Netflix Recommendation System - Group 12

Nisarg Vinchhi - nvinchh
Rohit Mandge - rnmandge
Vaibhav Rajadhyaksha - vrajadh

Abstract

The concept of recommendation systems of late is becoming a norm in this data driven world. Popular entertainment websites like Netflix, Hulu, Spotify etc are increasingly leaning towards learning the interests of the users, and recommending them with similar titles which will help boost their market. The application of machine learning techniques to make sense of unstructured data has transformed the paradigm of the movie recommendation and rating prediction to a much more reliable, accurate model. There are certain challenges in building such systems given the individual preferences of each user in rating the movies. However, normalizing the data and observing patterns of user ratings over similar movies will help in predicting the rating of a movie.

This paper presents one such prediction and recommendation system which includes grouping similar movies and users over various similarity metrics as a main agenda and building a prediction system to achieve lesser error than Netflix's own prediction system. For grouping similar entities, we have used Pearson correlation coefficient as the similarity metric as well the Alternating Least Squares method which uses the latent factors to identify similarity. For rating prediction system, we learned the linear regression model over the normalized rating data for each user/movie pair available and predicted the ratings of unseen data.

Introduction

Recommendation systems try to recommend items (movies, music, web pages, products, etc) to interested potential customers, based on the information available. A successful recommendation system can significantly improve the revenue of e-commerce companies or facilitate the interaction of users in online communities.

Recommendation systems are generally constructed using two different methods, content based filtering and collaborative filtering. Content based filtering uses the features of users or items. In some cases, it is difficult to extract features from the items which makes the recommendation task difficult. Collaborative Filtering on the other hand just uses the ratings of items given by users to predict ratings of new user-item pairs. The main idea behind collaborative filtering is that two users probably continue choosing similar products if they have already chosen similar ones. We will consider Collaborative Filtering for building recommender systems in this project.

Collaborative filtering tries to identify relationships and interdependencies among the users and the movies that they are rating. The only required information is the past habits of the users like the ratings they have given or the times they have watched a particular movie. Explicit feedback where users rate movies directly are most convenient as they are a direct indication of the amount of interest the user has in a particular movie. However, there are certain situations in

which explicit feedback is not enough. For e.g.: initially a user watched a movie and gave it an average rating. However, he watched it again and liked it and went on to watch it many more times. Based on the rating which we have, we feel the user didn't like the movie much but that is not the case. This is when implicit feedback comes into the picture. Implicit feedback indirectly reflects the opinions of users by observing their past behavior.
Collaborative filtering can be achieved by implementing various algorithms such as regression, clustering, matrix factorization, latent class models and Bayesian models. In this project we have used the Pearson Correlation Coefficient in the explicit feedback model and Alternating Least Squares latent class model in the implicit case for finding similarity between entities.

System Requirements

- Software Requirements:
  - The data structure that we used is RDDs and Nested dictionaries in Python.
  - Frameworks used for parallel and distributed computing is Apache Spark
  - Operating System used: Ubuntu 14.04.4 LTS

- Hardware Requirements:
  - Memory (RAM) Requirements. 2.9 GB per node (total 25.7 GB)
  - Storage (ROM) Requirements. 6 GB minimum (size of the dataset)
  - Processor Requirements. 8 cores per node (total 72 nodes)

Data

The Netflix Prize dataset consists of around 200 million records. Each movie name is mapped to a movie ID. For each movie ID a separate record is available which is of the form "userid, rating, time". Userid is an integer and rating defined as an integral scale from 1 to 5. Date is represented as a timestamp.

Related Work

Recommendation systems have been in use since a long time and primarily there are two main classes of these. Collaborative filtering and Content based filtering. As described before we will be using the collaborative filtering approach, based on the dataset that we are using. Since we know the ratings given to the movies by different users, it was logical to use this past history to identify relationships and recommend new movies to the users. Initially we decide to use the euclidean distance as a metric to identify similarity between the movies .
However we were faced with the grade inflation problem. If two users like the movie equally and one is a bit more critical than other ,and gives a lesser rating, then even though both have liked the movie equally there would be a vast difference in the similarity scores among the movies based on the ratings. To normalize things we have gone for the pearson correlation coefficient our similarity metric.

