TEXT RETRIEVAL USING WAVELET TREE

A Thesis Presented to the Department of Computer Science,

African University of Science and Technology

In Partial Fulfilment of the Requirements for the Degree of

Masters of Computer Science

By

Isah Hauwa Yakubu

(ID No. 40807)

Abuja, Nigeria

July, 2021

CERTIFICATION

This is to certify that the thesis titled “Text Retrieval Using Wavelet Tree” submitted to the school of postgraduate studies, African University of Science and Technology (AUST), Abuja, Nigeria for the award of the Master's degree is a record of original research carried out by Isah Hauwa Yakubu in the Department of Computer Science.

TEXT RETRIEVAL USING WAVELET TREE BY:

ISAH HAUWA YAKUBU

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

APPROVAL BY

Supervisor

Surname: Rajesh

First name: Prasad

Signature: 

The Head of Department

Surname: Rajesh

First name: Prasad

Signature: 

ABSTRACT

The wavelet tree is a flexible data structure that permits representing sequences S[1; n] of symbols over an alphabet of size (n), within compressed space and supporting a wide range of operations on S. It has been used to index the document in the past. Text mining is the process of retrieving information from a huge body of text. During mining, we can extract the keywords from the document and convert them using term indexing, to numbers. Wavelet tree can then be used as an index to allow fast access of keywords within the documents using rank and select operations. The focus of this research is to design a hybrid model using wavelet tree and text mining to retrieve the keywords from the text.

Keywords: Wavelet tree, text mining, term indexing, rank operation, select operation.

DEDICATION

This project is dedicated to the Almighty, the most benevolent. Also friends and well-wishers for their prayers and non-stop support.

ACKNOWLEDGEMENT

I give the Almighty the glory and honour for knowledge and guidance. I would like to record my sincerest thanks to my colleagues for sharing their pearls of wisdom with me and immense gratitude to family and friends for their support and non-stop encouragement during this thesis. To my supervisor for his patience, perseverance and guidance. I say thank you. Finally, I thank African University of Science and Technology (AUST), for providing me with the scholarship to complete this program. May the Almighty bless you all.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

1.1 Text mining

It is no doubt that data is everywhere and is rapidly increasing every second. On the internet, a large amount of text is being exchanged in textual forms such as blogs, social media and e-mails (Sagayam, 2012).

Text mining is the technique of extracting interesting and significant patterns from textual data sources in order to ascertain knowledge. (Fan, Wallace, Rich, & Zhang, 2006). It deals with natural language text which is stored in a semi-structured and unstructured format (Weiss, Indurkhya, Zhang, & Damerau, 2010). It's a new tool for evaluating large data sets of texts for easy identification of patterns that are both fascinating and non-trivial. It can be seen as a leap from data mining or knowledge discovery from (structured) databases (Fayyad, Piatetsky-Shapiro, & Smyth, 1996).

Text mining has a lot of financial potential because written words are the most natural way of storing and exchanging information. A recent study indicated that 80% of a company's information was contained in text documents, such as emails, memos, customer correspondence, and reports. The ability to filter this untapped source of information provides substantial competitive advantages for a company to succeed in the era of a knowledge-based economy (Tan, 2002).

It is a multidisciplinary field, involving information retrieval, text analysis, and information extraction, clustering visualization, database technology, machine learning and data mining (Brinda, K.Prabha, & S.Sukumaran, 2016).

Text mining applications can extend to any area where text document exists (Stavrianou, Andritsos, & Nicoloyannis, 2007). These include social media, business intelligence, life science and health, fraud detection, digital libraries, academic and research field.

Text mining had received lots of attention since its inception, gaining a significant importance in research. This is due to the availability of a huge amount of textual documents from multiple sources and the increasing demands to obtain knowledge.

1.2 Wavelet tree

The field of wavelets has been growing on both the theoretical and application fronts since its invention. (Abbas & Rain, 2018).

A wavelet tree is a robust data structure that is useful in problem domains such as data compression and string processing. It was first introduced by Grossi, Gupta, and Vitter in 2003 as a way to obtain a faster rank and select query times on compressed suffix arrays.

It's a data structure for representing a sequence of elements across a fixed, possibly huge alphabet in a concise manner. It is large and versatile even when its compression capabilities are not considered (Castro, Lehmann, P´erez, & Subercaseaux, 2016)

A wavelet tree is a self-indexing binary tree used to index text characters. It implements three major operations: rank (which returns the number of occurrences of a symbol), select (which returns the position of a given occurrence of a symbol) and search (which returns the symbol at any position of the text) (Brisaboa, Cillero, Fari˜na, Ladra, & Pedreira, 2007). It can also be used with different types of coding schemes to increase the performance of search operations.

1.3 Problem statement

Text mining, deemed the "next wave of technology" by many, is a real hassle because it entails dealing with text data that is inherently unstructured and ambiguous. (Tan, 2002).

A lot of issues occur during text mining that affects the efficiency and effectiveness of decision making. The majority are posed by the peculiarities of natural language processing (Stavrianou, Andritsos, & Nicoloyannis, 2007).

Aside from the fact that it is powerful and easy to code, Wavelet trees have proven to be a viable approach in the past, particularly in terms of access time and space usage. Its ability to handle enormous text, the ease with which it can be implemented, and the fact that it can be seamlessly combined with other data structures gives it a significant benefit.

Exploring its potential, on the other hand, may lead to discoveries and, potentially, a long-term solution to the indexing problem's complexity.

1.4 Aim and Objectives

This project aims to design a hybrid model that uses wavelet tree and text mining to retrieve the keywords from texts.

Objectives include:

1. Numerical representation of text.
2. Indexing using wavelet tree**.**
3. Drawing the wavelet tree of text representation.
4. Retrieving keywords from the tree using rank, search and select operations.

1.5 Limitation

This thesis focuses on single keyword retrieval which is not the final goal of an application, but a key step of larger text mining systems (Beliga, 2014) and (Siddiqi & Sharan, 2015).

1.6 Project outline

This paper is divided into five chapters. The research aim and objectives are introduced in chapter one. A literature review of prior works connected to this research is described in chapter two. The third chapter explained the methodology that was used. Performance

analysis and evaluation, future works, and conclusion are contained in chapter four, followed by a summary and conclusion of the work in chapter five.

