MOVIE RECOMMENDATIONS USING MAP REDUCE

CS 579 - ONLINE SOCIAL NETWORK ANALYSIS

Illinois Institute of Technology

J V P S Avinash | Rakshith Muniraju

INTRODUCTION

Recommendation Systems are quite popular among shopping sites and social networks these days. These recommendations help the site visitor find products to buy, new music to listen to, movies that should be watched etc. Generally, the user interaction data is available from the items and products in various shopping sites, movie websites. We extract this data to build the recommendation engine. In the current report, we will talk about Movie Recommendations and use Map Reduce Framework for analyzing this data. For this, we need to find the similar movie to the movie user is interested in and recommend him. To evaluate this, we need to compute the similarity between two movies. Various Similarity methods such as Jaccard, Cosine, Pearson, Regularized Pearson, and Generalized Jaccard are introduced in this report to find the relation between two items.

THE PROBLEM

The data coming from any movie website will be very large. Suppose, if we have a website which has 1000 movies, to find movie-movie similarity, we need to compute at least 10 lakh computations. Besides this computations, the correlation data will be sparse. This results in a matrix which contains more number of zeros than user ratings because it is unlikely that every pair of items will have some user interested in them.

Also another problem we have is that we have also to deal with the temporal aspect since the user interests in the product changes over the time. So the similarity between the ratings of movies should be done periodically so that the results are up to date.

The last problem which we are going to discuss is to handle large amounts of data. Since the data origins from various sources, we need a system to store this huge amounts of data. This section is related to the future work in this area.

THE HYPOTHESIS

For the above problems we encounter, the best way to handle this scenario is designing an algorithm which is most suitable for this divide and conquer patterns. Map Reduce is a powerful framework designed to approach these problems. “mrjob” in python helps us to write and run Hadoop Streaming jobs. It is used as an interface for Hadoop and it can be run in our Hadoop Cluster or can be run without installing the cluster. In the current project, we focused on running the job in local mode. In future, we store the data in Hadoop environment called Hadoop Distributed File System. HDFS provides a high throughput access to the stored data and is suitable for applications that have large data sets.

DATA

For this project, we thought of extracting the movie-user ratings from Rotten Tomatoes Movie Database API. From the response, we found out that it does not provide us with any details of that information. Also API failed to provide the information about all the movies released in those years.

We first wrote a web crawler to get all the movies released in the past two years. After extracting the movie information, we had written a web crawler to parse the contents in the URL Page to get the required information. We dumped the obtained output from the page into a text file.

METHODS

MAP REDUCE ALGORITHM

After loading the data to HDFS, we need to design Map-Reduce Framework to calculate the similarity between two movies. The following diagram illustrates the idea.

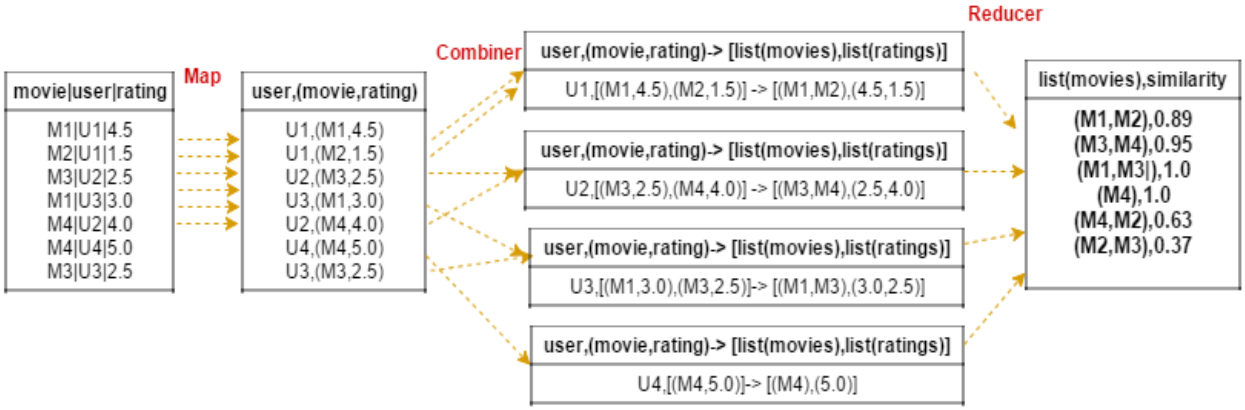

Figure 1 Typical Map Reduce Flow to calculate Similarity between Movies

We have used a Mapper, Combiner and Reducer to obtain the required output. The following gives the abstract information of how it is used.

