[COEN 242]

Content-based Book Recommendation System

Team15
Naga Padmavathy Kancharla
Youra Cho
Rahul Kilambi

Abstract

As Internet generates rapidly huge amounts of texts it is very important to process the data in a short time interval. This entails the necessity of fast, scalable methods for text processing. In this paper a method for pairwise document similarity on massive data-sets like a corpus, using the Cosine Similarity Function and the tf-idf (Term Frequency-Inverse Document Frequency) normalization method is used to suggest a list of similar books matching the user's query. This approach is mainly focused on the Map Reduce paradigm, a model for processing large data-sets in parallel manner, with a distributed algorithm on computer clusters. Through Map Reduce model application on each step of the used method, text processing speed and scalability is enhanced in reference to other traditional methods. The CSMR (Cosine Similarity with Map Reduce) method's [1] implementation is used to calculate similarity between two documents. These methods have been discussed in detail and implemented in the later sections of the observation report.

Keywords: Map Reduce, Hadoop, TF-IDF, Text Mining, Cosine Similarity, Big Data

1. Introduction

Content-based systems examine properties of the items recommended. It measures similarity by looking for common features of the items. One example of such a content-based system is job recommendation in LinkedIn. We aim to build a recommendation system that suggests similar books matching a user's query. We implemented this recommendation system on Hadoop Map Reduce framework using Java programming language. The Map Reduce program calculates TF-IDF (Term Frequency ×Inverse Document Frequency) score for each term first. The TF-IDF is the formal measure of how important a word is to a document within a corpus. And it calculates the similarity between each pair of books by using similarity measure. Based on similarity matching results, the system returns K most similar books within a reasonable response time.

TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following formula. This can be done with the help of logarithms.

$$IDF(t) = \log_{10}(\text{Total number of documents} / \text{Number of documents with term t in it})$$

Cosine Similarity

This metric is frequently used when trying to determine similarity between two documents. Since there are more words that are incommon between two documents, it is useless to use the other methods of calculating similarities (namely the Euclidean Distance and the Pearson Correlation Coefficient discussed earlier). As a result, the likelihood that two documents do not share the majority is very high (as with the Tanimoto Coefficient) and does not create a satisfactory metric for determining similarities.

In this similarity metric, the attributes (or words, in the case of the documents) is used as a vector to find the normalized dot product of the two documents. By determining the cosine similarity, the user is effectively trying to find cosine of the angle between the two objects. For cosine similarities resulting in a value of 0, the documents do not share any attributes (or words) because the angle between the objects is 90 degrees.

2. Design

We defined 6 Map Reduce algorithms to execute the job.

1. Word Count in a document

Calculate the number of occurrences of the word in document

Mapper: ((*docID, term)*, 1) Reducer: (*(docID, term), n)*

The first job is word count. This phase counts the number of occurrences of the word in a document. The input is document ID and the contents are values. The Mapper assigns 1 to each appeared term in a document. Reducer produces the number of occurrences of the term, small n (raw frequency) in each document.

2. Total Number of Terms for TF normalization

Calculate the total number of words of each document

Mapper: (*docID*, (*term*, n)) Reducer: (*(docID, N), (term, n))*

Next, we calculate the total number of words in each document, N. We need to know N for normalization of term frequency in the next step.

3. TF-IDF Score

Mapper: (term, (docID, n, N)) Reducer: ((term, docID), n/N*log (D/df))

In the third Map Reduce job, we calculate TF-IDF score for each term. So, the key here is term and Reducer returns the term as key, all the documents containing the term and its TF-IDF as value.

4. Cosine Similarity

Mapper: ((docID(i), docID(j)), (docID(i).TF-IDF, docID(j).TF-IDF))

Reducer: ((docID(i), docID(j), cosin_sim)

In this job, we calculate cosine similarity for every possible pairs of books to determine similarity between books.

