INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Course Project

Introduction to Parallel and Scientific Computing

Spring 2019

Text Mining using Parallel Computing

Author:
Tarun Vatwani
Roll no. 2018201075
Rajat Yadav
Roll no. 2018201077
Vivek Patare
Roll no.2018201078

Supervisor:
Prof. Pawan Kumar

May 3, 2019

Abstract

Text Mining describes extracting useful information from large unstructured data from collection of texts. Text Mining has become very prevalent for applications such as business analytics, exploratory data research, medical applications, sentiment analysis etc. It employs various machine learning techniques and methods for getting relevant results. By the nature of dataset size and text mining techniques used, they are generally slow in nature. We propose solution using parallel computing applications to speed up process and analyze how it fares to normal methods to see if we get better results.

Contents

List of Figures

Chapter 1

Introduction

Text mining in context of this project means method of extracting useful information from large and often unstructured collections of texts. Another, closely related term is *information retrieval.* A typical application is searching databases of abstracts of scientific papers. For instance, in a medical application one may want to find all the abstracts in the database that deal with a particular syndrome. So one puts together a search phrase, a *query*, with keywords that are relevant to the syndrome. Then the retrieval system is used to match the query to the documents in the database and presents to the user the documents that are relevant, ranked according to relevance.

the use of induced hypothermia in heart surgery, neurosurgery, head injuries, and infectious diseases.

This is a test query for Medline dataset[1] which we are using for analysis. Dataset has 1033 documents and 30 queries. For simplicity we use 399 documents and 1 query. For text mining, simple word matching is not enough. We use one of the most common method for text mining, namely, *Vector Space Model*[2] we use variant of vector space modelling knowns as Latent Semantic Indexing(LSI), which uses SVD of the term-document matrix.

Chapter 2

Preprocessing the Documents and Queries

We discuss preprocessing that needs to be done before the vector space model of a particular collection of document is set up. In information retrieval, keywords that carry information about the contents of a document are called terms. A basic step in information retrieval is to create a list of all the *terms* in a document collection, a so-called index. For each term, a list is stored of all the documents that contain that particular term. This is called an inverted index.

But before the index is made, two preprocessing steps must be done: elimination of all stop words and stemming.

2.1 Eliminating stop words

Stop words are words that one can find in virtually any document. Therefore, the occurrence of such a word in a document does not distinguish this document from other documents. The following is the beginning of one particular stop list[3].

> a, a's, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, another, any, anybody, anyhow, anyone, anything, anyway, anyways, anywhere, apart, appear, appreciate, appropriate, are, aren't, around, as, aside, ask, ...

given this stop words list, for each document string we eliminate stop words from string.

2.2 Stemming

Stemming is the process of reducing each word that is conjugated or has a suffix to its stem. Clearly, from the point of view of information retrieval, no information is lost in the following reduction:

$$[\text{hbt!}] \quad \begin{array}{c} \text{computable} \\ \text{computation} \\ \text{computing} \\ \text{computed} \\ \text{computational} \end{array} \quad \rightarrow \quad \text{comput}$$

There are many stemming algorithms available. For given dataset, we use *Porter stemmer*[4] some of the results obtained are:

Parsing all documents with removing stop words and stemming, we get 9357 terms. Final term-document matrix has shape of 4914 * 400. We also applied *term weighing scheme* where the elements of A are weighted depending upon characteristics of document collection. One can define element in A as:

$$a_i j = f_{ij} * log(n/n_i) \tag{2.1}$$

without stemming	with stemming
action	action
actions	
activation	activ
active	
actively	
activities	
activity	
acts	
actual	actual
actually	
acuity	acuiti
acute	acut
ad	ad
adaptation	adapt
adaptations	
adaptive	
add	add
added	
addition	addit
additional	

Table 2.1: sample index for dataset with porter stemmer

where f_{ij} is term frequency, the number of times term i appears in document j, and n_i is is the number of documents that contains term i (inverse document frequency). If a term occurs frequently in only a few documents, then both factors are large. In this case the term discriminates well between different groups of documents, and the log-factor in formula gives it a large weight in the documents where it appears.

