Recommender System

Challenges

- Provide good user experience to every user
- Satisfy the business objectives
- Measure its impact
- Scaling
- Expired items
- Cold start - new users, new products
- Sparsity of user preference

Types of Recommenders

- Content based
  - using explicit features of the users and/or items
- Collaborative filtering
  - implicit features
  - based on observed interactions rather than metadata
- Hybrid

Alternating Least Square(ALS) - Method of Choice

There are many techniques for generating recommendations, such as

- Matrix factorization
- co-occurrence analysis
- content based filtering
- graph based algorithms
- hybrids

In our case, my dataset mainly contains only implicit features. I choose ALS recommender, which is a widely popular matrix factorization algorithm that uses alternating least squares with weighted lambda regularization. It factors the user-to-item matrix into the user-to-feature matrix and the item-to-feature matrix. This recommendation was also successfully used in the Netflix competition.

One of the big strengths of ALS based recommender, compared to the user or item based recommender, is its ability to handle large sparse data sets and its better prediction performance. It is best suited for our datasets.

In [1]:

```
from pyspark import SparkContext
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
from pyspark.mllib.linalg.distributed import CoordinateMatrix, MatrixEntry
from pyspark.sql import Row, SQLContext
from pyspark.sql.functions import lit, udf
import pandas as pd
import numpy as np
```

executed in 166ms, finished 21:05:01 2017-03-27

The data file contains the following 3 columns:

- timestamp
- user_id
- product_id

I drop the timeaxis, which is not relevant for my model

In [2]:

```
df = pd.read_csv("purchases.csv", sep=";", names=["timestamp", "user_id", "product_
df = df.drop("timestamp", axis=1)
```

executed in 878ms, finished 21:05:04 2017-03-27

Here, I group users and the **unique products** that they have purchased (few users have purchased more than 1 item. In those cases, it is redundant and memory intensive to use those details)

In [3]:

```
rawdata = df.groupby("user_id").product_id.apply(set)
```

executed in 27.3s, finished 21:05:32 2017-03-27

For the implementation of ALS using spark machine learning library, it is important that the input is integer. Strings such as product_ids cannot be used.

So, it is necessary that the product_ids should be integer. For this purpose, I create a dictionary to assign an integer to a product_id and another dictionary to reconvert integer to their corresponding product_ids

---

In [4]:

```
product_to_ix = {prod:i for i, prod in enumerate(df.product_id.unique())}
ix_to_product = {i:prod for i, prod in enumerate(df.product_id.unique())}
```

executed in 263ms, finished 21:05:41 2017-03-27

Convert the pandas dataframe to a format readable by the spark dataframe for applying ALS method

In [5]:

```
def expand_user(a):
    return [Rating(user, product_to_ix[item], 1) for user in a.index for item in a[
ratings = expand_user(rawdata)
```

executed in 9.14s, finished 21:05:51 2017-03-27

In [6]:

```
ratingsRDD = sc.parallelize(ratings)
```

executed in 2.14s, finished 21:05:53 2017-03-27

In order to predict and improve the accuracy of my model, I randomly split the data into training and test.

In [7]:

```
training_RDD, test_RDD = ratingsRDD.randomSplit([8, 2], seed=123)
test_for_predict_RDD = test_RDD.map(lambda x: (x[0], x[1]))
```

executed in 3ms, finished 21:06:04 2017-03-27

Now, the data is ready for training and testing.

Things to note:

```
- It's hard to test recommender systems, unless you put it into production
- User's feedback from the recommended products is the best way to test, ho
w well your recommender works
```

Tuning parameters:

```
- lambda or the regularization parameter
    - This fights against overfitting of the model
- rank
    - Number of latent factors
    - Increasing rank will decrease RMSE
    - Increasing rank is **computationally expensive**
- Number of iterations
    - Tune this until RMSE stops decreasing. Best is somewhere between 5 an
d 20
- Alpha - confidence parameter
    - Relevant with implicit data like the one we have
    - Original paper suggests 40, spark default is 0.1.
```

Note: I am getting memory errors for higher ranks. So, for this working example, I am restricting the ranks to a maximum of 25