Hadoop is a distributed data infrastructure, that distributes data among multiple nodes, with a cluster of commodity servers. It can be used along with the map-reduce framework to for

processing data parallely. The mapreduce workflow look like : Read data from the cluster, perform an operation, write results to the cluster, read updated data from the cluster, perform next operation, write next results to the cluster.Data is written to disk after every operation and hence it's pretty resilient.
In our approach we have used the apache spark cluster computing framework. It was an ideal choice considering the fact that it processes data in-memory and near real-time. The PySpark module exposes the spark programming model to python. The resilient distributed dataset(RDD) is the basic data structure in spark. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.These were the key factors that influenced our decision to go for spark. Spark also provides MLib which is a rich machine learning library consisting of common learning algorithms.

Analysis

Data Extraction and Preprocessing:

The original data consisted of multiple txt files in the training set, per each movie consisting of userid, rating and the date of the review. We wrote a python code (*preprocess.py*) to preprocess this data into a desired format that consisted of multiple csv files with different sized data sets to identify the performance of our algorithm and spark. The resulting file, *movie_data.csv*, consists of the tuples in 'movieId, userId, rating and timestamp' format.
To measure the performance of the project, we wrote a python code (create_datasets.py) to divide the originally obtained *movie_data.csv* into four different datasets according to the size as following.
small_movie_data.csv - 20% of the original dataset
small_medium_movie_data.csv - 40% of the original dataset
medium_movie_data.csv - 60% of the original dataset
large_movie_data.csv - 80% of the original dataset
The results and performance measures are discussed in the experiment and results section.

Movie Recommender System:

For developing the recommendations system we have used the Collaborative filtering approach. In this technique we make predictions about interests of a user by considering information we have about similar users or users with similar tastes. The logic behind this is that if a Person X has the same opinion as a person Y on an issue, X is more likely to have Ys opinion on some different issue than have the same opinion as a randomly chosen person Z.

Recommending similar movies using Pearson Correlation coefficient:

In this approach we are trying to identify movies that haven't been watched but are similar to the ones already rated by the user. We will be recommending the top x movies to the users based on the similarity scores obtained. The pearson correlation coefficient was identified as the metric to measure similarity. Below is the formula for the same.

$$r = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{\sqrt{\left(\sum X^2 - \frac{(\sum X)^2}{n}\right)\left(\sum Y^2 - \frac{(\sum Y)^2}{n}\right)}}$$

Figure 1: Pearson Correlation Coefficient formula

Here,
X - The rating given to movie one by different users
Y - The rating given to movie two by different users
n - Total number of movies

It measures the strength of the linear relationship among the 2 entities being compared. Its value lies between -1 to 1. So a value of 1 indicates a strong correlation while a value of -1 indicates an extremely weak relationship. We chose this method as it corrects the problem of grade inflation, which is not the case with Euclidean distance. The grade inflation problem refers to when 2 users both like the movie a lot but one gives 7 as the best rating while other one gives a 10.
If we use Euclidean distance than even though both considered it as a great film, they would be far apart since there rating varies. PCC helps eliminate this problem by normalizing the results within a scale of -1 to 1. PCC indicates the correlation among the objects. The accuracy of the score increases when the data is not normalized . Hence this metric can be used when data is not normalized. It can also correct for any scaling within an attribute ,when the final score is being tabulated.

Working of the Algorithm:

The first step is to load the data from CSV file to an RDD. Then we filter the dataset using the year of rating. Then, we are loading the data into a dictionary that consists of the movieId mapped to the users and the rating that they have given the movies. We are also storing all the movies watched by the user into an RDD and a total list of movies.
For recommendation purposes, we will be filtering the movies based on the year as we feel are those are the most relevant recommendations to the user. We are calculating the PCC score of each movie that hasn't been watched by the user against the watched movies by comparing the number of users that have watched both those movies. We are multiplying that with the rating given to the watched movie which gives us the weighted similarity score. Finally we are sorting the results of the similarity scores and then sorting them, to return the top n movies.

Architecture and Execution:

We are using the apache spark cluster computing framework that uses the master slave architecture. We are using 10 ubuntu hosts to set up our cluster. One of the hosts is the master, and one other is being used for Zookeeper. ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization. 8 nodes are assigned as executor nodes that share the computing tasks. The *create_dataset.py* runs on the entire

dataset and create batches of datasets that are the input to the main program. The *recommender.py* consists of the main logic of the recommendation.
For the faster execution, we have placed the data files *movie_data.csv, small_movie_data.csv etc.* on each worker node along with the master node.

