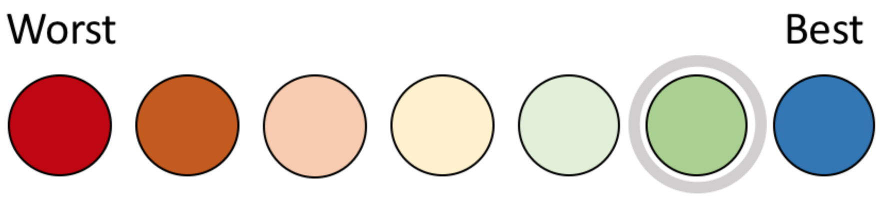
BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Introduction to the Million Songs Dataset

Jamen Long
Data Scientist

Explicit vs Implicit

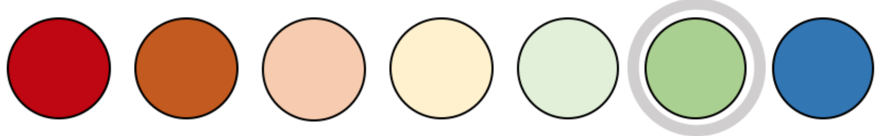
Explicit Ratings



Explicit vs Implicit (cont.)

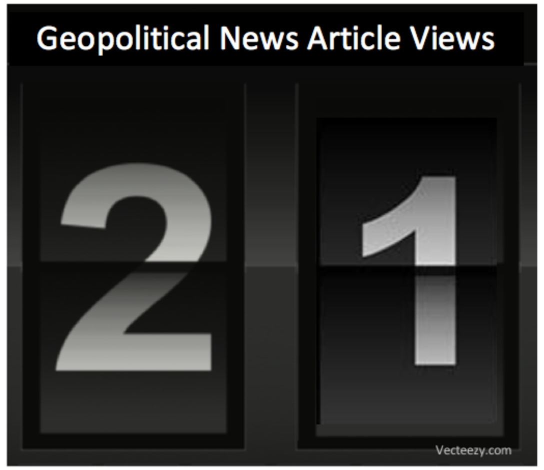
Explicit Ratings

Worst Best



Implicit Ratings

= Low Confidence Rating

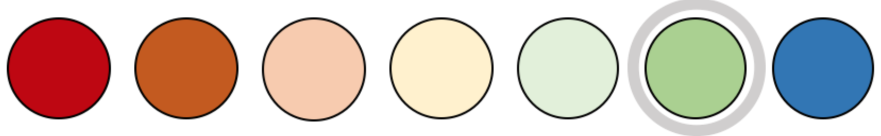


= High Confidence Rating

Implicit Refresher II

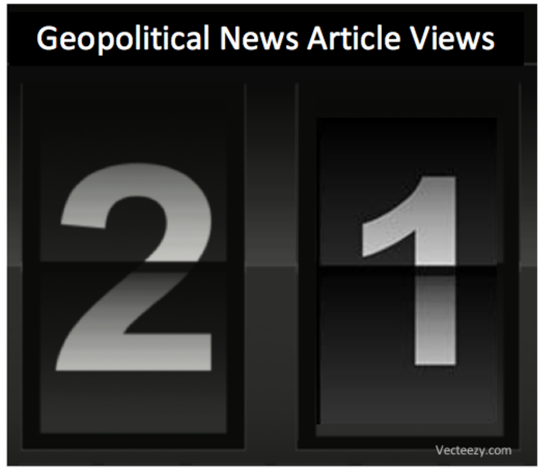
Explicit Ratings

Worst Best



Implicit Ratings

= Low Confidence Rating



= High Confidence Rating

Collaborative Filerting for Implicit Feedback Datasets
Yifan Hu, Yehuda Koren and Chris Volinsky

THE ECHO NEST TASTE PROFILE DATASET

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (SIMIR 20122), 2011.

