Thesis for the Degree of Master of Engineering

Design and Implementation of an Efficient Web Crawling

By

Md. Tanvir Ahmed

Department of Computer Engineering

The Graduate School

Pukyong National University

August 23, 2019

Design and Implementation of an Efficient Web Crawling

효율적인 웹크롤링 설계 및 구현

Advisor: Prof. Mokdong Chung

By

Md. Tanvir Ahmed

A thesis submitted in partial fulfillment of the requirements
for the degree of

Master of Engineering

in Department of Computer Engineering, The Graduate School,
Pukyong National University

August 23, 2019

Design and Implementation of an Efficient Web Crawling

A dissertation
by
Md. Tanvir Ahmed

Approved by:

Jeong-Mo Yeo	[illegible]
(Chairman)	
Kyung-Ryong Seo	Kyung Ryong
(Member)	(Member)
Mokdong Chung	Mokdong Chung
(Member)	(Member)

August 23, 2019

Contents

Appendix A

List of Tables

List of Figures

Design and Implementation of an Efficient Web Crawling

Md Tanvir Ahmed

Department of Computer Engineering, The Graduate School,

Pukyong National University

ABSTRACT

The key part for the attainment of the World Wide Web (WWW) is its large size and the lack of centralized control over its contents. Both manifestations are also the most important source of problems for locating information. The web is a context in which traditional Information Retrieval (IR) methods are challenged, and given the volume of the web and its speed of change, the coverage of modern search engines is comparatively petty. Moreover, the distribution of quality is very skewed, and interesting pages are scarce in comparison with the rest of the content.

The web crawler is a program, which is commonly used by search engines to find the new brainchild on the internet. The use of crawlers has made the web easier for users. In this thesis, we have used unstructured data by structuralization to collect data from the web pages. Our system is able to choose the word near our keyword in more than one document using unstructured way. Neighbor data were collected on the keyword through word2vec. In this research, we have introduced (Word2vec-Tiling) a new technique to collect data from the web pages. In our research, we used word2vec in the tiling method. Word2vec is considered as one of the easiest and most popular algorithms for identifying the relationship weight among the documents. Thus, the user will get accurate information related to his keyword. In this thesis, specific keywords were used by Word2vec-Tiling to identify naturally fused word frequencies, semantic relationships, and directional text-ranks. Finally, our system proposed a competent web search crawling algorithm that is derived from word2vec and Reinforcement Learning (RL) and Term Frequency and Inverse Document Frequency (TF-IDF) web search algorithm to enhance the searching efficiency for the relevant information. Therefore, the neural network is an advanced mechanism for verifying the semantic relationship between words and texts in a particular document. Our approach uses the Word2vec-Tiling to capture the unstructured data semantic features of words in the selected text while naturally integrating the word frequency and semantic relation.

The Word2vec-Tiling is a technique for subdividing texts into multi paragraph units that represent passages, or subtopics. The discourse cues for identifying major subtopic shifts are patterns of semantic co-occurrence and distribution. The algorithm (Word2vec-Tiling) is fully implemented and is shown to produce segmentation that corresponds well to Artificial Intelligence (AI) judgments of the subtopic boundaries of multiple texts. Multi paragraph subtopic segmentation should be useful for many text analysis tasks, including text semantic relation and word embedding. By following the Word2vec-Tiling method that we have proposed, it is easily possible to find out the related information about the searched keyword on Internet that we have already proved by improving the equation of TF and RL algorithm.

Chapter 1

Introduction

1.1. Introduction

With the flourishing availability of text documents on the World Wide Web (WWW), it has become very difficult for an average reader to gather information on individual issues, events, or topics by reading each and every document with an aim to find out the useful information. Therefore, the smart crawling system able to play a major role to overcome this problem. Because the crawler is a program that retrieves and stores pages from the web, commonly for a web search engine [2]. In this research, we have discussed how web crawling is highlighted in an efficient way using unstructured data. We preferred unstructured data in this article and the data was collected from the web page using a specific keyword. The unstructured data is a data or information that do not have any predefined data model or not organized pre-defined model [8], [12]. Earlier, in the research on unstructured data [1], [8] in web pages similarity but was very condensed. For this reason, their accuracy rate is not up to the satisfaction levels. Therefore, to solve this phenomenon, on this research effective method of collecting unstructured data is demonstrated. First of all, we have collected automatically keywords related urls. Then we created the data tree using the Breadth First Search (BFS) method from the all collected urls [20]. Unstructured data was accounted and collected from the web page. We have collected multitudes of unstructured data.

In recent years, most research has been focused on finding new categorization algorithms; however, little research has been done on the improvement of document representation models, which comes from unstructured data tools text semantic relation. The early research on web crawler used word2vec semantic function to investigate unstructured data word frequency. As a result, its accuracy rate is not up to satisfactory levels. In this thesis, we proposed a new algorithm named Word2vec-Tiling, based on Word2vec, a widely used weighing method in text segmentation or categorization [19]. In this thesis, we propose an improvement on word2vec weighting in relation to the tiling space model. Word2vec-Tiling considers both the term frequency and inverse document frequency; in this method, if the term frequency is high, and the term only appears in a little part of the documents, then the term has a very good differential ability. Although this approach emphasizes the ability to differentiate different classes more, it ignores the fact that a term that frequently appears in documents belonging to the same class can better represent the characteristic of that class. Therefore, we introduce a new method (Tiling) to represent the in-class characteristic, following which we conducted some experiments to compare the effects. The result shows that this improvement yields better accuracy. Finally, we used Reinforcement Learning (RL) or Q-learning algorithm by improving the equation to find out the semantic relation between these unstructured data words in different documents and the analysis of the information collected through the Word2vec-Tiling algorithm. The RL or Q-learning is known to learn unrealistically high action values because it includes a maximization step over estimated auction values that tends to prefer overestimated to underestimated values.

Figure 1.1. The search engine's view of the web. As the crawling process takes time and the web is very dynamic, the search engine's view of the web represents the state of web pages at different times. This is similar to watching the sky at night, as the stars we see never existed simultaneously as we see them.

1.2. Research Purpose

By this time, we describe the relationship between the crawler and the web to their workmanship in figure 1.1. Now we will describe why we are interested in research on web crawler and their data tool. Our main aim is to deliver accurately the result of searching data of users at very less time. For this reason, the thesis we introduced a new method of crawler search. In order to reach this goal, we first emphasized on collecting data from the web. According to the report of previous web research, Google crawler uses word2vec to collect its data from the web [1], [22]. For this reason, we have focused on collecting data from the web at the beginning of our research. So we proposed Word2vec-Tiling in the proposed method. Using this recommended method, the user will be able to collect information very quickly and accurately for his search. Finally, we used the TF, RL algorithm to analyze the performance of the tiling method. The ability to work in the efficiently way of our proposed algorithm we tried to prove through RL and TF algorithms.

1.3. Scope and organization of this thesis

This thesis focuses on Web crawler in Tiling process, and we research Web Crawling at many different levels. Our starting point is a crawling model, and in this framework, we develop algorithms for a web crawler data collection. Our aim at designing an efficient web crawling architecture and developing a scheduling policy to download pages from the web that is able to download the most “valuable” pages early during a crawling process.

The topics covered in this thesis are shown in figure 1.2. The topics are entangled, such as, there are several relationships that make the development non-linear. The crawler implementation is required for web characterization, but a good crawler design needs to consider the characteristics of the collection. Also, the crawler architecture is required for implementing the scheduling algorithms, but the result of the scheduling experiments drives the design of the crawler’s architecture.

We try to linearize this research process to present it in terms of chapters, but this is not the way the actual research was carried out: it was much more cyclic and iterative than the way it is presented here.

The following is an outline of the contents of this thesis. The first chapters explore theoretical aspects of Web Crawling:

Chapter 2, reviews selected publications related to the topics covered in this thesis, including web search, unstructured data analysis, and web crawler background. The next chapters are organized into two parts: one theoretical and one practical.

Chapter 3, introduces a new model for web crawling (Tiling), and a novel design for a web crawler that integrates it with the other parts of a search engine which is illustrated by pseudocode. Several issues of the typical crawling architectures are discussed and the architecture of the crawler is presented as a solution to some of those problems.

Chapter 4, details implementation issues related to the design and to algorithms presented in the previous chapters, including the data structures and key algorithms used.

Finally, **Chapter 5,** summarizes our contributions and provides guidelines for future work in this area.

We have described a new method for completing data from the web in thesis is research, which means that we can easily implement keyword related data from the web. We have also included in appendix-A a list of practical issues of web crawling that were detected only after carrying several large crawls. We propose solutions for each problem to help other crawler designers.

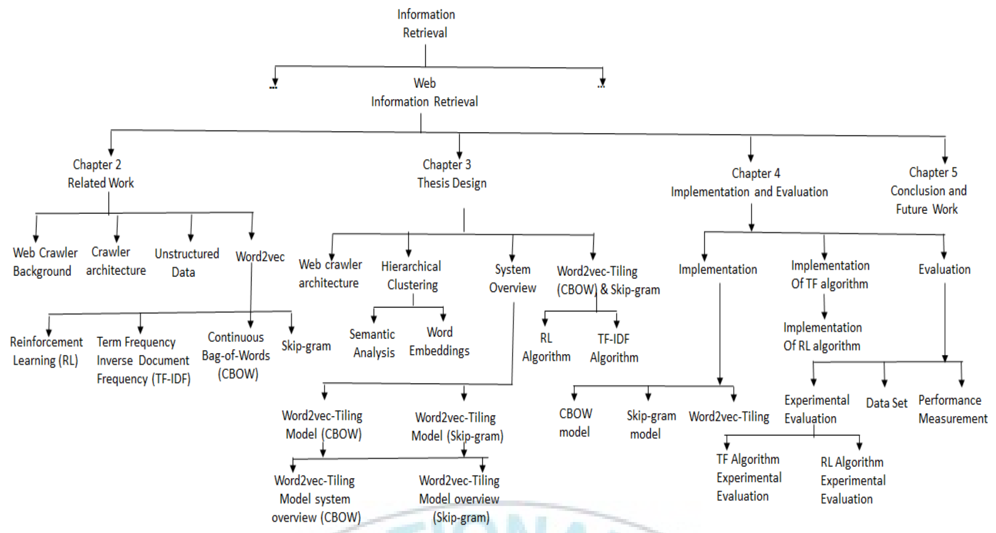

Information Retrieval
Web Information Retrieval
Chapter 2 Related Work
Chapter 3 Thesis Design
Chapter 4 Implementation and Evaluation
Chapter 5 Conclusion and Future Work
Web Crawler Background
Crawler architecture
Unstructured Data
Word2vec
Reinforcement Learning (RL)
Term Frequency Inverse Document Frequency (TF-IDF)
Continuous Bag-of-Words (CBOW)
Skip-gram
Web crawler architecture
Hierarchical Clustering
Semantic Analysis
Word Embeddings
System Overview
Word2vec-Tiling (CBOW) & Skip-gram
RL Algorithm
TF-IDF Algorithm
Word2vec-Tiling Model (CBOW)
Word2vec-Tiling Model (Skip-gram)
Word2vec-Tiling Model system overview (CBOW)
Word2vec-Tiling Model overview (Skip-gram)
CBOW model
Skip-gram model
Word2vec-Tiling
Implementation
Implementation Of TF algorithm
Implementation Of RL algorithm
Evaluation
Experimental Evaluation
Data Set
Performance Measurement
TF Algorithm Experimental Evaluation
RL Algorithm Experimental Evaluation

Figure 1.2 Topics covered in this thesis. Main topics covered in this thesis. Web crawling is important in the context of Web information retrieval because it is required for both Web search and Web characterization

Chapter 2

Related Work

2.1 Web Crawler Background

The first Internet "search engine", a tool called "Archie" (shortened from "Archives"), was developed in 1990. It downloaded the directory listings from specified public anonymous FTP (File Transfer Protocol) sites onto local files around once a month [9]. At that time, "Gopher", which indexed plain text documents, was created. The introduction of the World Wide Web in 1991 had numerous of these Gopher sites changed to web sites that were properly linked by HTML links [1]. "Jughead" and "Veronica" programs are useful for exploring the aforementioned Gopher indexes. In 1992 'Tim Berners-Lee' sets up the Virtual library of the web (also known as VLib), which plays an important role in maintaining relevant socialist links. In the year 1993, the "World Wide Web Wanderer", the first crawler, was formed. Although this crawler was initially used to measure the size of the Web, it was later used to retrieve URL’s that were then stored in a database called "Wandex" the first web search engine. Another early search engine, "Aliweb" (Archie-like Indexing in the Web) allowed users to submit the URL of the manually constructed index of their site. The index contained a list of URL’s and a list of user-written keywords and descriptions. In the year 1994, the first "full text" crawler and search engine, "WebCrawler", was launched. web crawler permitted users to explore the web content of documents, rather than the keyword’s and descriptors written by the web administrators, reducing the possibility of confusing results and allowing better search capabilities. From 1994 to 1997, commercial search engines began to appear. In the early 2000s, researchers on crawlers made some changes to their research [10]. They attempted to improve the quality of web search engines. For this reason, they split the web search engine through "Hypertext" and "Hyperlink" [11]. More changes on web crawler were achieved in 2010. The crawler or web search engine's "hypertext" and "hyperlink" are described as "data web mining" and "focus web". The crawler or web search engine's "hypertext" and "hyperlink" are divided by "data mining" and "focus web", where “deep web crawler” and “web crawler hidden data” are described.

2.2 Web search and Web crawling

The typical design of search engines is a “cascade”, in which a web crawler creates a collection which is indexed and searched [20]. Most of the designs of search engines consider the web crawler as just a first stage in web search, with little feedback from the ranking algorithms to the crawling process. This is a cascade model, in which operations are executed in strict order: first crawling, then indexing, and then searching.

Our approach is to provide the crawler with access to all the information about the collection to guide the crawling process effectively. This can be taken one step further, as there are tools available for dealing with all the possible interactions between the modules of a search engine, as shown in figure 2.1.

The indexing module can help the web crawler by providing information about the ranking of pages, so the crawler can be more selective and try to collect important pages first. The searching process, through log file analysis or other techniques, is a source of optimizations for the index, and can also help the crawler by determining

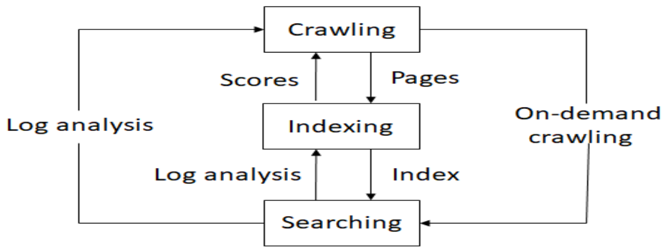


Figure 2.1 Cyclic architecture for search engines, showing how different components can use the information generated by the other components. The typical cascade model is depicted with thick arrows.

the “active set” of pages which are actually seen by users. Finally, the Web crawler could provide on-demand crawling services for search engines. All of these interactions are possible if we conceive the search engine as a whole from the very beginning.

2.3 Crawler Architecture

A general web crawler consists of webpage fetcher (downloader) for retrieving webpage contents, URL’s queue (frontier) for storing unvisited URL’s, and webpage processor for extracting text and URL’s out of a webpage’s HTML. Crawlers model the World Wide Web (WWW) as a graph G (V, E) where nodes (V) are webpage’s and edges (E) are links between webpage’s [15]. So, two webpage’s (nodes) will have an edge between them if one webpage has a link pointing to the other webpage. Similar to general Web crawling, a focused crawler has a webpage fetcher, URL’s queue, and webpage processor. In addition, a focused crawler has a topic or domain specific model, and a module for estimating the relevance of URL’s and webpage’s. Typically, a focused crawler takes as input: (1) the desired number of pages to collect, and (2) seed URL’s to start crawling from. It outputs the set of webpage’s found.
One of the important aspects of (focused) crawlers is the ordering of the URL’s in the queue, which specifies the order of visiting the nodes of the graph. In the focused crawler literature as example, the best-first search is the most commonly used technique and is considered the state-of-the-art focused crawler, taking into consideration the estimated relevance of the URL’s/webpage’s during crawling.
In Web like an Internet focused crawler starts from a seed URL. It downloads the corresponding webpage and extracts the text of that webpage. The focused crawler then estimates the relevance of the webpage textual content with regard to the topic/event of interest. In the next step, there are two design options. One option is that the focused crawler decides whether the webpage is relevant or not by comparing its estimated score to a pre-defined threshold. If the webpage is considered relevant, then the focused crawler extracts the embedded URL’s from the webpage and inserts them into the queue. The other option is that the focused crawler extracts all

embedded URL’s from the webpage and then inserts those into the queue, not being constrained by the webpage score. The second option takes into consideration the tunneling phenomena in crawling, where a non-relevant webpage links to relevant webpage’s, either directly or through several steps.