5. Query Finder

Find all documents contain the query and returns document most close to the query

Mapper: (docID, TF-IDF) Reducer: (docID, Sum (TF-IDF))

The fifth job requires user's input as a query. If a user enter a query then the system finds all documents containing the query and returns a document that is close to the query.
Query varies in its length. It may consist of one or more than two terms.

6. Recommendation

Return list of top 30 similar document

Map :(docID(i),docID(j)) Reduce: (DocID,list<Similar documents>)

The last job is Recommendation. It extract documents with high TF-IDF values related to the query documents and returns list of 30 similar books.

3. Experiment

3.1. Infrastructure

We do the Programming on Map Reduce paradigm. Java programming language is used for implementation. We also experimented on Hadoop 2.7.0 (single node) for comparison of execution time.

3.2. Dataset

The dataset for this experiment contains about 2587 books provided by Gutenberg EBook and their contents are scanned. The size of dataset is ~1GByte. The link we got the dataset is as follows.
http://www.cs.colostate.edu/~cs480/DataSet.zip

4. Implementation

1. Calculate the TF-IDF (TF_{ij} x IDF_i) values for each term in the corpus. We used the normalized TF by dividing it by the total number of words of each document N.

$$TF_{ij} = tf_{ij} / N * \quad IDF_i = log\ (D/df_i)$$

2. Calculate a similarity for every pair of books using cosine similarity.

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

3. Find all documents contain the user query and document and return the document most related regarding the query.

4. Find all the documents related to the query documents from similarity matrix and return top K most similar books

The fig 1. is a flowchart of the book recommendation system. It clearly summarizes the implementation steps.

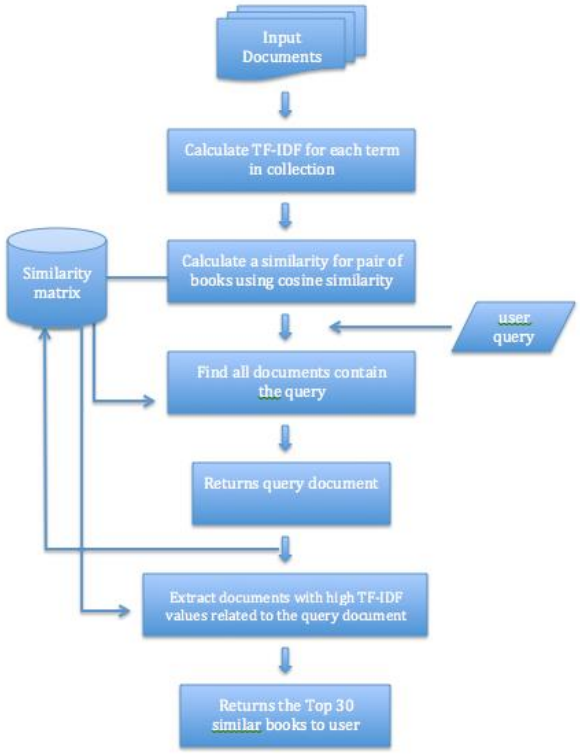

Figure 1. Flow chart for book recommendation system

The offline computations are all computed and their output partitions are stored in paths. Each path contains the Term Count, Total Word count, Calculated TfIdf and Similarity pairs. The Online computations have two map reduce jobs which perform the task of responding to user query and returning similar books. The Query Job makes use of the Calculated TfIdf partitions as its input and gives all the documents that could be related

to the query and their respective TfIdf value. For now our implementation needs the user to the go through this list of results, select document with highest TfIdf value and run the next Map Reduce job giving this query result as the input argument. The next Map Reduce job returns the results to another path which are a list of all the documents that could be similar to the query.

