RECOMMENDATION ENGINE IN SPARK WITH AN EXAMPLE OF MUSIC RECOMMENDATION

Lavanya K.[1], Nayna Agarwal[2]
School of Computer Science and Engineering,
VIT University, Vellore

[1]lavanya.k@vit.ac.in
[2]agarwal.nayna1996@gmail.com

***Abstract* – The volume of data has been increasing in the past years. It is often difficult to sift through the variety and identify the things the users would like. Therefore, effective techniques are needed to handle the large amount of data generated daily and provide recommendation to the users. Recommendations help these services solve the problem of discovery by helping user navigate the maze of the catalogues to find what they are looking for but don't know of. Recommendation systems take data such as what users bought, what users browsed, what users rated as input to the recommendation engine and give top picks for you as the desired output. Recommendation engine predicts the rating a user would give to a product, whether a user would buy a product and filters relevant products to the user. Collaborative filtering is a general term for any algorithm used by most recommendation engines. Collaborative filtering relies only on user behavior (history, ratings, similar users etc) and normally predicts users ratings for products they haven't yet rated. Latent factor analysis is one of the algorithm to perform collaborative filtering. It identifies hidden factors that influence a user's rating by a user-product rating matrix. It represents users by their ratings for different products. This paper is a framework for recommending music artist for a particular by using Alternating Least Squares (ALS) method of Sparks MLlib. Alternating least squares (ALS) is used to solve the equations in user-product rating matrix. Sparks MLlib has a built-in module for recommendation system that performs machine learning algorithms. It has built in functionality for applying ALS on any user-product-rating matrix. MLlib abstracts the programmer from the technical implementation details of ALS algorithm and running it across the cluster. Since Sparks RDDs are in-memory it can make multiple passes over the same data without doing disk writes.**

***Keywords* — Recommendation engine; Collaborative filtering; Latent factor Analysis; MLlib; Alternating least squares**

I. INTRODUCTION

The data is becoming more and more diverse and complex in the recent years. With the increasing volume and velocity of data, it is often difficult to process the big data; sift through the variety and predict the things the users would like. Therefore, effective techniques are needed to handle the large amount of data generated daily and provide recommendation to the users. Recommendation engines enable users to filter through large number of products. They personalize the service experiences and solve the problem of discovering the product that might be relevant for the user. They help users to find what they are looking for but don't know of. Recommendation systems take past information about the users as input to the recommendation engine and provide the desired output. Recommendation engine filters relevant products by predicting the implicit and explicit ratings that a user might give to a product. Collaborative filtering is a general algorithm that filters information by using recommendations of other people. It is user based collaborative filtering. The predictions are specific to a person, but it uses information gathered from many users. It relies only on user behavior. It is based on the idea that people often get recommendations from someone who have the similar taste as themselves. Latent factor analysis is used to identify the hidden factors in the system. The hidden factors might be abstract factors without any meaning. They influence the user's ratings. It is represented by a user-product rating matrix. The user product ratings matrix represents users by their ratings for different products. A framework is proposed in this paper for recommending music artist for a particular by using Alternating Least Squares (ALS) method of Sparks MLlib. Sparks MLlib has built-in modules for classification, regression, clustering, recommendations etc. algorithms. It provides built-in machine learning functionality for applying ALS on any user-product-rating matrix. MLlib completely abstracts away the technical implementation details of ALS. Spark's library completely abstracts the programmer from implementing the ML algorithm intricacies of running it across a cluster. Machine Learning algorithms are iterative, which means you need to make multiple passes over the same data. Since Sparks RDDs are in-memory, it can make multiple passes over the same data without doing disk writes.

II. LITERATURE SURVEY

Recommendation systems are required to handle the large amount of data generated daily and sift through the variety and provide recommendation to the users. This section briefs about the techniques used for recommendation systems.