When inserting the extracted URL’s into the queue, the focused crawler has to make another decision. One option is to insert all extracted URL’s, along with the estimated score of the webpage from which they were extracted. Another option is to estimate the relevance of each URL based on the tokens in both the URL’s address and anchor text, and insert the URL and its resulting estimated relevance score into the correct position in the priority queue. We adopt a hybrid approach where we use the average of a URL’s score and the score of the parent webpage from which the URL was extracted.

Next, the focused crawler pulls from its queue the URL with highest score, and repeats the process. In pseudocode shows a focused crawler algorithm that handles tunneling (such as, extracts the URL’s from the webpage regardless of score) estimating the score of each URL and inserting it into the queue with its estimated score. We consider this approach as the foundation for the baseline for evaluation comparisons.

```
START
Input: Seed URLs, pagesLimit, PageScore, urlScoreThreshold
Insert seed URLs in priority queue
# Topic Representation
topicVector = Build topic representation from seed pages
# Crawling
While pagesCount < pagesLimit and priorityQueue is not empty:
  URL = pop (priorityQueue)
  append URL to visited list
  page = download (URL)
  # Preprocessing (Vector Space Model)
  page_Score = calculate_Score(page_Vector, topic_Vector)
  page_Count += 1
  if (page_Score >= page_Score_Threshold):
    page.get_URLs()
    relevant_Pages_Count += 1
    save page to event-related collection
    for link in page.out_going URLs:
      URL = link.address
      validate(URL)
      if URL not in visited list and URL not in priority_Queue:
        # Preprocessing (Vector Space Model)
        url_Vector = process(URL text)
        # Relevance Estimation (Cosine Similarity)
        url_Score = calculate_Score(url_Vector, topic_Vector)
        if url_Score >= url_Score_Threshold:
          push (URL, priority_Queue)
END
```

2.4 Unstructured Data

Unstructured data, [20] in contrast, refers to data that does not fit neatly into the traditional row and column structure of relational databases. In web pages, it often includes text and multimedia content. Examples of text files like data processor,

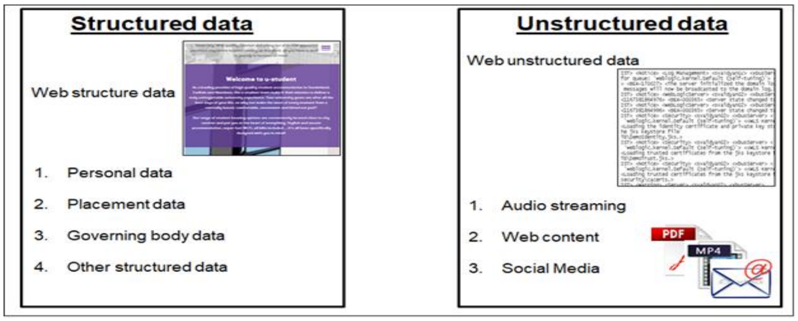


Figure 2.2 Type of data and its application area

spreadsheets, presentations, webpage's and many other kinds of business documents. While these sorts of files may have an internal structure, they are still considered unstructured because the data they contain does not fit neatly in a relational database. The learning resources and social networking sites are examples of unstructured data [15]. The video and audio streaming of classroom is also unstructured data figure A.2. Our thesis research focuses on learning resources and web pages multimedia data for unstructured data. The education system generates, maintains and analyzes large amount of data through various sources. This data is related to academic, non-academic, learning, examination, admiration, training and placement. The nature of such data as shown in figure 2.2 is varied in nature as given.

2.5 Word2vec

The word2vec model was proposed by Google, and its applications have recently attracted a great deal of attention from the machine learning community [21]. It is an algorithm for learning embedding's using a neural language model. Its input is a text corpus, while its output is a set of vectors. It is a neural network language model that can process text data and generate word vectors. The dense vector representations of words learned by word2vec can capture semantic relations, such as, the logical relationships between words; words that we know to be synonyms are more likely to have very similar vectors, while antonyms tend to have dissimilar vectors. Besides, these vectors adhere surprisingly well to our intuition and obey the laws of analogy.

Google introduced two main learning algorithms under word2vec: Continuous bag-of-words (CBOW) and Skip-gram (SG) model [22]. The CBOW model uses the context to predict a target word, while the Skip-gram model uses a word to predict a target context. Both algorithms learn the representation of a word that is useful for the prediction of other words in the sentence. In this thesis, we have researched the Skip-gram model of word2vec.

More formally, given a large training corpus represented as a sequence of words w_1,

$w_2 \ldots w_n$ the objective of the CBOW model is to maximize the log probability.

$$\sum_{n=1}^{n} \quad \sum_{c \in C_n} \log P\,(w_p|w_k) \tag{1}$$

Where n, is the web document and the context C_n, is the set of indices of words surrounding keyword w_k. The probability of observing a context page, w_p, given w_k, will be parameterized using the aforementioned word vectors. For now, let us assume that we are given a total number of web pages |P| that maps pairs of (word, context) to scores in k_i. One possible choice for defining the probability of a context word is the softmax.

$$\mathrm{P}\left(w_p|w_k\right) = \frac{\exp(|P|\left(w_p|w_k\right))}{\sum_{i=1}^{W} \exp(|P|(w_p,i))} \tag{2}$$

However, such a model is not adapted at this instance as it implies that given a, word w_k, we can only predict one context page, w_p.

The problem of predicting context words can instead be framed as a set of independent binary classification tasks. Then, the goal is to independently predict the presence of context words. For the word at position n, we consider all context words as positive examples, and sample negatives at random from the dictionary. For a chosen context position c, using the binary logistic loss, the negative log probability we obtained is subsequently discussed.

$$\log\left(1 + e^{-|P|(w_p|w_k)}\right) + \sum_{c \in C_n} \log(1 + e^{|P|(w_p,n)}) \tag{3}$$

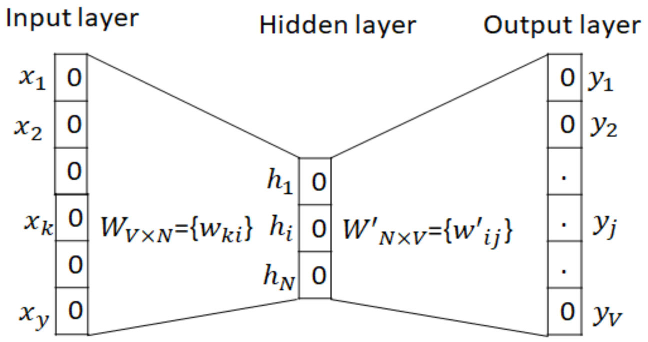


Figure 2.3 A simple CBOW model

2.5.1 Continuous Bag-of-Words Model (CBOW)

Consider a very simple version of CBOW model where only one context word c is given as input and the model has to predict the next word w. In figure 2.3 shows the simplified CBOW model where V, is the size of vocabulary and N is the size of the hidden layer and the dimension of the word embedding produced by this model [22]. The size of the output layer is same as the input layer. The size of output y, is same as the cardinality of the vocabulary. The value at each index of the output vector represent the propability that next word is the word at index i, of the vocabulary and so on. Thus

$$y_i = P(w|c) \quad (4)$$

$$\sum_v y_i = 1 \quad (5)$$

The input to this network x_v, is one hot vector representation of the context word. Thus $x_k = 1$ and $x_k = 0$ for $k' \neq k$

There are two sets of weight matric one between input layer and hidden layer and is denoted by W, and the other weight matrix between the hidden layer and output layer denoted by W'. The size of the matrix W is V×N and of matrix W' is N×V.

$$h = W^T x \quad (6)$$

Since x, is a sparse vector with only one dimension on at the kth position so the h, is k row of W. Let w be the kth , row of W_k, h is the represented as

$h = W^T$, $x = W_k = v_c$

Thus, activation function of the inner layer is a linear function. Suppose

$$y_i = \varphi\ (u_i) \quad (7)$$

and,

$$u = W^T h \quad (8)$$

$$u_i = W_i\ h \quad (9)$$

Let us denote W_i as V_ω therefore

$$u_i = V_w\ h \quad (10)$$

From equation $h = W^T x = W_k = v_c$ therefore

$$u_i = v_w,\ v_c \quad (11)$$

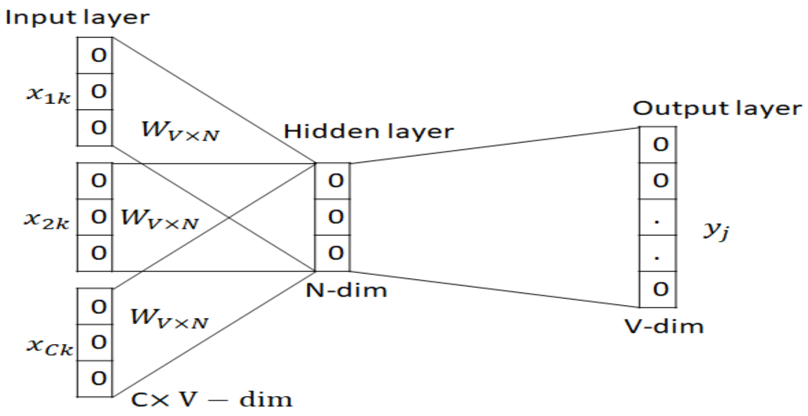


Figure 2.4 A CBOW with C words (C > 1) in the context

y_i is the non liner function φ, applied to u_i. Since we want to interpret y_i, as a probability so we will use Word2vec-Tiling classification model to obtain the probability distribution of the worlds.

$$y_i = \frac{e^{ui}}{\sum_i e^{ui}} \tag{12}$$

From equation (11)

$$y_i = \frac{e^{v_w v_c}}{\sum_{w\prime} e^{v_w v_c}} \tag{13}$$

Where W∈ all the word in vocabulary V. v_w and v_c are the parameter of this model that we need to find in a way that the likelihood of the model is maximized. Let θ denote the parameters of the CBOW model such that θ = v_w, v_c. Let L(θ) denote the likelihood of the parameter.

$$L(\theta) = \prod_{w\in T} P(w|c) \tag{14}$$

On taking the log if the likelihood L(θ) denoted by l

$$l(\theta) = \sum_{w\in T} \log P(w|c) \tag{15}$$

From equation (13)

$$l(\theta) = \sum_{w\in T} \log \frac{e^{v_w v_c}}{\sum_{w\prime} e^{v_w v_c}}$$

$$l(\theta) = \sum_{w\in T} v_w, v_c - log \sum_{w\prime} e^{v_w v_c} \tag{16}$$

On taking the derivative of log likelihood l(θ)

$$\frac{\delta l(\theta)}{\delta v_w} = v_c - \frac{1}{\sum_{w\prime} e^{v_w, v_c}} e^{v_w, v_c} v_c$$

$$\frac{\delta l(\theta)}{\delta v_w} = v_c - P(w|c)v_c$$

$$\frac{\delta l(\theta)}{\delta v_w} = v_c\big(1 - P(w|c)\big) \tag{17}$$

Using stochastic gradient descent, we can obtain the weight updating for hidden → output layer,

$$v_w = v_w - \mu\, v_c\big(1 - P(w|c)\big) \tag{18}$$

Similarity, on taking derivative of log likelihood $l(\theta)$ also v_c the weight updating equation for input → hidden layer is estimated by

$$v_c = v_c - \mu\, v_w\big(1 - P(w|c)\big) \tag{19}$$

Multi-word context

In this case the input is not the single word context but multiple words $x_1, x_2, x_3 \ldots x_c$

The hidden layer output h, is given by

$$\text{h} = \frac{W^T(x_1 + x_2 + \cdots + x_c)}{C}$$

$$\text{h} = \frac{v_{w_1} + v_{w_2} + \cdots + v_{w_c}}{C}$$

C is the context window size. Let $\mathrm{L}(\theta)$ denotes the likelihood of the parameter θ.

$$\mathrm{L}(\theta) = \prod_{w\in T} P(w|c_1, c_2 \ldots c_c) \tag{20}$$

On taking the log of the likelihood denoted by $\mathrm{l}(\theta)$

$$\mathrm{L}(\theta) = \prod_{w\in T} P(w|c_1, c_2 \ldots c_c)$$

$$\mathrm{l}(\theta) = \sum_{w\in T} log \frac{e^{v_w h}}{\sum_w e^{v_w h}}$$

$$\mathrm{l}(\theta) = \sum_{w\in T} v_w h - \sum_w \log e^{v_w, h} \tag{21}$$

Using stochastic gradient descent, we can obtain the weight updating equation for hidden → output layer.

$$v_c = v_c - \tfrac{1}{C}\mu v_w\big(1 - P(w|c_1, c_2 \ldots c_c)\big) \tag{22}$$

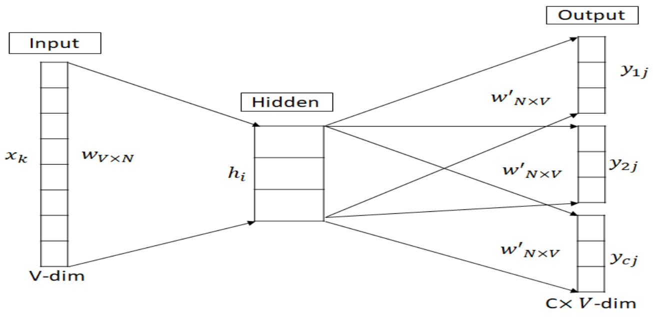


Figure 2.5 The Skip-gram model

2.5.2 Skip-gram Model

In this model the aim is to predict the neighboring word given the central word as the input [21]. The input is one hot vector representation of the central word and is represent by x.

The input vector for the word x, is denoted by v_c. Similar to the single word context model of CBOW, there are two weight matrices denoted by W and W'.

The output of hidden layer is given by

$$h = W^T x = W_k = v_c$$

Score vector for the output word is generated using

$$u_{cj} = u_j = W^T h = v_{wj} h = v_{wj} v_c$$

For c =1,2,3…C

v_{wj} is the output vector of jth word in the vocabulary and it is also the jth column of the output matrix W'. Using this function, each of these score is converted to a corresponding probability.

The objective function of this model is that probability vector generated to match the actual probability which is $y^{(i-c)}$, $y^{(i-1)}$, $y^{(i+1}$…$y^{(i+c)}$ the one hot vector of actual output. This probability is estimated by the formula below

$$P(W_{t+i}, W) = y_{c,j} = \frac{\exp(V_{w_{t+i}}{}^T v_w)}{\sum_V^{V'} \exp(V_{w_{t+i}}{}^T v_w)} \tag{23}$$

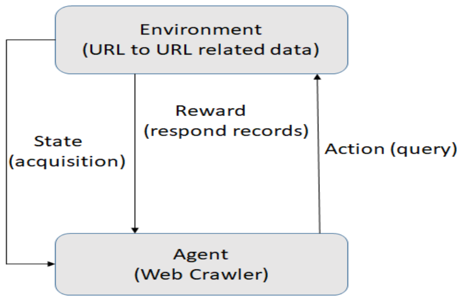


Figure 2.6 Overview of the reinforcement learning frame work

We need to define an objective function to evaluate the skip-gram model. Using the Gupta Saurabh assumption, given the central word, all the neighboring words are completely independent [22].

2.5.2.1 Reinforcement Learning Framework

In this chapter, we propose a formal framework for the deep web crawling based on RL and formalize the crawling problem under the framework. First of all we give an overview of the RL framework. The relation between a crawler and a deep web database is illustrated in Figure 2.6.