Chapter 3

Query Matching and Performance Modeling

Query matching is the process of finding the documents that are relevant to a particular query q. This is often done using the cosine distance measure: a document a_j is deemed relevant if the angle between the query q and a_j is small enough. Equivalently, a_j is retrieved if

$$cos(\theta(q, a_j)) = \frac{q^T a_j}{||q||_2||a_j||_2} > tol, \tag{3.1}$$

where *tol* is predifined tolerance. If the tolerance is lowered, then more documents are returned, and it is likely that many of those are relevant to the query. But at the same time there is a risk that when the tolerance is lowered, more and more documents that are not relevant are also returned.

We applied query matching for query-9. By varying tolerance values result of precision vs recall are as below.

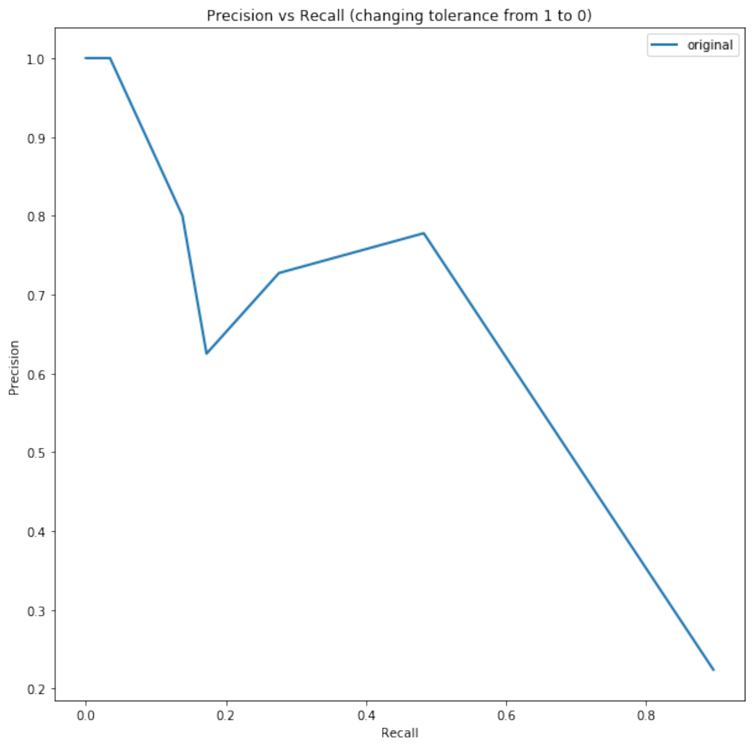


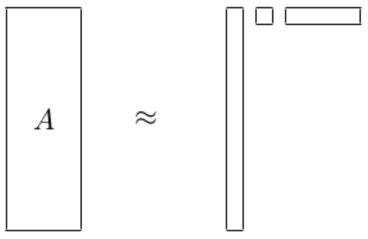
Figure 3.1: Query matching for sample query using the vector space method. Recall versus precision.

Chapter 4

Latent Semantic Indexing

Latent semantic indexing (LSI) "is based on the assumption that there is some underlying latent semantic structure in the data that is corrupted by the wide variety of words used" and that this semantic structure can be discovered and enhanced by projecting the data (the term-document matrix and the queries) onto a lower-dimensional space using the SVD.

Let $A = U^T$ be the SVD of the term-document matrix and approximate it by a matrix of rank k:



$$= U_k \Sigma_k V_k^T =: U_k H_k.$$

Figure 4.1: SVD

The columns of U_k live in the document space and are an orthogonal basis that we use to approximate the documents. Write H_k in terms of its column vectors, $H_k = (h_1, h_2, ..., h_n)$. From $A \approx U_k H_k$ we have a $j \approx U_k h_j$, which means that column j of H_k holds the coordinates of document j in terms of the orthogonal basis. With this rank-k approximation the term-document matrix is represented by $A_k = U_k H_k$ and in query matching we compute $q^T A_k = q^T U_k H_k = (U_k^T q)^T H_k$. Thus, we compute the coordinates of the query in terms of the new document basis and compute the cosines from

$$cos\theta_i = \frac{q_k^T h_j}{||q_k||_2 ||h_j||_2}, q_k = U_k^T q. \tag{4.1}$$

This Means that the query matching is performed in a k-dimensional space.

Chapter 5

Implementation

Main focus of this project was to implement parallel algorithm and to compare result of text mining with or without SVD. Most of the preprocessing stuff is done in python. Dataset is parsed line by line and documents are recorded as string. From each string stop words are removed. For stemming, Porter stemmer is used. Term- Document matrix is calculated keeping rows as terms and columns as documents. Last document is text query. This results are stored in csv format to be parsed by c++ code to run svd in parallel using openmp.