Jaschar Domann (2016) presented a highly scalable recommender system optimized for the processing of streams. The system ensures the scalability by enabling the distributed processing of the recommendation request. [15]

Wenyuan Xu (2017) proposed a parallel method to provide a precise and efficient food recommendation system in Apache Spark. The method gives the recommendation for the user by finding similar users based on food features and the evaluation given by the users for the food. [8]

Siping Liu (2017) said about the ranking model that unified explicit feedback data and implicit feedback data together to give the top N recommendations. It is a parallel optimization model based on distributed and parallel computing implemented in Apache Spark. [10]

Sina Gholamian (2017) introduced UW Incremental Spark Analytics (UWISA) which is an incremental smart meter data platform. It calculates the energy-temperature model by applying efficient incremental techniques. It also compares the incremental and non-incremental method of Spark streaming. [12]

Sadanand Howal (2017) proposed a hybrid algorithm using different collaborative filtering algorithms i.e. Tanimoto Algorithm, Pearsons Algorithm, Slope Algorithm and SVD Algorithm to give the best results for implementation. [14]

Sasmita Panigrahi (2016) implemented User Oriented Collaborative Filtering method by using ALS and K mean clustering and improves the running time of the algorithm by using Spark cluster. This gives better result in terms of throughput compared to standard algorithms. [1]

Bobin K Sunny (2017) explained a system that provides recommendations in real time by using a self-adaptive approach that uses distributed processing power of Spark with scalability. the data processing method of Lambda Architecture manages the large amount of data in the system. [11]

Bipul Kumar (2016) proposed a latent factor model that reduce the number of latent factors of either users or products accurately and efficiently. It makes it simpler than Funk-SVD (Singular Value Decomposition), where latent factors of both users and product are often larger. [2]

Tong Yu (2016) developed One-sided Least Squares method of Matrix Factorization models with incremental learning and demonstrated its parallel implementation in Apache Spark. The method is integrated with batch learning with ALS. Web-scale recommender systems is improved by the fast and accurate incremental learning method. [5]

Manda Winlaw (2016) presented ALS-NCG (nonlinear conjugate gradient) algorithm to increase the convergence of collaborative filtering and provided its parallel implementation in the Apache Spark. The acceleration mechanism can also be used in optimization methods for collaborative filtering. [4]

III. BACKGROUND STUDY

A. COLLABORATIVE FILTERING

Content based and Collaborative filtering are the two methods used by the recommendation system. Content based filtering analyses the product by different attributes like genre, description etc. It recommends the products with the similar attributes to a product that the user already likes. Collaborative filtering relies on user's behavior like history, ratings, similar users etc. It analyses the user product relationship to identify the new user-product associations. Collaborative filtering is based on the basic principle that two users having same view about a group of products are likely to have the same view about the other products too. Collaborative filtering algorithms normally predict ratings for products, the users haven't rated yet.

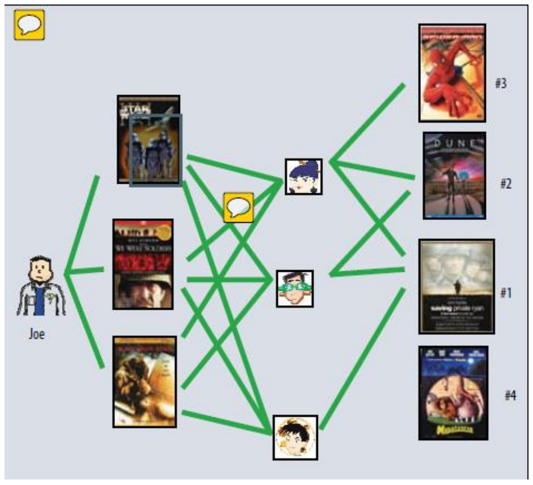


Fig 1: The user-oriented neighborhood method

To make prediction for Joe, the system finds similar users who also liked the movies liked by him, and then determines which other movies they liked.

1. MEMORY BASED COLLABORATIVE FILTERING

User based and Item based collaborative filtering are two techniques to perform memory based collaborative filtering. User based collaborative filtering finds similar users to predict missing ratings. It produces recommendations according to the preferences of similar users. Item based collaborative filtering finds similar products to predict missing ratings. It produces recommendations based on the relationship between products in the user-product matrix.

2. MODEL BASED COLLABORATIVE FILTERING

It predicts the missing ratings by building a model of the rating data (clustering etc.)