From the figure, one can conclude that at any given step, an agent (crawler) perceives its state and selects an action (query). The environment responds by giving the agent some (possibly zero) reward (new records) and changing the agent into the successor state [20]. In the total procedure of RL-algorithm in the proposed method is described by the figure 2.7.

1. Link Queue: Current set of links that have to be visited. Fetch link with highest score on queue.
2. Evaluate page this link points to: Based on set of text/content attributes. If relevant, store on Good Pages.
3. Get links from page
4. Evaluate links, add to link queue.
5. In the RL link scorer evaluate the links.

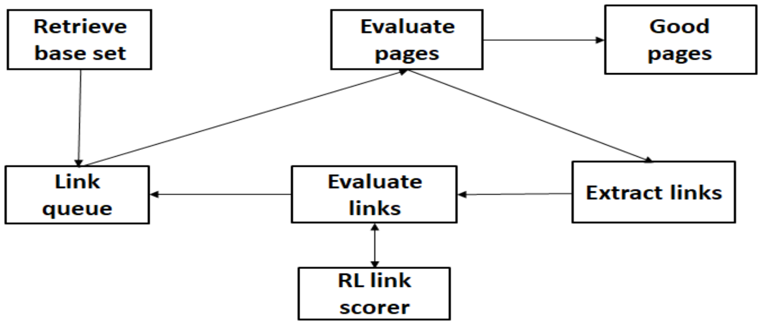


Figure 2.7 Reinforcement Learning working Overview

2.5.2.2 Reinforcement Learning Overview

Reinforcement Learning or Q-learning can acquire optimal control policies from pending rewards, even when the agent has no former knowledge of the environment [13]. For a discrete case, a Q-learning algorithm assumes that the initial state set s, and action set a, can be divided into separated values. At the next step n, the agent observes the state s_n, and then chooses an action a_n. After executing the action in time t, the agent receives a reward r, which reflects how good the action is (in a short-term sense). The state will change into the next state s_{n+1}, under action a, in time t. Then, the agent will choose the next action, a_{n+1}, according to the established knowledge.
Conceptually, Q-learning works by learning an action-value function that gives the desired utilization of taking , given action in a, given state. Q-learning is defined thus:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r+\gamma.\arg.max_a(Q(s_{n+1},a_n) - Q(s,a))] \qquad (24)$$

Where r is the reward observed, the parameter, a, controls the learning rate (0 < a < 1) and ɣ is a factor discounting the rewards obtained so far (0 < ɣ < 1).
Given the learned Q-values, when reaching a state s, Q-learning algorithms typically select the greedy action, a, which has the highest Q(s, a) value with a probability 1−∈. The algorithms will then select a random action with a probability ∈ (with 0 < ∈ < 1). Typically, ∈ is set to be very small (such as, close to 0); thus, most of the time, the greedy action is selected.

This method of choosing between the greedy action and the random action is called "∈-greedy", and it enables Q-learning to escape local maxima. After a potentially large number of iterations, Q-values are learned such that when the greedy action is chosen every time, the return at time t is maximized; the return is defined as the summation of the rewards obtained after time t, until the episode ends.

Reinforcement Learning or Q-Learning is learning method, which renews each synapse weights according to output signals. The learning rule can be applied to feed forward networks and recurrent networks. The flow of measurement of the renewal of the syndrome is shown by the pseudocode as follows.

```
1. Initialize state ($s_n$, $a_n$)
2. calculate the reward of action an
3. **for** each document $d_i \in r(s_{n-1}, a_n)$
4.     **for** each keyword k in $d_i$ do
5.         **if** action $a(k) \neq A$ then $A = A \cup a(k)$
6.             **else** then restart Reinforcement-learning (RL) of action a(K)
7.     **end for**
8. **end for**
9. change the current state to $s_{n+1}$
10. $P_r = P_r \cup [a_n]$ restart candidate set S; $S = S / [a_n]$
11.         **for** each $a_i \in S$ update its reward $r_t$
12.             **for** each $a_i \in S$ update its each value $\gamma$
13.                 **for** each $a_i \in S$ calculated its RL-value
14. **return** arg.max$_a$ Q(s', a)
$Q(s,a) \leftarrow Q(s,a) + \alpha[r+\gamma. \arg. max_a(Q(s_{n+1}, a_n) - Q(s, a))]$
15. **End**
```

Specifically, the surfacing algorithm first calculates the reward of the last executed action and then updates the action set through step 3 to Step 8, which motive the agent to transit from its state 's_n' to the successor state 's_{n+1}'. Then the training and candidate set are restart in accord with the new action set in step 10. After that, the algorithm catalog the reward and RL-value for each action in candidate set in step 11, step 12 and step 13 respectively. The action that maximizes RL-value will be returned as the next to be executed action.

2.5.2.3 An overview of TF-IDF

We will first introduce the mathematical background of the algorithm and also describe the TF-IDF algorithm working process.

2.5.2.4 Mathematical Framework

Basically, TF-IDF works by determining the relative frequency of words in a specific document compared to the inverse ratio of that word over the entire document corpus [20]. Intuitively, this calculation determines how relevant a given word is in a special document. Words that are common in a single or a small group of documents tend to have higher TF-IDF numbers than common words such as articles and prepositions [25]. The formal procedure for implementing TF-IDF has some minor differences over all its applications, but the overall approach works as follows in table 2.1. Their research first defines a topic keyword dictionary, at the same time the web document is represented by x heft words, and then use TF-IDF method to calculate keywords (K), webpage, (P) and Term Frequency (TF) as shown in equation (25).

$$\text{TF(P, K)} = \text{fre (P,K)} / \sum_{K_i \in K} fre\ (P,\ K_i) \tag{25}$$

Here, P, K indicate the number of things of the keyword K, in the web document P. As shown in equation (26) Inverse Document Frequency (IDF).

$$\text{IDF(P, K)} = |\text{P}| / \sum_{P_i \in P} dfre\ (P_i, K) \tag{26}$$

For the keyword, K with the webpages, P. Their interrelation coefficients are calculated as shown in equation (27).

$$\text{T (P,K)} = \text{TF(P,K)} \times \text{IDF(P,K)} \tag{27}$$

TF-IDF is the multiple of the value of TF and IDF for a particular word. The value of TF-IDF increases with the number of occurrences within a document and with rarity of the term across the corps. Finally T has been used to establish a connection between TF and IDF [6], [25].

Table 2.1 TF-IDF algorithm tools

Methods	**Denoted by**
Term Frequency	TF
Web document	x
Key words	K
Web page	P
Number of keyword document	K_i
Number of webpage document	P_i
Their interrelation coefficients	T
Total number of web page	$\lvert P \rvert$
Total number of Keyword	$\lvert K \rvert$
1st equation	$TF(P, K) = fre(P,K) / \sum_{K_i \in K} fre(P, K_i)$
2nd equation	$IDF(P, K) = \lvert P \rvert / \sum_{P_i \in P} dfre(P_i, K)$
3rd equation	$T(P,K) = TF(P,K) \times IDF(P,K)$

Chapter 3

Design of Web Crawler

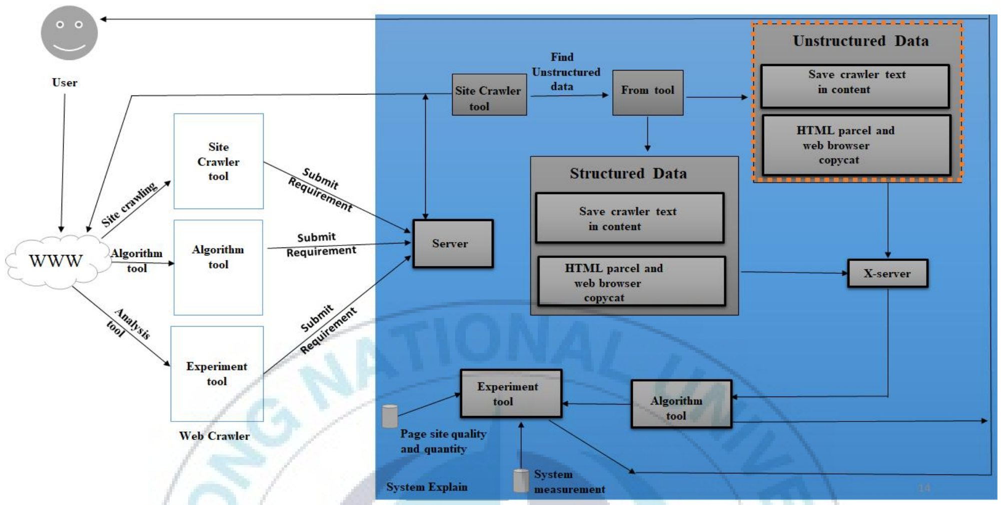


Figure 3.1 Web Crawler Architecture

3.1 Web Crawler Architecture

Web data acquisition is the foundation of web data mining. The web crawler is an important tool for web data acquisition. A web crawler is a program that is an essential web search engine to find the URL to the web page then save and download it. Web crawlers are an important component of web search engine, where they are used to collect the corpus of web pages index by the search engine as shown in figure 3.1 [4].

The web crawler's work methods are described [20]. The web page has different types of data. It is not possible to separate, when we are collecting this data, such as structured data or unstructured data in figure 2.2. Therefore, we created the architecture to separate these data. According to this architecture, we collect data from the web page and it is divided into structured and unstructured ways. Actually structure data is a data which is stored in row and column format using traditional database management system. It follows all relational database management system properties like ACID, Normalization and etc. examples are Oracle, MySQL, ms-access, and so on. On the other hand, unstructured data is a data which is not stored in a table format. Which has no structure. It does not have a specific pattern like column or row. It includes audio, video, pdf file, webpage's url etc. Therefore, we preferred web pages unstructured data. Firstly, we have chosen a web page, after which we have separated all the unstructured data associated with that web page, which we saved on our server. The biggest advantage of this architecture is that the collected data will not be lost, all data can be stored separately on the server. In this step of the algorithm tool, we will design how to use the algorithm. Finally, the data reaches the experimental tool. The experimental tool uses the metrics computation tool's output to show how a given design differ from highly rated designs with a similar purpose. It uses several statistical models which derived from an analysis, that were rated according to their quality and usability and finally, it sends the result to the user.

3.2 Hierarchical Clustering

The imperialist competitive algorithm is widely used to solve different problems [5]. In this research, we use Word2vec-Tiling to identify the optimal topic boundaries of text segmentation in a document. At first basic preprocessing applied to the text, punctuation and the not useful words are eliminated by using a list of stop words specific to the Python language. Then we have used clustering for text segmentation. The hierarchical text segmentation produces a tree that can be used as a visual illustration of the underlying hierarchical structure of a document. The benefit of this clustering tree is that it represents different levels of granularity of the document, which in turn means that the document can be segmented into different segmentation levels. This is a powerful criterion in the hierarchical representation of text. In contrast to a linear representation, in each level of the structure (tree) segmentation with different levels of details could be obtained and can be usefully applied to many other tasks needs show in figure 3.2. After which each word is represented as a term

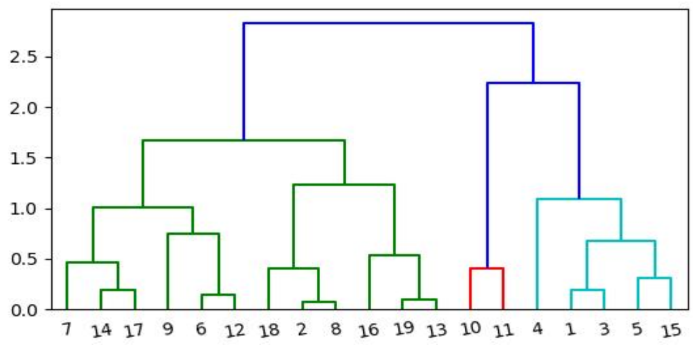


Figure 3.2 Sentence Clustering

frequency vector [6]. Then the similarity between a middle pair of sentences is computed using cosine similarity, this is applied to all sentence middle pairs to generate a similarity matrix. Finally, the optimal boundaries are created by Word2vec-Tiling according to the sentence similarity matrix. In order to improve the performance of our algorithm, we used the optimal boundaries which are found by the word2vec-Tiling (CBOW and SG) algorithm.

3.2.1 Semantic Analysis

Semantic analysis of the text is the key issue for relations extraction form unstructured text. In our approach to relationship extraction we used simple semantic rule such that a sentence describing the subject entity and object entity, and predicate in between them act as a candidate relation. Linguistic output is input to semantic analysis process that executes defined Tiling semantic rule and extract set of entities and semantic relations between them. For example, the rule to extract semantic relation is pseudocode formulated as follows:

```
Phase: Priority
Input: Document Token
Options: Control = applet
Rule: Prior_verbs(
{Token.category == CBOW}   {Token.category == CBOW}
{Token.category == SG}  {Token.category == SSG}
{Token.category == HC}  {Token.category == HC}
)
:priori →
{
  gate.Annotation set priori = (gate.Annotation set)bindings.get("priori")
  gate.Fearure map features = Factory.new feature map()
  feature.put("rule", "priori_varbs")
  output PREDICT.add(predi.firstNode(), predi.lastNode(); "Predicate", feature)
}
```

3.2.2 Word Embeddings

Word embedding's is a preferred technique for transforming text into a vector in text classification, document clustering, speech tagging, entity identification, emotion analysis, and many other natural language processing tasks. These distributed representations aim to transform the words into vectors in order to relate the similarity between the vectors to the semantic similarity between the words. Word2vec-Tiling are one of the best known word embedding's approaches.

In text classification tasks such as sentiment analysis, word embedding's created with word2vec are used to improve performance and obtain more accurate classification results. The Word2vec-Tiling model shown in Figure 3.3 has two architectures: Continuous Bag of Words (CBOW) shown in Figure 3.3(B) and Skip-gram (SG) shown in Figure 3.3(A). Although the CBOW model is reported as several times faster to train in the literature, it is observed that the skip-gram model works better in predicting non-frequent words. So, we prefer the skip-gram model for obtaining pre-trained word embedding's research.

The skip-gram model tries to predict the words around the center of a given word, that is, context words. Figure 3.3(A) shows a classifier that takes a word vector and a context vector as inputs. The classifier in Figure 3.3(A) is first trained to learn and

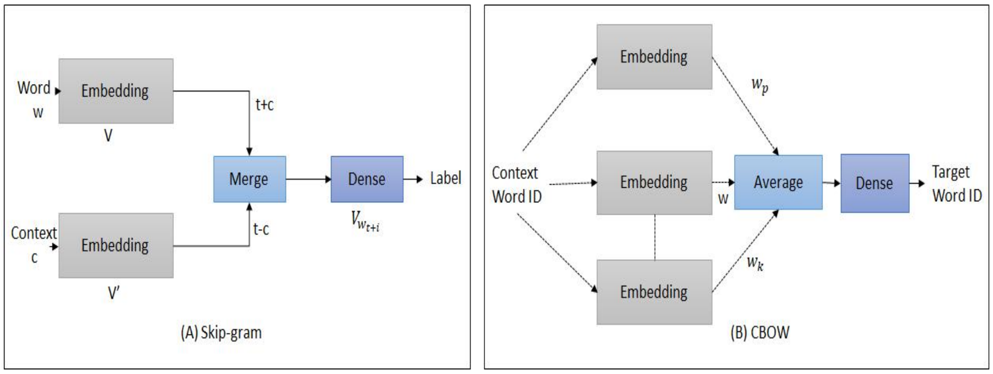


Figure 3.3 Skip-gram and CBOW Word2vec-Tiling models Overview

then to predict the actual label of a review, which is 0, if the sentiment polarity is negative and 1 if the polarity is positive.

In the skip-gram word2vec model, each word $w \in w'$ is represented by a word vector embedding $V \in V'$ and similarity each context $C \in C_n$ is represented as a context vector, V_r, V_{wt}

where r and w_t is the vector size (embedding size). W is the word and C is the contextual dictionary. Inner multiplication is performed between words and vector representations of contexts. The output of this trained network is the weight of the word layer. Because of the very low computational complexity, the skip-gram model can calculate very accurate high-dimensional word vectors from very large data sets.

