Recommendation System using Amazon Reviews

Term Project for CS435 - Introduction to Big Data

Spring 2017

by

Nikhila Chireddy

Avinash Konduru

1.Problem Statement

All users search for static information for potentially buying products, we can build a system to increase the interaction among users to provide rich user experience dynamically using Recommendation Systems. These systems derive recommendations for each individual user based on their past purchases and their reviews.

The goal of this recommendation system is to suggest new items to a user based on the similarity between products.

The main advantages of building the recommendation system are:

- Shoppers become more engaged in the site when personalized product recommendations are made. They are able to delve more deeply into the product line without having to perform search after search.

- Converting shoppers into customers takes a special touch. Personalized interactions from a **recommendation engine** show your customer that he is valued as an individual. In turn, this engenders his loyalty.

- Average order values typically go up when a *recommendation engine* in uses to display personalized options.

- The volume of data required to create a personal shopping experience for each customer is usually far too large to be managed manually. Using an engine automates this process, easing the workload of your IT staff and your budget.

This recommendation system is already being using by Amazon. Our goal is to re-design this system and to get familiar with various approaches.

2. Dataset

The amazon reviews data is downloaded from: http://jmcauley.ucsd.edu/data/amazon/

The dataset consists of ***JSON*** files for each product category like Books, Electronics, Movies and TV, etc. The files have size about 100 MB to 2GB, totaling to ~4.5 GB and has about 1 Million reviews.

2.1.Attributes Description

- *reviewerID* - unique id given to a reviewer
- *reviewerName* - Name of the reviewer
- *asin* - Product number
- *helpful* - specifies if the product was helpful or no
- *reviewText* - review written by the customer
- *overall* -rating given by the customer
- *summary* - summary of the review written by the customer
- *unixReviewTime* - timestamp of the review
- *reviewTime* - date on which review was written

A sample tuple from the dataset.


```
{
        "reviewerID":"AIXZKN4ACSKI",
         "asin":"1881509818",
        "reviewerName":"David Briner",
        "helpful":[0,0],
        "reviewText":"This came in on time and I am veru happy with it, I haved used it
                        already and it makes taking out the pins in my glock 32 very easy",
        "overall":5.0,
        "summary":"Woks very good",
        "unixReviewTime":1390694400,
        "reviewTime":"01 26, 2014"
}
```


3.Strategy

There are different algorithms for implementing the Recommendation Systems. Memory-based Algorithms, Model-base Algorithms. **Memory-based** [2] recommendation systems are not always as fast and scalable as we would like them to be, especially in the context of actual systems that generate real-time recommendations on the basis of very large datasets. To achieve these goals, model-based recommendation systems are used.

Model-based recommendation systems [2] involve building a model based on the dataset of ratings. In other words, we extract some information from the dataset, and use that as a "model" to make recommendations without having to use the complete dataset every time. This approach potentially offers the benefits of both speed and scalability.

Item-based collaborative filtering[1] is a model-based algorithm for making recommendations. In the algorithm, the similarities between different items in the dataset are calculated by using

one of a number of similarity measures, and then these similarity values are used to predict most similar items for user-item pairs not present in the dataset.

The similarity values between items are measured by observing **all the users who have rated both the items**. As shown in the figure 1, the similarity between two items is dependent upon the ratings given to the items by users who have rated both of them:[1]

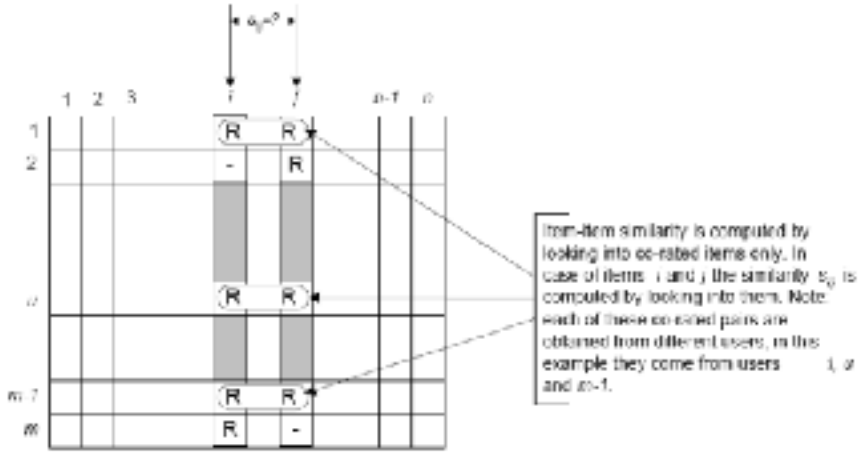


figure 1: This is picture is taken from http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/itembased.html

Our Approach involves 2 stages and each stage uses the Hadoop MapReduce to perform the tasks.

