HTPI: Hadoop Text Processing Interface

Disha Kangar[1] Dr. Kanwal Garg[2]
Scholar, M.Tech(Computer Science and Engineering)[1] Assistant Professor in Computers[2]
[1,2] Department of Computer Science and Applications,Kurukshetra University,Kurukshetra,India.

Abstract---Text mining is a practice which is regarded as the supporting pillars of Information Retreival. This paper is in simple terms dedicated to text mining and bear a prime focus on mining academic papers. An architecture is proposed by the authors is presented in the paper, which they have named HTPI. This framework is built upon Java eclipse using Apache Hadoop.

The problem under consideration for the paper is the reference metamorphosis of the references mentioned in the references section of any scientific paper based upon the similarity score(between the referenced paper and the paper whose reference list is being re-ordered) retrieved. Various notions have been used in the paper like stemming, skipping and similarity calculation using Jaccard Coefficient.

Keywords:- Document Similarity, Hadoop, Information Retreival, Jaccard Coefficient, Map-Reduce, Skipping, Stemming.

I. INTRODUCTION

In the 21st century, the truth of I.T. world is exponential growth of data that has became the biggest challenge in scientific applications [1]. Useful information can ne gathered from this huge data, but an utilitarian technique is required. To meet this expectation, Google engineers, Jeffrey Dean and Sanjay Ghemawat [2] have proposed MapReduce paradigm which has proven its worth well in industry since a decade and has stepped in academia recently.

Also, the research practice has matured in the near past. The course of every researcher includes studying of papers of one's interest and surfing through the reference list of screened papers. While going through this list, no idea can be gained by him, that which paper will prove more beneficial for him thus leading to wastage of time and effort by may be studying low-relevance papers. The reference list has to be written according to some conventions and their ordering does not give any clue to the researcher as to which next paper one must proceed to.

This very problem has been addressed by the author in HTPI. Author has chosen Map-Reduce as programming paradigm because it can be perceived that when scalability of application will increase, then the data would be of large-scale nature and the task at hand becomes computation intensive task[3]. For these types of problems, Hadoop has become the trend-setter for analyzing large-scale data because of two major achievements in its architecture i.e. reliability and data motion for applications.

Reliability is maintained in Hadoop by using Hadoop Distributed File System(HDFS) as this file structure stores the data across the nodes. As a result, an ultimately high aggregate bandwidth over the entire cluster can be realized[1]. In Hadoop, because the Map-Reduce model is realized, thus application's workload is divided into smaller loads and assigned to multiple nodes present in the cluster. This type of architecture fits well in designing applications like those of Information Retrieval.

The idea behind this research work is to find out the most relevant research paper from the reference list of a given research paper. The suitability of research purpose is advocated by the "Probability Ranking Principle" according to which; –' if" provided with a reference retrieval system and the task is to arrange the references in decreasing order of usefulness to the user who is interested in the base paper, then the overall effectiveness of the system to its users will be the best that is obtainable on the basis of that data[4]. The flow of work is divided into six sections. Section 2 describes the architecture of HTPI. Section 3 explores and analyzes the pre-processing steps. Section 4 is all about similarity metrics present in the literature. In section 5, algorithm proposed by the authors to calculate Jaccard's coefficient between two documents is given. The paper concludes with a conclusary note presented in section 6.

II. HTPI ARCHITECTURE

Figure 1 explains the overall flow of activities that take place internally. The architecture can be partitioned into three layers i.e. input, applications and output layers. At the input layer, base paper is submitted as a query to the system and then through an extractor, all the papers whose links are cited in the query paper are fetched from the database.

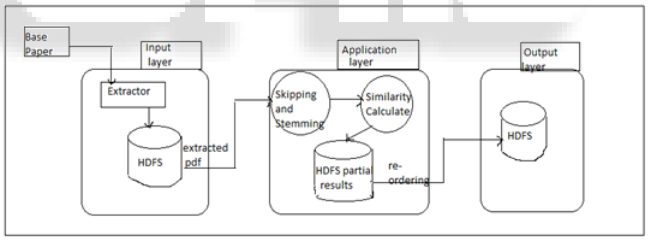


Fig. 1: HTPI architecture

This task is traced-out and an index of query paper and reference papers are stored in the HDFS (1-n) fashion. In this way, an instance of database is created for each and every query submitted at the interface. The HDFS gives a route to the extracted pdfs to the Application layer, where pre-processing i.e. stemming and skipping is performed on the data of individual documents. This processed data then moves out for similarity calculation with the query paper, which is here done using Jaccard's Coefficient.

