CS 724- High Performance Computing & Big Data

Course Project Report

Amazon Product Recommendation System

Submitted by
Raja Harsha Chinta
UIN: 01043488

Project Abstract

We have chosen Amazon product sales data set comprising of sales activity and user ratings for each product. The idea is to create a product suggestion/recommendation system for each user based on his previous purchases and his rating for each one. A Collaborative Filtering model is built to predict the virtual ratings for the product that the user did not purchase. The system predicts the user rating for all the items and we display the products which user may be like, buy and rate higher.

Secure | https://www.amazon.com
Recommendations for You, Raja

Figure 1.1

Project Description

Today's marketing world offers multiple products to meet multiple needs of multiple customers. Unlike the earlier days, where one standard product is available for one particular need. A vast number of product options are being offered by a seller for the customer to choose from. An efficient system is required to showcase the products a user may like and interested in buying. This improves the reach of a product to the relevant customer and increases marketing ability.

A recommendation system here uses collective information of the whole customer user group to make individualized user response predictions. This recommendation system generates recommendations for a particular user based on their previous product ratings and for a non-existing/new user we prompt for providing sample ratings for few random products. The problem solving approach we have chosen here is Collaborative filtering. We aim at filling missing entries of a user's item association matrix. Users and Products are described by a set of latent factors that can be used to predict missing entries. We use Alternating Least Squares (ALS) algorithm to learn these latent factors. This ALS algorithm logic is suggested to be implemented manually instead of using native spark.mllib libraries. This project is planned to be developed on distributed environment using Apache Spark and code in Python 2.7 using Hortonworks Sandbox with Hadoop version 2.2.4.2. Recommendations are generated automatically for existing users with a purchase history and ratings. For new users, default top rated products in the category are planned to be recommended.

Introduction

Recommender systems is a family of methods that enable filtering through large observation and information space in order to provide recommendations in the information space that user does not have any observation, where the information space is all of the available items that user could choose or select and observation space is what user experienced or observed so far.

Collaborative Filtering

Collaborative Filtering (CF) is a subset of algorithms that use extensively other users and items along with their ratings and target user history to recommend an item that target user does not have ratings for. Fundamental assumption behind this approach is that other users preference over the items could be used recommending an item to the user who did not see the item or purchase before. CF differs itself from content-based methods in the sense that user or the item itself does not play a role in recommendation but rather how (rating) and which users (user) rated a particular item.

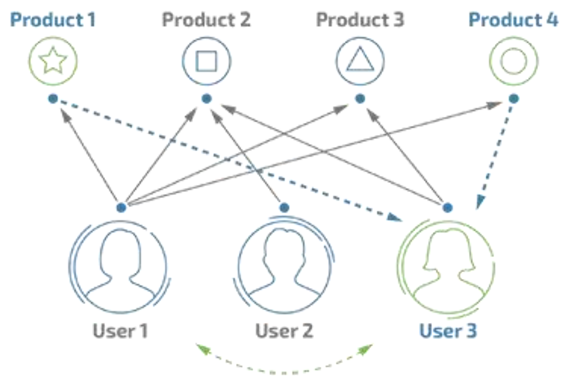


Figure: 1.2

Model-based algorithms build a model from users past behavior, and then use that model to recommend items to any user. They have two main advantages over memory-based approaches: they can provide recommendations to new users and they can provide instant recommendations, since most of the computation is moved into the pre-processing phase of the model creation. Memory-based algorithms merge model creation and instant recommendations into a single procedure. Due to this fact, their computation requirements may be inadequate for instant recommendations, and they also suffer from the new user problem. Latent factor algorithms explain user preferences by characterizing products and users with factors that are automatically inferred from user feedback. Some of the most successful realizations of latent factor models are based on matrix factorization. In its basic form, matrix factorization characterizes items and users by vectors of factors inferred from item usage patterns. High correspondence between item and user factors leads to a recommendation. These methods have become popular since they combine predictive accuracy with good scalability.