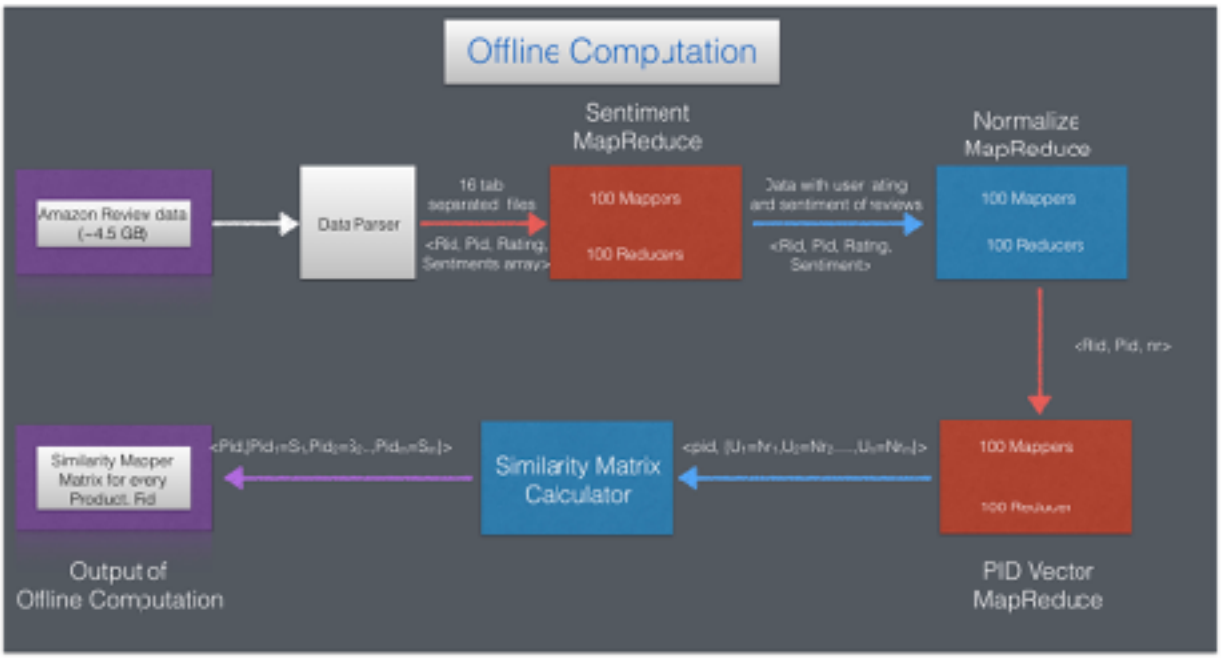
- Offline Computation - In this stage we try to build a similarity matrix as mentioned above for all the products we have in the corpus, based on their ratings and reviews.

- Online Computation - In this stage we take some users and their reviews and use the Similarity Matrix computed in the Offline Computation to recommend similar products to the users.

3.1.Offline Computation

Figure 2 represents the sequence of Hadoop MapReduce operations that we have ran on the data corpus to find the Similarity Matrix. The first step in Offline computation is to convert the JSON files into tab separated files which consists of the required fields only. For this purpose we use a JSON parser. The output fields are Reviewer Id, Product Id, Rating and Review Text for each product.

The first step in Offline computation is to convert the JSON files into tab separated files which consists of the required fields only. For this purpose we use a JSON parser. The output fields are Reviewer Id, Product Id, Rating and Review Text for each product.



Then we use the Stanford NLP [4] to identify the sentiment of review Text. The Review Text may contain multiple lines so to increase the computation speed, we have divided the review text into smaller sentences and then calculated the sentiments for each of these smaller sentences. This gives output in the form comma-separated values. Now in order to calculate the actual sentiment of the review text, we have used an approach which checks the positivity levels and negativity levels of the text and based on these levels, we decide if the sentence is positive or negative.