5.Results

Our Infrastructure used 6 map reduce jobs to retrieve a response to a user query and return some k more similar books as recommendations. We scaled our responses by keeping the reducers constant to a level of about 1100. As the size is varied from 250,500,750 to 1000 MB. The response time varied much from 250 to 500 but stayed quite constant at 500 to 750 from figure 2. In this figure the doc return refers to time taken by the MR to select k documents and return them as the size increases. These responses to returning query and the documents are part of the online or interactive queries done by the user.
Figure 3 refers to the time taken for the offline job to be completed as the size is increased keeping the reducers constant. Interestingly in the figure, the time taken for building the offline jobs was smaller for the 1GB cluster than the 750 MB size. We assume that this is due to the presence of excessive reducers than needed resulting in taking more time waiting for completion. Figure 3 relates to the time taken to run the job in a single node. While we could run a small set of data in small time, it took about 55 minutes to run the single offline computations. Hence we had to scale the data to MB from 10 limiting to 250.
(Figure2 and 3 are to the scale in seconds while figure 3 is in minutes.)

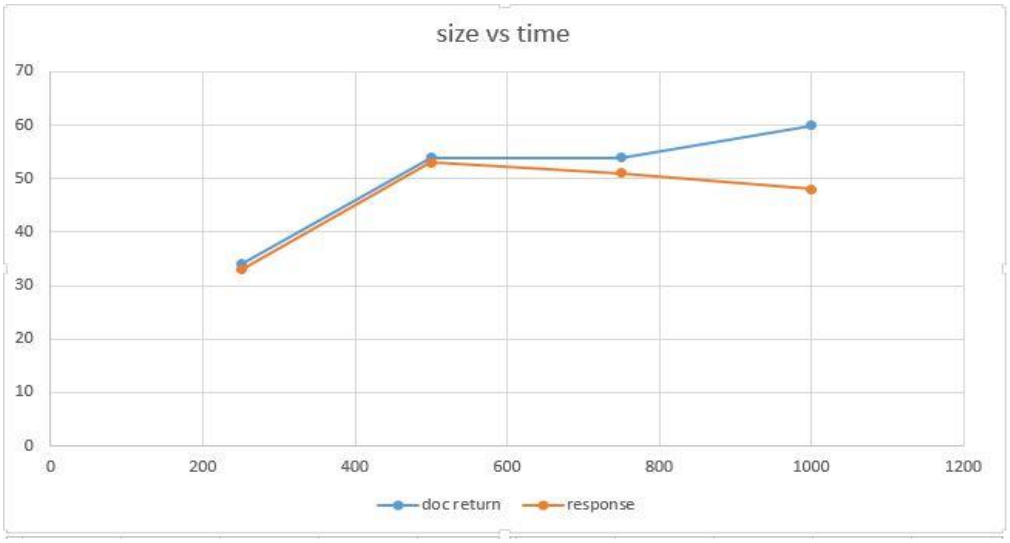

Figure 2(Latency of response as size is varied)

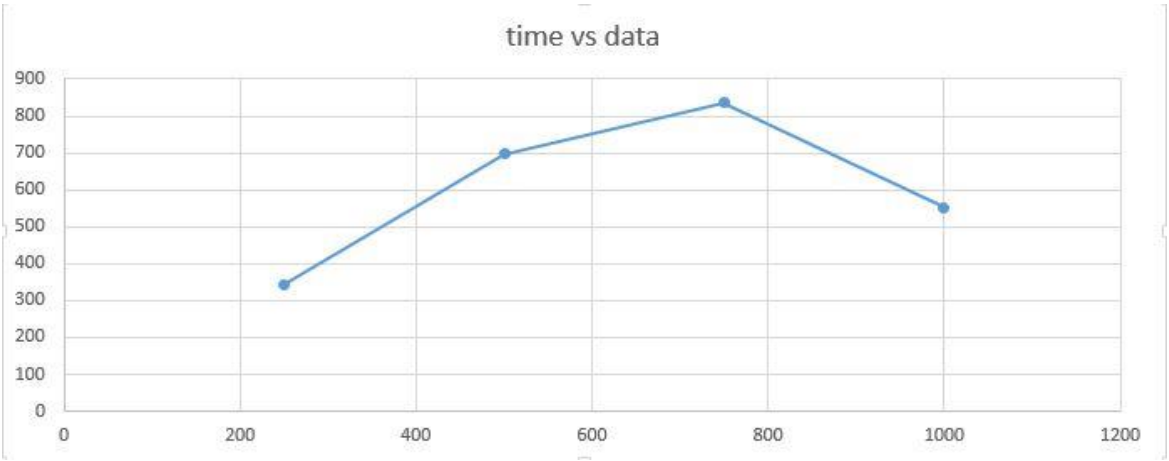

Figure 3 (offline Job latency as size varies)