3.3 System Overflow

Our proposed model is shown in Figure 3.4. Our total work process will start by collecting unstructured data from Google web pages. The process of our web crawler work is described below.

- Next step will be a list of keyword's related to the unstructured data.
- In the next step, we will extract the data of collected sentences through word2vec. We will use the Tiling method to select the data related to our keyword's. Which is defined as Word2vec-Tiling.

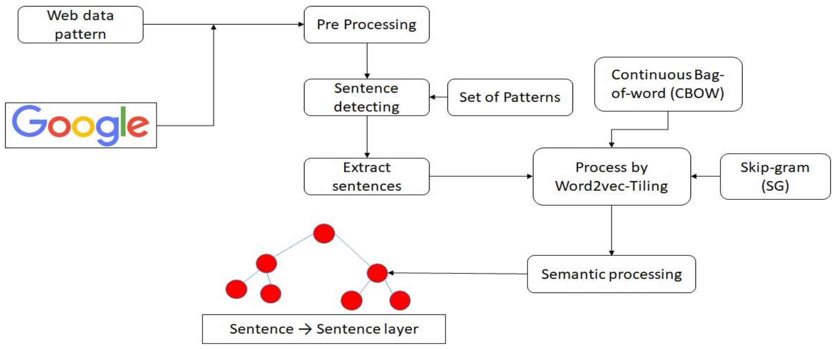


Figure 3.4 Overall working process

- After collecting the data using our Tiling method, we will process this data semantically.

In figure 3.4, we have described our overall working process. But since we have focused on collecting data for which word2vec has been selected. We know in word2vec divided by two part one is CBOW and another is Skip-gram. We have described the existing method CBOW and Skip-gram using process in chapter 2 and chapter 3 we described proposed method using process by pseudocode. In this chapter, we will design their work method.

3.3.1 Continuous Bag-of-Words (CBOW) Model

Word2vec-Tiling represents each word W_p in a in a full document or text of size T as a low dimensional dense vector j_w in an embedding space K_i. It attempts to learn the continuous word vector j_w from a training corpus such that the spatial distance between words then describes the similarity between words, the closer two words are in the embedding space, the more similar they are semantically and syntactically. Next, the sentence similarities are illustrated by considering a theoretical matrix gap between every pair of sentence similarity matrix. The sentence similarity matrix is constructed by computing all pair wise sentence similarities. The flow of the measurement of the Word2vec-Tiling algorithm working process is shown in figure 4.4, also describe document working method in figure 4.1 and 4.2 in implementation chapter, and described by the pseudocode as follows.

Table 3.1 Algorithm of CBOW model

The CBOW algorithm attempts to partition a finite collection of elements w_p and w_k into a collection of dist < d_{min} Tiling clusters with respect to some given criterion.

The following is a step description of the CBOW algorithm

Step 1: At the initial state select the number of clusters dist < d_{min} exponential dist $\leftarrow$distance (left, right) initial partition element left, right.

Step 2: Calculate dist < d_{min} the Tiling cluster center closer pair $\leftarrow$left, right and the terminal criterion $d_{min} \leftarrow$ distance.

Step 3: Calculate web text word [left] to web text word [right] then there is at least one more point in Word2vec-Tiling.

Step 4: Select next word dist $\leftarrow$distance [right] flowing dist < d_{min} that's meaning closer pear which is define by n, divided by 2 its meaning 2^n according to tiling rule.

Step 5: According to previous step the new condition there is at one more point in Tiling dist $\leftarrow$distance (next [Tiling], right)

Step 6: Calculate the new partition element right $\leftarrow$next [Tiling] left, which is closer pear and d_{min}

Step 7: Therefore the previous step satisfied then stop otherwise go to step 2.

In pseudocode, we show the main structure of our proposed CBOW model algorithm. The most inside condition evaluates the fitness value of each agent and updates information for the whole system step 1. Using its current solution, a keyword first finds out the best match from each document, calculates the frequency value, and updates w_p_keyword, w_p_doc if necessary (step 1-6). Step 2, is the main part of calculating document gap value. The document will be a gap with the gap jumps excluded from the previous and the next value. In that case, the previous value was assumed to be 2^n and by reducing it to 1, the gap value is determined (2^n -1) (step 3). Step 4, is the main part of calculating document word frequency value. From a random initial solution, each document continuously searches for better solutions in the neighborhood, taking information from its own experience (w_p_keyword), and the experience of all document (w_p_doc). The actual movement of each document is determined by the update value. Finally, as a Tiling algorithm, the final solution of Word2vec-Tiling depends on its starting solutions, the purpose of step 6 is, therefore, to restart the system several times, from independent random solutions, to ensure a high overall success rate. Step 1-6, have been used for the web page keyword and web document used to match compatibility.

In table 3.2, we show our original input text document and according in CBOW model proposed pseudocode, the text gape value calculation process describe in below table 3.2. In table, Tiling output result flow black value show the text original value and red value indicate text gape value calculation.

3.3.2 CBOW Model System Overview

Web crawlers are an essential component to search engines running a web crawler is a challenging task [5]. There are tricky performance and reliability issues and even more importantly, there are social issues. Crawling is the most fragile application since it involves interacting with hundreds of thousands of web servers and various name servers, which are all beyond the control of the system.

Table 3.2 Proposed CBOW working process

Input text	Tiling Output result
['william', 'shakespeare', 'was', 'born', 'in', 'stratford-upon-avon', 'in', '1564'],	['0.038329', '0.2771', '0.35388', '0.07023', '0.09793', '1.31594', '1.2116', '0.3786', '0.46297', '0.25444', '0.32705', '0.85704', '0.89301', '0.1166', '0.1592',],

Web crawling speed is governed not only by the speed of one's own Internet connection but also by the speed of the sites that are to be crawled. Especially if one is a crawling site from multiple servers, the total crawling time can be significantly reduced, if many downloads are done in parallel. Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same. The process of our CBOW model working process is described below.

> Parse through the downloaded page and restore all the links. For each link restore, repeat the process.

> Every object related data will be split through the Word2vec-Tiling process. This process is done in few steps.

- Data of each object will be divided by following the clustering method.
- Each data will be split up separately and each pair of data will separate the gap between pairs of the sentence.
- This trend will continue until it is completely sentence. Then go to next step.

> Verify the semantic relation between the words and texts in a particular document.

> If the semantic relation is able to detect then the work is success. If not, then introduce escape and avoidance response.

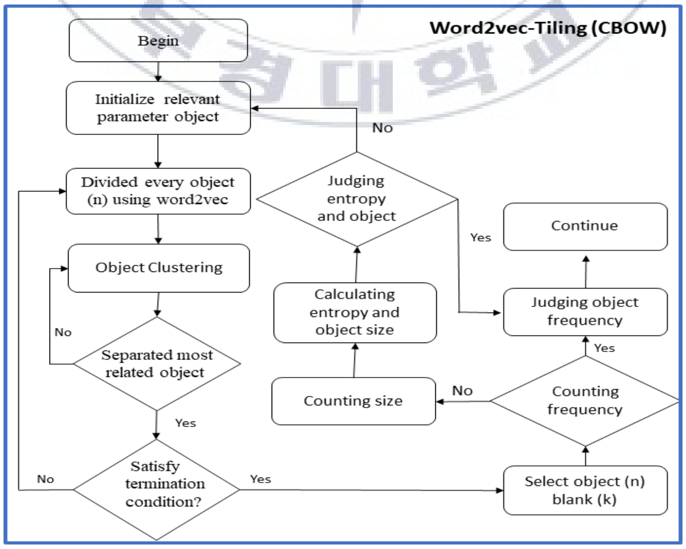

Word2vec-Tiling (CBOW)
Begin
Initialize relevant parameter object
No
Divided every object (n) using word2vec
Judging entropy and object
Yes
Continue
Object Clustering
Calculating entropy and object size
Judging object frequency
No
Yes
Separated most related object
Counting size
No
Counting frequency
Yes
No
Satisfy termination condition?
Yes
Select object (n) blank (k)

Figure 3.5 CBOW model system overview

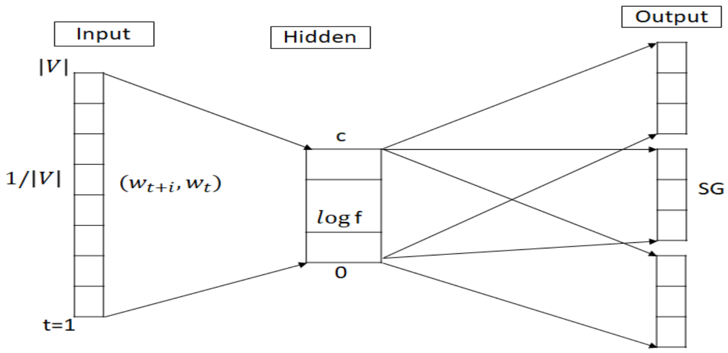


Figure 3.6 Proposed method Skip-gram working method

3.3.3 Skip-gram Model

The SG model is a popular choice to learn word embeddings by leveraging the relation between a word and its neighboring words. In detail, the SG model is to predict the context for each given word w_t and maximizes

$$SG = \frac{1}{|V|}\sum_{t=1}^{|V|}\sum_{0<|i|<c}\log f\ (w_{t+i}, w_t) \tag{28}$$

on a given corpus with vocabulary V, where w_{t+i} denotes the context words in a window w_{t-c}^{t+c} with c, denoting the window size. Here in $f(w_{t+i}, w_t)=P(w_{t+i}|w_t)$ and the probability to predict context word is estimated by

$$P(w_{t+i}|w_t) = \frac{\exp({V_{w_{t+i}}}^T v_w)}{\sum_{w_{t+i}\in V}\exp({V_{w_{t+i}}}^T v_w)} \tag{29}$$

Where V_{wt} is the embedding for w_t and V and V' refer to input and output embedding respectively. The training processing of SG model is thus to maximize SG over a corpus iteratively. For a large vocabulary Word2vec-Tiling uses hierarchical clustering to address the computational complexity that requires |V|×d matrix multiplications.

Table 3.3 Complexities of different SG models. The column of "Parameters" and "Operations" reports space and time complexity, respectively. And d: embeddings dimension, C: corpus size. 0: unit operation of predicting and updating one words embedding. n: the number of negative samples. The total Skip-gram model we described by pseudocode as flows.

Model	**Parameters**	**Operations**
SG	2|V|d	2cC(n+1)0
SSG	(2c+1) |V|d	$4c^2$C(n+1)0

3.3.4 Skip-Gram Model Overview

The Structured Skip-gram (SSG) model is an adaptation of SG model with consideration of words order. The overall likelihood of SSG model shares the same from of SG model as equation (29), however, with an adapted f(w_{t+i},w_t) where the probability of predicting w_{t+i} consider not only the word to word relations but also its relative position to w_t. In practice, each word in w_{t-c}^{t+c} is not predicted by a single predictor operating on the output embeddings $V_{w_{t+i}}$. Instead, w_{t+i} is predicted by 2c predictors according to where it appears in w_t context. As a result, every word is SSG should have 2c output embedding for the 2c relative positions. The probability of predicting w_{t+i} in SSG or SG is thus formulated as

$$P(w_{t+i}|w_t) = \frac{\exp(\sum_{r=-c}^{c} V_{r,\ w_{t+i}}{}^T v_{wt})}{\sum_{w_{t+i} \in V} \exp(\sum_{r=-c}^{c} V_{r,\ w_{t+i}}{}^T v_{wt})} \tag{30}$$

Where $V_{r,w_{t+i}}$ define the positional output embeddings for w_{t+i} at position r with respect to w_t. The embedding of w_t is thus updated with word order information implicitly recorded in $V_{r,w_{t+i}}$. Now we will describe the proposed method Skip-gram using method by pseudocode and flow diagram 3.7 as follows.

Table 3.4 Skip-gram model pseudocode

```
The Skip-gram algorithm attempts to partition a finite collection of elements N=
{$n_1 \ldots n_n$ } into a collection of c, SG clusters with respect to some given criterion.
Given a finite set of data, the algorithm returns a list of c cluster centers.
V ← $V_{w_{t+i}}$ Extract Vocabulary
N ← $N_c$ Number of Document
Where c∈C is the condition of Document $N_c/N$ that tells the degree of which the e
lement $text_c$←V

The following is a step description of the Skip-gram algorithm

Step 1: Parameters vocabulary V and document number N then count tokens of te
rm $T_{V_{wt}}$ initial partition matrix $N_c$

Step 2: Count tokens of term $T_{V_{wt}}$ of all document in class $N_c$ then the condition a
re
    $\frac{\exp(V_{w_{t+i}} T_{V_{wt}})}{\sum_{w_{t+i}} \exp(V_{w_{t+i}} T_{V_{wt}})}$
Step 3: Calculate new parameter $V_{r,w_{t+i}} T_{V_{wt}}$ by using condition of c∈-c where tex
t embedding $w_t$ also V and V' are input and output embedding for each word trun
cated [T]
    $[w_{t+i}], [w_t] = \frac{\exp(\sum_{r=-c}^{c} V_{r,\ w_{t+i}}{}^{T} V_{wt})}{\sum_{w_{t+i} \in V} \exp(\sum_{r=-c}^{c} V_{r,\ w_{t+i}}{}^{T} V_{wt})}$
Step 4: Update embedding parameter $V_{r,w_{t+i}}$ for all document in logarithm [c] co
ndition problem [t] and [c] then the probability equation are
    $[w_{t+i}], [w_t] = \frac{\exp(\sum_{r=-c}^{c} V_{r,\ w_{t+i}}{}^{T} V_{wt})}{\sum_{w_{t+i} \in V} \exp(\sum_{r=-c}^{c} V_{r,\ w_{t+i}}{}^{T} V_{wt})}$
Step 5: The Skip-gram embedding
    $SG = \frac{1}{|V|} \sum_{t=1}^{|V|} \sum_{0<|i|<c} \log f(w_{t+i} w_t)$
```

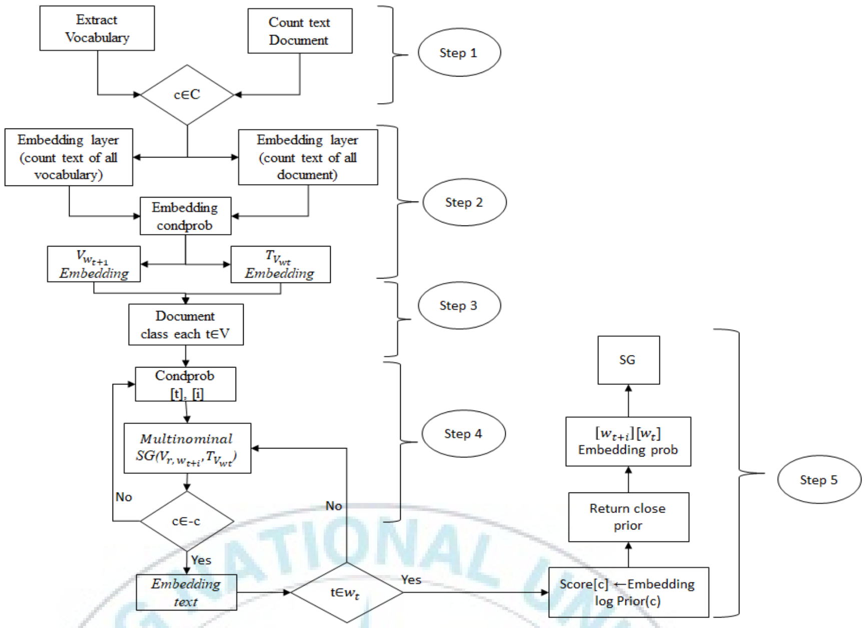
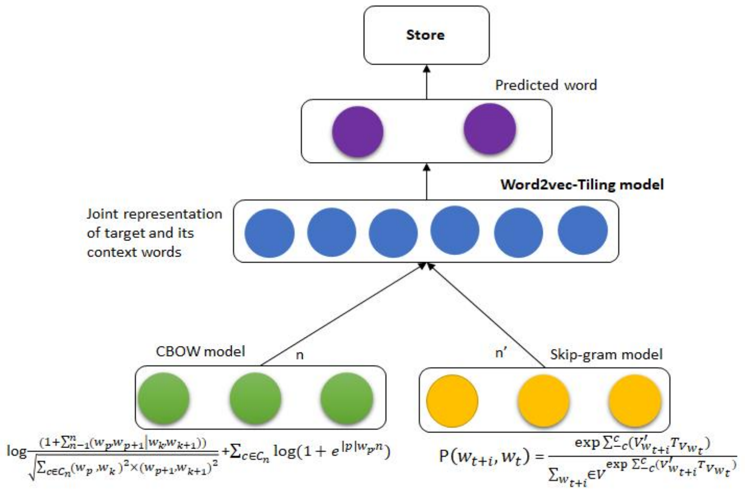


Figure 3.7 Proposed method Skip-gram working process

3.3.5 Optimizing Word2vec-Tiling Computational Efficiency

In both CBOW and Skip-gram the model (figure 3.8) learns two distributed representation of word in the vocabulary input and output V and V'. At the output of theses models we have the Hierarchical Clustering (HC) which gives us the probability of observing each word in the vocabulary following the given input context. One problem with this approach is that the dimension of the output HC is in the order of the size of the vocabulary which can be in hundreds of thousands or even millions. When we are performing the calculations we need to compute all the activations in the hidden layer and calculate the HC probabilities. This is computationally expensive both in terms of memory and time required to train the models as it is required for every training instance. Thus, we need an efficient way of defining this probability distribution function. One approach is to limit the number of output result to be updated per training instance. The HC method working process is described below by in figure 3.2.