Add Zeros Sample

```
ratings.show()
```

```
+------+------+---------+
|userId|songId|num_plays|
+------+------+---------+
|    10|    22|        5|
|    38|    99|        1|
|    38|    77|        3|
|    42|    99|        1|
+------+------+---------+
```

Cross Join Intro

```
users = ratings.select("userId").distinct()
users.show()
```

```
+-------+
|userId|
+-------+
|     10|
|     38|
|     42|
+-------+
```

```
songs = ratings.select("songId").distinct()
songs.show()
```

```
+-------+
|songId|
+-------+
|     22|
|     77|
|     99|
+-------+
```

Cross Join Output

```
cross_join = users.crossJoin(songs)
cross_join.show()
```

```
+------+------+
|userId|songId|
+------+------+
|    10|    22|
|    10|    77|
|    10|    99|
|    38|    22|
|    38|    77|
|    38|    99|
|    42|    22|
|    42|    77|
|    42|    99|
+------+------+
```

Joining Back Original Ratings Data

```
cross_join = users.crossJoin(songs)
                  .join(ratings, ["userId", "songId"], "left")
cross_join.show()
```

```
+------+------+---------+
|userId|songId|num_plays|
+------+------+---------+
|    10|    22|        5|
|    10|    77|     null|
|    10|    99|     null|
|    38|    22|     null|
|    38|    77|        3|
|    38|    99|        1|
|    42|    22|     null|
|    42|    77|     null|
|    42|    99|        1|
+------+------+---------+
```

Filling In With Zero

```
cross_join = users.crossJoin(songs)
                  .join(ratings, ["userId", "songId"], "left").fillna(0)
cross_join.show()
```

```
+------+------+---------+
|userId|songId|num_plays|
+------+------+---------+
|    10|    22|        5|
|    10|    77|        0|
|    10|    99|        0|
|    38|    22|        0|
|    38|    77|        3|
|    38|    99|        1|
|    42|    22|        0|
|    42|    77|        0|
|    42|    99|        1|
+------+------+---------+
```

Add Zeros Function

```
def add_zeros(df):
    # Extracts distinct users
    users = df.select("userId").distinct()

    # Extracts distinct songs
    songs = df.select("songId").distinct()

    # Joins users and songs, fills blanks with 0
    cross_join = users.crossJoin(items) \
            .join(df, ["userId", "songId"], "left").fillna(0)

    return cross_join
```

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Let's practice!

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Evaluating Implicit Ratings Models

Jamen Long
Data Scientist

Why RMSE worked before

userId	movieId	rating	explicit rating prediction
1	2112	5	4.88
1	303	3.5	3.96
2	5	3	2.78
2	77	2	2.89
3	913	1.5	2.11
3	44	4	3.56
3	6	4.5	4.67

Predictions reflect actual ratings.
RMSE makes sense here.

Why RMSE doesn't work now

userId	movieId	num_plays	implicit rating prediction
1	2112	16	1.755
1	303	3	.88
2	5	1	.01
2	77	2	.5
3	913	1	.08
3	44	21	1.98
3	6	4	.98

Different metrics.
RMSE doesn't make sense here.

(ROEM) Rank Ordering Error Metric

$$\mathrm{ROEM} = \frac{\sum_{u,i} r^t_{u,i} \mathrm{rank}_{u,i}}{\sum_{u,i} r^t_{u,i}}$$

ROEM Bad Predictions

```
bad_prediction.show()
```

```
+--------+------+-----+--------+--------+
|userId  |songId|plays|badPreds|percRank|
+--------+------+-----+--------+--------+
|     111|    22|    3|  0.0001|   1.000|
|     111|     9|    0|   0.999|   0.000|
|     111|   321|    0|    0.08|   0.500|
|     222|    84|    0|0.000003|   1.000|
|     222|   821|    2|    0.88|   0.000|
|     222|    91|    2|    0.73|   0.500|
|     333|  2112|    0|    0.90|   0.000|
|     333|    42|    2|    0.80|   0.500|
|     333|     6|    0|    0.01|   1.000|
+--------+------+-----+--------+--------+
```