CHAPTER TWO

LITERATURE REVIEW

This chapter contains an extensive review of different indexing techniques. Some techniques used over the years were illustrated, some of which include suffice tree, signature files and inverted index.

2.1 Indexing

Text indexing also known as document indexing is a powerful technique for the retrieval of documents from repositories that contain thousands of documents (Chauhan & Asthana, 2017)

It's a collection of an organized, systematic arrangement of language, signs or symbols that represent ideas (Cleveland & Cleveland, 2013).

Joudrey & Taylor (2009) described indexing as a process that helps in evaluating the content of the source of information and determining the about-ness of an item.

Indexing forms the main functionality of the text retrieval process. It simplifies the document to informative terms (Kaur & Guptal, 2016). It has become an important tool in the area of information Retrieval such that whenever information is to be systematized or organized, retrieved or used, the need for indexing grows (Shah, 2015).

The difficulty of compressing and indexing excessively repeated sequence collections is discussed in (Makinnen, Navarro, Siren, & Valimaki, 2009), (Siren, Valimaki, Makinnen, & Navarro, 2008). Such collections can be found in a variety of applications, including version control systems and the storing of biological data in computational molecular biology.

2.2 Indexing techniques

There exist lots of indexing techniques, the most common which include an inverted index, signature files and suffix tree. These techniques differ in many ways from their simplicity of

application to their stability, performance, limitation as well as advantages. However, accuracy expectation usually cuts across all techniques.

2.2.1 Inverted index

An inverted file index consists of a record, or inverted list, for each term that appears in the document. A term's record contains an entry for every occurrence of the term in the document collection identifies the documents and, possibly, gives the location of the occurrences or weight associated with the occurrences (Brown, Callan, Croft, & Moss, 1994).

According to (Belew, 2006), the concept of inverted files is mostly used in commercial library systems. This is due to its enhanced efficiency in searching which is paramount when dealing with files that comprise large texts.

Inverted indices can be implemented as sorted arrays, tries, B-trees and various hashing structures. It needs to frequently undergo re-organization under intensive insertion/updating procedures (Chen, 2001).

2.2.2Suffix tree

Suffix trees have been extensively explored and utilized to fundamental string problems such as determining the longest repeated substring (Weiner, 1973) and text compression (Rodeh, Pratt, & Even, 1981).

Suffix trees have seen some uses in information retrieval fields. They are used for computing document similarities and related tasks.

The suffix tree algorithm was ranked one of the fastest at that time although the algorithm was all linear time for size alphabet that was constant. Its running time is generally given as O(nlogn). It supports insertion, deletion and modification of strings, which is its unique feature as compared to other techniques (Malki, 2016).

Weiner was the first to introduce suffice tree in 1973 (Weiner, 1973). He called them position trees. His Linear pattern matching algorithm had little follow-up because it was difficult to understand.

Mc Creight came along in 1976, he introduced another algorithm with better space efficiency which was considered too difficult to be extended for generalized suffice tree (McCreight, 1976)

Ukkonen in 1992 introduced an algorithm that conceptually builds the tree differently than either of the previous algorithms. His algorithm has generally been the preferred suffix tree construction algorithm mostly due to the paper being considered easier to understand than the previous two (Ukkonen, 1992).

2.2.3 Signature files

Text signatures and superimposed coding have been attractive in information retrieval systems as they are efficient in the amount of memory used and the reduction in processing time that they bring to searching operations. They are well-suited to the Boolean operations which are used to combine search terms, and are common to text retrieval systems (Colomb, 1985).

This technique is more efficient in dealing with insertions and queries on parts of words as compared to other techniques. However, it suffers from inefficiency in query processing since for each query processed the entire signature file needs to be scanned. According to (Chen, 2001), this disadvantage led to the development of the signature tree which improves the query processing efficiency of signature file significantly.

Eastman (1989) also explained that the use of bit comparisons in signature searching is an improvement in the text comparison process used in text scanning. He also suggested that the signature file technique is more suited to structured databases.

2.3 Article Review

Researchers have established comparisons between indexing systems, most of which are performance-oriented, based on their tests throughout the years.

Nobel, Moffat, & Romamoharao, in their paper, concluded that for current architectures and typical applications of full-text indexing, inverted files are superior to signature files in almost every respect, including speed, space, and functionality.

In the book modern information retrieval by Ricardo Baeza-Yates and Berthier Ribeiro-Neto, some implementation issues of the three main techniques with regards to their performances were cited; such that inverted file requires less search cost, while signature requires less space overhead (10- 20 %) than inverted file (Baeza-Yates & Ribeiro-Neto, 1999).

The wavelet tree structure was used as a range structure in (Arroyuelo & Navarro, 2011), (Russo & Oliveira, 2008) for their compressed self-index on texts compressed with the LZ78 data compression algorithm, and as a component of various substructures in (Kreft & Navarro, 2010) for their implementation of a self-index on texts compressed with the LZ77 data compression algorithm. This is particularly intriguing since LZ77 can benefit from the presence of many repetitions in a text, which appears in a variety of applications.

Navarro (2014) shows the most important practical and theoretical achievements in the literature, as well as applications in a variety of cases, in this outstanding study of wavelet tree data structure. In contrast to Navarro's survey, (Castro, Lehmann, P´erez, & Subercaseaux, 2016) focus is less on the properties of the structure in general, and more on its practical applications, some adaptations, and also implementation targeting specifically the issues encountered in programming competitions.

2.4 Research Gap

Text mining have many areas including classification, text analysis, text retrieval and more. Our work focuses on single keyword retrieval. We made use of small and medium size dataset, although larger dataset can be used to further confirm the performance of the indexing technique. Also previous algorithms of text retrieval was not practically implemented, thus their running time was not shown practically in our working environment.

CHAPTER THREE

METHODOLOGY

The methodology used to achieve the aim of building the hybrid model was the empirical model.

3.1 Hybrid model

The focus of this research is to design a hybrid model using wavelet tree and text mining (text retrieval) to retrieve the keywords from text. The wavelet tree is used to ensure sequential representation of data which makes use of repetitions as an advantage to facilitate fast retrieval. It can handle sequences of elements over a fixed but potentially large alphabet; after an initial pre-processing, the most typical queries can be answered in time O(logn), where n is the size of the array of words.

The concept of existing text retrieval models was studied. The wavelet tree was chosen, due to its versatility and easy implementation.

3.2 Implementation

The proposed model was implemented using Jupyter Notebook, a python programming environment with built-in functions and a lot of packages for data pre-processing procedures.

