CENG 790: Big Data Analytics, Fall 2020

Assignment 2: Recommender Systems

This assignment is part of the evaluation of CENG 790 course and it should be done individually. The deadline is 6 December 2020 23:59. The late submissions will receive a penalty of %25 for each day, starting from 7 December 2020 00:00.

You should submit a report containing the answers of the questions for each part. You may provide the screenshots of the output where necessary. You should also submit the code you produced to answer the questions. Make sure that your code is understandable, use comments extensively to explain what code piece is written for which question.

Make sure you follow the METU Academic Integrity guide. Copying code from internet is also considered as cheating. Make sure the code you produced is original.

Recommender systems are among the most popular applications of data analytics. The ultimate goal is to predict whether a customer would like a certain item: a product, a movie, a song, etc. A key concern is the scaling factor of such systems, since computational complexity increases with the size of a company's customer base. In this assignment, you will learn how to use Spark's MLlib to build a movie recommender system.

Dataset

The dataset comes from the MovieLens Website. It is very often used by researchers and industrials to experiment with a variety of recommender systems and compare their performance. Since this course advertises Web-Scale analytics, we obviously select the largest dataset available: MovieLens 20M Dataset. This dataset contains 20M user ratings for over 27k movies (ratings.csv). In this file, users and movies are identified with integers, and ratings are floating point values from 1 to 5. Using this information, we can compute recommendations using a collaborative filtering approach. Furthermore, meta-data about movies is available. For this assignment, we will use the 18 movie genres that are used to annotate movies. This information is available in movies.csv along with the title of each movie. Further information about the dataset can be found from the website.

Part 1: Collaborative Filtering

Collaborative filtering recommender systems rely on user ratings to predict unknown ratings. For this part, we will use a method of the Matrix Factorization family called ALS that was presented in class. ALS is available in Spark through the MLlib library.

Train a model and tune parameters

First, you need to load the dataset into the RDD ratings. The tuples in this RDD are instance of:

org.apache.spark.mllib.recommendation.Rating (user: Int, product: Int, rating: Double)

In order to get more accurate predictions, you should normalize the ratings per user with avgRatingPerUser.

You are ready to train an ALS model. The train method of ALS, has the following parameters:

• rank - is the number of latent factors in the model.

• iterations - is the number of iterations of ALS to run.

• lambda - specifies the regularization parameter in ALS.

Different values of these parameters result in different models and consequently different quality in predictions. To compare different models, you will use Mean Squared Error (MSE) as an evaluation criterion. To be more objective in the evaluation process, we split the data in train and test sets of sizes 8:2. We train and tune parameters on the 80% of the data, and then test on the 20% we put aside at the beginning.

In a file named ParameterTuningALS.scala, provide the code for the split and calculation of MSE. MSE is a simple sum of the errors we make in our prediction compared to the true ratings.

Ideally, we want to try a large number of combinations of parameters in order to find the best one, i.e. the one with lowest MSE. Due to time constraints, you should start by testing only 12 combinations:

1. Change the values for rank, lambda and iteration and create the cross product of 2 different ranks (8 and 12), 3 different lambdas (0.01, 1.0 and 10.0), and two different numbers of iterations (20 and 30). What are the values for the best model? Store these values, you will need them for the next question.

Getting your own recommendations

Now that you have experimented with ALS, you know which parameters should be used to configure the algorithm. Create another file named CollabFiltering.scala. Our objective now is to go beyond aggregated performance measures such as MSE, and see if you would be satisfied by the recommendations of the system you just built. To do this, you need to add you as a user in the dataset.

2. Build the movies Map[Int,String] that associates a movie identifier to the movie title. This data is available in movies.csv. Our goal is now to select which movies you will rate to build your user profile. Since there are 27k movies in the dataset, if we select these movies at random, it is very likely that you will not know about them. Instead, you will select the 100 most famous movies and rate 25 among them.
3. Build mostRatedMovies that contains the 100 movies that were rated by the most users. This is very similar to wordcount, and finding the most frequent words in a document.

   Obtain selectedMovies List[(Int, String)] that contains 25 movies selected at random in mostRatedMovies as well as their title. To select elements at random in a list, a good strategy is to shuffle the list (i.e. put it in a random order) and take the first elements. Shuffling the list can be done with scala.util.Random.shuffle.

4. You can now use your recommender system by executing the program you wrote! Write a function getRatings(selectedMovies) gives you 25 movies to rate and you can answer directly in the console in the Scala IDE. Give a rating from 1 to 5, or 0 if you do not know this movie.
5. After finishing the rating, your program should display the top 20 movies that you might like. Look at the recommendations, are you happy about your recommendations? Comment.

Part 2: Content-based Nearest Neighbors

For collaborative filtering, you relied purely on rankings and did not use movie attributes (genres) at all. For this part of the assignment, you will use a different method: content-based recommendation. Our goal here is to build for each user a vector of features (genres) describing their interests. Then, we will find the k users that are most similar using those vectors and cosine similarity to obtain a recommendation.

1. For this part, you will transform ratings into binary information. There are movies the user liked and movies the user did not like. In a file named NearestNeighbors.scala, build the goodRatings RDD by transforming the ratings RDD to only keep, for each user, ratings that are above their average. For instance, if a user rates on average 2.8, we only keep their ratings that are greater or equal to 2.8.
2. Build the movieNames Map[Int,String] that associates a movie identifier to the movie name. You have already done this in the previous part of this assignment.
3. Build the movieGenres Map[Int, Array[String]] that associates a movie identifier to the list of genres it belongs to. This information is available in the movies.csv file, in the third column, and movies are separated by "|". If you use split, you will need to write "\\|" as a parameter.
4. Provide the code that builds the userVectors RDD. This RDD contains (Int, Map[String, Int]) pairs in which the first element is a user ID, and the second element is the vector describing the user. If a user has liked 2 action movies, then this vector will contain an entry ("action", 2). Write the userSim function that computes the cosine similarity between two user vectors. The mathematical formula is available on the slides. To perform a square root operation, use Math.sqrt(x).
5. Now, write a function named knn that takes a user profile named testUser. Then the function selects the list of k users that are most similar to the testUser, and returns recommendation, the list of movies recommended to the user.
6. Congratulations, you can now experiment with your recommender system by modifying the vector of testUser and see which recommendations you get. Use the profile you built for yourself in Part 1 and list the recommendations. Comment on the performance of recommendations. Also, compare the two methods you implemented in Part 1 and 2.

HAVE FUN and GOOD LUCK !