To calculate the positivity or negativity, we take the individual count of positive sentences, negative sentences and neutral sentences. Then we add the count of neutral sentences to positive and negative sentence count and then finally compare them. If positive count is greater than negative count, then the text is considered as positive(vice-versa). If both the counts are equal, then we consider it as neutral.

The next step is to compare the sentiment of each review with the corresponding rating. To do this, we calculate the average ratings given by the user and then subtract this average from each rating. If the result is greater than zero and the sentiment of review is positive, then we take the rating. Similarly if the result is less than zero and the the sentiment is negative or the result is equal to zero and the sentiment is neutral we consider it. But if these conditions are not met, we consider that the review given by the user and the rating do not match and hence we ignore that rating.

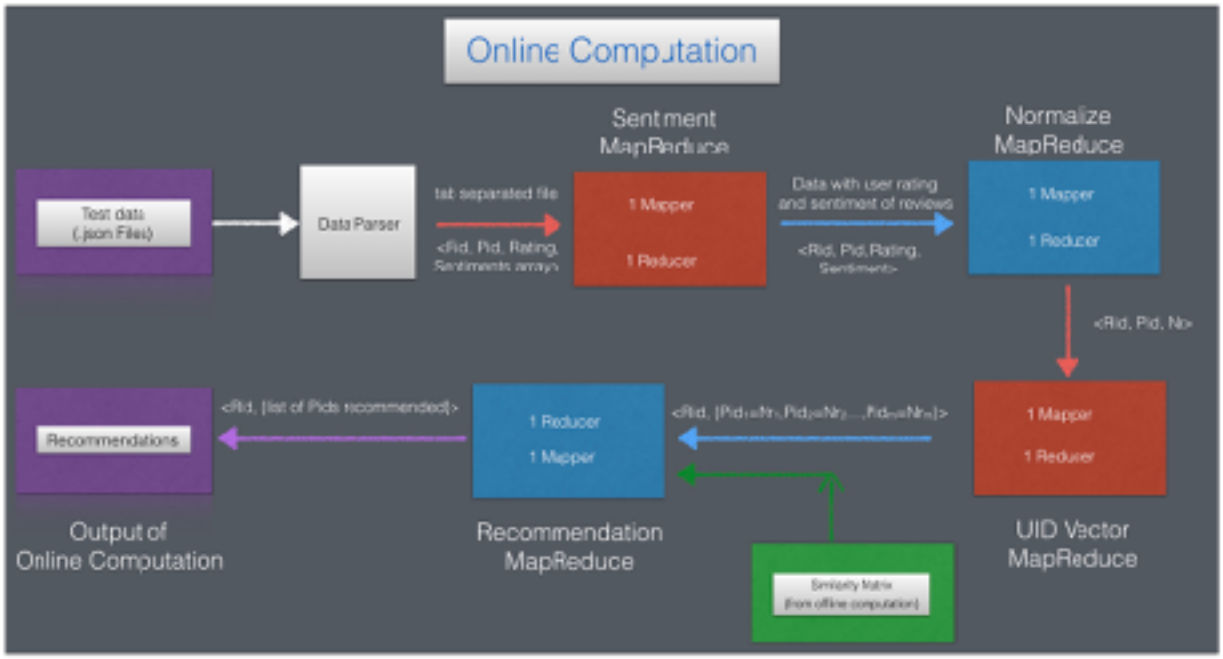
Now since we have got all the correct reviews given by the users for each product, we try to identify the similarity among all the items. To do this, we have used the Pearson's Correlation.

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_i)^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R}_j)^2}}$$

Now using the above formula, we calculate the similarity among all the products and store them in a file.

3.2.Online Computation

In the Offline computation we have calculated the Similarity matrix of all the products in the corpus. Prior to the execution of Offline Computation we have separated a test data set of 6 MB which contains ~5000 reviews from the Corpus. This data set contains of a mixture of user-product pairs which are already used in corpus and also new ones. As shown in figure 3, we repeat the same steps that we have done in Offline Computations to parse the data, identify the Sentiments, normalize ratings and finally create UID Vectors.