Some techniques to perform model based collaborative filtering are –

- Recommending products to the user by clustering users and identifying the closest cluster the active user likes.
- Recommending products to the user by mine association rules (for binary data)
- Finding significant deviation from the null-model that is a stochastic process which models usage of independent products
- Discovering factors by learning a latent factor model from the data and find products with expected high rating

B. LATENT FACTOR ANALYSIS

Latent factor analysis is one of the many different algorithms to perform collaborative filtering. Latent factor model explains both products and user ratings by describing several factors derived from the rating patterns. It identifies hidden factors that influence a user's rating.

Example, for movies, the hidden factors might be comedy vs drama, commercial vs arty etc. Sometimes the factors may have meaning or they might be abstract factors with no real life meaning.

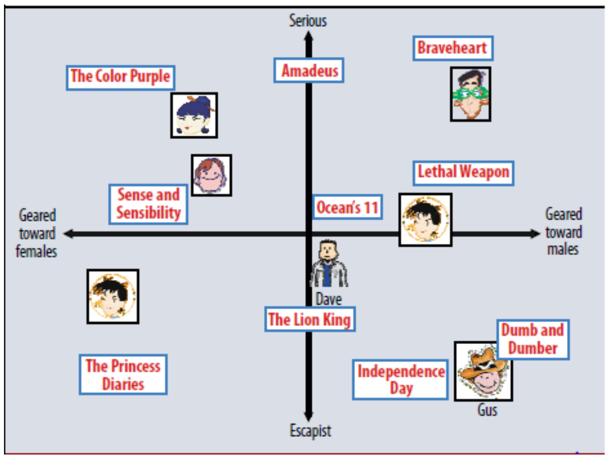


Fig 2: Simplified illustration of the latent factor approach

It characterizes both users and movies using two axes – male vs female and serious vs escapist.

Figure 2 gives a simple example for this idea in two axes. X axis represents female vs male oriented and Y axis represents serious vs escapist. Some popular movies and fictitious users are represented in the two axes. In this model, the users predicted the movie rating with respect to the movies average rating will be same as the dot product of the users and movies locations on the graph. Some movies and users might be considered neutral on these two axes.

C. MATRIX FACTORIZATION MODEL

Matrix factorization is used by most of the latent factor models that is a user product rating matrix. The user product rating matrix represents both products and users by their ratings for different products. The matrix is decomposed into user factor matrix where each row is a user and product factor matrix where each column is a product of interest to identify the hidden factors.

Variety of input data are placed in the matrix of the recommendation systems. The most convenient data is explicit rating, which is a rating the user has explicitly given to the product. For example, Netflix asks users to rate a movie once they have watched it. As any user might have rated only a few percentage of products, explicit rating is generally a sparse matrix. Recommendation systems use implicit rating, when explicit rating is not available, which is based on a preference the user has somehow indicated. For example, by the number of purchases made, the browsing history of the user etc. As it represents an event's presence, implicit rating is generally a dense matrix.

To implement a recommendation system using a matrix factorization approach a large user-product rating matrix is broken down into lower dimensional user factor matrix and product factor matrix. Each rating is decomposed into 2 vectors.

$$\mathbf{r_{AB} = p_A * q_B} \qquad \text{– } \mathbf{(1)}$$

We can then estimate the user rating by multiplying the factors according to the following equation:

$$\mathbf{r_{ui} = p_{u},\ q_i} \qquad \text{– } \mathbf{(2)}$$

This is the equation for each rating of a prod i by user u.

This set of equations is solved for the set of ratings which exist. The resulting p's and q's is used to find the rating of any user for any product. After solving this set of equations, the rating that the user would give to any product is predicted. The predicted ratings are then sorted in descending order to find the top recommendations for a user.

D. LEARNING ALGORITHMS

Two approaches to solve the user product rating matrix are Stochastic Gradient Descent (SGD) and Alternating Least Squares (ALS).

1. STOCHASTIC GRADIENT DESCENT

Stochastic Gradient Descent (SGD) optimization algorithm iterates through all ratings in the data set. It gives the associated prediction by predicts r_{ui} for each given training case. This approach has easy implementation and has a relatively fast running time.