Advantages of python language include less coding, flexibility, simplicity, and interoperability, making it the perfect language for implementation.

3.3 Hardware Requirement

The hardware requirements are:

- All CPU of 64 bits and operating systems (Linux, Windows, Macintosh, etc.)
- 4GB RAM with at least 30GB HDD space

3.4 Theoretical Framework

The proposed framework consists of the following stage: data set collection, pre-processing, text representation, code implementation and lastly, performance analysis stage. The data collected is pre-processed to facilitate easy indexing which converts a document into a list or array of words, which in turn are given IDs, used as the root of the wavelet tree. Hence, a keyword can be retrieved and the rank and select query can be performed on the tree.

These have various applications which include crime detection, sentiment analysis, and analysis of customer feedback in business, risk management, etc.

3.4.1 Data set

A large part of these studies uses small-scale data sets, gotten from open source applications or public repositories. Some of which include a comprehensive collection of data and articles or write-ups from Google.

3.4.2 Data Pre-processing

Pre-processing of text data means cleaning of noise such as cleaning of stop words, punctuations, etc. It is an essential step as here, we prepare the text data ready for mining. If not applied there is a high tendency of the data inconsistency, thus jeopardizing the intended results. The majority of these pre-processing tasks are language-specific, and they often vary. So in pre-processing of the data, the sequence of strings was divided into words, all punctuation, stop words were removed. Words were grouped into groups, all missing values were replaced with some values and the case of text was transformed into a single one.

3.4.3 Stop words

Stop words are terms which doesn’t carry much weightage in context to the text. They are a set of commonly used words in English. Examples include ‘is’, ‘the’, ’a’, ‘we’ and ‘they. Removing these terms from the dataset can significantly improve retrieval performance

(Zaman, Matsakis, & Brown, 2011). Stop word lists are one of the key parameters when retrieving keywords.

3.4.4 Tokenization

Tokenization is as simple as splitting the text into white spaces and punctuation marks that do not correspond to the abbreviations found in the previous phase.

3.4.5 Case changing

A lot of techniques propose to lowercase only the words at the beginning of the sentence leaving the words in the middle capitalized. The standard approach is to lowercase all words, although this may result in the loss of information. This process is paramount to avoid duplications in further analysis.

3.4.6 Text representation

The pre-processing of a document gives a bag of words only. There is a need to convert this bag of words into numerical form for easy indexing.

Indexing is the most crucial part of the retrieval system. It was used for the conversion of the data set into numerical form which eases the indexing process. It represents the text of the document as a sequence of numbers, which can then be used as the root for the construction of wavelet tree.

Indexing starts at the beginning of a text, giving each unique word a unique ID number. This is done on the pre-processed text, hence only relevant words are indexed. Because it is sequential, the text can be represented using the created IDs.

This is implemented using dictionaries in python, where the words are the values with their corresponding ids as keys. Figure 3.1 explains the text retrieval process.

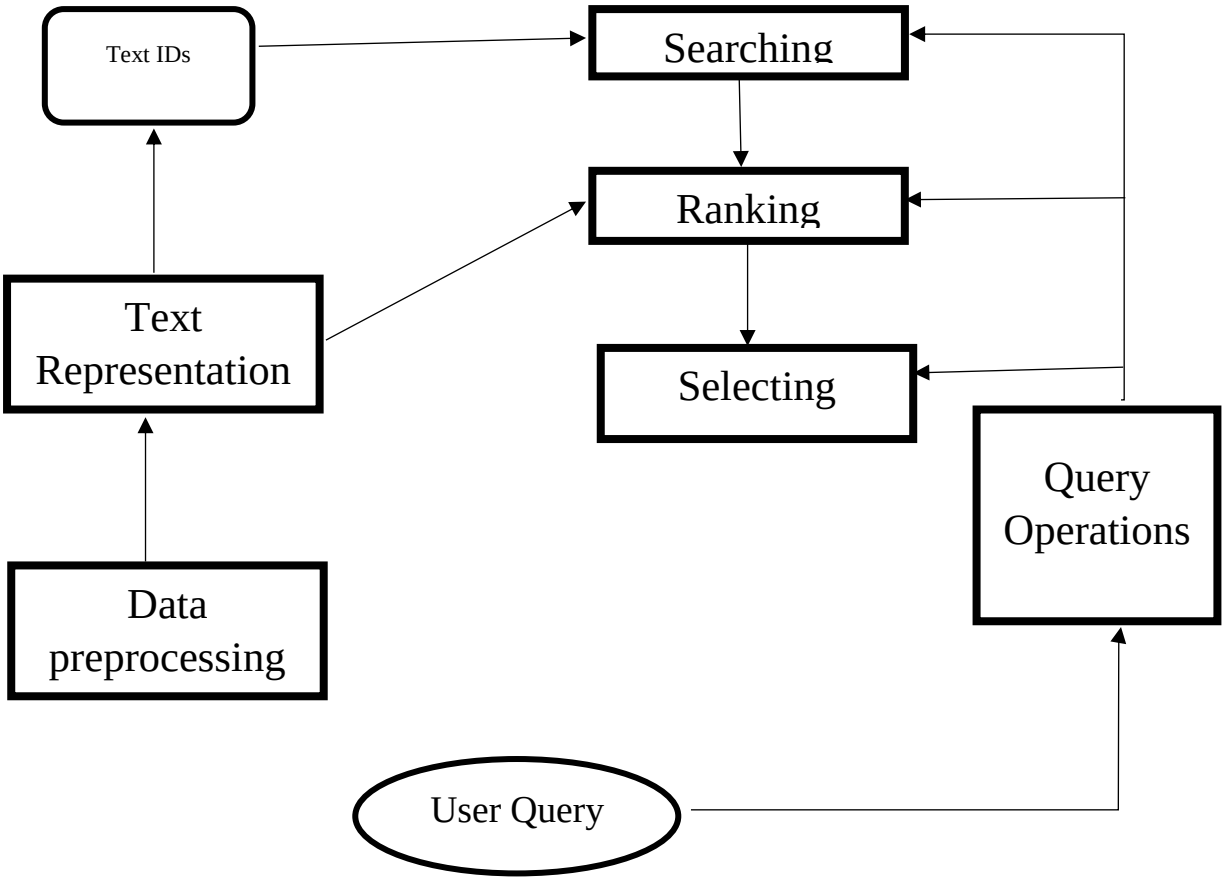

Figure 3.1: Text retrieval process

3.5 Constructing the wavelet tree

The wavelet tree is used to store the data set and run queries on it. It takes a numerical representation of the sentence as the root node and partitions it into two parts, the left child and the right child. The left child contains words whose IDs are less than the average A, of the smallest and largest id of the words in the sentence. The right child contains IDs greater than A. This is done recursively until just one word remains, which becomes the leaf.