Store
Predicted word
Word2vec-Tiling model
Joint representation of target and its context words
CBOW model
n
n′
Skip-gram model
$\log\frac{(1+\sum_{n-1}^{n}(w_p w_{p+1}|w_k w_{k+1}))}{\sqrt{\sum_{c\in C_n}(w_p, w_k)^2\times(w_{p+1}, w_{k+1})^2}}+\sum_{c\in C_n}\log(1+e^{|p|w_p n})$
$P(w_{t+i}, w_t)=\frac{\exp\sum_{-c}^{c}(V'_{w_{t+i}}{}^T v_{w_t})}{\sum_{w_{t+i}\in V}\exp\sum_{-c}^{c}(V'_{w_{t+i}}{}^T v_{w_t})}$
Figure 3.8 The structure of CBOW and Skip-gram model

3.4 Word2vc-Tiling (CBOW and Skip-gram model)

Traditionally, Word2vec-Tiling models are trained on textual corpus data but here we utilize it on the purpose data of web users. We are therefore treating each item as a word. Then the original word2vec method can be applied in the recommendation scenario. Based on the idea of utilizing tiling distance to measure the similarity between items, we apply distributed representation approach to build our recommender system. The procedure described in our recommended word2vec using the tiling method is described at 3.2.1 and 3.2.2 above. Now how we use Word2vec-Tiling in the combination of CBOW and Skip-gram models, we will describe it as figure 3.8.

According in figure 3.8, we have described CBOW and skip-gram overall module, describes the strategy that we propose for finding a text word semantic relation for a given input slice using the trained Word2vec-Tiling model. Note that the word embedding in a semantic may or may not represent a text embedding. If the input slice is not in the Word2vec-Tiling (CBOW) vocabulary, the algorithm returns the original input slice as its own substitute. Otherwise, it calculates the gap distance between all text in the vocabulary and the input text words. It then collects a list of n text word that are the closest to the input, where n' is an input parameter of the algorithm. Because the text considered in this research is word, the algorithm uses the top-n slices to calculate a score for the text semantic relation analysis, which can be used to inter the keyword. The score of a particular text semantic analysis is its normalized frequency of occurrence in the top-n slices. A sentence embedding is represented by averaging the embeddings of words in that sentence and use word-gap distance (CBOW) to measure the distance between two consecutive sentences. The distance between the last sentence of the previous part and the first sentence of the next part is usually far, resulting in a peak in the line plot Skip-gram (SG). One example for a single abstract is shown in figure 3.3. Purify the information of purpose corpus obtained from the SG step by removing the words whose frequency of occurrence being the top of the purpose corpus to get the purified purpose corpus. This corpus has higher concentration of topic information than the purpose corpus, let alone preprocessed corpus. Train a new Word2vec-Tiling model, by using the purified-purpose corpus obtained from the CBOW and SG step, in which similar words have smaller n-tiling distance compared with Word2vec-Tiling in CBOW model. Note that all the word embeddings used after CBOW step are from Word2vec-Tiling in Skip-gram model. Finally, find the clustering category of each text document by computing the Tiling similarity of document embeddings (Predicted word).

3.4.1 Reinforcement Learning (RL) Algorithm

Reinforcement Learning or Q-Learning is learning method, which renews each synapse weights according to output signals. The learning rule can be applied to feed forward networks and recurrent networks. The flow of measurement of the renewal of the syndrome is shown by the pseudocode as follows. Capital and lowercase letters when used as symbols.

At initial state as input the starting state s, a waiting time t the collection of Q-value for each individual classes q, the set of total document δ, the set of all possible action

Table 3.5 Text episode reward value calculation process
Step 1: Set n=1 and for all $s \leftarrow s_n$ and $a \in A(i)$, $Q(s, a)=Q(s_{n+1}, a_n)=0$ specify $g \in \delta$ **Step 2:** For each episode rewards $r \in \delta$ then **Step 3:** Calculate every text word gap value by $\delta = (1/r)^{(1/1+r)}$

a, the step-size parameter α, the discount rate parameters ɣ, from Q-learning and the parameter ∈ for the ∈-greedy selection of action.

Out total working process we divided by two way. In the begging, we use the Q-value learning in previous episodes to control the reward and total document. Finally, we update the Q-value from the sequence of initial state (s, a) and next state (s_{n+1}, a_n) triples that occurred in the episode that just ended.

In initializes s, to the initial next state s_n and starts the episode. During an episode, Q-value periodically waits and then observes the current state s_n. For each document δ in the set of classes g, create the current state s, for the class δ by customizing the observed state s_n. The reason why we have to do this is because different classes need different kinds of information. The size of the observed state s_n is very large and contains various kinds of information. Each class requires some of the information uniquely. We will detail the kind of information extracted in the next chapter.

According table 3.6, the next step is to create the set of valid actions A, of class δ under current situation. We cannot use α_c directly because some of the actions might not be applicable in the current state. For example, cultivators can edify his land. However, without sufficient funds, etymology will ignore this command. Therefore, Step 2 shows us the invalid actions. Any action randomly chosen from this set of actions is guaranteed to be a valid action. Next, retrieve the Q-table of class δ from the collection of Q-tables q. For each unit g if the unit δ is idle, Q-value retrieves an action a from the Q-table using μ-greedy exploration. Notice that the algorithm choose an action a from the set of valid action a, not from the set of possible actions

α_δ. Then execute the action a step 3. Because this is remote learning method, step 3, saves the set of s_δ, a_δ and the current state s_n for the Q-learning updates in the second phase. We wait until the end of the episode to update the Q-values because we have found experimentally to be more effective to use the outcome at the end of the episode. This is why we have to save the list of state-action to perform the remote-line update later. Step 3 we also updates the previous state s_G and the previous action a_δ.

Table 3.6 Outline of modified algorithm based on Q-value
Step 1: Set n=1, and select an initial middle state (a_δ, s) **Step 2:** Initialize the Q-value for all states and actions to initial middle value Q← q(δ) Use the following transformation repeatedly until the Q-values converge in an ∈-approximate sense. For all episode if random(1) ≥μ $\alpha \leftarrow \text{arg.max}_a \in A(\delta(s_{n+1}, a_n))$ **Step 3:** Generate a new episode L_G using the following relation $L_G < s_\delta, \alpha_\delta, s_n >$ $S_G \leftarrow s_n, \alpha_\delta \leftarrow \alpha$

According in table 3.7, in the second phase, after calculating the reward r, we use the Q-learning update on all value in the list L_δ of each individual unit q to update Q-value of each value δ. Finally, return the Q-value show in Step 2.

Table 3.7 The update reward function Q-value
Step 1: Let L_δ denote the next episode of action s_{n+1}, a_n such as, the sample obtained in the s_n iteration.
Step 2: Update reward value $\delta \in g$ using , the current sample $q \leftarrow \delta$ such as $<s,a,s_n>L_\delta$ Finally update Q-value $Q(s,a) \leftarrow Q(s,a) + \alpha\delta(s_{n+1},a_n)[r+\gamma.\arg.max_a(Q(s_{n+1},a_n) - Q(s,a))]$

3.4.2 TF-IDF Algorithm

In a good segmentation, sentences among different segments should belong to different subtopic's. Hence, the goal is to find the segmentation with the maximum external (two consecutive segments) dissimilarity. There are two types of segmentation in the field of the web such as structured and unstructured data (figure 2.2). Web researchers usually use these two types of data to complete their web research. In this thesis, we have preferred only web pages of unstructured data which we have collected by web pages using BFS method. In the existing method, they have preferred web pages data. As a result, they are not able to find their keyword data equally. The equation (25) used in the existing method, in the primary stage if the Web page (P), Keyword's (K) and Number of keyword document (K_i) any value are "0" so, the result of the total TF value is zero, it means that the keyword is unable to choose a similar or closer word from the same document, which is not a good result. In this reason, the accuracy rate is not up to the satisfaction levels.

Therefore, to solve this problem we have used the total number of keyword document "|K|" in equation (31) [20]. In previous research, the total number of words has not been calculated. In this thesis research, we have calculated the total word value. This means that in the same document, the keyword can be correctly selected as a similar word. Therefore, if the more documents a term appears, the less important that term will be, and the weighting will be less. So the user can find their information easily and within in a short time.

$$\text{TF}(P, K) \leftarrow \text{fre}\ (P,K)\ /\ \sum_{K_i \in K} fre\ (|P|,\ K_i) + |K| \tag{31}$$

$$\text{IDF}(P, K) = |P|\ /\ \sum_{P_i \in P} dfre\ (P_i, K) \tag{32}$$

The flow of measurement of the TF-algorithm of the syndrome is shown by the pseudocode and working flow diagram figure 3.9, as follows.

Table 3.8 TF-algorithm proposed method pseudocode

```
The TF-IDF algorithm attempts to partition a finite collection of element k=1...n, into a finite collection of p=x=0; TF clusters with respect to some given criterion. Given a finite set of data, the algorithm feeds forward a list of TF cluster centers. Where |P| the total number of web page |P| and keyword |K| then the total Term Frequency step are
Step 1: Select list the x square score for each term p category.
Step 2: Select the new condition the keyword |P| is not empty and total number of keyword |K| is queue then
        $|P|[K_i] > 0$ and $t > x[K_i]+x$
Step 3: Then go to search web page p, number of keyword document $K_i$ with web page document $P_i$
        $|P|[K_i] = P_i[K_i] - 1$ or $x[k_i] = t$
Step 4: If previous condition is satisfy then go to step 4, k==n then reject web page total number of keyword value. Otherwise condition are
$|P|[|K|] < T$ and $|P|\ [|K|] > 0$ and $t > x\ [|K|] + x$
Step 5: Select new state of web page total number with total number of keyword and subtraction with 1, $|P|[|K|] -1$ then set $x\ [|K|] = t$ where  is a controller by p
Step 6: Select the new value interrelation coefficients T with web page total number and total keyword
$|P|[|K|] > + T$ and $P\ [|K|] <$ fre P and K / (get_total (P and $K_i$))
Step 7: Push total keyword value in the internal keyword queue value $x\ [K_i] = t$
Step 8: Keyword queue is empty for initial and next state s and s' the set $P[K_i]<$fre P and $K_i$
```

Step 9: If previous is satisfy the x [K_i] = t frequency total value are

$$TF(P, K) \leftarrow fre\ (P,K) / \sum_{K_i \in K} fre\ (|P|,\ K_i)+|K|$$

Invocation and initialization (Step 1 and 3). The block functionality is executed in an infinite loop, activated every time interval. The current P state is set to the lowest value, P_i and the vector of time of the previous P state change, x, is set to 0.

Task fetching, the tasks external queue is checked if empty end, if not a task total keyword value |K| is fetched.

The conditional rejection (Step 3 to 5), The code related to this step is executed inside a loop whose iterator k ranges from 1 to n, which is the number of cores in the processor. If the output value of the K_i is controller (|P|[K]) is negative and the corresponding K_i core operates with highest performance and the interval between the previous P state switching time and current time t, is long enough this value is assumed to have no capacity to execute task K_i. Consequently, if all the cores in the analyzed processor have no capacity, task |K| is rejected. Moreover, if P, state of the K_i core is different from P and its P_i state has not been switched for at least time web document x, P state in the K_i core is decreased and the K_i element of vector x, storing the previous time of the P_i state alteration of the value K_i, is updated. As the previous errors in the k controller have been observed in a different P state. Hence, these obsolete values do not influence future admittance decisions.

Task conditional admittance (Step 6 to 8), If the output value |K| controller is lower then threshold +T, |P| state in the |K| core is long enough, the |P| state is lower, and x is updated accordingly. In case the |K| dependent controller output value is above threshold +T, P state of the K_i core is different from the highest P state available in the processor (frequency p) and the previous switching of P, state in the |K| core is increased, P|K| and x is updated. Finally, |K| is sent to the every state queue.

P, state conditional increase, the functionality of this next state step is activated a' and s'. Under such condition, P, state of all n cores are analyzed in a loop. If the core performance is not the lowest possible then the core's P state is increased, the web document value continues updated. Invocation (Step 9), the block functionality is executed in an infinite loop, activated every time interval.

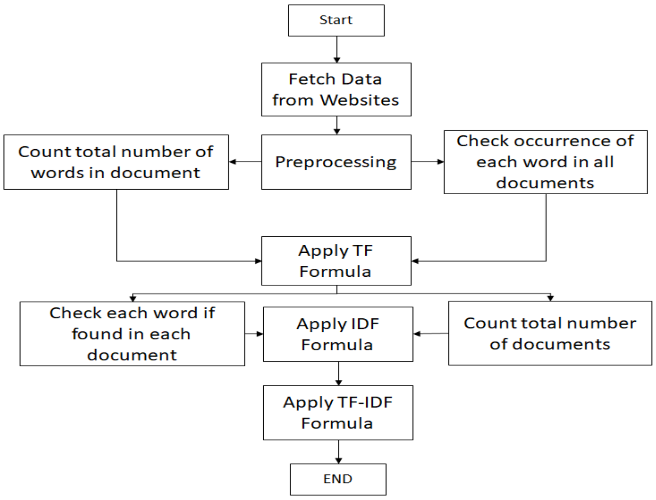

Start
Fetch Data from Websites
Count total number of words in document
Preprocessing
Check occurrence of each word in all documents
Apply TF Formula
Check each word if found in each document
Apply IDF Formula
Count total number of documents
Apply TF-IDF Formula
END

Figure 3.9 TF-IDF working diagram

3.4.3 TF-IDF Working Diagram

First of all, data has to be fetched from websites. Secondly, in preprocessing phase, one has to look for HTML/CSS and stop words and remove all of them as they are unnecessary has no importance in this scenario. Third, one need to count total number of words and their occurrences in all documents. Once these steps are performed, one can apply Term Frequency formula to calculate TF as discussed in equation (31). After calculating TF, one has to check, if each word is found in every document or not and has to count total number of documents in hand. Once these steps are completed, one can apply Inverse Document Frequency (IDF) formula to calculate IDF as discussed in chapter 3 and equation (32). In the last, after obtaining TF and IDF, one can easily calculate TF-IDF by applying its formula as described. The algorithm can be implemented in any programming language of your choice. For this research, it was implemented using PHP for the sake of simplicity.

The TF-IDF process can be observed using a diagram here which is showing all major and minor steps that has to be taken in order to successfully implement the algorithm using computer programming.

