**Business Intelligence using Product Star Rating & Customer’s Reviews**

*Implementation of Recommender Systems and Natural Language Processing on Amazon Dataset*

**(Bhuvan Chopra, Fatemeh Rostamzadeh Renani, Junaid Qazi)**

ETL:

Cassandra related stuff!

Data cleaning and analysis.

Data Visualization:

Natural Language Processing using NLTK

Recommender Systems:

Collaborative Filtering with Alternating Least Square (ALS) in Spark

Collaborative Filtering using Python’s library “surprise”

Outcomes & suggestions

Learning:

Data Cleaning

Recommender systems is a family of methods that filters through large information (what the users have experienced/observed/bought/rated) and provides recommendations to users for which the user has not experienced. 35 percent of Amazon's revenue and about 70 percent of everything users watch in Netflix are generated by their recommendation engine. By narrowing down the selection options for the costumer, it is more likely that they make a purchase. Therefore, many businesses would greatly benefit from recommender systems.

In this project we focus on the Collaborative filtering algorithm. In order to recommend appropriate products to costumers based on the costumer's rating we have implemented the recommended system using two technologies. We have investigated and learned how recommender systems work in Spark using Collaborative Filtering. We also implemented the Surprise Library (a Python scikit) which builds and analyzes recommender systems. Each approach has its own advantages and requirements.

Recommender System Collaborative Filtering: Alternating Least Squares (ALS) in Spark

The mane idea is to predict whether a customer would like a certain product (an item, a movie, or a song) to find the targeted customers for the product. The computational complexity increases with the size of a company’s customer base and products. Therefore the scalability is an important concern. This challenge can be address using Apache Spark MLlib which enables us to build recommendation models from billions of records.

Spark MLlib implements a collaborative filtering algorithm called Alternating Least Squares (ALS). ALS models the user-item association matrix (R) as the product of two matrices U and V. Where U is a [u by f ] matrix (users and hidden features) and V is a [f by i ] matrix (items and hidden features). The idea is to learn these two matrices by minimizing the reconstruction error of the observed ratings.

ALS implementation in spark.ml has the following parameters:

*NumBlocks* is the number of blocks the users and items will be partitioned into in order to parallelize computation (defaults to 10). *Rank* is the number of latent factors in the model (defaults to 10). *MaxIter* is the maximum number of iterations to run (defaults to 10). *RegParam* specifies the regularization parameter in ALS (defaults to 1.0). *ImplicitPrefs* specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data (defaults to false which means using explicit feedback). *Alpha* is a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations (defaults to 1.0). *Nonnegative* specifies whether or not to use nonnegative constraints for least squares (defaults to false).

We have performed cross-validation to find the best values for ***rank, maxlter, regParam*** parameters in a small dataset (1000 reviews). These are some interesting concepts that we learned using spark ml:

1- The "regParam" is the regularization parameter. Fig. a shows the grid-search actually found the parameters that minimizes the error.

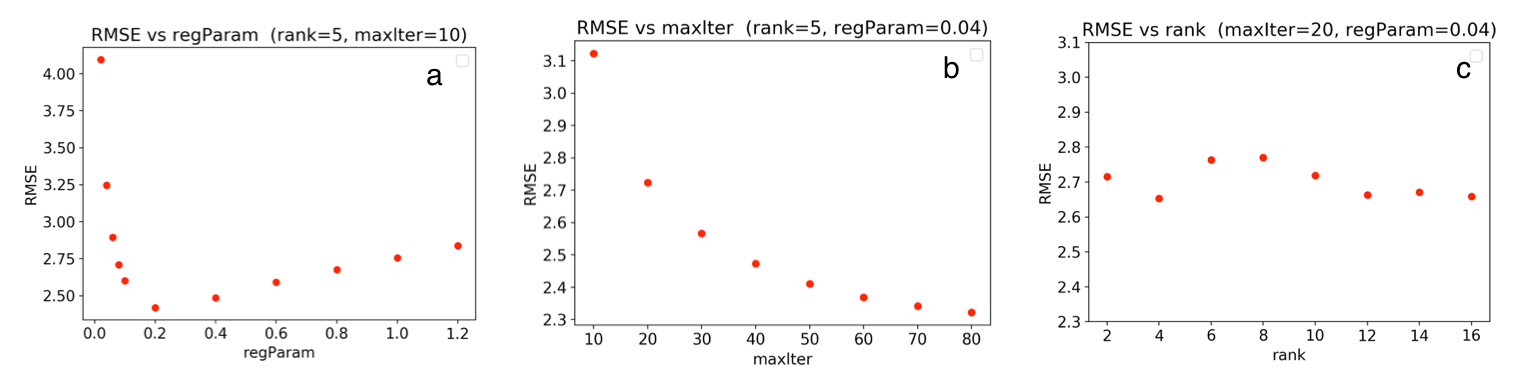


Fig. Optimization of RMSE based on a) regParameter regularization parameter, b) maxIter maximum iteration, c) rank number of hidden features.