Let L be the length of the root node. Each node stores a list of size n, containing all the IDs of words in the document. The number of leaf nodes is equivalent to the length of the root node. Figure 3.2 is an example of wavelet tree construction for the list of words in a sentence

([dog, ran, across, farm, house, ran, dog, across, farm, house, behind, lake] with corresponding IDs [1,2,3,4,5,2,1,3,4,5,6,7]).

Algorithm 1 below is used in the practical implementation of text representation.

Algorithm1 (Text representation)

function TextRep(sentence)

dict←{}

IdList←[]

newList←[]

for all w in sentence **do**

if w not in newlist **then**

x← x+1

newList.append(w)

IdList.append(x)

dict[w]← x

else

IdList.append(dic[w])

end if

end for

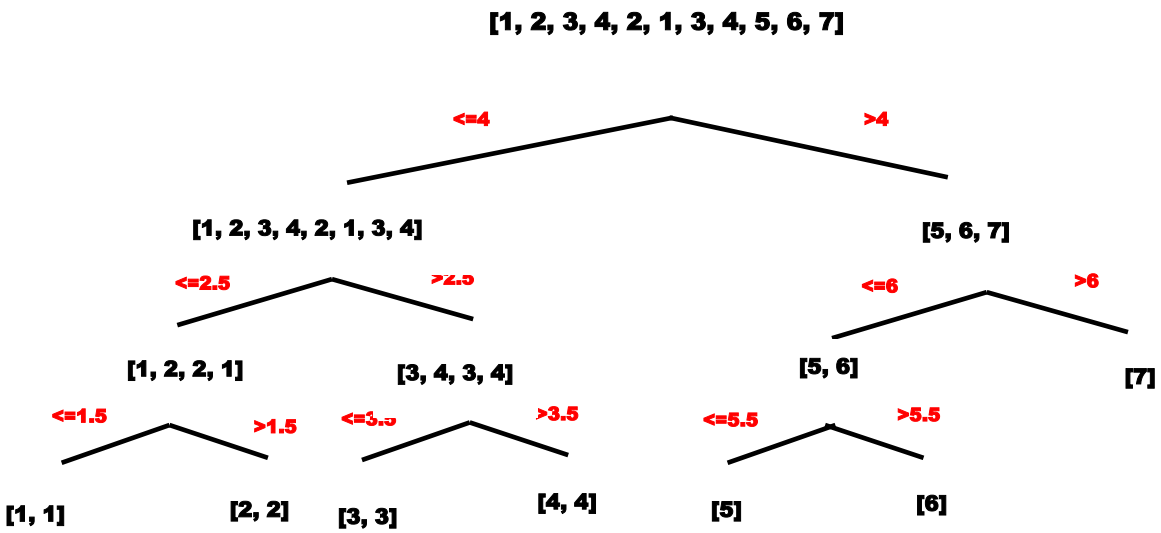

Figure 3.2: An example of wavelet tree construction

Note: only the IDs are stored in the tree.

Algorithm 2 (Knudsen & Pedersen, 2015) explains the creation of wavelet tree

Algorithm 2 (creating wavelet tree)

function NODE(sentence)

rep ← sorted(sentence)

If |sentence| = 1 or |rep| = 0 **then**

return self

end if

(R_{left},R_{right}) ← sentence

splitWord← R_{left} []

for all w in sentence **do**

if w > SplitWord **then**

S_{right}. Append(w)

Self.Bitmap.Append(1)

else

S_{left}.Append(w)

Self.Bbitmap.Append(0)

end if

end for

RightNode ← NODE(S_{right} , R_{right})

leftNODE ← NODE (S_{left}, R_{left})

return Self

end function

3.5.1 Search Operation

This involves finding a keyword present in the wavelet tree. Traversing through the entire wavelet tree is not done. This is because all IDs contained on the wavelet tree are at the root node, which is a list or array of all words present on the tree. Hence, a serial search is done to check if an ID corresponds to the word being search. Searching stops where it hits an ID that corresponds to the keyword. Also, the whole array is searched when the word is not present or is present at the final index.

For instance, searching for the keyword ‘house’ at the node of the wavelet tree in Figure 3.2. Searching begins at the first index 0. It compares ID 1 with its value, which returns false. The index is incremented, this is done recursively until it reaches index 4 with ID 5, whose value corresponds to the search term. Hence, it returns true. Figure 3.3 depict searching for the keyword ‘house’ in array of IDs in relation with the dictionary {‘dog’:1, ‘ran’:2, ‘across’:3, ‘farm’:4, ‘house’: 5, ‘ran’:2, ‘dog’:1, ‘across’:3, ‘farm’:4, ‘house’:5, ‘behind’:6, ‘lake’:7}.

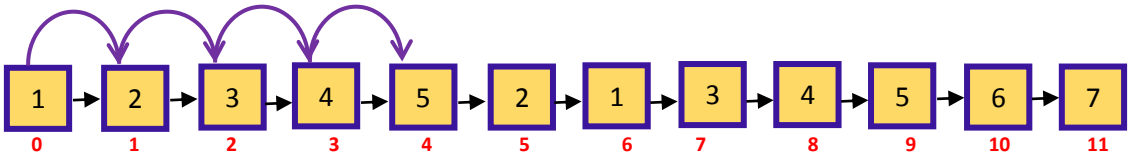

Figure 3.3: searching for keyword

Algorithm 3 explains the search operation on the wavelet tree.

