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|A2IBPI20UZIR0U|ca
ssandra tu "Yea...|                good|    1393545600|
|1384719342|[13, 14]|    5.0|The product does ...|03 16, 2013|A14VAT5EAX3D9S|
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|A2C00NNG1ZQQG2|Ru
styBill "Sunday...|GOOD WINDSCREEN F...|    1392336000|
|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
```

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|A2C00NNG1ZQQG2|
|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*1.0 / 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|A195EZSQDW3E21|   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
```

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)
```

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

```
Row(reviewerID_idx=471, recommendations=[Row(asin_idx=357, rating=5.994929790496826), 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=5.702645301818848), Row(asin_idx=360, rating=5.70063591003418), Row(asin_idx=427, rating=5.692359924316406), Row(asin_idx=346, rating=5.683685302734375), Row(asin_idx=746, rating=5.678509712219238), Row(asin_idx=829, rating=5.665043830871582), Row(asin_idx=368, rating=5.654289245605469), Row(asin_idx=805, rating=5.633547782897949), Row(asin_idx=350, rating=5.57622766494751), Row(asin_idx=723, rating=5.573838233947754), Row(asin_idx=396, rating=5.543375492095947), Row(asin_idx=632, rating=5.540017127990723), Row(asin_idx=734, rating=5.527731895446777)])


Row(reviewerID_idx=1342, recommendations=[Row(asin_idx=731, rating=6.399199485778809), Row(asin_idx=427, rating=6.240170001983643), Row(asin_idx=746, rating=6.126894950866699), Row(asin_idx=328, rating=6.101090431213379), Row(asin_idx=232, rating=6.0971221923828125), Row(asin_idx=852, rating=6.052910327911377), Row(asin_idx=476, rating=6.034964561462402), Row(asin_idx=645, rating=6.031756877789917), Row(asin_idx=544, rating=6.0315022468566895), Row(asin_idx=829, rating=6.031264781951904), Row(asin_idx=682, rating=6.015640735626221), Row(asin_idx=853, rating=6.006399154663086), Row(asin_idx=734, rating=6.0043182373046875), Row(asin_idx=346, rating=5.98906135559082), Row(asin_idx=723, rating=5.986504077911377), 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)])


Row(reviewerID_idx=463, recommendations=[Row(asin_idx=746, rating=5.707493305206299), Row(asin_idx=476, rating=5.547768592834473), Row(asin_idx=544, rating=5.540889739990234), Row(asin_idx=368, rating=5.476436614990234), Row(asin_idx=805, rating=5.46685266494751), Row(asin_idx=731, rating=5.438949108123779), Row(asin_idx=465, rating=5.411510944366455), Row(asin_idx=427, rating=5.398251056671143), Row(asin_idx=698, rating=5.384541988372803), Row(asin_idx=853, rating=5.3731842041015625), Row(asin_idx=306, rating=5.3723273277282715), Row(asin_idx=183, rating=5.372140407562256), Row(asin_idx=664, rating=5.357931613922119), Row(asin_idx=232, rating=5.35223913192749), Row(asin_idx=328, rating=5.336521148681641), 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)])


Row(reviewerID_idx=833, recommendations=[Row(asin_idx=734, rating=6.090943336486816), Row(asin_idx=476, rating=5.975735187530518), Row(asin_idx=465, rating=5.958075523376465), Row(asin_idx=746, rating=5.939692974090576), Row(asin_idx=368, rating=5.900303840637207), Row(asin_idx=437, rating=5.891534328460693), Row(asin_idx=396, rating=5.8877973556518555), Row(asin_idx=853, rating=5.885824680328369), Row(asin_idx=427, rating=5.86740779876709), Row(asin_idx=402, rating=5.852911949157715), Row(asin_idx=888, rating=5.833059310913086), Row(asin_idx=684, rating=5.827024936676025), Row(asin_idx=791, rating=5.814032554626465), Row(asin_idx=705, rating=5.810422420501709), Row(asin_idx=645, rating=5.807799816131592), 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)])

Row(reviewerID_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=5.279950141906738), 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'
    ]
```

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: 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 [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