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|rev
iewerName|             summary|unixReviewTime|
+----------+-------+-------+--------------------+-----------+--------------+---
---------+--------------------+--------------+
|7806397051| [3, 4]|    1.0|Very oily and cre...|01 30, 2014|A1YJEY40YUW4SE|
Andrea|Don't waste your ...|    1391040000|
|7806397051| [1, 1]|    3.0|This palette was ...|04 18, 2014| A60XNB876KYML|  J
essica 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
```

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| A60XNB876KYML|
|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
```

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)
```

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

```
Row(reviewerID_idx=1580, recommendations=[Row(asin_idx=9900, rating=7.452762603
759766), Row(asin_idx=7885, rating=7.445200443267822), Row(asin_idx=8386, ratin
g=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, ratin
g=6.905022621154785)])



Row(reviewerID_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(asi
n_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)])



Row(reviewerID_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(asi
n_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)])



Row(reviewerID_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(asi
n_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=7.043422222137451), 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, rati
ng=6.954199314117432), Row(asin_idx=9900, rating=6.791838645935059), Row(asin_i
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)])


Row(reviewerID_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=
8406, rating=5.722962379455566), Row(asin_idx=6098, rating=5.7152099609375), Ro
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, r
ating=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'
     ]
```

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.reviewerID])\
                                .apply(list).reset_index()
```

```
c:\program files\python36\lib\site-packages\ipykernel_launcher.py:10: SettingWi
thCopyWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  # 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	A00700212KB3K0MVESPIY	[(B000ALBJ40, 6.009731292724609), (B00H8JPMX6,...
3	A0078719IR14X3NNUG0F	[(B000PHP8L4, 7.588184833526611), (B000ALBJ40,...
4	A01198201H0E3GHV2Z17I	[(B00HHECHLC, 6.440869331359863), (B000VOHH56,...
...	...	...
22356	AZZNK89PXD006	[(B0009OAHQY, 3.877105236053467), (B000052YMG,...
22357	AZZQXL8VDCFTV	[(B001CB2OQO, 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")
```