Algorithm 3 (search operation)

Function exist(dict, textRep)

Input keyword k

if k is in dict then

if dict[k] is in textRep then

Output← ‘keyword found’

Output← ‘keyword found’

else

Output← ‘keyword not found’

Print output

end if

else

Output ← ‘doesn’t exists’

Print output

end if

end function

3.5.2 Rank Operation

The rank is an operation that counts the occurrences of value w till an ID i of S in a sequence S. It is usually denoted by $rank_w(S,i)$. That is, if S = (s1,...,sn) then

$$rank_w(S,i) = |\{e \in \{1,...,i\}| se= w\}|$$

For the non-leaf node S of W, we encapsulate the rank operation above in two abstract functions, mapLeftW(S,k) and mapRightW(S,k). The map function shows the new index k, which the index i from the previous level of the tree, maps to. As shown in Figure 3.4 mapLeftW (S^0,10)=7, mapLeftW(S^1,7)=3, and mapLeftW(S^2,3)=2. The superscripts 0, 1 and 2 depict the level of the tree at which the mapping occurs.

The rank operation on a wavelet tree begins at the root and travels down the tree until it hits the leaf node that fits to the input word. The index rank obtained in the root node when the leaf node is reached is the rank of the input symbol up to the original input point. This means that the rank of a word up to a point in a sentence containing only that word is the same as the

location in a sentence containing only that word. For instance, in Figure 3.4, a rank query was performed on the wavelet tree in Figure 3.3 thus showing how the rank query works.

In Figure 3.4 we have that $rank_3$ (10) = 2. Assume that W is a wavelet tree for S, then $rank_w$(S,i) can be easily computed with the following strategy. If w ≤ A then we know that all occurrences of w in S appear in the sequence LeftW(S), and thus $rank_w$ (S,i) = $rank_w$ (LeftW (S), mapLeftW(S,i)). Similarly, if w > A then $rank_w$(S,i) = $rank_w$(RightW(S), mapRightW(S,i)). These process is repeated until we reach a leaf node; if we reach a leaf S with this process, we know that $rank_w$(S,i) = i.

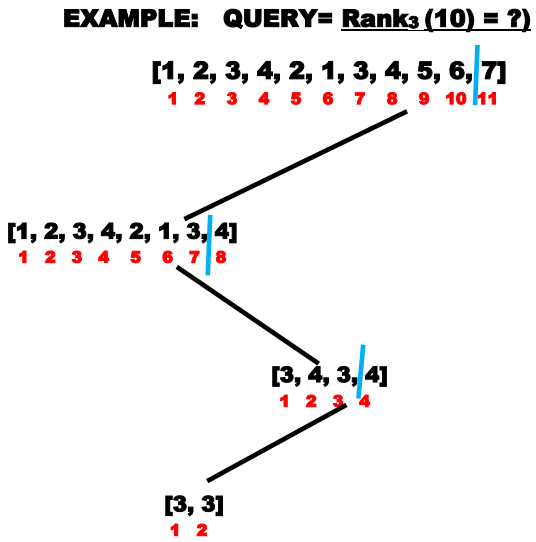

Figure 3.4: illustration on how rank query works

In the example, the rank operation looks for the number of occurrences of the word before offset 10. Traversal begins from the root node and moves downward until it reaches the leaf node. At each level the query $rank_w$ (i) is performed where i is index or value gotten from the previous level, thus i becomes the new offset of the next level.

The execution of $rank_3$(S, 10) is shown with blue lines. Index 10 was mapped down the tree using either mapLeftW or mapRightW depending on the A value of every node in the path. We first map 10 to 7, then 7 to 3 and finally 3 to 2, reaching a leaf node. Thus, the result of

the overall query, $rank_3$ (10)=2. Algorithm 4 (Knudsen & Pedersen, 2015) was used in the practical implementation of the rank operation.

Algorithm 4 (rank operation)

```
Function rank (S,w,i)
    if S is a leaf then
        return i
    else if w ∈ leftW(S) then
        return rank (leftW(S), w,mapleftW(S, i))
    else return rank (rightW(S), w,mapRightW(S,i)
    end if
end function
```

3.5.3 Select Operation

The select query can be used to find the position of a character's occurrence. Traversal on the wavelet tree begins at the leaf node that matches a word up until it reaches the root node. This is the reverse of the rank query. Thus, it is paramount to determine the leaf node which corresponds to the word whose position is to be determined.

After the leaf node has been determined, an upward traversal is done to determine the position of the word. Algorithm 5 was used in implementing the select operation.

Algorithm 5 (select operation)

```
select(S,w,i)
    if S is a leaf then
        return i
    else if w ∈ labels then
        return select0 (Bv,select(vl,w,i))
```

```
else
        return select1(Bv,select(vr,w,i))
end if
```

CHAPTER FOUR

PERFORMANCE ANALYSIS AND RESULTS

This chapter includes the implemented framework and results with the programming language used, from the pre-processing phase to the testing and debugging phase of the hybrid model. The following steps were taken to achieve this research work: data pre-processing; removal of stop words and punctuation, evaluation of the model's performance in terms of accuracy, precision and sensitivity are presented; all the phases of pre-processing, text representation and testing of data, and measures of accuracy are implemented using created classes and functions using Jupiter notebook in python programming

Screenshots of results were presented to support our framework.

4.1 Analysis

The focus was to improve the keyword search, rank and select functionality. The main thing needed is to randomly assign IDs to the available data from the dataset to make the ranking, searching and selecting process less tedious. This was achieved using Jupyter IDE and python programming language. The hybrid model is based on text mining and single keyword retrieval and any text format can be converted into numbers. Searching is made efficient for the query sent by the user. The complexity of the rank, select and search process is hidden from the user. In the ranking of keywords, the rank frequency is displayed to the user. In searching, it shows the existence of a keyword that returns a positive result, if the keyword is an exact match to the user's query. This model will be effective whenever keywords are needed to be retrieved from structured or unstructured text.

4.2 Performance

Our model is efficient in searching, ranking and retrieving the data. So when a query is sent, the system can search the information relevant to the query and retrieve relevant keywords

with their frequencies of appearance, from the document. The frequencies depict how relevant a keyword in the text is. The results from our model give the user insight of what the whole text entails. It is understood by the rank given to keywords present in the data. These frequencies were given after the entire pre-processed text had been scanned and converted. But in the case of rank gotten from a keyword search, the size of the data is reduces in each step.

A small data set of size 1384 gotten from Google was used. A dictionary of values and keys was established, these were used to determine the result from our model. Table 4.1 shows the different times it takes to run different rank, select and search queries.

Table 4.1: search, rank and select execution time

Keyword	Search (secs)	Rank (secs)	Select (secs)
Domestic	1.562…	1.435…	2.443…
Health	1.875…	1.243…	1.989…
application	1.278…	1.604..	2.333…
Theft	1.334...	2.041…	1.232…
Tracking	1.644…	0.992…	1.876…

Based on the results gotten, we concluded that our code has a logarithmic complexity of O(log n). Where n is the number of keywords in the dataset.