The UID Vector is a pair of userID and ProductID with the normalized ratings. The output of the UID Vector map-reduce looks like $<U_i, \{P_1:NR_1,P_2:NR_2, \ldots\ldots P_n:NR_n\}>$

Here Ui represents the UserID of user-i , P1,P2,….Pn are the products and NR1, NR2,….NRn are the normalized ratings given by the user to each of the products
Now to recommend items to these users, we take this file and the the Similarity Matrix obtained from offline computation as input and then based on the ratings given by the user to each product, we identify similar products and predict the rating the user might give. Based on

these predicted ratings, we order all the products from high to low and then select the top-10 items.

In our system, we also make sure that the products purchased by the user are not recommended again. To do this, we maintain a list of items that have been rated by the user and then before recommending any product, we verify if the item is present in the list or not.

4.Experimental Results

4.1.Offline Computation

Once we built the system as mentioned in Section 3, we ran our tests on files with multiple file sizes, 100 MB, 1Gb and 4.5 Gb of the data corpus. The below table shows the times taken for each experiment.

Data Set Size	Time Taken (in Minutes)	Similarity Matrix Size
100 MB	5	2.5 MB
1 GB	33	649 MB
4.5 GB	145	5.5 GB

One observation we made is, the time taken for the offline computation is not linear. The reason could be because the size of data is set is directly proportional to number of products that we have in the corpus. As the number of products (N) increase, the size of the Similarity Matrix also increases as its size would be $N \times N$ dimensional.

There are some interesting results on Sentiment Analysis from the "Data Parser". Most of the reviews with rating 4, where "StandfordNLP" identified the review Text to be Negative. And for most of the users who has this kind of situation have $averagerating > 4$. This might conclude following things

- These users are very liberal at giving the ratings, even though they are not happy with the product.
- Or they purchased the product looking for one feature, which might not working properly, but the overall product could be good. Since they don't have features they are looking for, the review might be written as negative.

4.2.Online Computation

We have run experiments on the system by collecting data from different input files and then running the online computation using this file. This data contains the users who has given

ratings to different products, which exists in the Similarity Matrix. So when ever we find a product in the test data which matches with the one in the Similarity Matrix, we pick the TOP 10 recommendations for this test user. (Top 10 is what we considered in our testing because it those are the ones which will have higher similarity, how ever our system can be extended by taking an input on number of recommendations that user is looking for.)

4.3. Challenges

One of the biggest challenges we have faced in applying Sentiment Analysis for the dataset.

We were able to get some third party api's which returns the sentiment analysis of a text using a REST service call. This api call is fast and effective, as it does not execute on our own cluster and which would not impact our MapReduce program. But the problem with these is, they are not free versions, and they have limitations on number of requests, 30 Requests per minute and 500 requests per day, which would not suit our dataset as we have reviews in Millions.

After this, with the help of Prof Sangmi, we get to know that we have one open NLP package called StanfordNLP. But, Stanford NLP has limitations, as length of the sentence increases, there will be high consumption of heap space in the JVM. Due to which, the MapReduce program that we have written started continuously failing with "Java Heap Space Error". To avoid this problem, we tried passing the data to MapReduce, by splitting the reviewText into multiple lines and then try to get the sentiment of the sentence. This time MapReduce worked for small input files of 100 MB, but when we run it for the whole data set, it ended up having issue with Heap space. So Instead of running the MapReduce program, we have created one job to pick file from the input folder, parse it and generate the out put file with tab delimited as well as with Sentiments. Instead of running this on one machined, we have distributed the job over 20 machines, which took almost 12 hours to parse whole data. And rest of the offline computation is executed using Hadoop MapReduce.

5. Contributions

Before we start the actual implementation, Design and Research was done to together. The below table will give the detailed break down on tasks.

Nikhila	Avinash
Design and Research on required methodology to implement the recommendation system.	Design and Research on required methodology to implement the recommendation system.
Online Computation (except Data Parser with Sentiment Analysis)	Data Parser with Sentiment Analysis and running the job over different machines
	Online Computation
Testing on 4.5 GB dataset	Testing on 100 MB and 1 GB dataset
Report for Offline Computation (along with diagram)	Report for Online Computations (along with diagram)

6.References

1. http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/itembased.html
2. http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/modelbased.html
3. https://en.wikipedia.org/wiki/Collaborative_filtering
4. http://stanfordnlp.github.io/CoreNLP