Project Design

The recommendation system build in the project involves three input files, ratings.dat, users.dat and products.dat. The input files which are in nested .json format are converted into .dat files using python scripting. After this the files are transferred to Hadoop file storage system in an input directory. Input files are then read into an array/collection/dictionary and are distributed across Hadoop clusters to form a Resilient Distributed Dataset. We use Collaborative filtering technique to calculate user's similarities among the items and taking similar user's interest we do recommendation for the users again. Collaborative filtering is one case of neighborhood methods. Here we use matrix factorization technique using Alternative least square algorithm to perform collaborative filtering. A prediction matrix is generated after n number of iterations and then we correlate user with the products not rated earlier and have a predicted rating greater than a value, like 3 or 4 rating. Each individual user specific product recommendation is written into a file, which is usually cached and used in real-time environment to make suggestions.

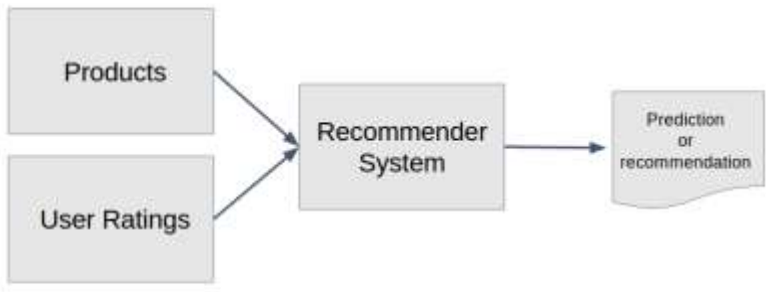


Figure: 1.3

Implementation

Data Source and Metadata Description:

The dataset chosen for the project is a small piece from Amazon product data for last 18 years in various product categories are available in the below link. Citation for the resource provider is made under References and Citations section in the end of the document. Out of the entire data, we have chosen ratings data related to Musical Instruments sold on Amazon.

http://jmcauley.ucsd.edu/data/amazon/links.html

Ratings Metadata: Highlighted fields are drawn to a ratings.dat file.

```
{
"reviewerID": "A2SUAM1J3GNN3B",
"asin": "0000013714",
"reviewerName": "J. McDonald",
"helpful": [2, 3],
"reviewText": "I am having a wonderful time playing these old hymns. The music is at times hard to read
because we think the book was published for singing from more than playing from. Great purchase
though!",
"overall": 5.0,
"summary": "Heavenly Highway Hymns",
"unixReviewTime": 1252800000,
"reviewTime": "09 13, 2009"
}
```

Product Metadata: Highlighted fields are draws to a products.dat file.

```
{
"asin": "0000031852",
"title": "Girls Ballet Tutu Zebra Hot Pink",
"price": 3.17,
"imUrl": "http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
"salesRank": {"Toys & Games": 211836},
"brand": "Coxlures",
"categories": [["Sports & Outdoors", "Other Sports", "Dance"]]
}
```

Data Extraction

The data source made available is zipped and in nested .json format, this data is extracted to .csv files using json, pandas libraries in Python. The data extraction code is submitted in a separate file data_extraction.py for reference. There are many categories of products available in the data source, out of which we have chosen Musical Instruments category. Due to limitation of processing capability using the Hadoop Virtual machine having 2 clusters, we used this dataset which fits well here.

Parallel Matrix Factorization Approach – Alternative Least Square

We start with randomly chosen U, V values and adjust the values of U, V to make RMSE smaller.

Step 1: Fix V and Update all entries of U.
Step 2: Fix U and Update all entries of V.
Step 3: Repeat first and second steps till we reach a least saturation of RMSE value.

