Big Data Analytics (20 points)

Assignment 6 (Experimenting with Recommender Systems)

Problem 1 (5 points). Experimenting with parameters for movieLens dataset on a single node.

For this assignment, you should build a recommender system for the MovieLens users using the 100K dataset available in (http://grouplens.org/datasets/movielens/), experiment with various parameters, and evaluate your recommendation system using both prediction and IR based measures.

What you need to do (Detailed instructions)

1. Download the dataset. Your input file will be "ua.base" which contains all the ratingsinformation. This data set includes 100000 ratings by 943 users on 1682 items
2. **Decide whether you need to use item-based or user-based recommender for this data**. (Choose the one that is more efficient for thisdata)
3. Use Mahout to build your recommender system and evaluate it using the following measures:
    - **a.** Mean absolute error (i.e., AverageAbsoluteDifferenceRecommenderEvaluator)
    - **b.** Root Mean Squared Error (i.e., RMSRecommenderEvaluator)
    - **c.** Precision
    - **d.** Recall

Create a java class called *EvaluateRecommender* with a main method and add an inner class called *MyRecommenderBuilder* that implements the *RecommenderBuilder* interface and builds your (item or user-based recommender). Then in your main method create an instance of each evaluation metric above and call the evaluate method for each metric as explained in the lectures.

Do not create different classes for different evaluation metrics. Instantiate all the evaluation metrics in the main method of your EvaluateRecommender class.

4. Once you successfully run your EvaluateRecommender code and get the evaluation measures, experiment with different parameters when building your recommender system. If you are using a user-based recommender try your code for various neighborhood sizes (specified in the constructor of **NearestNUserNeighborhood** **http://archive-primary.cloudera.com/cdh4/cdh/4/mahout-0.7-cdh4.3.2/mahout-core/org/apache/mahout/cf/taste/impl/neighborhood/NearestNUserNeighborhood.html)** and different similarity metrics. If you are using an item-based recommender, try running your recommender for various similarity metrics we studied in module 11. You can find a list of available similarity measures here: https://builds.apache.org/job/Mahout-Quality/javadoc/

What you need to turn in

1- Your EvaluateRecommender.java file.

2- A chart that visualizes the performance of recommenders in your experiment. You can create the chart in Excel and submit your excel file. Below is the description of what you should include in your chart:

   a. **If you are using a user-based recommender**, you should create a chart (similar to what is in slide 16 of the lab lecture notes for module 8). Your x-axis should be various neighborhood sizes and your y-axis should be the performance value (use Root Mean Squared Error metric). Plot for different similarity measures (Euclidean, Pearson, and LogLikelihood) and put all the plots in one chart for comparison.

   b. **If you are using an item based recommender**, the only parameter that you can experiment with is the type of similarity measure that you use. So your chart will be a bar chart with different similarity measures on the x-axis and their performances on the y-axis.

Problem II (15 points). Building Artist Recommendation System

Your task for this assignment is to use spark ALS matrix Factorization recommender system for recommending music to the last.fm users and evaluate your recommender system.

The Dataset

The dataset we will be using is a snapshot of the AudioScrobller dataset and can be downloaded from here: http://bit.ly/1K iJdOR . This dataset is about 500MB uncompressed. Download the archiveand extract it and inspect the"readme" file to understand the format of each file in the dataset. The dataset includes the following files:

- **User_artist_data.txt :** This file records about 24.2 million users' plays of artists and has three columns separated by spaces:

  ```
  userid artistid playcount
  ```

  Where the *playcount* is the number of times that *userid* played a record from *artistid*.

- **artist_alias.txt:** when plays are scrobbled, the client application submits the name of the artist being played. This name could be misspelled or nonstandard, and this may only be detected later. For example, "The Smiths," "Smiths, The," and "the smiths" may appear as distinct artist IDs in the data set, even though they are plainly the same. So, the file artist_alias.txt maps artist IDs that are known misspellings or variants to the canonical ID of that artist. The file artist_alias.txt has two fields:

  badid goodid

Data Preparation and pre-processing:

The rating values are implicit. Therefore, you should use ALS.trainImplicit (http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.recommendation.ALS$) method in spark to build the recommender system.