3.4.4 Development tools

For model development, Matlab Mint was used because it is easier to install and hold the environment on the Unix-based operating system. The language chosen was Python 3.6, because it was used to create the system and has a lot of efficient packages for machine learning tasks. Scikit-learn, which is built for machine learning tasks, NumPy very good library for scientific computations, pandas is developed for data analysis, ScraPy scientific computing tool for Python. ScraPy is a full framework for web crawling which has the tools to manage every stage of a web crawler. Gensim is an easy to use library for generating similarities. Something very useful is that all these libraries communicate perfectly with each other. As a main development environment, PyCharm CE 2018.3.5. was used because it highlights errors, suggests better code styling, provides autocomplete and includes other comfortable features. Anaconda is a virtual environment system for python to handle package installation. It was used to install all necessary libraries, as well as the Jupyter Notebook. Jupyter Notebook helps to save intermediate results and allows to use them in other code cells inside one notebook, what is very convenient and important for data analysts, because it saves a lot of time that otherwise would be used saving and loading files for every intermediate result.

Chapter 4

Implementation and Evaluation

4.1 Implementation

In order to prove that the accuracy of the semantic analysis can be improved by applying web data, two experiments are conducted in this research. The first is an experiment to distinguish document features by extracting features of document semantics; the other one is word embedding [12], which is used to distinguish the meaning of words in a document. Object distribution vector and word embedding such as Word2vec-Tiling are all distributed representations of words. Word2vec-Tiling uses a fixed-size sliding window to collect contextual information on the current word in documents that only reflects the semantics of word items at sentence-level. Topic distribution vector is derived from the Latent Dirichlet Allocation (also known as LDA) model, which can characterize the relationship between words at document-level [7]. In this thesis research, Word2vec-Tiling word embedding based on topic distribution vector is proposed. The topic segmentation process is the basis core of Tiling system. It can be launched in two different ways [8]. The first one allows the user, once he visualizes the results of his research, to launch this process for every document appearing in the list of research results. The second method gives the user a chance to segment a text outside the research interface. From the segmentation interface, the user must choose a text to segment; he can see the different stages of the segmentation process after activating the segmentation synthesis option. Tiling segmentation can be done in two ways, pre-processing and segmentation. In this research, we followed this method.

The pre-processing phase consists of the succession of the following operations: normalization, stop words elimination, and stemming; while the segmentation phase is constituted of the following operations: TF-IDF weights, blocks similarity, cohesion score, and boundaries selection.
First of all we will describe our data collect process from the web page. By using the specific keywords, we collected all hyperlinks which belong to in that web page. In this thesis research, firstly we have presented the working design of our crawling system by an architecture in chapter 3 by figure 3.1. Here the process of execution of RL and TF-IDF algorithm work is displayed. When the crawler finishes downloading a page from web, the relevance of the page needs to be identified to improve the efficiency of the crawling process and save the storage space. For this reason we have used Breadth-first search (BFS) and Word2vec-Tiling algorithm. At first we explain an interesting feature of the BFS algorithm [11]. The BFS algorithm traverses the graph by following its links. The distance of each crawled page from the root (seed) is always less than or equal to that of the uncrowded pages. The BFS ordering is not the best method for crawling but breadth-first has an interesting feature. It can discover pages with high page rank in the initial stages of the crawling process. According to Word2vec-Tiling, it works by dividing it into two steps such as CBOW and Skip-gram. Now we will describe these two methods to follow the existing procedure.

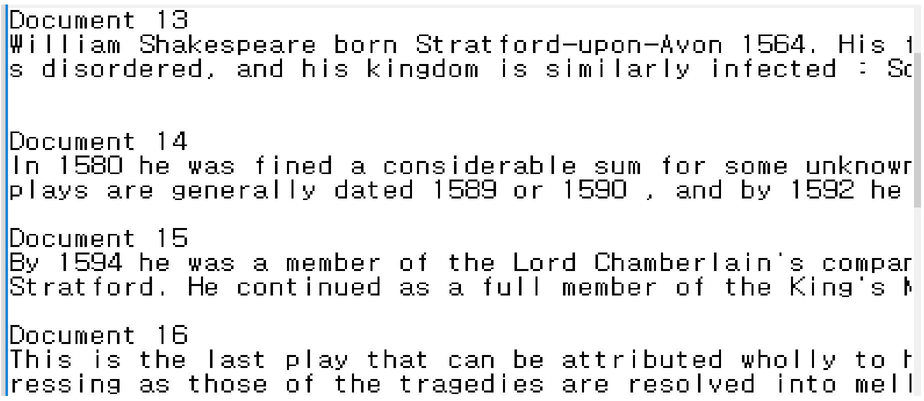
4.1.1 Implementation of CBOW model

We collected unstructured data from web pages using the BFS algorithm. Then, we used Word2vec-Tiling (CBOW) to divide the closer neighboring words in our used keywords. According in CBOW model we proposed our equation as

$$\log\frac{(1+\sum_{n=1}^{n}(w_p,w_{p+1}|w_k,w_{k+1}))}{\sqrt{\sum_{c\in C_n}(w_p,w_k)^2\times(w_{p+1},w_{k+1})^2}}+\sum_{c\in C_n}\log(1+e^{|P|(w_p n)}) \tag{33}$$

According to the equation (33), the word gap value is calculated by a logarithm measure, where given the two text word blocks w_p and w_k each with going to next step w_{p+1} and w_{k+1} where n, ranges over all the terms that have been registered during the tiling tokenization step and w_p,w_{p+1} in the weight assigned to term n, in block w_{p+1}. In our algorithm, the weights on the terms are simply their frequency (term frequency) within the block. However, because the similarity is measured between two blocks and w_{p+1} and w_{k+1}, where w_{p+1} spans token sequences 2^n-1 through n, which show the results of the block Word2vec-Tiling algorithm on the semantic text. The boundaries are determined by changes in the sequence of similarity scores. The text sequence gap numbers are ordered according to how steeply the slopes of the plot are to either side of the text sequence gap, rather than by their absolute similarity score. For a given text gap n (our defined gap value), the algorithm looks at the scores of the gaps to the left, as long are their values are changing figure 4.2. When the values to the left peak are found, the difference between the score at the peak and the score are recorded. The same procedure takes place with the text gaps to the right of n; their scores are examined as long as they continue to rise, as shown in figure 4.4. The relative height of the peak to the right of n is added to the relative height of the peak to the left. These new scores, called depth scores, corresponding to how sharply a change occurs on both sides of the token sequence gap, are then sorted. Segment boundaries are assigned to the text sequence gaps with the largest corresponding scores, adjusted as necessary to correspond to true paragraph breaks.

On the other hand, in previous researches on tiling, they tokenize each of the text by 20, under a gap. In this way, they calculate the distance between multiple gaps [3], [4]. After tokenization, the next step is the comparison of adjacent pairs of blocks of text-sequences for overall text similarity. Another important parameter for the algorithm is the block size, the number of token sequences that are grouped together into a block in comparison with an adjacent group of token sequences. They do not use actual paragraphs or text because their lengths can be highly irregular, leading to unbalanced comparisons. However, in our system, we can use the actual paragraph or text. Using our equation (33), our overall result is shown in figure 4.6. In the figure, we gave the lifestyle of Shakespeare, with about 10,000 words, as an input document. However, we get 3,686 words related to our inputted keyword. On the other hand, using the equation (3) of the existing method, from 10,000 words in figure 4.5, we get about 5,816 words, some of which are not related to our keyword.



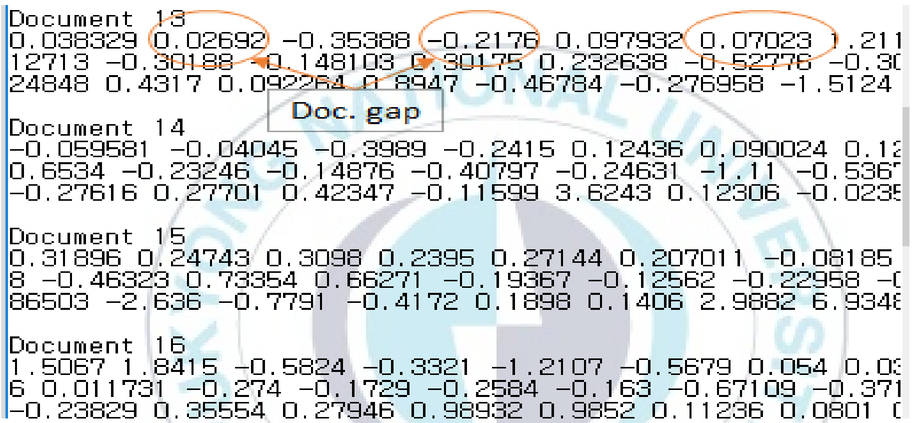
```
Document 13
William Shakespeare born Stratford-upon-Avon 1564. His i
s disordered, and his kingdom is similarly infected : Sc


Document 14
In 1580 he was fined a considerable sum for some unknowr
plays are generally dated 1589 or 1590 , and by 1592 he

Document 15
By 1594 he was a member of the Lord Chamberlain's compar
Stratford. He continued as a full member of the King's M

Document 16
This is the last play that can be attributed wholly to h
ressing as those of the tragedies are resolved into mell
```

Figure 4.1 Example of applied document



```
Document 13
0.038329 0.02692 -0.35388 -0.2176 0.097932 0.07023 1.211
12713 -0.30188 -0.148103 0.30175 0.232638 -0.52776 -0.30
24848 0.4317 0.092264 0.8947 -0.46784 -0.276958 -1.5124

Document 14
-0.059581 -0.04045 -0.3989 -0.2415 0.12436 0.090024 0.12
0.6534 -0.23246 -0.14876 -0.40797 -0.24631 -1.11 -0.5367
-0.27616 0.27701 0.42347 -0.11599 3.6243 0.12306 -0.0235

Document 15
0.31896 0.24743 0.3098 0.2395 0.27144 0.207011 -0.08185
8 -0.46323 0.73354 0.66271 -0.19367 -0.12562 -0.22958 -0
86503 -2.636 -0.7791 -0.4172 0.1898 0.1406 2.9882 6.9348

Document 16
1.5067 1.8415 -0.5824 -0.3321 -1.2107 -0.5679 0.054 0.03
6 0.011731 -0.274 -0.1729 -0.2584 -0.163 -0.67109 -0.371
-0.23829 0.35554 0.27946 0.98932 0.9852 0.11236 0.0801 0
```

Figure 4.2 Example of word transaction using Tiling method

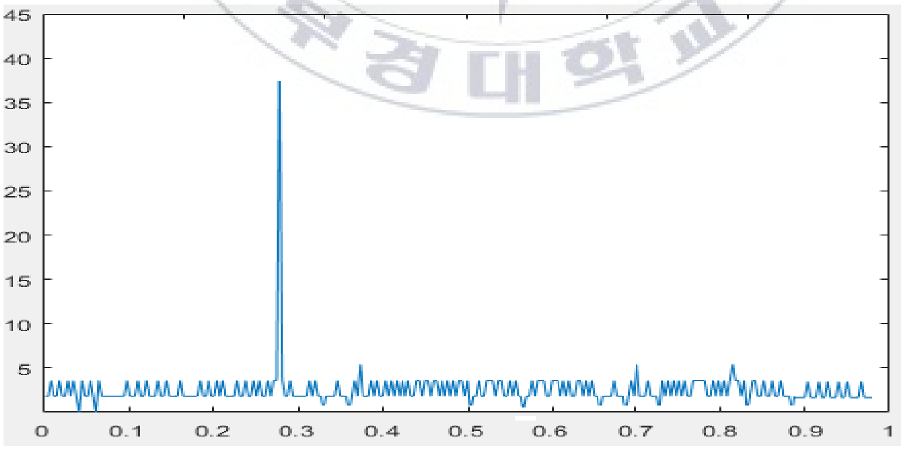


Figure 4.3 Represent document (figure 4.1) working process

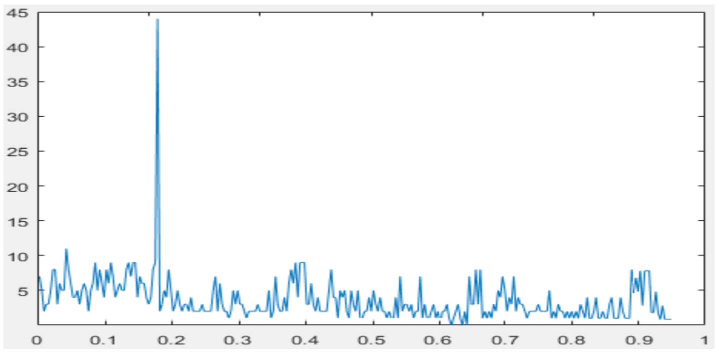


Figure 4.4 Word2vec-Tiling (figure 4.2) working process

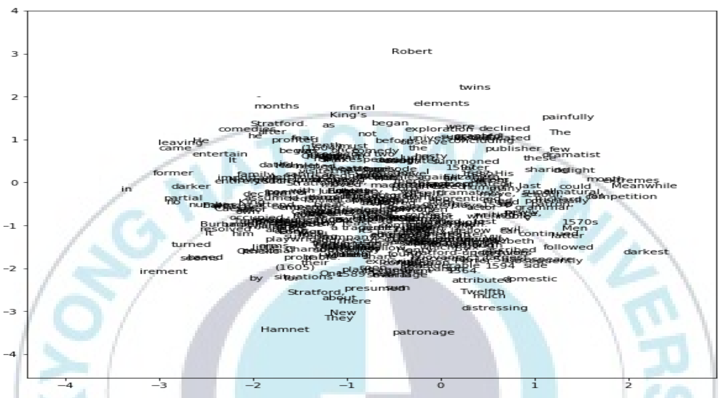


Figure 4.5 Existing method word2vec overview

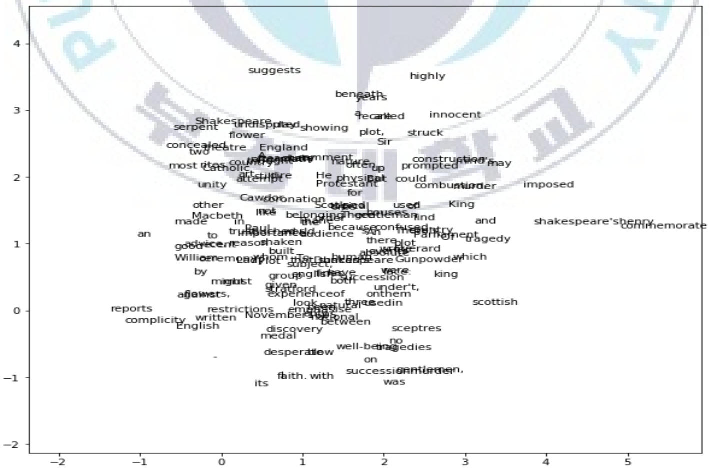
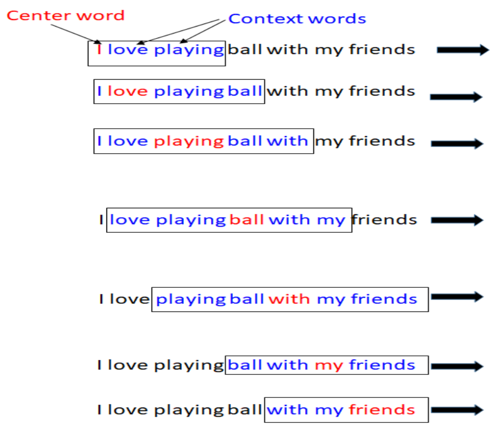


Figure 4.6 Proposed method Word2vec-Tiling overview


Center word
Context words
I love playing ball with my friends
I love playing ball with my friends
I love playing ball with my friends
I love playing ball with my friends
I love playing ball with my friends
I love playing ball with my friends
I love playing ball with my friends

Center Words	Context Words
[1 0 0 0 0 0 0]	[0 1 0 0 0 0 0] [0 0 1 0 0 0 0]
[0 1 0 0 0 0 0]	[1 0 0 0 0 0 0] [0 0 1 0 0 0 0] [0 0 0 1 0 0 0]
[0 0 1 0 0 0 0]	[1 0 0 0 0 0 0] [0 1 0 0 0 0 0] [0 0 0 1 0 0 0] [0 0 0 0 1 0 0]
[0 0 0 1 0 0 0]	[0 1 0 0 0 0 0] [0 0 1 0 0 0 0] [0 0 0 0 1 0 0] [0 0 0 0 0 1 0]
[0 0 0 0 1 0 0]	[0 0 1 0 0 0 0] [0 0 0 1 0 0 0] [0 0 0 0 0 1 0] [0 0 0 0 0 0 1]
[0 0 0 0 0 1 0]	[0 0 0 1 0 0 0] [0 0 0 0 1 0 0] [0 0 0 0 0 0 1]
[0 0 0 0 0 0 1]	[0 0 0 0 1 0 0] [0 0 0 0 0 1 0]

Figure 4.7 Center and context words resulting from the text corpus "I love playing ball with my friends".