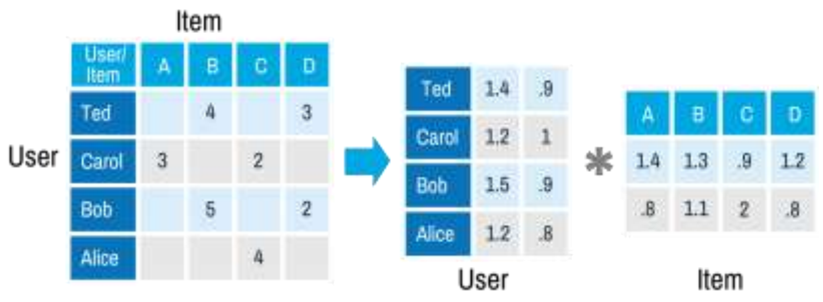


Figure 1.4

ALS Logical deduction

```
var U = sc.broadcast(User Matrix)
var V = sc.broadcast(Item Matrix)
var R = sc.broadcast(Ratings Matrix)

for ( i in range(Iterations)):

         us = sc.parallelize(range(U), partitions)
                .map(lambda x: updateUV(x, vsb.value, Rb.value))
                .collect()
         usb = sc.broadcast(us)


         vs = sc.parallelize(range(V), partitions)
                .map(lambda x: updateUV(x, usb.value, Rb.value.T))
                .collect()
         vsb = sc.broadcast(vs)
```

Psuedocode

```
# import libraries

  numpy, pyspark, etc.

# check for sufficient arguments

  if (number of arguments passed !=5):
       sys.exit(1)

# declare functions for loading, parsing input files

  def loadRating(ratingsFile), def parseRating(line), def parseProduct(line), def parseUser(line)

# declare function for compute RMSE

  def computeRMSE(R, us, vs):
      diff = R - us * vs.T
      rmse_val = np.sqrt(np.sum(np.power(diff, 2))/(U * V))
      return rmse_val

# declare function to calculate update U,V matrix after each iteration

  def updateUV(i, uv, r):
      uu = uv.shape[0]
      ff = uv.shape[1]
      xtx = uv.T * uv
      xty = uv.T * r[i, :].T

      for j in range(ff):
         xtx[j, j] += lambdas * uu
```

```
    updated_val = np.linalg.solve(xtx, xty)
    return updated_val
```

# setup Environment

```
  conf = SparkConf().setAppName("AmazonALS").set("spark.executor.memory", "2g")
  sc = SparkContext(conf=conf)
```

# read ratings, products, users files into respective RDD's

```
  ratings = sc.textFile(join(hdfs_src_dir, "ratings.dat")).map(parseRating)
  products = sc.textFile(join(hdfs_src_dir, "products.dat")).map(parseProduct
  users = sc.textFile(join(hdfs_src_dir, "users.dat")).map(parseUser)
```

# declare number of partitions, iterations, parameters

```
  Partitions_num = 4, Iterations_num = 20, seed, regularization parameters
```

# Generate Random Matrix U,V

```
  U = matrix(rand(u, n_factors))
  V = matrix(rand(v, n_factors))
```

# convert ratings data into a Users vs Products and Values matrix R.

```
  Z = np.zeros((U,V))
  R = np.matrix(Z)
  for i in range(numRatings):
        r_local = r_array[i]
        R[(r_local[0]-1),(r_local[1]-1)] = r_local[2]
```

# broadcast the values to all the processors

```
  Rb = sc.broadcast(R)
  ub = sc.broadcast(U)
  vb = sc.broadcast(V)
```

# run for 20 iterations while adjusting and obtain optimum RMSE value.

```
  for i in range(n_iterations):
       U = parallelize(Fix V and solve for U value)
       V = parallelize(Fix U and solve for V value)

       RMSE_current = computeRMSE(R, us, vs)
```