In [9]:

```
seed = 4242
iterations = 10
regularization_parameter =[i * 0.01 for i in range(1, 20, 2)]
ranks = [3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25]
errors = [[0]*len(regularization_parameter)] * len(ranks)

min_error = float('inf')
best_lambda = -1
best_lambda_index = -1
best_model = None
best_rank = -1
best_rank_index = -1


# Loop over all possible value fr lambda and rank to find the best parameters for o
for i, rank in enumerate(ranks):
    for j, regParam in enumerate(regularization_parameter):
        model = ALS.train(training_RDD, rank, seed=seed, iterations=iterations, lam
        predictions = model.predictAll(test_for_predict_RDD).map(lambda r: ((r[0],
        rates_and_preds = test_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2
        error = np.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean(
        errors[i][j] = error
        print('For lambda %s and rank %s the RMSE is %s' % (regParam, rank, error))
        if error < min_error:
            min_error = error
            best_lambda = regParam
            best_model = model
            best_rank = rank
            best_rank_index = i
            best_lambda_index = j
        with open('sparkLogging', 'a') as f:
            f.write("RMSE on testing set: {}, with rank: {}, lambda: {}\n".format(e


print('The best model was trained with lambda %s, rank %s and RMSE: %s' % (best_lam

with open('sparkLoggingBest', 'a') as f:
    f.write("RMSE on testing set: {}, with rank: {} at index {}, lambda: {} at inde
```

