Recommender Systems using Collaborative Filtering

Pratik Nikhil Chaudhari

Aditi Anil Raghuwanshi

12/12/2021

—

ITCS 6190: Cloud Computing for Data Analysis

—

Prof. Srinivas Akella, Ph.D.

Contents

Introduction

The aim of our project is to implement a recommender system for movies. The system will be implemented in Spark such that it can be executed on the DSBA Hadoop cluster.

Motivation

One of the key reasons why we need recommendations is that now people have too many options to choose from. Given the history of user's opinions about an item, having a system that can generate choices keeping in mind the history of these opinions helps to provide a more user-oriented experience.

Having said that, the aim of our project is to develop a movie recommendation system that will suggest new movies based on the movies the user has previously watched and rated. We will be using collaborative filtering algorithms to implement the system.

Challenges

Since we are using Spark on DSBA cluster, we faced issues in terms of creating matrices and collecting on a single node for eigen values and eigen vectors calculation for SVD.

Dataset Information

The following statistics is available in the README file of the Movie Dataset.

Dataset size – The dataset has 3 files – ratings.csv, users.csv and movies.csv. Files have data present in the following format –

File name	Available attributes	Project specific attributes
ratings.csv	UserID, MovieID, Rating, Timestamp	UserID, MovieID, Rating
users.csv	UserID, Gender, Age, Occupation, Zip-code	NA*
movies.csv	MovieID, Title, Genres	MovieID, Title

**Since we already have user ID information available in ratings.dat, we won't be using the users.dat file.*

The ratings file has approximately 1 million ratings – each user has rated at least 20 movies.

For implementing SVD, we had to perform some data analysis and filtering task since decomposing matrices and performing eigen computations was very memory exhaustive.

Collaborative Filtering using Neighborhood Method

Neighborhood Based Collaborative Filtering leverages the behavior of other users to know what our user might enjoy. It may find people similar to our user and recommend stuff they liked or recommend stuff that other people bought after buying what our user has bought. This concept can be extended to items as well.<Add reference here>. For example, if a person A has same opinion as a person B on an issue. Then A is more likely to have B's opinion on different issue.

Basic idea of neighborhood method is –

1. Given a target user *U* find the movies U likes M_u.
2. Based on the movie names we get, find a set of users S_u who have rated these movies.
3. Find other movies rated by S_u and calculate similarity between M_{Su} and M_u
4. Recommend the most similar movies.

For the project implementation, we have used item-based collaborative filtering in which we create item-item pairs and calculate similarity between them.

We use two variables that help to find and filter movies which are the most similar to our input movie –

1. *score threshold*: how good the cosine similarity score is
2. *coOccurence threshold*: how frequently the pair is rated

Algorithm –

Main program –

- *Initialize Spark Context*
- *Initialize score threshold and coOccurence threshold*
- *Read input movie ID*
- *Read input csv files of ratings and movies into a data frame. Filter out userID, movieID, rating column (as these are the only ones required)*
- *Create all possible movie pairs of each user by performing self-join operation on the data frame. Later filter our redundant pairs.*
- *Compute similarity between the movie pair. We use cosine_similarity function to calculate that.*
- *For each movie pair the cosine similarity score and num_pairs will be part of the data frame.* ***We will cache this data frame as it'll be needed for filtering***
- *Based on the score threshold and coOccurence threshold, we will filter the data frame to get movie IDs of the movies to be recommended.*
- *Find these IDs in the movie data frame and fetch titles.*

Cosine similarity calculation –

```
func cosine_similarity(ratingPairs) :
  for pair in list(ratingPairs) do
      rating1 = pair[0] #first term of the pair
      rating2 = pair[1] #second term of the pair

      sum_11 += rating1 * rating1
      sum_22 += rating2 * rating2
      sum_12 += rating1 * rating2
```

```
    num_pairs <- num_pairs + 1    #number of terms the summation will be calculated

   numerator:Double = sum_12
   denominator = sqrt(sum_11) * sqrt(sum_12)
done
```

Program Implementation –

The spark job is wrapped in a bash script that will require a movie ID to be passed as command line argument. This ID will be read and submitted to the job. The internal working of the code is entirely based on data frame – thanks to the structured APIs that Spark 2.0+ gives.
On successful execution of the program, the output is displayed in form of a list of top 10 movies that are ranked high based on our threshold parameters and its corresponding similarity score.

Collaborative Filtering using Singular Value Decomposition

Results

Drawbacks of the current approach

One of the major drawbacks of Collaborative filtering is that when new items are added to the system, they need to be rated by a substantial number of users before they could be recommended to users who have similar tastes to the ones who rated them. Similarly, to recommend a movie to a new user, that user should have rated a sufficient number of movies.

For SVD, we have observed that with the current cluster configuration, we were able to work with a max of 600 movies. As the user-movie matrix increased in dimension, the matrix increased in density that caused Java heap space overflow issues.

Project Aspects covered

Expectation and Delivery –

Part	Task	Completion Status
What we said we will definitely accomplish	Implement item-based collaborative filtering	Complete
	Dataset cleaning and preprocessing (for SVD)	Complete
	Implement Singular Value Decomposition (SVD)	Complete
	Wrapper scripts/program to take user input, process it and send as command-line input to Spark program	Complete
	Recommend upto 10 movies that rank highly similar to input movie ID	Complete
What we said we will likely accomplish	Implement item-based collaborative filtering	Complete
	Dataset cleaning and preprocessing (for SVD)	Complete
	Implement Singular Value Decomposition (SVD)	Complete
	Wrapper scripts/program to take user input, process it and send as command-line input to Spark program	Complete
	Recommend upto 10 movies that rank highly similar to input movie ID	Complete
	Add flexbility of adding new movies with their ratings and include them in calculations	Could not achieve
What we said we ideally will accomplish	Implement item-based collaborative filtering	Complete
	Dataset cleaning and preprocessing (for SVD)	Complete
	Implement Singular Value Decomposition (SVD)	Complete
	Wrapper scripts/program to take user input, process it and send as command-line input to Spark program	Complete
	Recommend upto 10 movies that rank highly similar to input movie ID	Complete
	Add flexbility of adding new movies with their ratings and include them in calculations	Could not achieve
	Develop an HTML interface to get this information to get the rating and movie information from the user	Could not achieve

Division of Tasks –

Task	Owner
Implement item-based collaborative filtering	Aditi
Documenting approach for item-based collaborative filtering	Aditi
GitHub page for uploading report	Aditi
Wrapper scripts/program to take user input, process it and send as command-line input to Spark program	Aditi
Dataset cleaning and preprocessing (for SVD)	Pratik
Implement Singular Value Decomposition (SVD)	Pratik
Documenting approach for SVD	Pratik
Final deliverable package with README consisting of run instructions	Pratik