Before feeding the data to the ALS model you need to map every artistid in the file user_artist_data.txt to its canonical artistid using the file artist_alias.txt. Note that most artistids in user_artist_data.txt are already in their canonical form; which means they do not appear as badid in artist_alias.txt. You have to replace only the ones that are badids with their corresponding goodids. For example, If user_artist_data has:

1,2,10
2,3,10

and artist_alias has:

2,4 (2 is a bad id and 4 is its corresponding good id)
1,3 (1 is a bad id and 3 is its corresponding good id)

After replacing the bad_ids in your user_artist_data you should get the following:

1,4,10
2,3,10

Where bad id 2 is replaced by its corresponding good id 4.

ATTENTION: The fields in artist_alias.txt are predominantly delimited by tab. However, in artist_alias.txt there are quite a few records that are delimited by space instead of tab and in some records first or second columns are empty.

Running ALS recommender in spark:

Once you preprocess your data, you can split it to training and testing set and use ALS.trainImplicit method on the training set to build your recommender mode.

In iterative algorithms, rdd lineage can get deeply nested and causes the program to throw stack overflow. To avoid this problem, set spark checkpoint directory using the following method:

sc.setCheckPointDir(<a directory on hdfs for checkpointing>)

This enables spark to truncate rdd lineage graph and store it on hdfs instead of keeping it all in memory. If this property is set, ALS checkpoints every 10 iterations by default.

Using ALS to recommend products for users and evaluating your ALS model

As explained in the paper "Collaborative Filtering for Implicit Feedback Datasets", by Yifan Hu et. al.,

(http://yifanhu.net/PUB/cf.pdf) the precision based measures and root mean squared based measures are not appropriate for a dataset with implicit rating. The precision based metrics require knowing which products are undesirable for the user (false positive rate) and that information is not available with implicit data. At the other hand RMSE measures are also not appropriate because they use explicit rating values for evaluation.

For this assignment, you should use the measure explained in the paper (section 6, Evaluation Methodology). To this end, first you need to use ALS.predict method to predict the ratings **for users and products in the testing set**. Then given the actual and predicted ratings for each user u, you need to compute rank(u) as follows:

$$Rank(u) = \frac{\sum_i actualRating(u,i) * r(u,i)}{\sum_i actualRating_{(u,i)}}$$

Where actualRating(u,i) is the rating that user *u* gave to item *i* in the test set and *r(u,i)* is the percentile ranking for product *i* in the sorted recommended set. This measure basically compares how the products are ranked in the recommendation set compared to the testing set. The lower values of rank(u) are better. For completely random predictions, the expected rank is 50%. Thus any value above 50% indicates that our model performs no better than a random predictor.

For example, if for user u=1, the actual and predicted ratings are as follows:

Actual= {Rating(1, 2, 3.0), Rating(1, 3, 5.0), Rating(1,4, 4.0), Rating(1,1,1.5)}

Predicted= {Rating(1,2,2.5), Rating(1,3,1.5), Rating(1,4,3.0), Rating(1,1,5)}

Then we first sort the predicted set in descending order of rating values:

Predicted_sorted = {Rating(1,1,5), Rating(1,4,3.0), Rating(1,2,2.5), Rating(1,3,1.5)}

Then for each product in the actual rating set, find the percentile rank of that product in the testing set and calculate the rank for user u=1 as stated in formula above.

For example, the percentile rank of product 2 in the predicted_sorting is:r(1,2)= 3/4. The percentile rank of product 3 is r(1,3) =4/4= 1, the percentile rank of product4 is: r(2,4) is 2/4 = 0.5, and the percentile rank of product 1 is r(1,1)=1/4.

Hence:

$$rank(1) = \frac{3.0 * \frac{3}{4} + 5.0 * \frac{4}{4} + 4.0 * \frac{2}{4} + 1.5 * \frac{1}{4}}{3.0 + 5.0 + 4.0 + 1.5}$$

The overall performance of your recommender is then the average of rank(u) over all users:

$$\overline{rank} = \frac{1}{n} \sum_{u=1}^{n} rank(u)$$

When I run my program on user_artist_data.txt I get 0.03 for average rank which is <<0.5 and indicates that my model performs much better than a random predictor. You can play with different values of parameters for ALS.implicitTrain and see which combination gives you a lower average rank.

What you need to turn i

1- Your .scala or .py spark program including all your spark statements for preprocessing and cleaning data, building the ALS model and evaluating your model.

2- The best value your model achieved for average rank. You can just write down the value with your submission.

Good luck and let me know if you have any questions