Dinosaur) Kale, Organic Carrot Bunch, Organic Broccoli, Organic Black Beans, Banana, Broccoli Crown, Organic Banana]"),
Row(order_id=1591, items='[Cracked Wheat, Strawberry Rhubarb Yoghurt, Organic Bunny Fruit Snacks Berry Patch, Goodness Grapeness Organic Juice Drink, Honey Graham Snacks, Spinach, Granny Smith Apples, Oven Roasted Turkey Breast, Pure Vanilla Extract, Chewy 25% Low Sugar Chocolate Chip Granola, Banana, Original Turkey Burgers Smoke Flavor Added, Twisted Tropical Tango Organic Juice Drink, Naval Oranges, Lower Sugar Instant Oatmeal Variety, Ultra Thin Sliced Provolone Cheese, Natural Vanilla Ice Cream, Cinnamon Multigrain Cereal, Garlic, Goldfish Pretzel Baked Snack Crackers, Original Whole Grain Chips, Medium Scarlet Raspberries, Lemon Yogurt, Original Patties (100965) 12 Oz Breakfast, Nutty Bars, Strawbery Banana Smoothie, Green Machine Juice Smoothie, Coconut Dreams Cookies, Buttermilk Waffles, Uncured Genoa Salami, Organic Greek Whole Milk Blended Vanilla Bean Yogurt]', prediction="Organic Strawberries, Organic Baby Spinach, Large Lemon, Organic Avocado, Organic Hass Avocado, Strawberries, Limes, Organic Raspberries, Organic Blueberries, Organic Whole Milk, Organic Cucumber, Organic Zucchini, Organic Yellow Onion, Organic Garlic, Seedless Red Grapes, Asparagus, Organic Grape Tomatoes, Organic Red Onion, Yellow Onions, Organic Baby Carrots, Honeycrisp Apple, Organic Cilantro, Sparkling Water Grapefruit, Raspberries, Organic Fuji Apple, Small Hass Avocado, Broccoli Crown, Organic Baby Arugula, Red Peppers, Organic Large Extra Fancy Fuji Apple, Original Hummus, Organic Blackberries, Organic Gala Apples, Fresh Cauliflower, Organic Half & Half, Michigan Organic Kale, Organic Small Bunch Celery, Organic Garnet Sweet Potato (Yam), Organic Tomato Cluster, Green Bell Pepper, Carrots, Organic Peeled Whole Baby Carrots, Half & Half, Cucumber Kirby, Organic Red Bell Pepper, Organic Granny Smith Apple, Blueberries, 100% Whole Wheat Bread, Apple Honeycrisp Organic, Red Vine Tomato, Unsweetened Almondmilk, Boneless Skinless Chicken Breasts, Organic Whole String Cheese, Roma Tomato, Bunched Cilantro, Jalapeno Peppers, Organic D'Anjou Pears, Orange Bell Pepper, Grape White/Green Seedless, Red Raspberries, Clementines, Bag, Unsweetened Original Almond Breeze Almond Milk, Bartlett Pears]"),
Row(order_id=4519, items='[Beet Apple Carrot Lemon Ginger Organic Cold Pressed Juice Beverage]', prediction='[]')]
```

In [36]: DF\_cast.write.csv('mostPopularItemInABasket.csv')