# Print the Iteration and RMSE

```
      print("Iteration Number")
      print("RMSE value")
```

# prediction Matrix

```
  RP = np.dot(pb.ub);
```

# write the products recommended to each user into a file

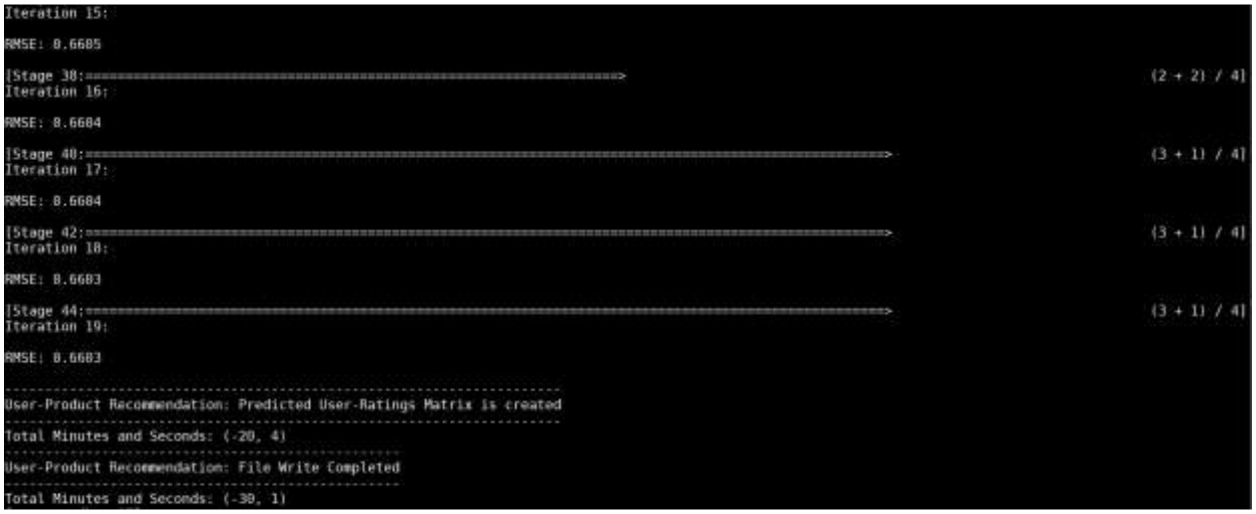
```
    output = open(outputfile,'w')

    for i in range(U):
        for j in range(V):
        pRating = recoB.value[i,j]
            if((Rb.value[i,j]==0 and pRating>3)):
                output.write(str(i)+","+str(j)+","+str(pRating)+"\n")

    output.close()
```

# stop spark context

Results

A final weighted matrix is considered after n iterations (20 in our case) to draw the predicted ratings. We observed the root mean square error, consider the dot product of UV matrix where the error value is low. In program runtime, RMSE value for each iteration is displayed and the Recommendation data is written to a file with columns: userid, recommended product, rating predicted

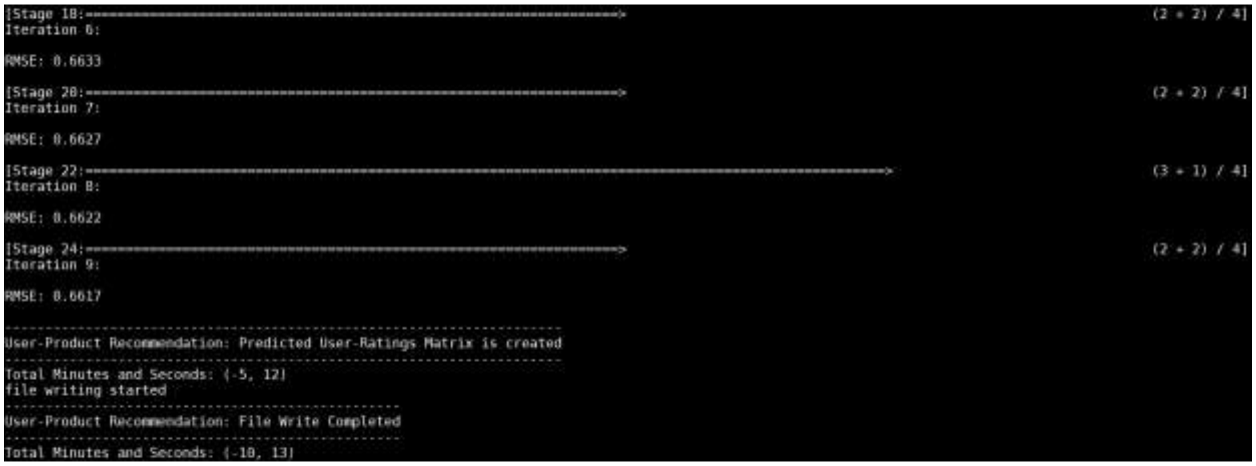
Sample run with 20 iterations is accomplished in 30 minutes, 1 second.



```
Iteration 15:

RMSE: 0.6685

[Stage 38:=================================================================>                               (2 + 2) / 4]
Iteration 16:

RMSE: 0.6684

[Stage 40:================================================================================================>   (3 + 1) / 4]
Iteration 17:

RMSE: 0.6684

[Stage 42:================================================================================================>   (3 + 1) / 4]
Iteration 18:

RMSE: 0.6683

[Stage 44:================================================================================================>   (3 + 1) / 4]
Iteration 19:

RMSE: 0.6683

-------------------------------------------------------------------
User-Product Recommendation: Predicted User-Ratings Matrix is created
-------------------------------------------------------------------
Total Minutes and Seconds: (-20, 4)
-----------------------------------------------
User-Product Recommendation: File Write Completed
-----------------------------------------------
Total Minutes and Seconds: (-30, 1)
```