4.3 Result

The results were obtained after the model was tested and analyzed using random datasets from different repositories, especially Google.

4.3.1 Reading the text file

The data files used were downloaded to my local drive E in the folder 'Hauwa' as 'sampleWLT.text file'. Its directory "E:\Hauwa\sampleWLT.txt". The python functions open () and read () were used to load and read the content of the .text file as shown in Figure 4.1.

```
doc1= open("E:\Hauwa\sampleWLT.txt")
doc2=doc1.read()
```

Figure 4.1: reading the .text file

4.3.2 Data pre-processing

In the implementation of the data pre-processing both inbuilt and created functions were used.

Removing punctuations

This is the first pre-processing step performed on data. It involves the removal of all English punctuation like the comma, full-stop, bracket, etc. It is implemented using snapshot of the code of Figure 4.2.

```
##REMOVING ALL PUNCTUATIONS
punc= '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
doc4=""
for char in doc3:
    if char not in punc:
        #newlist.append(char)
        doc4= doc4+char
#print(doc4)
```

Figure 4.2: Removing punctuations

4.3.3 Removing stop word

The nltk package which contains specified English stop words was imported. These stop words were then downloaded and removed from text with the code as seen in Figure 4.3 below.

```
## dowloading packages using the following commands
import nltk
nltk.download('stopwords')
```

```
##this is nltk stopword collection
from nltk.corpus import stopwords
```

```
###REMOVING ALL STOP WORDS
relevant=[]
for word in doc4:
    if word not in stop_words:
        relevant.append(word)
#1)print(relevant)
```

Figure 4.3: Removing stop words

After running the code in Figure 4.3 above, all stop words were removed from the document file.

4.3.4 Case conversion

After the removal of stop words and punctuation, the doc which now contains a bunch of words was converted to a single case (lower case). This was done with the help of the lower () inbuilt method .This step is important to avoid duplication in subsequent phases.

Figure 4.4 contains the snap shot of case conversion code.

```
doc3=doc2.lower()      ##remove all caps
#print(doc3)
```

Figure 4.4: Case Conversion

4.3.5 Text to word conversion

The text gotten from the previous pre-processing steps was then changed to words. This was achieved with the split () function. Figure 4.5 is the snapshot of code used in text to word conversion.

```
##CONVERTING DOC TO WORDS
doc4= doc4.split()
#print (doc3)
```

Figure 4.5: text to word conversion

4.3.6 Words to numbers conversion

Finally, the collection of words gotten from the above step was converted to list or array of numbers. This simplifies the indexing phase that comes after the data pre-processing phase. A function called TextRep was created, in which contains dictionaries and list were used. All the words where taken as keys and the unique numbers generated as values. The process is depicted in Figure 4.6.

```
NewDict={}
###TEXT REPRESENTATION TO NUMBER
def TextRep(relevant):

    #NewDict={}
    Numlist=[]
    Emptylist=[]
    x=0

    for word in relevant:
        if word not in Emptylist:
            x+=1
            Emptylist.append(word)
            Numlist.append(x)
            NewDict[word]=x
        else:
            Numlist.append(NewDict[word])
    return Numlist
            #num_rep=Numlist
```

Figure 4.6: conversion of words to numbers

4.4 Data Visualization

4.4.1 Word cloud

This is a data visualization technique that displays all text in a single space. The words with the highest frequency distribution are displayed larger than those with lower frequencies. The word cloud of our data set containing 55 keywords is shown in Figure 4.7.

Figure 4.7: Word cloud showing words with different ranks.

Figure 4.7 shows the most frequent words are in descending order; health, domestic, violence, well, widely and so on.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

The amount of data created, generated, and saved is enormous. The retrieval of text would be tedious without appropriate knowledge of Information Retrieval procedures. Studies have also shown that text retrieval approaches are paramount for information storage and retrieval in information centers (such as sentiment analysis). Hence, the right text retrieval model can be of great improvement to the status quo of retrieval systems.

The present indexing techniques were investigated in this study. Problems related to them were researched. The concept of existing text retrieval models was studied, and the knowledge gained was used to design this hybrid system. It was successfully implemented using real-life data. It merely informs the user of the existence or non-existence and whereabouts of the keyword relating to the query and how frequently it appears. The model was built using an algorithm, to perform efficiently for different dataset formats.

5.2 Possible Applications

Documents contain the majority of an organization's knowledge that is obscured in electronic media. Acquiring this knowledge necessitates good data querying as well as the combining of data from many textual sources. Due to the wide range of applications, discovering such concealed knowledge is a crucial requirement for many organizations.

Some of these applications of keywords mined include:

- **Academic Research**: As a researcher, one of the major tasks while building up your paper is to mention or suggest keywords for your article. It helps both rookie and

experienced researchers to choose, organize, document, and interpret data in a way that produces legitimate and reliable knowledge for academic study.

- **Customer service**: The worldwide industrial advancement has surpassed the mass production era and has entered a period of mass customization. Specifically, the saturation of product supply generated a need to assess customer demand and preference as a strategy to promote purchases from consumers. This coincided with the birth of affective engineering, which converts a consumer's feelings toward a product into design factors (Nagamachi, 1995). Keywords mined from text collected from blogs, surveys, newspaper articles, social media, and other text information sources can inform owners of the status of their businesses. These pieces of information can be used to enhance a product or service and improve customer experience as the case may be.
- **Resume filtering**: Every day, large corporations and headhunters receive thousands of resumes from job seekers. Extracting information from this resume can be quite a challenging task. Moreover, these resumes are of different file formats (pdf, word, jpeg, etc.), and in different languages. Recruiters are concerned with mistakes, qualifications, fuzz words, employment history, job titles, and other relevant information. Extracting keywords elated to these concerns can be the major first step in filtering resumes. Hence, easing the resume filtering task.
- **Other applications** include Crime detection, Decision making, Spam filtering and sentiment analysis, etc.

5.3 Future work

This thesis focuses on single keyword retrieval. An extension can be done to accommodate phrases, multiple word combinations. Term frequency and inverse document frequency is can be used in future work, to help filter relevant keywords.