2. ALTERNATING LEAST SQUARES

Alternating Least Squares (ALS) is an approach to optimise the equations. Equation 2 is not convex as both q_i and p_u are unknowns. To solve the problem optimally, one of the unknows is fixed to make the optimisation problem quadratic. Thus, ALS techniques fixes the qi's and fixes the ps and rotates between them. The system fixes all p's and then the q_i's are recomputed by solving a least squares problem and the same is repeated for all q_i's. This decreases equation 2 until it converges. ALS is preferred when parallelization is used by the system. Each p_u and q_i is computed independently of the other user and product factors respectively. This parallelises the algorithm massively. It is also preferred when systems use implicit ratings. ALS is preferred in the cases where the data set is not sparse and looping over each single training case would not be practical.

IV. IMPLEMENTATION

Building a recommendation system in Spark is divided into two pats. The first part includes getting and parsing the data set into RDDs and persisting the RDD in disk. The second part is building the recommendation system using the complete dataset and printing the recommended artists for the user.

A RDD of strings is created from the raw User Artist data set. The data set is filtered to include only very strong ratings to reduce the amount of processing and the amount of data held in-memory. It is then converted to RDD of Rating objects before feeding it to ALS. ALS will pass over the RDD many times. So, it is persisted into disk that will make the computation faster. The model RDD returned by ALS has a recommendProducts method that takes the user ID and the number of recommendations the user wants. It gives the recommended Artist IDs for the user. The recommended artist names are printed using the lookup transformation applied on the RDD.

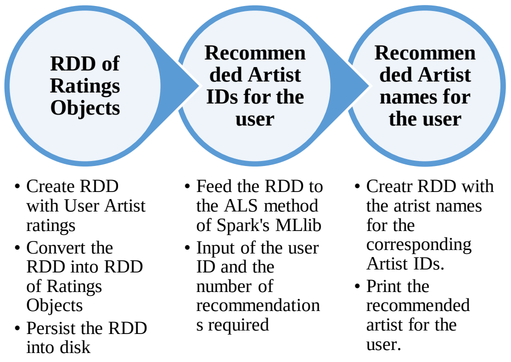

RDD of Ratings Objects
Recommended Artist IDs for the user
Recommended Artist names for the user
Create RDD with User Artist ratings
Convert the RDD into RDD of Ratings Objects
Persist the RDD into disk
Feed the RDD to the ALS method of Spark's MLlib
Input of the user ID and the number of recommendations required
Creatr RDD with the atrist names for the corresponding Artist IDs.
Print the recommended artist for the user.

Fig 3: Implementation steps

Step 1 – Create RDD with User-Artist ratings
RDD is created from the data set that contains the user artist ratings dataset. The ratings column from the data set is extracted using the stats method. Stats gives the descriptive measures of the ratings column like the number of ratings, the average number of times a user listened to an artist, the maximum and minimum rating. The data set may contain the artist that the user listened to very few times. The data set
is filtered to include only very strong ratings. Filtering low ratings will help reduce the amount of processing and the amount of data held in-memory.

Step 2 – Converting the data set into an RDD of Rating objects
rawUserArtitData is a RDD of strings. It is converted to RDD of Rating objects before feeding it to ALS. It takes the RDD of strings, splits the row into a list, filters out the ratings which are below a certain number and then converted into rating object.

Step 3 – Persisting it into disk
ALS will pass over the RDD many times. So, it is persisted into disk that will make the computation faster.

Step 4 – Feeding RDD to ALS method of Spark's MLlib
ALS has two methods that is train and trainImplicit. Our ratings are implicit, so we use the trainImplicit method. The parameters used in the method are the number of hidden factors, the maximum number of iterations, a lambda value that is used to control the quality of the ALS results. The choice of the parameters is driven by the dataset and the domain. There are separate techniques called the hyper-parameter tuning techniques to find the right values.

Step 5 – Input of the user ID and the number of recommendations required
The model RDD returned by ALS has a recommendProducts method that takes the user id and the number of recommendations the user wants. The recommendations are an RDD of Rating Objects with user, product and rating. The product column gives the recommended Artist IDs for the user.