Figure 1.5

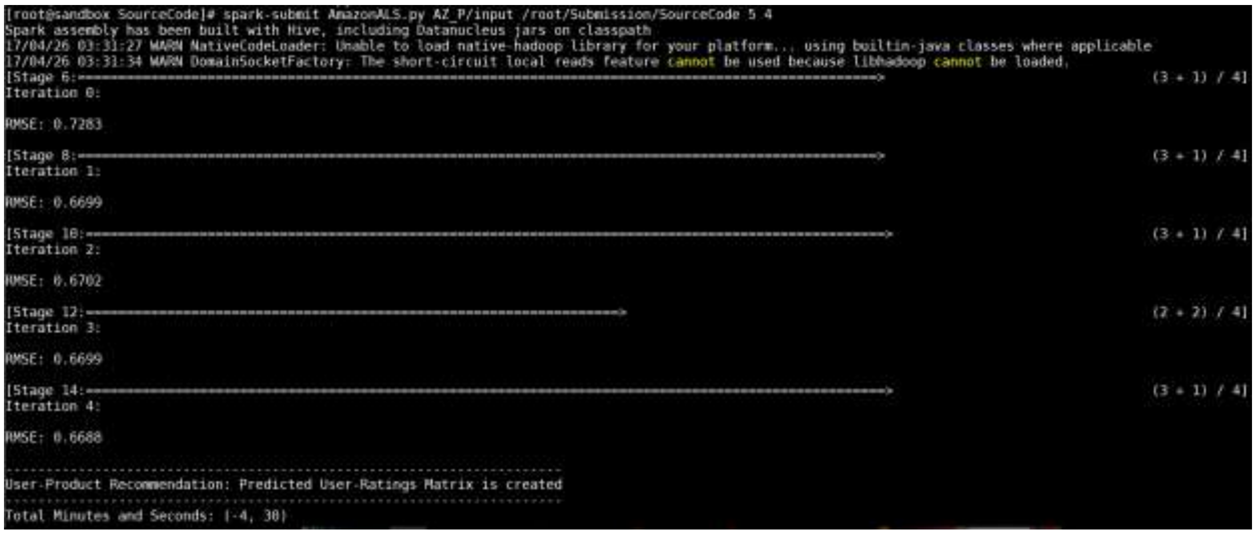
Sample run with 10 iterations is accomplished in 10 minutes, 13 seconds.



```
[Stage 18:=================================================================>                               (2 + 2) / 4]
Iteration 6:

RMSE: 0.6633

[Stage 20:=================================================================>                               (2 + 2) / 4]
Iteration 7:

RMSE: 0.6627

[Stage 22:================================================================================================>   (3 + 1) / 4]
Iteration 8:

RMSE: 0.6622

[Stage 24:=================================================================>                               (2 + 2) / 4]
Iteration 9:

RMSE: 0.6617

-------------------------------------------------------------------
User-Product Recommendation: Predicted User-Ratings Matrix is created
-------------------------------------------------------------------
Total Minutes and Seconds: (-5, 12)
file writing started
-----------------------------------------------
User-Product Recommendation: File Write Completed
-----------------------------------------------
Total Minutes and Seconds: (-10, 13)
```

Figure 1.6

Sample run with 5 iterations is accomplished in 4 minutes, 30 seconds.