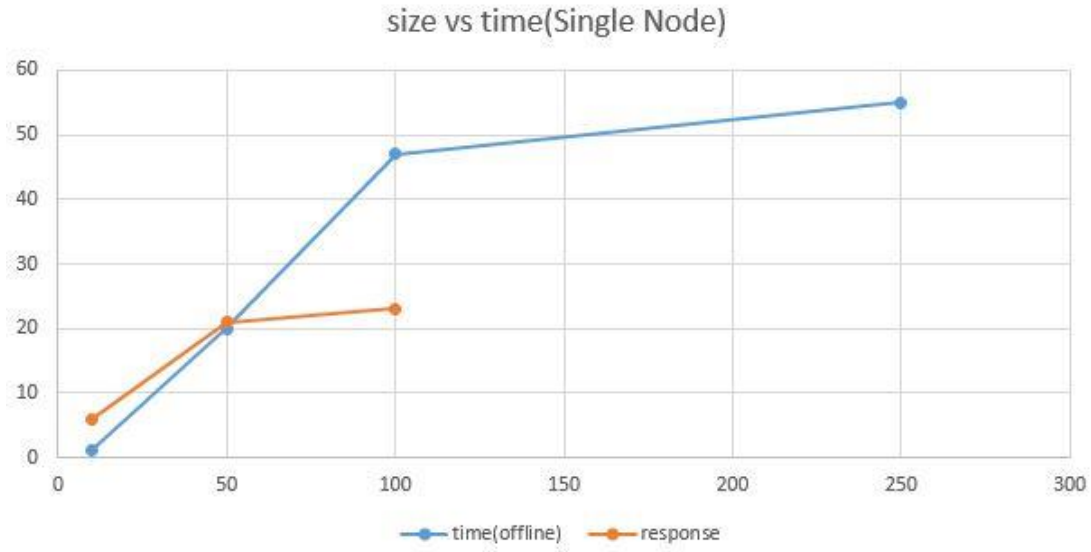

Figure 4(Latency in minutes as size varies)

6.**Further Implementations**

While more number of unique queries can result in a better filtering to the correct document, we are currently receiving a lot number of other documents with little matching too. So we need to correct this by implementing further weightage factors. If we can distinguish between a verb and a noun, we could provide high priority to noun, as they are searched more. Also in our present context, we haven't extracted the name of the author or the book, they are just a part of the book and hence are included just as content. But if we can categorize them as per their group, filtering could be done better and recommendation could be made better by returning books that the author has written.

7. References

1. http://link.springer.com/chapter/10.1007%2F978-3-662-44722-2_23#page-1
2. http://infolab.stanford.edu/~ullman/mmds/ch9.pdf
3. www.atlantis-press.com/php/download_paper.php?id=2562

4. https://janav.wordpress.com/2013/10/27/tf-idf-and-cosine-similarity
5. http://www.academypublisher.com/proc/iscsct09/papers/iscsct09p278.pdf
6. http://www.garph.co.uk/IJARIE/July2014/1.pdf
7. http://www.ipcsit.com/vol47/009-ICCTS2012-T049.pdf
8. https://code.google.com/p/hadoop-clusternet/wiki/RunningMapReduceExampleTFIDF
9. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5694030&url=http%3A%2F%2Fieeexplore.ieee.org%2Fstamp%2Fstamp.jsp%3Farnumber%3D5694030