The TF-IDF statistic reflects the importance of terms appearing in a text in comparison to a large corpus of texts. Its many different extensions are commonly used by search engines (Croft, Metzler, & Strohman, 2009). It is also frequently used in document classifiers for checking relevance between documents or between a given search query and a found document (Łażewski, Pikuła, Siemion, & Szklarzewski, 2005).

5.4 Conclusion

Within the context of time utilization, this model can be of tremendous help in solving some of the identified problems of text retrieval.

Of the many other ways data simplification can be done, the wavelet tree was used due to its ease of traversal. Making searching half as less tedious as it should be. Although this model has the problem of data size limitation, it can be very efficient. On increasing the query length the performance increases more than had been seen before.

APPENDIX

```
##############NLTK STOPWORDS COLLECTION

import time
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
   ################Loading the the data set from excel
#doc1= open("E:\Hauwa\sampleWLT.txt")
doc1= open("E:\Hauwa\sim.txt")
doc2=doc1.read()

text=re.sub(r'[^a-zA-Z]'," ", doc2)         #for word cloud
#print(text)

doc3=doc2.lower()     #####remove all caps
#print(doc3)

## ############USING NLTK TO FILTER STOPWORDS
stop_words = stopwords.words('english')
stop_words=set(stop_words)
#print(stop_words)

############REMOVING ALL PUNCTUATIONS
punc= '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
doc4=""
for char in doc3:
   if char not in punc:
      #newlist.append(char)
```

```
        doc4= doc4+char
#print(doc4)

#############CONVERTING DOC TO WORDS
doc4= doc4.split()
#(doc4)
#print(len(doc4))

##########REMOVING ALL STOP WORDS
relevant=[]
for word in doc4:
    if word not in stop_words:
        relevant.append(word)
#print(relevant)                    #1) print words
#print()
#print(len(relevant))

NewDict={}
################TEXT REPRESENTATION TO NUMBER
def TextRep(relevant):
    #NewDict={}
    Numlist=[]
    Emptylist=[]
    x=0

    for word in relevant:
        if word not in Emptylist:
            x+=1
```

```
            Emptylist.append(word)
            Numlist.append(x)
            NewDict[word]=x
        else:
            Numlist.append(NewDict[word])
    return Numlist
rep=TextRep(relevant)                    #2)print ids
#print (rep)
#print((end - start), 'secs')
#print (relevant)

start=time.time()
############TO CHECK IF A KEYWORD EXIST:
def exist(dict,numrep):
    keyword=input(' enter keyword  ')
    if keyword in dict:
        if dict[keyword] in numrep:
            output='found'
            print('word',output)
        else:
            output='not found'
            print('word',output)
    else:
        output='doesnt exist'
        print('word',output)
    return keyword,output

rep=TextRep(relevant)                #3)existance of a word
```

```
e=exist(NewDict,rep)

end = time.time()
print('time it takes to execute = ',(end - start), 'secs')
"""
#exist=list(exist(NewDict,rep))

start=time.time()
rep=TextRep(relevant)

###################### RANK QUERY
keyword=input('keyword is ')
count=0
#e=list(exist(NewDict,rep))
if keyword in NewDict:
    if NewDict[keyword] in rep:   output='found'
    else:  output='not found'
else:   output='doesnt exist'

if output=='found':
    for c in rep:
        if c==NewDict[keyword]:
            count+=1
    print('Rank of' ,keyword ,'is', count)
else:  print('please enter existing keyword on WLT')

end = time.time()
print('Time it takes to execute = ',(end - start), 'in secs')
```

```
##############SElECT QUERY
keyword=(input('keyword is '))
start=time.time()
rep=TextRep(relevant)
count=0
keyword=keyword.lower()
#e=list(exist(NewDict,rep))
if keyword in NewDict:
    if NewDict[keyword] in rep:   output='found'
    else:  output='not found'
else:   output='doesnt exist'

if output=='found':
    for c in rep:
        if c==NewDict[keyword]:
            print('found at position(s)',end=" ")
            for i, y in enumerate(rep):
                if y == c:
                    print (i , end =" " )
                   # print()
            break
    #print('Rank of' ,keyword ,'is', count)
else:  print('please enter an existing keyword on wavelet tree')
end = time.time()
print()
print('Time it takes to execute = ',(end - start), 'in secs'
```

REFERENCES

Abbas, Z., & Rain, P. (2018). A Study on Applications of Wavelets to Data Mining. *International Journal of Applied Engineering Research, 13*, 10886-10896 . Retrieved from http://www.ripublication.com

Arroyuelo, D., & Navarro, G. (2011). Space-efficient construction of Lempel-Ziv compressed text indexes. Information and Computation. *209*, 1070-1102.

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern Information Retrieval.* New York: ACM press.

Belew, R. (2006). Adaptive information retrieval. In *Machine Learning in Associative Networks* (pp. 78-83). Michigan: University of Michigan Press.

Beliga, S. (2014). Keyword Extraction: A Review of Methods and Approaches. . Rijeka: Google scholar.

Brinda, S., K.Prabha, D., & S.Sukumaran, D. (2016, September). The comparison of text based methods using text mining. *International Journal of Computer Science and Mobile Computing., 5*(9), 112-116. Retrieved from www.ijcsmc.com

Brisaboa, N. R., Cillero, Y., Fari˜na, A., Ladra, S., & Pedreira, O. (2007). A New Approach for Document Indexing Using Wavelet Trees. *International Workshop on Database and Expert Systems Applications (DEXA 2007).* Regensburg: IEEE.

Brown, E. W., Callan, J. P., Croft, W. B., & Moss, J. E. (1994). Supporting Full-Text information retrieval with a persistent object store. *International Conference on Extending Database Technology. 779*, pp. 365-378. USA: EDBT.

Castro, R., Lehmann, N., P´erez, J., & Subercaseaux, B. (2016). Wavelet Trees for Competitive Programming. *INTERNATIONAL OLYMPIADS IN INFORMATICS, 10*, 19-37.

Chauhan, E., & Asthana, D. A. (2017, July). Review of Indexing Techniques in Information Retrival. *International Journal of Engineering Science and Computing (IJESC), 7*.

Chen, Y. (2001). Signature files and signature trees. *Information processing letter, 82*, 213-221.