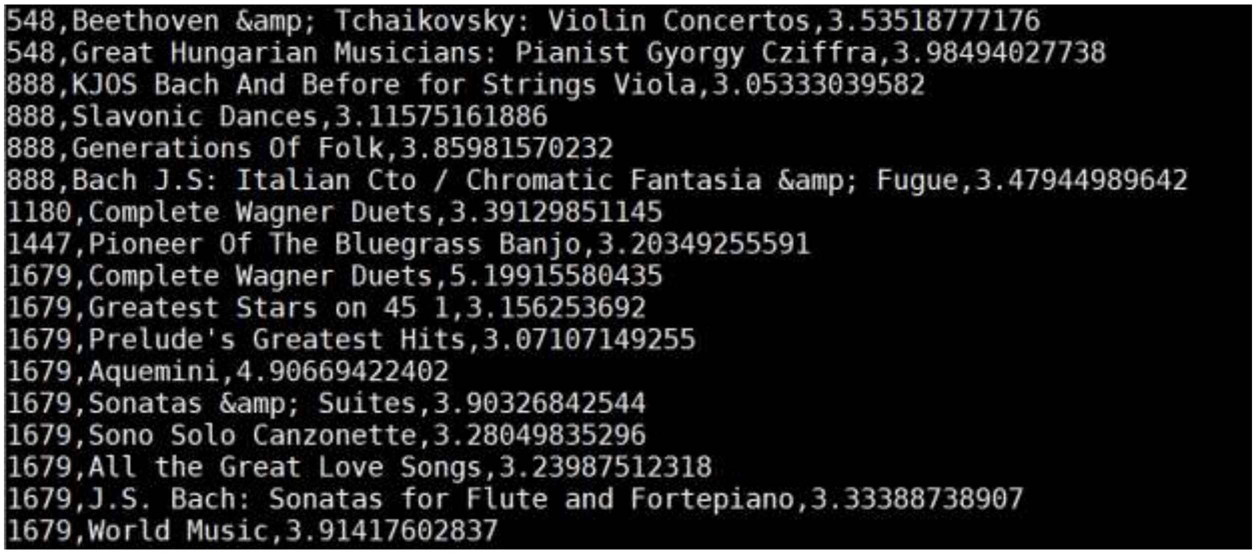


```
[root@sandbox SourceCode]# spark-submit AmazonALS.py AZ_P/input /root/Submission/SourceCode 5 4
Spark assembly has been built with Hive, including Datanucleus jars on classpath
17/04/26 03:31:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
17/04/26 03:31:34 WARN DomainSocketFactory: The short-circuit local reads feature cannot be used because libhadoop cannot be loaded.
[Stage 6:=====================================================================================>                    (3 + 1) / 4]
Iteration 0:

RMSE: 0.7283

[Stage 8:=====================================================================================>                    (3 + 1) / 4]
Iteration 1:

RMSE: 0.6699

[Stage 10:=====================================================================================>                   (3 + 1) / 4]
Iteration 2:

RMSE: 0.6702

[Stage 12:============================================================>                                            (2 + 2) / 4]
Iteration 3:

RMSE: 0.6699

[Stage 14:=====================================================================================>                   (3 + 1) / 4]
Iteration 4:

RMSE: 0.6688

------------------------------------------------------------------
User-Product Recommendation: Predicted User-Ratings Matrix is created
------------------------------------------------------------------
Total Minutes and Seconds: (-4, 30)
```

Figure 1.7

Sample Output

Structure: userID, MusicalPRoductName, predictedRating



```
548,Beethoven &amp; Tchaikovsky: Violin Concertos,3.53518777176
548,Great Hungarian Musicians: Pianist Gyorgy Cziffra,3.98494027738
888,KJOS Bach And Before for Strings Viola,3.05333039582
888,Slavonic Dances,3.11575161886
888,Generations Of Folk,3.85981570232
888,Bach J.S: Italian Cto / Chromatic Fantasia &amp; Fugue,3.47944989642
1180,Complete Wagner Duets,3.39129851145
1447,Pioneer Of The Bluegrass Banjo,3.20349255591
1679,Complete Wagner Duets,5.19915580435
1679,Greatest Stars on 45 1,3.156253692
1679,Prelude's Greatest Hits,3.07107149255
1679,Aquemini,4.90669422402
1679,Sonatas &amp; Suites,3.90326842544
1679,Sono Solo Canzonette,3.28049835296
1679,All the Great Love Songs,3.23987512318
1679,J.S. Bach: Sonatas for Flute and Fortepiano,3.33388738907
1679,World Music,3.91417602837
```

Figure 1.8

RMSE value is reduced steadily from 1st iteration to 20th iteration. We observe only a minute reduction in the value from 15th iteration and observer saturation near 20th one.