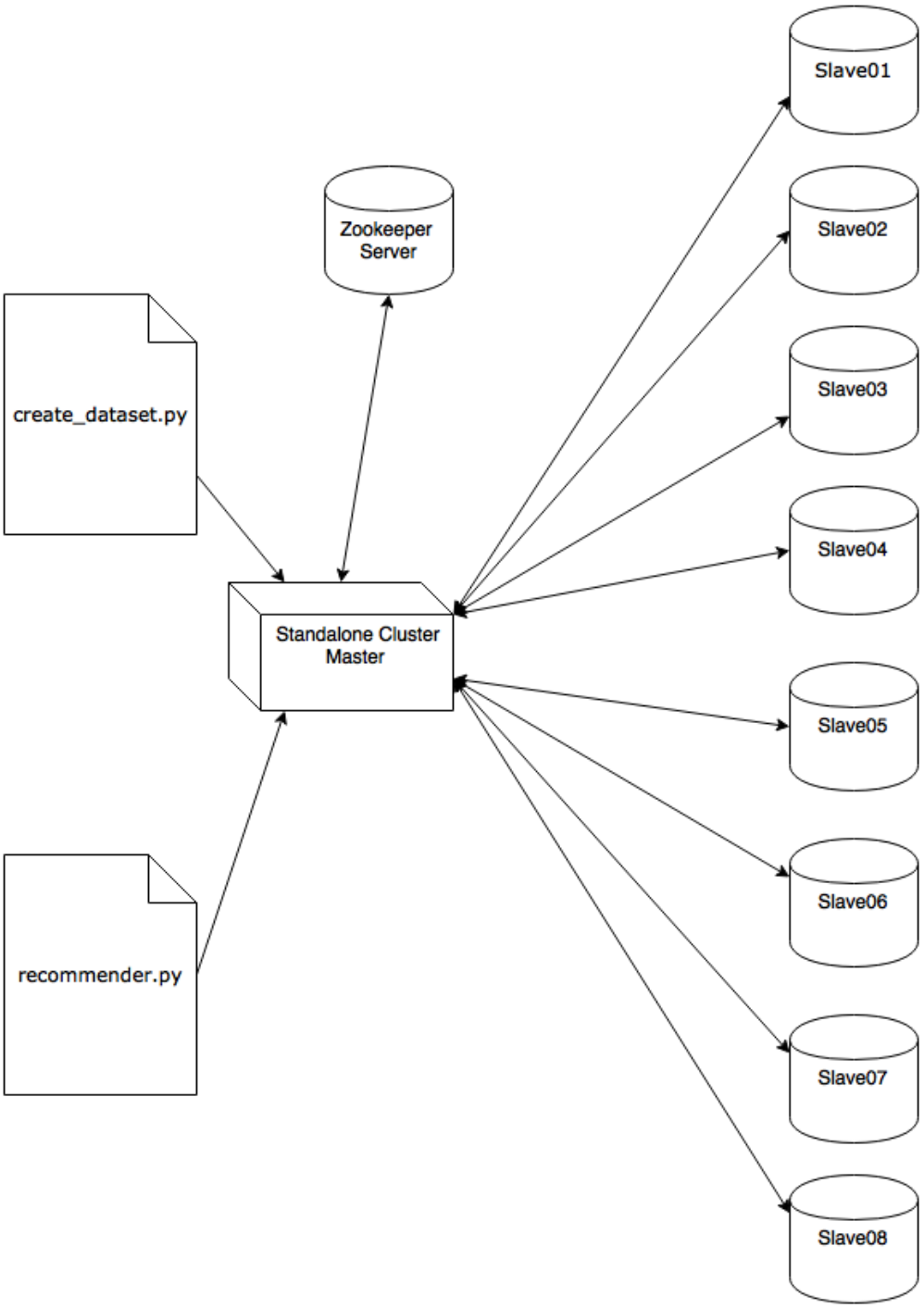

Figure 2: Architecture Diagram

Results

Below is a screenshot of the Spark UI that displays the various attributes of the executor nodes. It details the core, the memory in use, ip address as well as the state of the machine.