map() function: - Mapper takes the input from the text file (movie-user ratings) and for each user emits a row containing their "postings" (item,rating).

combine() function: - Combiner takes the output from the Mapper as input. Each output is partitioned by key (user) and fed to the combiner. For each user in combiner, we generate a combination of item-item pairs and ratings for them. The output of the Combiner will be a combination of movies and ratings pair for an individual user.

reducer() function: - Reducer takes the output from the Combiner as input. Each output is partitioned by list of movie pairs and fed to the reducer. Reducer then generates two Movie Vectors. Similarity measures like Pearson Correlation, Cosine Similarity, Jaccard Similarities are applied on these vectors. These similarity measure along with movie pair is returned as output. From this measure, we can find the movies most similar to one another. The similarity measures employed are discussed below.

SIMILARITY MEASURES

Cosine Similarity: - We find the cosine of angle between two items. For a value of 0, it indicates both the items are not similar. For a value of 1, it indicates both the items are similar

Jaccard Similarity: - The Jaccard coefficient measures the similarity between finite ratings between the movies. It is the ratio of the intersection divided by the union of all the ratings.

Generalized Jaccard Similarity: - This is the most generalized jaccard similarity where we take the minimum rating from all the vectors and divide it with the maximum rating from all the vectors.
Pearson's Correlation: - This measures the strength of association between the movie vectors. In this case, nearer the scatter of ratings is to a straight line, higher the strength of association between the movie vectors.
Normalized Pearson's Correlation: - To bring the above Pearson's correlation to a value in between [0,1], we use this measure.

EXPERIMENTS

From the above points, we believe Map Reduce Framework helps us in running parallel jobs. The following are the experimental steps we followed.

- ✓ Load all the user-movie ratings to a file in local file system.
- ✓ Write a mapper function to read this file information into Map Reduce Framework.
- ✓ Write a Combiner function, such that, for every pair of movies, it finds all the people who rated both the movies.
- ✓ Form two vectors from these ratings.
- ✓ Write a Reducer function, such that, we find the relation between those two vectors by above measures.
- ✓ After finding the similar items, we can recommend the movies to the user that is most similar with it. The following is the sample output for our data.

```
In [16]: ! python movieSimilarities_with_MapReduce.py samp.csv
no configs found; falling back on auto-configuration
no configs found; falling back on auto-configuration
creating tmp directory /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399

PLEASE NOTE: Starting in mrjob v0.5.0, protocols will be strict by default. It's recommended you run your job
 with --strict-protocols or set up mrjob.conf as described at https://pythonhosted.org/mrjob/whats-new.html#r
eady-for-strict-protocols

writing to /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399/step-0-mapper_part-00000
Counters from step 1:
  (no counters found)
writing to /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399/step-0-mapper-sorted
> sort /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399/step-0-mapper_part-00000
writing to /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399/step-0-reducer_part-00000
Counters from step 1:
  (no counters found)
Moving /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399/step-0-reducer_part-00000 -> /tmp/mo
vieSimilarities_with_MapReduce.root.20151203.013408.703399/output/part-00000
Streaming final output from /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399/output
["Just My Luck", "Lady in the Water"]   [0.93601, 0.16667, 0.66667, -0.94491, 0.02754]
["Just My Luck", "Snakes on a Plane"]   [0.95762, 0.0, 0.65517, -0.33333, 0.33333]
["Just My Luck", "Superman Returns"]    [0.92657, 0.125, 0.6129, -0.42289, 0.28855]
["Just My Luck", "The Night Listener"]  [0.97489, 0.25, 0.7037, 0.55556, 0.77778]
["Just My Luck", "You, Me and Dupree"]  [0.91841, 0.125, 0.73913, -0.48566, 0.25717]
["Lady in the Water", "Snakes on a Plane"]      [0.99774, 0.1, 0.77778, 0.76376, 0.88188]
["Lady in the Water", "Superman Returns"]       [0.9838, 0.1, 0.7, 0.48795, 0.74398]
["Lady in the Water", "The Night Listener"]     [0.98188, 0.3, 0.875, -0.61237, 0.19381]
["Lady in the Water", "You, Me and Dupree"]     [0.97818, 0.125, 0.84, 0.33333, 0.66667]
["Snakes on a Plane", "Superman Returns"]       [0.97988, 0.35714, 0.83051, 0.1118, 0.5559]
["Snakes on a Plane", "The Night Listener"]     [0.97065, 0.16667, 0.78723, -0.56635, 0.21682]
["Snakes on a Plane", "You, Me and Dupree"]     [0.92003, 0.16667, 0.65217, -0.6455, 0.17725]
["Superman Returns", "The Night Listener"]      [0.96123, 0.16667, 0.78, -0.17985, 0.41008]
["Superman Returns", "You, Me and Dupree"]      [0.96808, 0.08333, 0.61224, 0.65795, 0.82898]
["The Night Listener", "You, Me and Dupree"]    [0.95266, 0.0, 0.74286, -0.25, 0.375]
removing tmp directory /tmp/movieSimilarities_with_MapReduce.root.20151203.013408.703399
```