4.1.2 Implementation of Skip-Gram model

Now we are going to describe about Skip-gram model that we proposed in this thesis. The first step in our implementation is to transform a text corpus into numbers. Specifically, into one-hot encoded. Recall that in Word2vec-Tiling we scan through a text corpus and for each training example we define a center word with its surrounding context words. Depending on the algorithm of choice (Skip-gram), the center and context words may work as inputs and labels, respectively, or vice versa. Typically the context words are define as a symmetric window of predefined length, on both the left and right hand sides of the center word. For example, suppose our corpus of the sentence "I love playing ball with my friends". Also, let's say that we define our window to be symmetric around the center word and of length two. Then, our one-hot encoded context and center words can be visualized as follows in figure 4.7.

The function above returns the corpus tokenized and the size V of the vocabulary. The vocabulary is not sorted in alphabetical order (it is not necessary to do so) but it simply follows the order of appearance. At this point we can proceed with the mapping from text to one-hot encoded context and center words using the function "corpus2io" which uses the auxiliary function to categorical.

To show that the functions defined above do accomplish the required task, we can replicate the example presented in figure 4.8. First, we define the size of the window and the corpus text. Then, we tokenize the corpus and apply the "corpus2io" function defined above to find out the one-hot encoded arrays of context and center words as flows:

1. window_size = 2

# Declare total text corpus size

2. corpus = ["**I love playing ball with my friends**"]

# Example our input text

3. corpus_tokenized, V = tokenize(corpus)

# Every word separate using tokenize condition

4. **for** i, (x, y) in enumerate(corpus2io(corpus_tokenized, V, window_size)):

# Every word divided with x and y with vocabulary V, it is condition for showing result

5.   **print**(i, "/n center word=",y, "/n context words =/n",x)

# Using the print command the total program show there final result

Note that at the boundaries the context words are not symmetric around the center words.

We are essentially interested in writing a few lines of code that accomplish this mapping, from text to one-hot-encoded vectors, as displayed in the figure above. In order to do so, first we need to tokenize the corpus text.

1. **def tokenize** (corpus):

"""" Tokenize the corpus text.

  :parameter corpus: list containing a string of text (example:["I love playing ball with my friends"])

  :return corpus_tokenized: index list of words in the corpus, in the same order as the original corpus(the example above would return [1, 2, 3, 4...])

  :return V: size of vocabulary

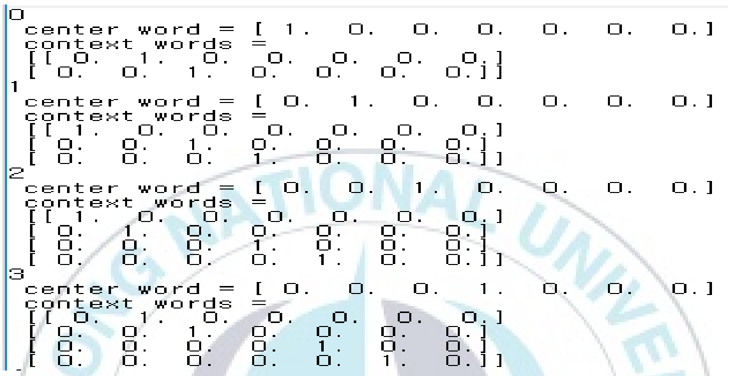
""""

2. tokenizer = Tokenizer()

# Every text word are divided using command

```
3. tokenizer.fit_on_texts(corpus)
4. corpus_tokenized = tokenizer.texts_to_sequences(corpus)
5. V = len (tokenizer.word_index)
# Text word vocabulary using tokenize function and split all text word
6. return corpus_tokenized, V
# Using this command the text word vocabulary complete there split up.
```

Overall pseudocode working figure as show in 4.8.



```
0
 center word = [ 1.   0.   0.   0.   0.   0.   0.]
 context words =
 [[ 0.   1.   0.   0.   0.   0.   0.]
  [ 0.   0.   1.   0.   0.   0.   0.]]
1
 center word = [ 0.   1.   0.   0.   0.   0.   0.]
 context words =
 [[ 1.   0.   0.   0.   0.   0.   0.]
  [ 0.   0.   1.   0.   0.   0.   0.]
  [ 0.   0.   0.   1.   0.   0.   0.]]
2
 center word = [ 0.   0.   1.   0.   0.   0.   0.]
 context words =
 [[ 1.   0.   0.   0.   0.   0.   0.]
  [ 0.   1.   0.   0.   0.   0.   0.]
  [ 0.   0.   0.   1.   0.   0.   0.]
  [ 0.   0.   0.   0.   1.   0.   0.]]
3
 center word = [ 0.   0.   0.   1.   0.   0.   0.]
 context words =
 [[ 0.   1.   0.   0.   0.   0.   0.]
  [ 0.   0.   1.   0.   0.   0.   0.]
  [ 0.   0.   0.   0.   1.   0.   0.]
  [ 0.   0.   0.   0.   0.   1.   0.]]
```

Figure 4.8 Text corpus using skip-gram result

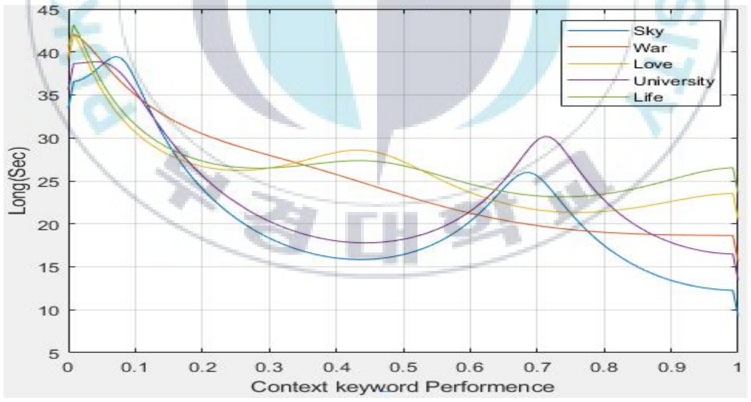


Figure 4.9 Proposed system using equation result of working skip-gram. Comparisons of training speed in logarithm against different context keyword size. Sec refers to thousand words per second.

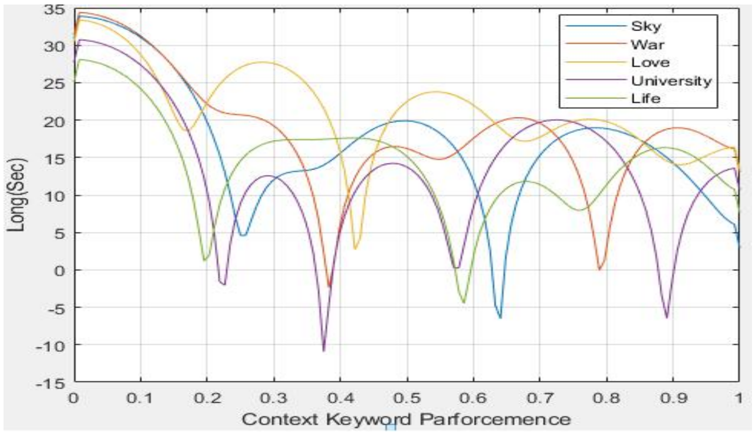


Figure 4.10 Existing system using equation result

According the figure 4.9 which is define in proposed method and figure 4.10 define existing method using equation (30) and (23). We can see in Existing method using equation (23) figure 4.10 the word value is going to lower because the input data is not finding their actual keyword, on the other hand, in Proposed method using equation (30) in figure 4.9 word value going to height velocity because input data is successful in finding their actual keyword. According to in figure the university related data almost 30% success for choosing their value.

4.1.3 Implementation of Word2vec-Tiling

Word2vec-Tiling is touted as one of the biggest, most recent break through in the field of Natural Language Processing (NLP). The concept is simple, elegant and (relatively) easy to grasp. A quick Google search returns multiple results on how to use them with standard libraries such as Matlab or TensorFlow.

According, in figure 4.11, a dashed grey line at y = 0.6 which represents the baseline against which we have measured both algorithms CBOW and Skip-gram. Any line above the 0.5 thresholds is performing at a rate that is better than chance. This adds another condition with which to evaluate CBOW and Skip-gram. While both models eventually end up performing below the threshold, the difference between when each crosses the 0.6 threshold is notable. Both algorithms start by performing at perfect accuracy. After the first 1% of paraphrases are generated, CBOW begins a swift dive towards the chance threshold. There is a slight recovery before it plummets again by the time it has generated the first 10% of paraphrases. Skip-gram, on the other hand, maintains perfect accuracy for the first 7% of the data it outputs. Accuracy then starts to degrade steadily for a few percentage points until it achieves an equilibrium that keeps it hovering between 60% and 70%. This trend is maintained until that Skip-gram has generated over 50% of the paraphrases it will generate during its execution. After this, a steady slump downwards starts, with the accuracy line intersecting the 0.5 threshold when Skip-gram has generated 59% of its total paraphrases. Only after 70% of the total data has been output does Skip-gram's performance degrade to equal that of CBOW.

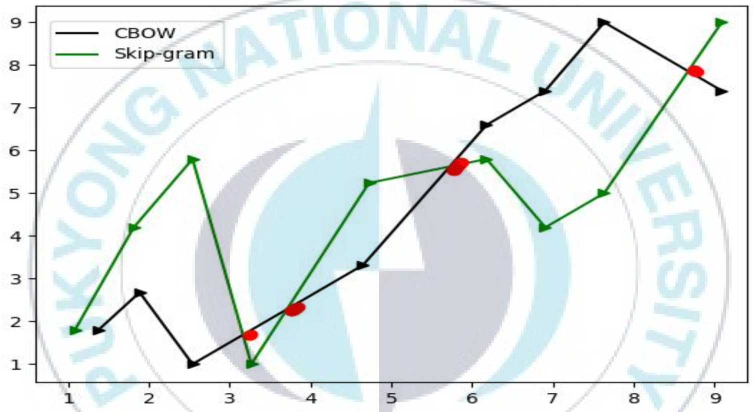
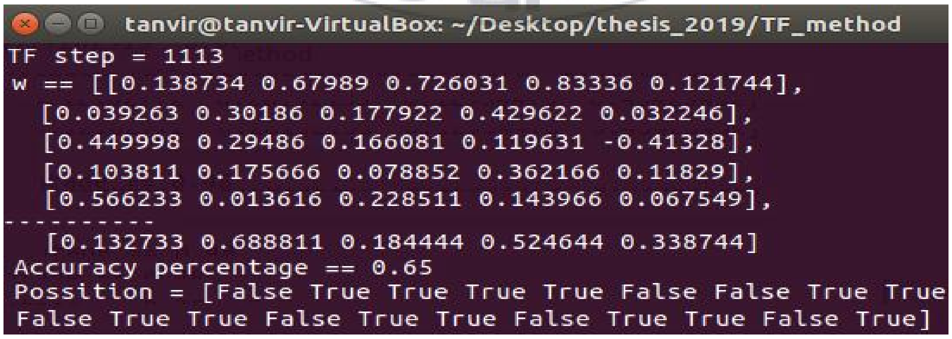
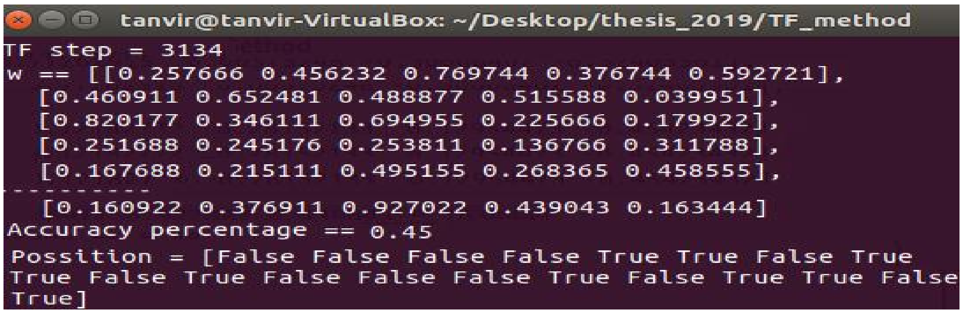


Figure 4.11 Graph comparing CBOW and Skip-gram. Algorithms above the dashed line are performing better than chance.



```
tanvir@tanvir-VirtualBox: ~/Desktop/thesis_2019/TF_method
TF step = 1113
w == [[0.138734 0.67989 0.726031 0.83336 0.121744],
  [0.039263 0.30186 0.177922 0.429622 0.032246],
  [0.449998 0.29486 0.166081 0.119631 -0.41328],
  [0.103811 0.175666 0.078852 0.362166 0.11829],
  [0.566233 0.013616 0.228511 0.143966 0.067549],
----------
  [0.132733 0.688811 0.184444 0.524644 0.338744]
Accuracy percentage == 0.65
Possition = [False True True True True False False True True
False True True False True True False True True False True]
```

Figure 4.12 Proposed method using TF-algorithm result.



```
tanvir@tanvir-VirtualBox: ~/Desktop/thesis_2019/TF_method
TF step = 3134
w == [[0.257666 0.456232 0.769744 0.376744 0.592721],
  [0.460911 0.652481 0.488877 0.515588 0.039951],
  [0.820177 0.346111 0.694955 0.225666 0.179922],
  [0.251688 0.245176 0.253811 0.136766 0.311788],
  [0.167688 0.215111 0.495155 0.268365 0.458555],
----------
  [0.160922 0.376911 0.927022 0.439043 0.163444]
Accuracy percentage == 0.45
Possition = [False False False False True True False True
True False True False False False True False True True False
True]
```

Figure 4.13 Existing method using TF-algorithm result.

4.2 Implementation of TF-IDF algorithm

In order to prove that the accuracy of the semantic analysis can be improved by applying web data, two experiments are conducted in this research. This of one is an experiment to distinguish document features by extracting features of document semantics and another one is word embedding, which is used to distinguish the meaning of words in a document [3]. The data used to test that is an online website corpus like NASA, Korean University and so on related to Sky, Love, Life, War and University and four data sets are created to check the performance of web devices by dividing mesne the data. We chose the closest data from the total data of the keyword using Word2vec-Tiling in figure 4.6 and table 4.1.

TF-IDF is a combination of two different words such as, Term Frequency and Inverse Document Frequency. Actually, in this thesis research we implemented TF method equation (31). Following this aforementioned equation, we implement data from Word2vec-Tiling in figure 4.12.

On the other hand, flowing existing method equation (25) we show the result in figure 4.13. According in figure 4.12 and 4.13, we can see that TF-algorithm's performance is not satisfactory. Because our keyword related document accuracy rate is not more than 65%. It’s meaning those TF algorithm even follows word frequency but can not use word frequency properly for multiple words. According in figure 4.12 and 4.13, here we have used twenty document, which can be seen as “True” and “False”. These results show that if the same word appears in every document, it excludes words with a value of “False”, which did not produce good results in learning. So accuracy based on learning outcomes was unsatisfactory. On the other hand, the results show that if the same word does not appear in every document, it excludes words with a value of “True”, which provides good results. So accuracy based on learning outcomes was satisfactory.