For actual cosine similarity measure, original data results and results obtained from svd are again stored in csv and parsed in python. Cosine similarity is measured for different values of tolerances. According to tolerance values, precision and recall values are calculated for sample query comparing with actual result. This process is done for different values of rank aproximation and results are recorded.

Chapter 6

Results and Analysis

We did query matching for query number 15 in medline dataset by taking 399 documents, approximating the matrix using the truncated SVD of various ranks and compared it with orignial results. For time measurement, we recorded running time of both parallel and sequential method. Reults obtained are:

6.1 Time Measure

By running algorithm for both sequential and parallel execution with different number of running threads results are:

- Sequential: 75.334 sec
- Parallel 2 Threads: 59.12 sec
- Parallel 4 threads: 86.67 sec

Result shows that we are getting marginal speedup on an average when run parallel.

6.2 Precision-Recall Measure

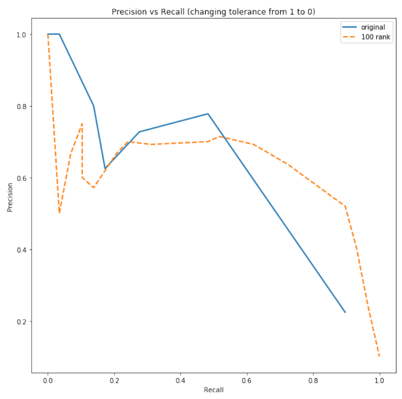


Figure 6.1: Recall versus precision for the full vector space model (solid line) and the rank 100 approximation

Looking at this results, It is seen that for this query, LSI improves the retrieval performance. Specifically for rank 200 we reach peak point where we can get optimal balance of precision vs recall which at specific tolerance which we can use for further measurements.

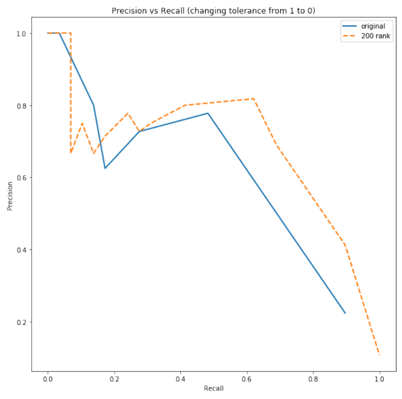


Figure 6.2: Recall versus precision for the full vector space model (solid line) and the rank 200 approximation

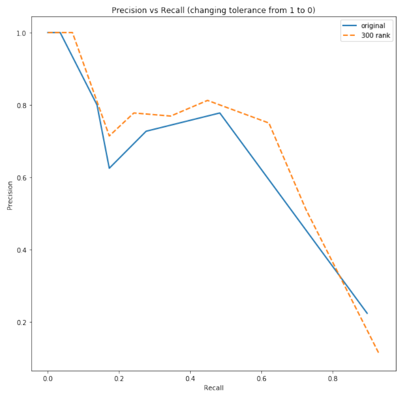


Figure 6.3: Recall versus precision for the full vector space model (solid line) and the rank 300 approximation

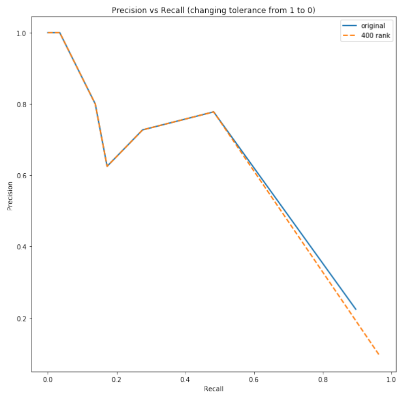


Figure 6.4: Recall versus precision for the full vector space model (solid line) and the rank 400 approximation

Bibliography

[1] Medline dataset. http://www.dcs.gla.ac.uk/idom/irresources/testcollections/.

[2] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[3] Stopwords. ftp://ftp.cs.cornell.edu/pub/smart/english.stop.

[4] Porter stemmer. http://www.tartarus.org/\begingroup\let\relax\relax\endgroup[Pleaseinsert\PrerenderUnicode{ĨČ}intopreamble]martin/PorterStemmer.