Figure 1 Randomly Picked Sample Output

From this sample output, we can say that movies "Just My Luck" and "The Night Listener" are similar to

each other and hence for a user who had seen Just My Luck, we can recommend him with The Night Listener (although the actual output can be different).

RELATED WORK

There are lot of ways in the market which demonstrates movie recommendations. The design complexity in implementing these type works is mainly due to algorithm constraints and time constraints.

In terms of algorithmic constraints, since the type of the data changes over time and user-interests changes over time, we need to develop algorithm constantly to be up-to-date. Since the Map-Reduce Algorithm designed can handle any type of data and is well suitable for streaming jobs. Integrating Map-Reduce with Python helps in utilizing the Python analytical tools like numPy and sciPy which makes the algorithm attain the better efficiency.

In terms of time complexity, since we need to compute the pairwise similarity between all items, and also there are many pairs of such items, the time taken would be large. Since Map-Reduce emits the set of item pairs, when it partitions the data, we can apply the similarity measures on this shuffled data. So, the time taken to shuffle the data compared to other methods is decreased.

CONCLUSIONS AND FUTURE WORK

From the analysis, we arrive at a conclusion that, given a large data set of points, we can apply various similarity measures on the data using Map-Reduce algorithm. We tend to cut down the time taken to calculate the similarity measures to 30%. Still it is not an acceptable value, there are lot of modifications needed to be done to increase performance. “mrjob” still fails in some aspects of making jobs run in Hadoop Cluster. For instance, setting various configuration parameters, moving the archives to a Distributed Cache, specifying the number of tasks framework can handle needs to be reviewed.

One such method would be creating a cluster with at least 2 × 6 core 2.9 GHz/15 MB cache CPU and of 64MB Memory. This would make the cluster proper to handle data. In this case, we can pull out the entire data (even from the past twenty years) and replicate it along the nodes in the cluster. A replication factor of 3 helps in data consistency and error removal. The data which we collected may not demonstrate the importance but movement to Hadoop Cluster would definitely improve the performance.

Another tool which can deploy the importance of Recommendations would be usage of “Mahout” engine which exemplifies the power of Map Reduce. It contains the in-built mechanism to handle the similarities. Many measures like compressing the data while writing to Hadoop Cluster, Indexing the data stored in the Hadoop Cluster, running Speculative Jobs on the framework can helps in effective utilization of resources.