ROEM: PercRank * Plays

```
bp = bad_predictions.withColumn("np*rank",
                                col("badPreds")*col("percRank"))
bp.show()
```

```
+-------+------+---------+--------+--------+-------+
|userId |songId|num_plays|badPreds|percRank|np*rank|
+-------+------+---------+--------+--------+-------+
|    111|    22|        3|  0.0001|   1.000|   3.00|
|    111|     9|        0|   0.999|   0.000|   0.00|
|    111|   321|        0|    0.08|   0.500|   0.00|
|    222|    84|        0|0.000003|   1.000|   0.00|
|    222|   821|        2|    0.88|   0.000|   0.00|
|    222|    91|        2|    0.73|   0.500|   1.00|
|    333|  2112|        0|    0.90|   0.000|   0.00|
|    333|    42|        2|    0.80|   0.500|   1.00|
|    333|     6|        0|    0.01|   1.000|   0.00|
+-------+------+---------+--------+--------+-------+
```

ROEM: Bad Predictions

```
+-------+------+---------+--------+--------+-------+
|userId |songId|num_plays|badPreds|percRank|np*rank|
+-------+------+---------+--------+--------+-------+
|    111|    22|        3|  0.0001|   1.000|   3.00|
|    111|     9|        0|   0.999|   0.000|   0.00|
|    111|   321|        0|    0.08|   0.500|   0.00|
|    222|    84|        0|0.000003|   1.000|   0.00|
|    222|   821|        2|    0.88|   0.000|   0.00|
|    222|    91|        2|    0.73|   0.500|   1.00|
|    333|  2112|        0|    0.90|   0.000|   0.00|
|    333|    42|        2|    0.80|   0.500|   1.00|
|    333|     6|        0|    0.01|   1.000|   0.00|
+-------+------+---------+--------+--------+-------+
```

```
numerator = bp.groupBy().sum("np*rank").collect()[0][0]
denominator = bp.groupBy().sum("num_plays").collect()[0][0]

print ("ROEM: "), numerator * 1.0/ denominator
```

```
ROEM: 5.0 / 9 = 0.556
```

Good Predictions

```
gp = good_predictions.withColumn("np*rank",
                                  col("goodPreds")*col("percRank"))
gp.show()
```

```
+-------+------+---------+---------+--------+-------+
|userId |songId|num_plays|goodPreds|percRank|np*rank|
+-------+------+---------+---------+--------+-------+
|    111|    22|        3|      1.1|   0.000|  0.000|
|    111|    77|        0|     0.01|   0.500|  0.000|
|    111|    99|        0|    0.008|   1.000|  0.000|
|    222|    22|        0|   0.0003|   1.000|  0.000|
|    222|    77|        2|      1.5|   0.000|  0.000|
|    222|    99|        2|      1.4|   0.500|  1.000|
|    333|    22|        0|     0.90|   0.500|  0.000|
|    333|    77|        2|      1.6|   0.000|  0.000|
|    333|    99|        0|     0.01|   1.000|  0.000|
+-------+------+---------+---------+--------+-------+
```

ROEM: Good Predictions

```
+-------+------+---------+---------+--------+-------+
|userId |songId|num_plays|goodPreds|percRank|np*rank|
+-------+------+---------+---------+--------+-------+
|    111|    22|        3|      1.1|   0.000|  0.000|
|    111|    77|        0|     0.01|   0.500|  0.000|
|    111|    99|        0|    0.008|   1.000|  0.000|
|    222|    22|        0|   0.0003|   1.000|  0.000|
|    222|    77|        2|      1.5|   0.000|  0.000|
|    222|    99|        2|      1.4|   0.500|  1.000|
|    333|    22|        0|     0.90|   0.500|  0.000|
|    333|    77|        2|      1.6|   0.000|  0.000|
|    333|    99|        0|     0.01|   1.000|  0.000|
+-------+------+---------+---------+--------+-------+
```

```
numerator = gp.groupBy().sum("np*rank").collect()[0][0]
denominator = gp.groupBy().sum("num_plays").collect()[0][0]
print ("ROEM: "), numerator * 1.0/ denominator
```

```
ROEM: 1.0 / 9 = 0.1111
```