executed in 33m 34s, finished 20:51:57 2017-03-27

```
For lambda 0.01 and rank 3 the RMSE is 0.561399001661
For lambda 0.03 and rank 3 the RMSE is 0.288752568934
For lambda 0.05 and rank 3 the RMSE is 0.216985722377
For lambda 0.07 and rank 3 the RMSE is 0.194333427767
For lambda 0.09 and rank 3 the RMSE is 0.18901181331
For lambda 0.11 and rank 3 the RMSE is 0.192734344792
For lambda 0.13 and rank 3 the RMSE is 0.201245941937
For lambda 0.15 and rank 3 the RMSE is 0.212475473862
For lambda 0.17 and rank 3 the RMSE is 0.22573564362
For lambda 0.19 and rank 3 the RMSE is 0.240504605629
For lambda 0.01 and rank 5 the RMSE is 0.551075230103
For lambda 0.03 and rank 5 the RMSE is 0.250500729443
For lambda 0.05 and rank 5 the RMSE is 0.184466149679
For lambda 0.07 and rank 5 the RMSE is 0.172463567727
For lambda 0.09 and rank 5 the RMSE is 0.175188894463
For lambda 0.11 and rank 5 the RMSE is 0.18363965509
For lambda 0.13 and rank 5 the RMSE is 0.195089331643
```

```
For lambda 0.13 and rank 5 the RMSE is 0.195089331643
For lambda 0.15 and rank 5 the RMSE is 0.208422095848
For lambda 0.17 and rank 5 the RMSE is 0.223080331657
For lambda 0.19 and rank 5 the RMSE is 0.238738560241
For lambda 0.01 and rank 7 the RMSE is 0.561816609573
For lambda 0.03 and rank 7 the RMSE is 0.239864694869
For lambda 0.05 and rank 7 the RMSE is 0.171373390022
For lambda 0.07 and rank 7 the RMSE is 0.162327198925
For lambda 0.09 and rank 7 the RMSE is 0.16714261096
For lambda 0.11 and rank 7 the RMSE is 0.176990091505
For lambda 0.13 and rank 7 the RMSE is 0.189436326902
For lambda 0.15 and rank 7 the RMSE is 0.203505873599
For lambda 0.17 and rank 7 the RMSE is 0.218718853139
For lambda 0.19 and rank 7 the RMSE is 0.234800889765
For lambda 0.01 and rank 9 the RMSE is 0.586214436069
For lambda 0.03 and rank 9 the RMSE is 0.242476154837
For lambda 0.05 and rank 9 the RMSE is 0.170564655614
For lambda 0.07 and rank 9 the RMSE is 0.161569363552
For lambda 0.09 and rank 9 the RMSE is 0.16665243772
For lambda 0.11 and rank 9 the RMSE is 0.176767675801
For lambda 0.13 and rank 9 the RMSE is 0.189429682518
For lambda 0.15 and rank 9 the RMSE is 0.203661106627
For lambda 0.17 and rank 9 the RMSE is 0.218991714184
For lambda 0.19 and rank 9 the RMSE is 0.23515741831
For lambda 0.01 and rank 11 the RMSE is 0.561454201226
For lambda 0.03 and rank 11 the RMSE is 0.225265943023
For lambda 0.05 and rank 11 the RMSE is 0.159418222588
For lambda 0.07 and rank 11 the RMSE is 0.15371076101
For lambda 0.09 and rank 11 the RMSE is 0.160165427677
For lambda 0.11 and rank 11 the RMSE is 0.171012688655
For lambda 0.13 and rank 11 the RMSE is 0.184166293774
For lambda 0.15 and rank 11 the RMSE is 0.198752472553
For lambda 0.17 and rank 11 the RMSE is 0.214331571418
For lambda 0.19 and rank 11 the RMSE is 0.230658891733
For lambda 0.01 and rank 13 the RMSE is 0.58593964145
For lambda 0.03 and rank 13 the RMSE is 0.230044766185
For lambda 0.05 and rank 13 the RMSE is 0.159983244297
For lambda 0.07 and rank 13 the RMSE is 0.153647487177
For lambda 0.09 and rank 13 the RMSE is 0.160102584847
For lambda 0.11 and rank 13 the RMSE is 0.171073634867
For lambda 0.13 and rank 13 the RMSE is 0.184360225977
For lambda 0.15 and rank 13 the RMSE is 0.199069995056
For lambda 0.17 and rank 13 the RMSE is 0.214764719215
For lambda 0.19 and rank 13 the RMSE is 0.23120270678
For lambda 0.01 and rank 15 the RMSE is 0.566561342241
For lambda 0.03 and rank 15 the RMSE is 0.221958841132
For lambda 0.05 and rank 15 the RMSE is 0.156420033504
For lambda 0.07 and rank 15 the RMSE is 0.151552837943
For lambda 0.09 and rank 15 the RMSE is 0.158412639395
For lambda 0.11 and rank 15 the RMSE is 0.169517096969
For lambda 0.13 and rank 15 the RMSE is 0.18287641082
For lambda 0.15 and rank 15 the RMSE is 0.197635860555
For lambda 0.17 and rank 15 the RMSE is 0.213357524654
For lambda 0.19 and rank 15 the RMSE is 0.229798821399
For lambda 0.01 and rank 17 the RMSE is 0.560512415674
For lambda 0.03 and rank 17 the RMSE is 0.21708806635
For lambda 0.05 and rank 17 the RMSE is 0.153716985809
For lambda 0.07 and rank 17 the RMSE is 0.149434352372
For lambda 0.09 and rank 17 the RMSE is 0.156563706509
For lambda 0.11 and rank 17 the RMSE is 0.16786725551
For lambda 0.13 and rank 17 the RMSE is 0.181397575738
For lambda 0.15 and rank 17 the RMSE is 0.196311251415
```

```
For lambda 0.17 and rank 17 the RMSE is 0.212173428003
For lambda 0.19 and rank 17 the RMSE is 0.228740475643
For lambda 0.01 and rank 19 the RMSE is 0.553110237724
For lambda 0.03 and rank 19 the RMSE is 0.215068441063
For lambda 0.05 and rank 19 the RMSE is 0.153685844052
For lambda 0.07 and rank 19 the RMSE is 0.149642431499
For lambda 0.09 and rank 19 the RMSE is 0.156782761382
For lambda 0.11 and rank 19 the RMSE is 0.16805187551
For lambda 0.13 and rank 19 the RMSE is 0.181553073809
For lambda 0.15 and rank 19 the RMSE is 0.196443725854
For lambda 0.17 and rank 19 the RMSE is 0.212278911717
For lambda 0.19 and rank 19 the RMSE is 0.228811088653
For lambda 0.01 and rank 21 the RMSE is 0.557945512733
For lambda 0.03 and rank 21 the RMSE is 0.216393825853
For lambda 0.05 and rank 21 the RMSE is 0.153584591862
For lambda 0.07 and rank 21 the RMSE is 0.149576787223
For lambda 0.09 and rank 21 the RMSE is 0.156771989706
For lambda 0.11 and rank 21 the RMSE is 0.168093045441
For lambda 0.13 and rank 21 the RMSE is 0.181638334211
For lambda 0.15 and rank 21 the RMSE is 0.196557629039
For lambda 0.17 and rank 21 the RMSE is 0.212407875873
For lambda 0.19 and rank 21 the RMSE is 0.228947621403
For lambda 0.01 and rank 23 the RMSE is 0.555842734281
For lambda 0.03 and rank 23 the RMSE is 0.214916425595
For lambda 0.05 and rank 23 the RMSE is 0.153484319177
For lambda 0.07 and rank 23 the RMSE is 0.149664483628
For lambda 0.09 and rank 23 the RMSE is 0.15691965133
For lambda 0.11 and rank 23 the RMSE is 0.168246825917
For lambda 0.13 and rank 23 the RMSE is 0.181775940869
For lambda 0.15 and rank 23 the RMSE is 0.196678477244
For lambda 0.17 and rank 23 the RMSE is 0.212517876574
For lambda 0.19 and rank 23 the RMSE is 0.229051589229
For lambda 0.01 and rank 25 the RMSE is 0.564210346149
For lambda 0.03 and rank 25 the RMSE is 0.220896136041
For lambda 0.05 and rank 25 the RMSE is 0.15552690567
For lambda 0.07 and rank 25 the RMSE is 0.150958053258
For lambda 0.09 and rank 25 the RMSE is 0.158037250112
For lambda 0.11 and rank 25 the RMSE is 0.169366543187
For lambda 0.13 and rank 25 the RMSE is 0.182946370547
For lambda 0.15 and rank 25 the RMSE is 0.197899569576
For lambda 0.17 and rank 25 the RMSE is 0.213784316733
For lambda 0.19 and rank 25 the RMSE is 0.230364549516
The best model was trained with lambda 0.07, rank 17 and RMSE: 0.14943
4352372
```