Step 6 – Create RDD with the Artist names
RDD is created from the data set containing the raw artist data. The row is split into a tuple of Artist ID and Artist Nam. The rows are filtered corresponding to the user. It is persisted into disk.

Step 7 – Recommended artist names for the user
The artist names are extracted and collected into a list. The recommended artist names are printed using the lookup action applied on the RDD.

V. DATA SET

Audioscrobbler dataset is used in this framework. Audioscrobbler is an online music recommendation service. It is acquired by Last.fm. It contains 49481 samples of user artist rating data.

1. DATA SET WITH USER ARTIST RATINGS

The data set has user ID, Artist ID and #times user has listened to the artist. The data set can be seen as a User-Product rating matrix as it is an implicit rating. The user row is the user ID and the product row is the artist ID. Each cell represents the #times user listened to the artist.

user ID	Artist ID	# times user has listened to the artist
1059637	1000010	238
1059637	1000049	1
1059637	1000056	1
1059637	1000062	11
1059637	1000094	1
1059637	1000112	423

Table 1: User Artist Ratings Data

2. DATA SET WITH ARTIST NAMES

Audioscrobbler also provides a file to lookup the artist name for an artist id. It is loaded into a text file and split into tuple of (Artist ID, Artist Name).

Artist ID	Artist Name
1240105	Andre Visior
1240113	Riow arai
1240132	Outkast & Rage Against the Machine
1030848	Ravers Nature
6671601	Erguner Kudsi
1106617	Bloque
1240185	Lexy & K. Paul
6671631	Rev. W.M. Mosley
6671632	Labelle Patti
1240238	The Chinese Stars

Table 2: Artist Data

PSEUDO CODE
The following algorithm filters the strong ratings given by different users, provide the data set to the ALS method and provide recommendation for the particular user.

```
INPUT: Audioscrobbler Dataset
OUTPUT: Recommended artist names for the user
Load the data set as text file
from pyspark. mllib. recommendation import Rating, ALS
uaData=rawUserArtistData
. map (lambda y: y. split (","))
. filter (lambda y: (y [2])>=20)
. map (lambda y: Rating (y [0], y [1], y [2]))
uaData.persist()
model=ALS.trainImplicit(Data set, hidden factors, iterations, lambda)
Enter the user ID
recommendations=model. recommendProducts (userID, number of recommendations)
```

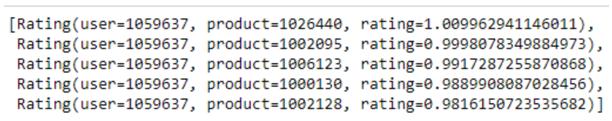
Load the Artist Data set
artistData.persist()
Print the recommendation

VI. RESULT

Collaborative filtering requires machine learning algorithms which are very complicated. Python and R have libraries which allow you to plug and play ML algorithms with minimal effort. These libraries are not suited for distributed computing. Hadoop MapReduce is great for distributed computing, but it requires you to implement the algorithms yourself as map and reduce tasks. Machine Learning algorithms are iterative, which means you need to make multiple passes over the same data. Hadoop MapReduce is heavy on disk writes which is not efficient for Machine Learning. Sparks ML Lib provides built-in Machine learning functionality in Spark. MLlib has built in modules for recommendation systems. Sparks library completely takes care of running the Machine learning algorithm across the cluster. It abstracts the programmer from the Machine Learning algorithm details of running it across the cluster. Since Sparks RDDs are in-memory it can make multiple passes over the same data without doing disk writes.

1. Recommended Artist ID for the user

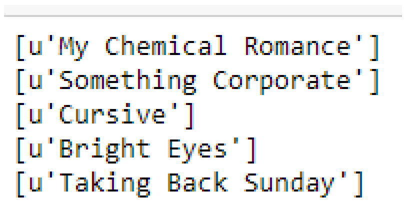
The recommendations are an RDD of ratings objects. The user column is the user ID. The product column gives the recommended Artist IDs.



```
[Rating(user=1059637, product=1026440, rating=1.009962941146011),
 Rating(user=1059637, product=1002095, rating=0.9998078349884973),
 Rating(user=1059637, product=1006123, rating=0.9917287255870868),
 Rating(user=1059637, product=1000130, rating=0.9889908087028456),
 Rating(user=1059637, product=1002128, rating=0.9816150723535682)]
```