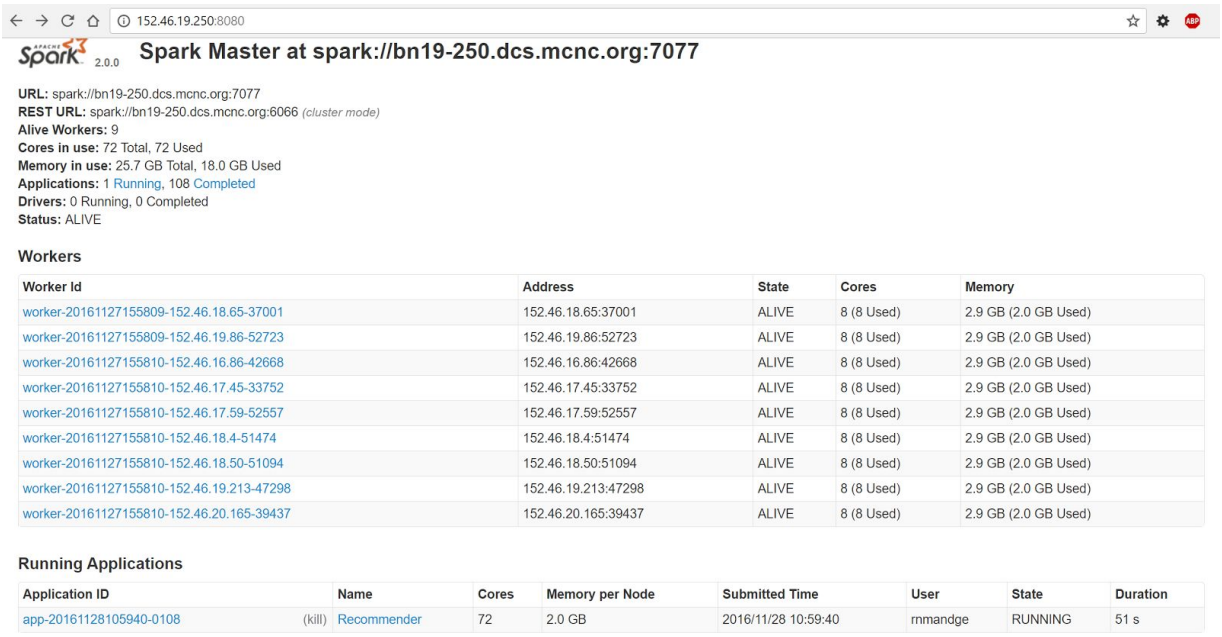

Figure 3: Spark Web UI

This is the final output where N movies are recommended to the user 2020475 for year 2005.

```
16/11/28 11:00:47 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 152.46.19.86:54875 (size: 4.3 KB, free: 912.3 MB)
16/11/28 11:00:55 INFO TaskSetManager: Finished task 15.0 in stage 2.0 (TID 47) in 8315 ms on 152.46.17.59 (1/16)
16/11/28 11:00:57 INFO TaskSetManager: Finished task 7.0 in stage 2.0 (TID 39) in 10515 ms on 152.46.19.86 (2/16)
16/11/28 11:00:57 INFO TaskSetManager: Finished task 8.0 in stage 2.0 (TID 40) in 10665 ms on 152.46.18.50 (3/16)
16/11/28 11:00:58 INFO TaskSetManager: Finished task 6.0 in stage 2.0 (TID 38) in 11095 ms on 152.46.17.59 (4/16)
16/11/28 11:00:58 INFO TaskSetManager: Finished task 10.0 in stage 2.0 (TID 42) in 11270 ms on 152.46.17.45 (5/16)
16/11/28 11:00:58 INFO TaskSetManager: Finished task 4.0 in stage 2.0 (TID 36) in 11473 ms on 152.46.18.65 (6/16)
16/11/28 11:00:58 INFO TaskSetManager: Finished task 11.0 in stage 2.0 (TID 43) in 11477 ms on 152.46.16.86 (7/16)
16/11/28 11:00:59 INFO TaskSetManager: Finished task 13.0 in stage 2.0 (TID 45) in 11686 ms on 152.46.18.65 (8/16)
16/11/28 11:00:59 INFO TaskSetManager: Finished task 1.0 in stage 2.0 (TID 33) in 11762 ms on 152.46.17.45 (9/16)
16/11/28 11:00:59 INFO TaskSetManager: Finished task 2.0 in stage 2.0 (TID 34) in 11887 ms on 152.46.16.86 (10/16)
16/11/28 11:00:59 INFO TaskSetManager: Finished task 9.0 in stage 2.0 (TID 41) in 12613 ms on 152.46.18.4 (11/16)
16/11/28 11:01:00 INFO TaskSetManager: Finished task 12.0 in stage 2.0 (TID 44) in 13117 ms on 152.46.20.165 (12/16)
16/11/28 11:01:00 INFO TaskSetManager: Finished task 5.0 in stage 2.0 (TID 37) in 13277 ms on 152.46.19.213 (13/16)
16/11/28 11:01:00 INFO TaskSetManager: Finished task 14.0 in stage 2.0 (TID 46) in 13363 ms on 152.46.19.213 (14/16)
16/11/28 11:01:00 INFO TaskSetManager: Finished task 0.0 in stage 2.0 (TID 32) in 13615 ms on 152.46.18.4 (15/16)
16/11/28 11:01:01 INFO TaskSetManager: Finished task 3.0 in stage 2.0 (TID 35) in 13703 ms on 152.46.20.165 (16/16)
16/11/28 11:01:01 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose tasks have all completed, from pool
16/11/28 11:01:01 INFO DAGScheduler: ResultStage 2 (collect at /home/rnmandge/final_machine_learning/Recommender/recommender.py:108) finished in 13.712 s
16/11/28 11:01:01 INFO DAGScheduler: Job 2 finished: collect at /home/rnmandge/final_machine_learning/Recommender/recommender.py:108, took 13.727548 s
One Man's Justice
The Godson
Option Zero
On Any Sunday
Dirty Work
16/11/28 11:01:37 INFO SparkContext: Invoking stop() from shutdown hook
16/11/28 11:01:37 INFO SparkUI: Stopped Spark web UI at http://152.46.19.250:4040
16/11/28 11:01:37 INFO StandaloneSchedulerBackend: Shutting down all executors
16/11/28 11:01:37 INFO CoarseGrainedSchedulerBackend$DriverEndpoint: Asking each executor to shut down
16/11/28 11:01:37 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
16/11/28 11:01:37 INFO MemoryStore: MemoryStore cleared
16/11/28 11:01:37 INFO BlockManager: BlockManager stopped
16/11/28 11:01:37 INFO BlockManagerMaster: BlockManagerMaster stopped
16/11/28 11:01:37 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
16/11/28 11:01:37 INFO SparkContext: Successfully stopped SparkContext
16/11/28 11:01:37 INFO ShutdownHookManager: Shutdown hook called
16/11/28 11:01:37 INFO ShutdownHookManager: Deleting directory /tmp/spark-85f3093d-fd95-4496-a56b-4cd57213de57/pyspark-52475fcf-2c73-4435-b9c7-28e763d9bf3e
16/11/28 11:01:37 INFO ShutdownHookManager: Deleting directory /tmp/spark-85f3093d-fd95-4496-a56b-4cd57213de57
rnmandge@bn19-250:~/spark-2.0.0-bin-hadoop2.7$
```