The papers along with their similarity scores are stored in HDFS. These partial results are then re-directed to output layer. In the output layer, by using a ranking function, the reference list is re-ordered and papers are arranged in the decreasing order of relevance with the base paper. The next sections goes through the details of algorithms used in the project.

III. PRE-PROCESSING

Term frequencies play a vital role in the functions employed for calculation of similarity score between two documents as

the tenet of similarity metrics is to find the number of common words in two documents. So, if one reduces the term frequencies, somehow, more relevant information can be retrieved. This fact is supported by Zipf's law[5] which states that the rank of a word is reciprocally proportional to its frequency. In the light of above facts, one can prove the importance of stemming and skipping.

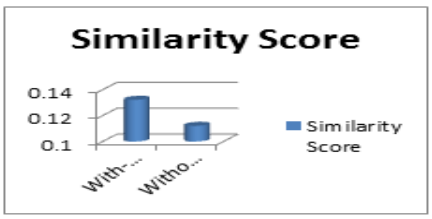
Skipping can be defined as removal of words that don't make any contribution to the similarity score between two documents because functional words are generally the most frequent words in English text, but neither are descriptive nor hold any importance for the document's subject. Example of such words are – 'a' ,'an', 'the', 'has', 'have', 'been', 'being', 'while', 'being' and so on. Skipping can be employed by using a filter program and the user can actively mention the stop-words to be used while filtering.

The importance of skipping can be demonstrated by two statements. First, if skipping is not employed, there is a great probability of false-match sore generation between two documents because of large contributions by functional words. Second, from a resource utilization perspective, keeping stop-words make index much larger and hence query evaluation becomes slow.

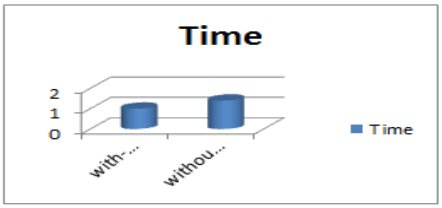
Stemming. It is the second solution for pre-processing and can be defined as the convergence of words to a common stem. Terms with a common stem have similar meaning. Example of stemming is – 'connected', 'connections', 'connecting', 'connection-oriented', can be mapped to the stem word 'connect'[6].

Two approaches have been mentioned in the literature for performing stemming. One is maintain a look-up table i.e. an index of words is generally maintained and all the derived words are mapped to the stem or root word. Usually, applications of B-tree or hash-table is preffered for this. Alternative is the usage of suffix-stripping algorithms in which a list of rules is maintained and when provided with a word, the word is processed based on the rules maintained in the database and mapped to the stem word.

When an experiment was conducted by the author, the efficiency gained in similarity score by applying the two processes together on a set of five documents and then taking an average of five runs, there was an increase in the similarity score and also time was decreased in calculating the similarity score. The results are explained in Figure2.



(a)



(b)

Fig. 2: Results of applying pre-processing.

In Figure 2.a, similarity score increased and in Figure 2.b, time taken to calculate similarity score decreases.

IV. Document Similarity

This paper has been written under the theme of document similarity, so an extensive research has been done on the similarity measures used till date and three very basic measures have been made a part of this paper i.e. Euclidean distance, Cosine similarity and Jaccard coefficient.

A text document can be modelled in many ways, "bag-of-words" being the most famous representation[6] in Information Retreival and text mining. A word count is maintained in the bag and each word corresponds to a dimension in the resulting data space and each document then becomes a vector consisting of non-negative values on each dimension[7].

Thus, words appearing in the document with a high frequency, contributes a high magnitude. This magnitude can be raised if stemming is employed as n-variants of a base-word add-up to the magnitude of a single word. Also, with the usage of skipping and stemming, dimensionality of vector space model can be reduced to a good extent. After pre-processing, similarity score is calculated for which any of the following metrics can be taken into consideration.

Three of the well known similarity metrics are:

A. Euclidean Distance: It is the very basic and simplest metric, generally used for geometrical problems. It can be defined in simple terms as a distance between two points and can be measured by even using a ruler in 2-d or 3-d space. When talking about text documents, Euclidean distance can be defined as follows.

Let A and B represents two documents and are represented by their term vectors i.e. t1 and t2 respectively. In that case Euclidean distance between A and B can be defined as

Euclidean distance $=(\sum |wt1\text{-}wt2|^2)^{½}$

where wt1=term freq* index document frequency(A,t)

B. Cosine Similarity: In this metric, each document is treated as a vector in the space of documents and the cosine between these vectors gives a measure of similarity. Given two documents, d1 and d2(represented in terms of vectors), their cosine similarity is

Sim(d1,d2) = (dot product of vectors) ÷ (product of magnitude of vectors)

Cosine Similarity is a monotonically decreasing function for the interval [0,180] which is shown in Figure3 and hence serves well in the ranking applications. For two documents to be similar enough, angle Θ should be as small as possible.