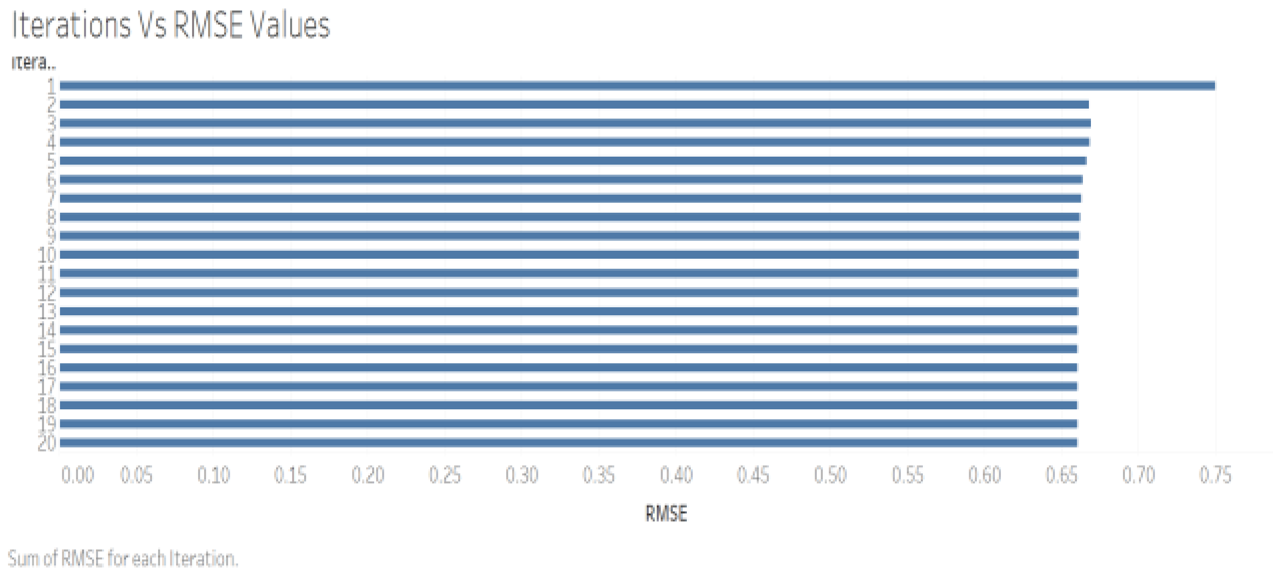


Figure 1.9

Performance Calculation

On a scaled data of 1 million rows the program approximately takes 20 minutes of time to run 20 iterations and 10 minutes to write the prediction data to a local file. An attempt was made to parallelize writing of recommendation data into file. But, it seems possible only in a real-time distributed environment which can perform multiple I/O operations.

We observed the program to run more than an hour, if we run it sequentially for 20 iterations. The memory resources available on the Virtual machine is not sufficient if we are trying to handle ratings data with users, product combination like 10000, 50000. This is a limitation observed in case of working with Hortonworks sandbox.

An improved RMSE value is observed with regularization parameter 0.01, 0.001.

Conclusion

As part of the course project we implemented Product Recommendation system with Amazon user ratings data. We solved the problem using matrix factorization method implemented Alternating Least Square algorithm to achieve high performance in calculations in distributed environment (Apache Spark). The results we observed are satisfactory and believe there is a lot of scope of improving the prediction model to bring accurate results. Also, various basic tuning techniques are tried to improve the results in this case.

References

1) https://math.stackexchange.com/questions/1072451/analytic-solution-for-matrix-factorization-using-alternating-least-squares
2) http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/
3) https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/
4) https://datascience.stackexchange.com/questions/2598/item-based-and-user-based-recommendation-difference-in-mahout
5) http://jmcauley.ucsd.edu/data/amazon/links.html
6) https://github.com/Mendeley/mrec/blob/master/mrec/mf/wrmf.py