ROEM: Link to Function on GitHub

```
+-------+------+---------+---------+--------+-------+
|userId |songId|num_plays|goodPreds|percRank|np*rank|
+-------+------+---------+---------+--------+-------+
|    111|    22|        3|      1.1|   0.000|  0.000|
|    111|    77|        0|     0.01|   0.500|  0.000|
|    111|    99|        0|    0.008|   1.000|  0.000|
|    222|    22|        0|   0.0003|   1.000|  0.000|
|    222|    77|        2|      1.5|   0.000|  0.000|
|    222|    99|        2|      1.4|   0.500|  1.000|
|    333|    22|        0|     0.90|   0.500|  0.000|
|    333|    77|        2|      1.6|   0.000|  0.000|
|    333|    99|        0|     0.01|   1.000|  0.000|
+-------+------+---------+---------+--------+-------+
```

```
numerator = gp.groupBy().sum("np*rank").collect()[0][0]
denominator = gp.groupBy().sum("num_plays").collect()[0][0]
print ("ROEM: "), numerator * 1.0/ denominator
```

```
ROEM: 1.0 / 9 = 0.1111
```

Building Several ROEM Models

```
(train, test) = implicit_ratings.randomSplit([.8, .2])

# Empty list to be filled with models
model_list = []

# Complete each of the hyperparameter value lists
ranks = [10, 20, 30, 40]
maxIters = [10, 20, 30, 40]
regParams = [.05, .1, .15]
alphas = [20, 40, 60, 80]

# For loop will automatically create and store ALS models
for r in ranks:
    for mi in maxIters:
        for rp in regParams:
            for a in alphas:
                model_list.append(ALS(userCol= "userId", itemCol= "songId",
                ratingCol= "num_plays", rank = r, maxIter = mi, regParam = rp,
                alpha = a, coldStartStrategy="drop",nonnegative = True,
                implicitPrefs = True))
```

Error Output

```
for model in model_list:
    # Fits each model to the training data
    trained_model = model.fit(train)

    # Generates test predictions
    predictions = trained_model.transform(test)

    # Evaluates each model's performance
    ROEM(predictions)
```

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Let's practice!

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Overview of binary, implicit ratings

Jamen Long
Data Scientist

Binary Ratings

```
binary_movie_ratings.show()
```

```
+------+-------+-------------+
|userId|movieId|binary_rating|
+------+-------+-------------+
|    26|    474|            0|
|    26|   2529|            1|
|    26|     26|            0|
|    26|   1950|            0|
|    26|   4823|            1|
|    26|  72011|            1|
|    26| 142507|            0|
|    26|     29|            0|
|    26|   5385|            0|
|    26|   3506|            0|
|    38|   2112|            1|
|    38|     42|            0|
|    38|     17|            0|
|    38|   1325|            0|
|    38|   6011|            1|
+------+-------+-------------+
```

Class Imbalance

```
getSparsity(binary_ratings)
```

```
Sparsity: .993
```

Item Weighting

- **Item Weighting:** Movies with more user views = higher weight

Item Weighting and User Weighting

- **Item Weighting:** Movies with more user views = higher weight
- **User Weighting:** Users that have seen more movies will have lower weights applied to unseen movies

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Let's practice!

BUILDING RECOMMENDATION ENGINES WITH PYSPARK

Course Recap

Jamen Long
Data Scientist

THREE TYPES OF DATA

- Explicit Ratings
- Implicit Ratings using user behavior counts
- Implicit Ratings using binary user behavior

THINGS TO BEAR IN MIND

- The more data the better

THINGS TO BEAR IN MIND

- The more data the better
- The best model evaluation is whether actual users take your recommendations

Resources

- McKinsey&Company: "How Retailers Can Keep Up With Consumers"
- ALS Data Preparation: Wide to Long Function
- Hu, Koren, Volinsky: "Collaborative Filtering for Implicit Feedback Datasets"
- GitHub Repo: Cross Validation With Implicit Ratings in Pyspark
- Pan, Zhou, Cao, Liu, Lukose, Scholz, Yang: "One Class Collaborative Filtering"