Best model was trained with lambda 0.07 and rank 17 the RMSE is 0.149434352372

Now, I tune the iterations. More than 25 iterations, I have memory error.

In [11]:

```
seed = 4242
iterations = [5, 10, 15, 20]
regParam = 0.07
rank = 17

for iteration in iterations:
    model = ALS.train(training_RDD, rank, seed=seed, iterations=iteration, lambda_=
    predictions = model.predictAll(test_for_predict_RDD).map(lambda r: ((r[0], r[1]
    rates_and_preds = test_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2])))
    error = np.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())
    print('For %d iterations, the RMSE is %s' % (iteration, error))
```

executed in 1m 23.4s, finished 21:10:28 2017-03-27

```
For 5 iterations, the RMSE is 0.33973580526
For 10 iterations, the RMSE is 0.149434352372
For 15 iterations, the RMSE is 0.135585874754
For 20 iterations, the RMSE is 0.133802020976
```

Now, I use the best fit parameters on my model to train

In [13]:

```
model = ALS.train(ratingsRDD, rank=17, seed=4242, iterations=20, lambda_=0.07)
```

executed in 27.4s, finished 21:12:57 2017-03-27

In [33]:

```
### replace users with list(df.user_id.unique()) to get recommendation for all user
users = [4, 1, 12, 11, 10]
for user in users:
    best_5_recommendations = model.recommendProducts(user, 5)
    print("_"*50)
    print("\nFor user %d we recommend the following 5 products:\n" %user)
    for recommendation in best_5_recommendations:
        print(" "*10, ix_to_product[recommendation.product])
```

executed in 196ms, finished 21:23:14 2017-03-27

```
__________________________________________________

For user 4 we recommend the following 5 products:

          AC016EL50CPHALID-1749
          AP082ELAD87SANID-176936
          IL086ELABGL9ANID-74783
          DR901TBAC245ANID-110616
          SA848ELAD130ANID-165269
__________________________________________________

For user 1 we recommend the following 5 products:

          AP082ELAD69IANID-173547
          KE263SPABEIOANID-71986
          NO749ELAD9EXANID-178702
          AP082EL51ANKALID-348
          DI747EL56RLNANID-50444
__________________________________________________

For user 12 we recommend the following 5 products:

          AP082EL51ANKALID-348
          NO749ELAD9EXANID-178702
          KE263SPABEIOANID-71986
          AP082ELAD69IANID-173547
          AC016EL58BKFALID-941
__________________________________________________

For user 11 we recommend the following 5 products:

          AC016EL58BKFALID-941
          AD029EL42BKVALID-957
          WI981EL52EFNALID-2847
          NO749ELAD9EXANID-178702
          AP082EL51ANKALID-348
__________________________________________________

For user 10 we recommend the following 5 products:

          NO749ELAD9EXANID-178702
          PE783EL91IKMANID-63112
          KE263SPABEIOANID-71986
          DI747EL56RLNANID-50444
          GR087ME12IADANID-22985
```

In [ ]:

```

```