2- Fig. b shows that the accuracy gets better as the number of iteration increase (agrees with our intuition). However, Spark was not able to perform well for large "maxlter" values when we used our whole dataset. Thus we have used the maxlter default value which is 10.

3- As represented in Fig. c the "rank" does not change the accuracy of the predictions. Therefore, we select the default value.

4- We have found that setting nonnegative= True (which allows to have negative contrains for least squares) improved the prediction, where is default value for this parameter is False.

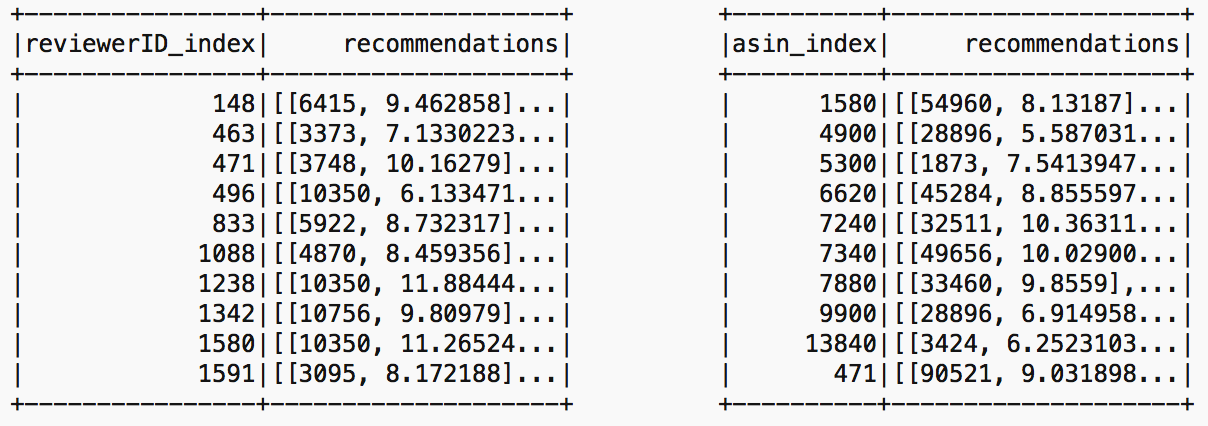


Fig. a) Spark recommended products to 10 users d) 10 potential customers for 10 products.

Here is the the best RMSE (validation) = 2.512477 for the model trained with rank = 10, lambda = 0.04, and numIter = 30

Challenges: Spark collaborative filtering (ALS) algorithm currently only supports integers for user and item ids. In our original dataset theses features are recorded as strings (combination of the numeric and alphabet). Thus, in order to perform the CF in spark we have to convert the review Id and item Id to unique integer IDs. We completed this task with two methods. First, we used Python numpy library (df['cod\_reviewerID'] = df.reviewerID.astype('category').cat.codes) for this conversion. Later we realized that this task can be down in spark using stringindexer module in spark ml.

Finally we run Spark setting rank=5, maxIter=10, regParam=0.2 for the full dataset reading from Cassandra table. The calculation has RMSE= 1.651045. Fig. represents the

Python using Surprise Library:

This library provides many prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based such as SVD, SVD++. Thus we were able to benchmark the RMSE for various algorithms with their default parameters (see Table.).

SVD and SVP++ are based on the matrix factorization-based method and produce better accuracy for our recommender system. Thus we investigated the recommender system using SVD++ in more detail to fine-tune its parameters. Here are the parameter of SVD++:

n\_ephocs : The number of iteration of the SGD procedure. Default is 20.

lr\_all : The learning rate for all parameters. Default is 0.007.

|  |  |
| --- | --- |
| algorithm | RMSE |
| SVD++ | 1.1931 |
| SVD | 1.1950 |
| BaselineOnly | 1.1996 |
| KNNBaseline | 1.1995 |
| KNNBasic | 1.2551 |
| KNNWithMeans | 1.2562 |
| NMF | 1.2733 |

Table. RMSE calculated for various algorithms using surprise library.

reg\_all : The regularization term for all parameters. Default is 0.02.

We found RMSE= 0.8793 implementing SVD++ where the parameters of the estimator are optimized by cross-validated grid-search over a parameter grid (n\_epochs=15, lr\_all= 0.009, reg\_all= 0.06).

Challenges:

We have to learn this library and write python codes to implement different algorithms, performing grid-search for optimization process, finding the top 10 recommendations for users. In order to use the library on cluster to run the python code for the full dataset, we have to install the Surprise library on the cluster.

We found our code in Python (using Surprise library) provides more accurate predictions however it is slower than spark.