Cleveland, D. B., & Cleveland, A. D. (2013). Introduction toIndexing and Abstraction (4th Edition). *Cataloging and classification quarterly, 52*, 337-338. Retrieved from https://doi.org/10.1080/01639374.2013.877113

Colomb, R. M. (1985). Use of Superimposed Code Words for Partial Match Data Retrieval. *The Australian Computer Journal, 17*, 181-188.

Croft, B., Metzler, D., & Strohman, T. (2009). Search Engines: Information Retrieval in Practice. Pearson Education Inc.

Eastman, C. M. (1989). Handling incrementally specified Boolean queries: a comparison of inverted and signature file organizations. Information processing and management.

Fan, W., Wallace, L., Rich, S., & Zhang, Z. (2006). Tapping the power of text mining. *Communications of the ACM, 49*(9), 76-82.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery: An overview. Cambridge: MIT Press.

Feldman, R., & Dagan, I. (1995). Knowledge discovery in textual databases (KDT). KDD 95. 112-117.

Grossi, R., Scott, J., & Xu, B. (2011). Wavelet tree from theory to practice. *International Conference on Data Compression, Communications and Processing* (pp. 210-221). IEEE. doi:10.1109/CCP.2011.16

Joudrey, D. N., & Taylor, A. G. (2009). *The organization of Information.* (3rd, Ed.) CT:Libraries Unlimited.

Kaur, H., & Guptal, V. (2016). Indexing Proess, Insights and Evaluation. *International Conference on Inventive Computational Technologies (ICICT).* Coimbatore: IEEE. doi:https://doi.org/10.1109/INVENTIVE.2016.7830087

Knudsen, J. H., & Pedersen, R. L. (2015). *Engineering Rank and Select Queries on Wavelet trees* . thesis. Retrieved from https://cs.au.dk/~gerth/advising/thesis/jan-hessellund-knudsen-roland-larsen-pedersen.pdf

Kreft, S., & Navarro, G. (2010). Self-indexing based on LZ77. In *Lecture Notes in Computer Science* (Vol. 6661, pp. 239-248). Heidelberg: Springer. doi:https://doi.org/10.1007/978-3-642-21458-5_6

Łażewski, L., Pikuła, M., Siemion, A., & Szklarzewski, M. (2005). Klasyfikacja Dokumentów Tekstowych (Text Document Classification). Poland: Google Scholar.

Makinnen, V., Navarro, G., Siren, J., & Valimaki, N. (2009). Storage and retrieval of highly repetitive sequence collections. *Journal of Computational Biology JCB*, 281-308.

Malki, Z. (2016). Comprehensive Study and Comparison of Information Retrieval Indexing Techniques. *International journal for advanced computer science and applications (IJACSA), 7*. Retrieved from www.ijacsa.thesai.org

McCreight, E. M. (1976). A space-economical su_x tree construction algorithm. *Journal of the ACM, 23*(2), 262-272. Retrieved from https://doi.org/10.1145/321941.321946

Mooney, R. J., & Nahm, U. Y. (2005). Text mining with Information Extraction. *Multilingualism and Electronic Language Management: Proceedings of the 4th International MIDP Colloquium*, (pp. 141-160). South Africa.

Nagamachi, M. (1995). Affective engineering: A new ergonomic consumer-oriented technology for product. *International Journal of Industrial Ergonomics, 15*, 3-11.

Navarro, G. (2012). Wavelet tree for all. *Combinatorial Pattern Matching:Lecture notes in computer science. 7354*, pp. 2-26. Heidelberg: CPM. Retrieved from https://doi.org/10.1007/978-3-642-31265-6_2

Nobel, J., Moffat, A., & Romamoharao, K. (1998). Inverted Files Versus Signature Files for Text indexing. *ACM transaction Database Systems, 23*, 453-490.

Rodeh, M., Pratt, V. R., & Even, S. (1981). Linear algorithm for data compression via string matching. *Journal of the ACM, 28*, 16–24.

Russo, L., & Oliveira, A. (2008). A compressed self-index using a Ziv-Lempel Dictionary. Information Retrieval. *SPIRE'06: Proceedings of the 13th international conference on String Processing and Information Retrieval*, (pp. 501-513). Retrieved from https://doi.org/10.1007/11880561_14

Sagayam, R. (2012). A survey of text mining: Retrieval, extraction and indexing techniques., *2*, pp. 1443-1446.

Salloum, S. A., Al-Emran, M., Monem, A. A., & Shaalan, K. (2018). Using Text Mining Techniques for Extracting Information from Research Articles. *Intelligent Natural Language Processing: Trends and Applications, Studies in Computational Intelligence* .

Shah, N. S. (2015). Review of Indexing Techniques Applied in Information Retrieval. *Pakistan Journal of Engineering, Technology & Science (PJETS), 5*, 27-47.

Siddiqi, S., & Sharan, A. (2015). Keyword and keyphrase extraction techniques: a literature review. International Journal of Computer Applications. *International Journal of Computer Applications*, 18-23.

Siren, J., Valimaki, N., Makinnen, V., & Navarro, G. (2008). Run-Length compressed indexes are superior for highly repetitive sequence collections. *Proceedings of the 15th International Symposium on String Processing and Information Retrieval*, (pp. 164-175). Retrieved from 10.1007/978-3-540-89097-3_17

Stavrianou, A., Andritsos, P., & Nicoloyannis, N. (2007, September). Overview and Semantic Issues of Text Mining. *ACM SIGMOD records, 36*(3), 23-34.

Talib, R., Hanif, M. K., Ayesha, S., & Fatima, F. (2016). Text mining: Techniques, Applications and Issues. *International Journal of Advanced Computer Science and Applications, 7*(11).

Tan, A. H. (2002). TEXT MINING:PROMISES AND CHALLENGES. Singapore.

Ukkonen, E. (1992). On-line construction suffix trees in linear time. *14*, pp. 249-260. Algorithmica . Retrieved from https://doi.org/10.1007/BF01206331

Weiner, P. (1973). *Linear pattern matching algorithms. In The 14th Annual Symposium on Foundations of Computer Science.* IEEE. Retrieved from 10.1109/SWAT.1973.13

Weiss, S. M., Indurkhya, N., Zhang, T., & Damerau, F. (2010). predictive methods for analyzing unstructured information. In *Text mining.* Springer Science and Business Media.

Zaman, A. N., Matsakis, P., & Brown, C. (2011). Evaluation of stop word list in information retrieval using latent semantic indexing. *International conference of digital information management*, (pp. 26-28). Melbourne.