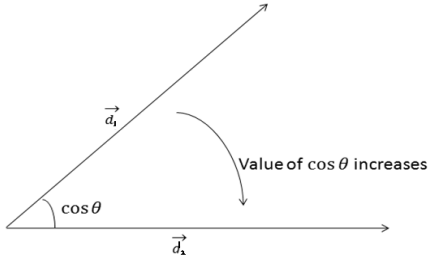


Fig. 3: vector space model of two documents that have an angle Θ between them.

The reason why cosine similarity is not very much preferred for text mining applications is that, vectors hold place for every term in the collection, so vectors are generally sparse (which is the major limitation of vector space model).

C. Jaccard Coefficient: Similarity between two documents A and B is measured by calculating |A∩B| ÷|AUB|. The result is between 0 and 1 while 0 means completely unmatched and 1 means identical documents (A and B are two documents here). In this function, intersection and union are given value as multiset-operations because multiple occurrences are taken into account. There is no implication that A and B must be of same size.

The fitness of Jaccard coefficient in case of document similarity can be proven by the procedure how it is calculated. Let there be two objects p and q, having only binary attributes (word present or absent). Four pairs of vales can be generated:

M(0,1) = No. of attributes where p is 0 and q is 1

M(1,0) = No. of attributes where p is 1 and q is 0

M(0,0) = No. of attributes where p is 0 and q is 0

M(1,1) = No. of attributes where p is 1 and q is 1

Jaccard Coefficient= (No. of M(1,1) matches)÷(No. of not-both-zero attribute values)

So, in case of document similarity, most of the documents are likely not to contain many of the keywords and thus 0-0 matches should be ignored by the distance measure.

By keeping this fact in consideration, author has designed algorithm in Map-Reduce style for Jaccard Coefficient calculation as described in the next section.

V. Proposed Algorithm

To compute the Jaccard Similarity Coefficient for a pair of documents, two quantities to be computed are minimum out of the count of words that occur in both documents, which is represented by the variable min and the second is the sum of the number of words that occur in both the documents, which is represented by the variable sum, along with their frequencies. The following algorithm has been proposed by the authors.

Pass One
{

Map phase

This map method simply takes document id and word as input and returns word and wordocc.

Input :- id,word

Output:- word,wordocc

Reduce Phase

This method takes input as a pair (word,wordocc) from different mappers and return id(word,wordocc).

Input:- word,wordocc

Output:- id(word,wordocc)
}

Pass Two
{

Map Phase

This map method just acts as identity and passes the data to the reducer.

Input:- page1(word,wordocc), page2(word,wordocc)

Output:- page1(word,wordocc), page2(word,wordocc)

Reduce Phase

After processing both documents, min and sum variables are counted in this phase

Input:- page1(word,wordocc), page2(word,wordocc)

Output:- word(min,sum)
}

Pass Two
{

Map Phase

It is a trivial step.

Input :- word(min,sum)

Output:- word(min,sum)

Reduce Phase

Input :- word(min,sum)

Output:- word (sim score)
}

VI. Conclusion And Future Scope

Authors have put an effort to deal with the problem of finding relevant documents, if provided a query document. This research work has only taken into account the structure of academic papers and ignored other references which may be books, wikipedia pages, blogs etc. The upcoming researchers are motivated to extend a hand to the problem and give a wise contribution to the academia.

References

[1] Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers,

[2] A.H.: Big data: The next frontier for innovation, competition, and productivity.

[3] Technical report, Mc Kinsey (May 2011)

[4] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In OSDI, 137-148{150, 2004}.

[5] Jimmy Lin. Brute Force and Indexed Approaches to Pairwise Document Similarity Comparisons with MapReduce

[6] Stephen E. Robertson, J. Documentation 1977 in Computing Relevance, Similarity: The Vector Space Model Chapter 27, Part B Based on Larson and Hearst's slides at UC-Berkeley

[7] Zipf's Law in Computer Science Tripos Part II Simone Teufel Natural Language and Information Processing (NLIP) Group

[8] Porter, M.F.: Readings in information retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997) 313–316

[9] R. B. Yates and B. R. Neto. Modern Information Retrieval. ADDISON-WESLEY, New York, 1999.

[10] Anna Huang "Similarity Measures for Text Document Clustering" Department of Computer Science The University of Waikato, Hamilton, New Zealand.