Figure 4 : Recommended Movies

Experimental Results

Dataset	Execution Time Single Node	Execution Time Cluster (8 workers)
small_movie_data (20%)	17 min	2.0 min
small_medium_movie_data (40%)	40 min	4.5 min
medium_movie_data (60%)	Out of memory error	9.3 min
large_movie_data (80%)	Out of memory error	14.0 min
full_movie_data (100%)	Out of memory error	21.0 min

As we can see from the above table that single node or a local machine could not handle such a large amount of data. To process the huge netflix dataset and achieve high performance, we adopted distributed computing approach using Apache Spark and obtained the results in a short span of time.

Conclusion

The biggest challenge was the processing of huge amounts of data, which was effectively handled by the spark computing framework. The distributed nature of loading and processing the data using the master slave architecture and RDDs ensured that we successfully obtained the desired results. Initially we tested the datasets on a single node (local) and it could not handle the data beyond 40% of the actual data. Then, we tested for these different sized data sets, ranging from 20% of the total data to the entire set on the spark cluster. The time taken to process went up incrementally, from 2 minutes to 20 minutes because of the intensive computation involved while calculating similarity metrics. The approach scales up as we add more nodes to the cluster for processing, ensuring effective data management. The in-memory processing capabilities and machine learning API's provide a unique edge to the Spark.

Future Scope

The recommendations are based on the logic developed using pearson correlation coefficient but aren't being validated. We would like to work on another approach for recommending movies based on a different algorithm and try to identify the match ratio between the two approaches.

References

- Apache Spark Documentation: http://spark.apache.org/docs
- Collaborative Filtering for Implicit Feedback Datasets, Yifan Hu,Yehuda Koren and Chris Volinsky

- Fast als-based matrix factorization for explicit and implicit feedback datasets, Istvn Pilszy, Dvid Zibriczky and Domonkos Tikk
- Large-Scale Parallel Collaborative Filtering for the Netflix Prize, Yunhong Zhou, Dennis Wilkinson, Robert Schreiber and Rong Pan