2. Recommended artist names for the user

It gives the recommended artist names for the corresponding recommended Artist IDs by using the lookup action.



```
[u'My Chemical Romance']
[u'Something Corporate']
[u'Cursive']
[u'Bright Eyes']
[u'Taking Back Sunday']
```

VII. CONCLUSION

With the increasing data, making recommendation system that are faster and provide accurate recommendation has become a growing challenge. It requires an efficient and scalable algorithm and implementations. Latent factor collaborative filtering is an accurate algorithm that is used in recommendation systems. It uses matrix factorization method to identify the hidden factors that influences a user's rating. ALS is an optimization approach to solve the equations of the factorization method. Spark's Mllib is best suitable for applying ALS algorithm. It has a built-in class for applying ALS on any user product rating matrix. The matrix is fed into ALS method of the Mllib. It is just required to have a good dataset with user-product ratings. The algorithm takes care of finding out the hidden factors that influence user's preferences.

VIII. REFERENCES

[1] S. Panigrahi, R. K. Lenka, and A. Stitipragyan, "A Hybrid Distributed Collaborative Filtering Recommender Engine Using Apache Spark," Procedia Comput. Sci., vol. 83, no. BigD2M, pp. 1000–1006, 2016.
[2] B. Kumar, "A novel latent factor model for recommender system," J. Inf. Syst. Technol. Manag., vol. 13, no. 3, pp. 497–514, 2016.
[3] B. Kupisz and O. Unold, "Collaborative filtering recommendation algorithm based on Hadoop and Spark," 2015 IEEE Int. Conf. Ind. Technol., pp. 1510–1514, 2015.
[4] M. Winlaw, M. B. Hynes, A. Caterini, and H. De Sterck, "Algorithmic Acceleration of Parallel ALS for Collaborative Filtering: Speeding up Distributed Big Data Recommendation in Spark," 2015.
[5] T. Yu, O. J. Mengshoel, A. Jude, E. Feller, J. Forgeat, and N. Radia, "Incremental Learning for Matrix Factorization in Recommender Systems," pp. 1056–1063, 2016.
[6] X. Zeng, B. Wu, J. Shi, C. Liu, and Q. Guo, "Parallelization of Latent Group Model for Group Recommendation Algorithm," 2016.
[7] A. Wijayanto, "Implementation of Multi-Criteria Collaborative Filtering on Cluster Using Apache Spark," pp. 0–4, 2016.
[8] Z. W. Wenyuan Xu, Yun Li, Liting Wei, "A Feature-based Food Recommendation on Apache," 2017.
[9] X. Tu, "Improving Matrix Factorization Recommendations for Problems in Big Data," pp. 193–197, 2017.
[10] S. Liu, X. Tu, and R. Li, "Unifying Explicit and Implicit Feedback for Top-N Recommendation," pp. 35–39, 2017.
[11] B. K. Sunny, P. S. Janardhanan, A. B. Francis, and R. Murali, "Implementation of a Self-Adaptive Real Time Recommendation System using Spark Machine Learning Libraries," pp. 1–7, 2017.
[12] S. Gholamian, W. Golab, and P. A. S. Ward, "Efficient Incremental Data Analytics with Apache Spark," pp. 2859–2868, 2017.

[13] T. Zhao, J. Mcauley, and I. King, “Improving Latent Factor Models via Personalized Feature Projection for One Class Recommendation Categories and Subject Descriptors,” CIKM 2015 Proc. 24th ACM Int. Conf. Inf. Knowl. Manag., pp. 821–830, 2015.

[14] S. Howal Assistant Professor, M. Mote, and M. Vanjari, “Movie Recommender Engine Using Collaborative Filtering,” vol. 141401, 2017.

[15] J. Domann, J. Meiners, L. Helmers, and A. Lommatzsch, “Real-Time news recommendations using apache spark,” CEUR Workshop Proc., vol. 1609, pp. 628–641, 2016.