4.3 Implementation of Reinforcement Learning Algorithm

Reinforcement Learning is a learning method, which renews each synapse weights according to output. The learning rule can be applied to feed forward networks and recurrent networks. The calculated flow of renewing synapse weights is shown in follows. Firstly, the influence of rewards ‘r’ and the influence of document ‘δ’ are obtained by

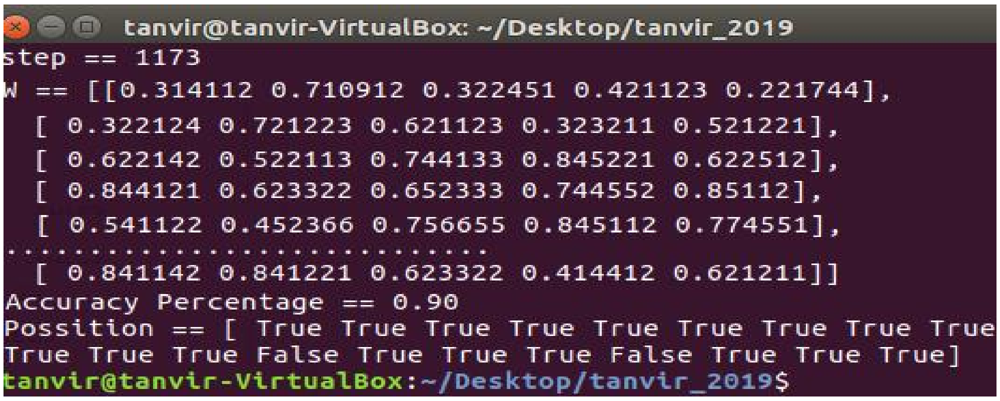
$$r = \sum_{i=0}^{n} \gamma^{i-1} r_i \tag{34}$$

$$\delta = (1/r)^{(1/1+r)} \tag{35}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r+\gamma.\,\arg.\,max_a(Q(s_{n+1}, a_n) - Q(s,a))] \tag{36}$$

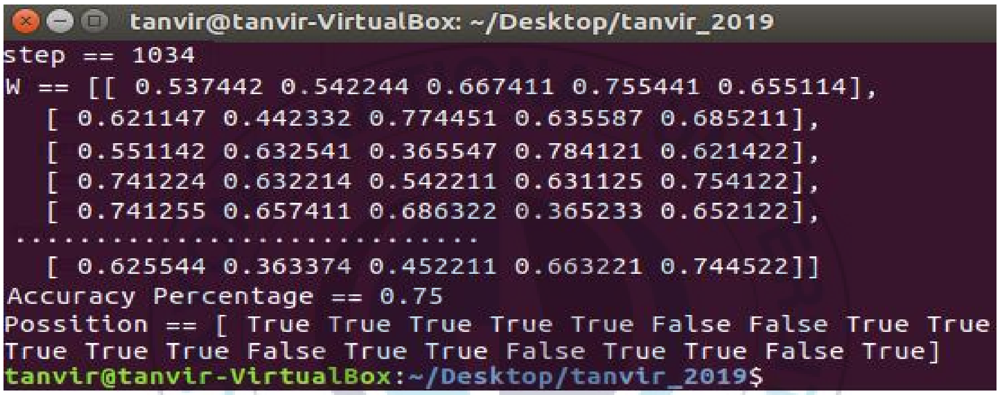
$$Q(s,a) \leftarrow Q(s,a) + \alpha\delta(s_{n+1},a_n)[r+\gamma.\,\arg.\,max_a(Q(s_{n+1}, a_n) - Q(s,a))] \tag{37}$$

Following the aforementioned equation, we compute the reward and value of each corpus. Here, we can see in figure 4.15, that the accuracy rate is 65%, because the baseline system selected the document value randomly, using equation (36). In order to randomly select the keyword related word, the existing method used the previous method (using equation ‘3’) of word2vec. In order to solve this problem, we multiply the value “$\delta(s_{n+1},a_n)$” used in equation (37) also describe in chapter 3 part. As a result, our accuracy rate has increased by 15%, because we have calculated the gap value in each sentence using “$\delta(s_{n+1},a_n)$”. Therefore, our accuracy rate is 90%, in figure 4.14.



```
tanvir@tanvir-VirtualBox: ~/Desktop/tanvir_2019
step == 1173
W == [[0.314112 0.710912 0.322451 0.421123 0.221744],
  [ 0.322124 0.721223 0.621123 0.323211 0.521221],
  [ 0.622142 0.522113 0.744133 0.845221 0.622512],
  [ 0.844121 0.623322 0.652333 0.744552 0.85112],
  [ 0.541122 0.452366 0.756655 0.845112 0.774551],
...........................
  [ 0.841142 0.841221 0.623322 0.414412 0.621211]]
Accuracy Percentage == 0.90
Possition == [ True True True True True True True True True
True True True False True True True False True True True]
tanvir@tanvir-VirtualBox:~/Desktop/tanvir_2019$
```

Figure 4.14 Proposed system equation (37) RL algorithm result



```
tanvir@tanvir-VirtualBox: ~/Desktop/tanvir_2019
step == 1034
W == [[ 0.537442 0.542244 0.667411 0.755441 0.655114],
   [ 0.621147 0.442332 0.774451 0.635587 0.685211],
   [ 0.551142 0.632541 0.365547 0.784121 0.621422],
   [ 0.741224 0.632214 0.542211 0.631125 0.754122],
   [ 0.741255 0.657411 0.686322 0.365233 0.652122],
 ...........................
   [ 0.625544 0.363374 0.452211 0.663221 0.744522]]
Accuracy Percentage == 0.75
Possition == [ True True True True True False False True True
True True True False True True False True True False True]
tanvir@tanvir-VirtualBox:~/Desktop/tanvir_2019$
```

Figure 4.15 Existing system equation (36) RL algorithm result

4.3 Evaluation

In this chapter, we have presented the performance results for each algorithm on the Tiling test sets with respect to the aforementioned evaluation metrics. We determine the effectiveness of the Tiling algorithm with respect to another word2vec-based approach to segmentation, the Word2vec-Tiling algorithm. For our dataset, we first performed the preprocessing steps discussed in chapter 2.

We have calculated the frequency of the word to obtain the semantic relation of the text. For this, we have used the Word2vec-Tiling method. Next, we ran the five segments (divided by 20) by employing the default values for any system parameters and by letting the systems estimate the number of thematic boundaries. In order to simplify the analyses, we provide the algorithms with the number of segments that they had to identify.

4.3.1 Experimental Evaluation

Data collection is very important for Web Crawler research because the evaluation of the research depends on it. Therefore, in the proposed method we have defined Word2vec-Tiling to collect data properly. Our approach uses the Word2vec-Tiling to capture the semantic features between words in the selected text. In the previous research on word2vec, most of the information in their collected data is not related to their input keyword because the word2vec technique is based on a feed-forward. For this reason, most of the time, it is unable to show the perfect result because the word2vec cannot detect the gap between texts. However, using the Tiling method, we can solve this problem, because it is possible to identify the value of the gap between the texts through the Tiling method. Therefore, in the proposed method, we defined Word2vec-Tiling algorithm for calculating the gap in text. We have described the procedure of the Word2vec-Tiling (CBOW) algorithm in chapter 3 by pseudocode and in the implementation chapter we have shown in figure 4.1, 4.2, 4.3 and 4.5. In figure 4.3, we have shown 10000 text results. In figure 4.1, we have divided the document separate process. We have chosen Williams Shakespeare life story as our input data. In figure 4.2, we calculated the gap of the document using the proposed Tiling method, except the stop word from the previous document. In figure 4.3, we have described our input document working process. According to figure 4.3, there are consistently words show in figure 4.1, so the figure 4.3 is not changing. On the other hand, figure 4.4, each word is vibration for the gap between the neighbor words. According to in figure 4.4, the lower efficiency of the vibrator indicates the negative value and upper efficiency of the vibrator indicates are positive value. That means our proposed algorithm is working successfully. Finally, we have collected the keyword-related data using the Tiling method shown in figure 4.6. In figures 4.5 and 4.6, we have shown the difference between word2vec using equation (3) and Word2vec-Tiling method using equation (35).

On the other hand, in Word2vec-Tiling (Skip-gram) model we have described in chapter 3 in chapter 3.3.4 by pseudocode. In the proposed method, by changing a previous Skip-gram model equation and we implement it, in the proposed method. The gap between texts in the previous equation (23) has not been used. Therefore, it is not possible to get the correct result of text word without there gap calculation. We calculated the value of the gap between the texts through the CBOW model and used it in the skip-gram model equation (30). In the implementation chapter figure 4.9, we have described the results of our using equation. In figure 4.7 we have used a text as an example. There the first word has been called "central word", according to figure 4.7. The "central word" means the first semantic word of input text, other than this (such as, central word), the text of word cannot be formed or text structure is not possible. Then the next word is "context word", it can be a partition in a few ways. In the proposed method, we divided the "context words" into two ways. Later we used the “center word” to be divided into [1 0 0 0 0 0 0]. Our input total word number 7 is so in the first step, central word 1 and all other 0. The next step in "context word" is "center word". Here the key value of "center word" is [1 0 0 0 0 0 0], but the value of the next two "center word" will be used. Because the text structure of the sentence "context word" depends on the next two "center words". We described the entire work process in figure 4.8 also show in figure 4.9 and 4.10. In figure 4.9, we have used 5 keyword’s. According to figure 4.9, we have seen the university related data accuracy higher because the university related data has correctly selected there input data or keyword. On the other hand, the existing method results in figure 4.10, their data accuracy is lower because correctly, the input data unable to select their keyword’s, so their values have decreased. In table 4.1, we describe the data quality of our CBOW and Skip-gram models. In addition, we discussed the differences between the existing method and the proposed method in table 4.2. According to the table in the existing method, the Precision, Recall, Accuracy and F-score values are less than the proposed method. So firmly we can say our proposed method equation performance is better than other methods.

Table 4.1 The difference between CBOW and Skip-gram using the tiling method

Keyword's	CBOW	Skip-gram

Love	modest=0.874555
	lust=0.623111
	remorse=0.366633
		dandle=0.455522
		canoodle=0.512011
		consistent=0.584411

War	hostility=0.622232
	estimated=0.754445
	prehistoric=0.611222
		significantly=0.655525
		destructive=0.745544
		morph=0.544414

University	integrated=0.744451
	vein=0.644114
	swamp=0.821142
		cavity=0.841451
		immune=0.685411
		radiation=0.655544

Sky	concentrating=0.855512
	deputize=0.674424
	follicle=0.547744
		engagement=0.665542
		exoplanet=0.744511
		aboriginal=0.776122

Life	existence=0.774522
	molecules=0.855512
	tedious=0.699881
		subsistence=0.877441
		aggravating=0.665514
		purification=0.874451

Table 4.2 Performance Measures

Existing System				
Algorithm	**Precision**	**Recall**	**Accuracy**	**F-score**
TF-IDF	46.3	42.1	51.4	44.1
Reinforcement Learning (RL)	63.1	53.2	61.2	57.7
Proposed System				
Algorithm	**Precision**	**Recall**	**Accuracy**	**F-score**
TF-IDF	**64.2**	**71.1**	**74.2**	**67.4**
Reinforcement Learning (RL)	**81.2**	**72.2**	**78.4**	**76.4**

Table 4.3 Each document separate value using TF algorithm

Purpose	Sky	War	Love	University	Life
Existing Method TF algorithm result	0.121142	0.233221	0.322122	0.28444	0.111021
Proposed Method modify TF algorithm result	**0.372211**	**0.522214**	**0.462212**	**0.451112**	**0.212232**

Table 4.4 Each document separate value using RL algorithm

Purpose	Sky	War	Love	University	Life
Before using "$\delta(s_{n+1},a_n)$" function (Existing Method)	0.352211	0.222322	0.322421	0.201522	0.331132
After using "$\delta(s_{n+1},a_n)$" function (Proposed Method)	**0.411222**	**0.408854**	**0.516628**	**0.414512**	**0.421422**

4.3.2. TF-IDF Algorithm Experimental Evaluation

In figure 4.12, we shows our TF-algorithm overall performance. When a user searches on his document, he will get results in a short time because our algorithm does not face any overlap and can show the correct results. In the evaluation, we will explain it more detail by table 4.3. First, we have selected five keyword's (Sky, War, Love, University, Life) and we have collected data from the web page. We have provided data with 100 hyperlink related data per webpage according to each keyword. We collected 500-hyperlink information from each of keyword's. We have created a data tree using the Breadth First Search (BFS) method. By using Word2vec-Tiling we have collected the keyword related data. Therefore, we use the TF algorithm to calculate the weight value of each hyperlink data. The results of the TF algorithm used in the existing and proposed systems are described in the table 4.3. Here we have described each keyword's category's average value different in existing method TF equation (25) and proposed method TF equation (31). Here we can see in proposed method keyword category's value is higher than existing method by improving equation.

The semantic relation approach provides a novel way of glancing at the matter of text deliverance, which links it with a lot of recent research in oratory. As word embeddings approach of TF-IDF algorithm emphasizes, the semantic relation approach to TF provides a different approach to scoring matches between documents and queries, and the prospect is that the word embeddings modeling establishment enhances the weights that are used, and consequently the representation of the TF model. The major manifestation is the estimation of the document model, such as choices of how to easy or smooth it effectively. The model has achieved very good salvation results. Compared to other probabilistic approaches, such as the existing method in TF from Chapter 2, the main difference originally appears to be that the TF approach does away with apparently modeling relevance. The TF semantic relation comprises that documents and expressions of information needs are objects of the same type, and comprises their match by exposing the tools and methods of text embeddings modeling from Natural Language Processing (NLP). The resulting model is mathematically appropriate, theoretically simple, intuitively appealing, and computationally tractable. Our TF-IDF semantic relation using has some limitation. Our TF-IDF is based on the Word2vec-Tiling model, therefore it does not capture the position in the text in different documents, so, the level of accuracy is less (show in figure 4.12) than the other algorithms. In figure 16, we have shown our TF-IDF algorithm semantic relation process.

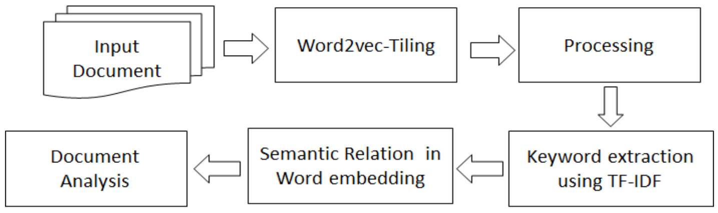

Input Document
Word2vec-Tiling
Processing
Keyword extraction using TF-IDF
Semantic Relation in Word embedding
Document Analysis

Figure 4.16 TF-IDF algorithm semantic relation working process

4.3.3. Reinforcement Learning Algorithm Experimental Evaluation

In this research, we applied the Word2vec-Tiling to capture the semantic features of words in text. However, we used the RL algorithm to calculate the semantic features of the document.

Where δ, is used to find out the value of the words separately, and $\delta(s_{n+1},a_n)$ [17] is used to identify the difference in the text. In the baseline method, the word value is randomly captured, and the total word gap number of the text is estimated (20 text gap); for this reason, their total word value is not clear (gap value fixed), and the accuracy rate is lower [20].

However, using "$\alpha\delta(s_{n+1},a_n)$", the proposed system demonstrated increased accuracy by 15% by using Word2vec-Tiling. Here, we have calculated the text gap value in the words. By calculating the value of the text, the overall performance of the accuracy rate is quite good, as we have described through figure 4.14, in the implementation chapter. The best aspect of the Word2vec-Tiling method is the ease of achieving the correct value of each word in the text. In table 4.4, we described the properties before and after using "$\alpha\delta(s_{n+1},a_n)$" in the RL-algorithm flowing equations (36) and (37).

4.3.2 Data Set

At the beginning of any research, collecting data is especially important. In table 4.5, we have described the dataset, data types, and data amount of both our method and that of existing research. We have collected 20 types of data from five keywords (Love, War, University, Sky, and Life).

Table 4.5 Comparison of different data sources

Title	Data Set	Data amount	Data amount
Existing Method	Wikipedia and social media	Average 2.47MB 600000 texts	These texts are short, some disorderly and have a lot of modern and Network Technology.
Proposed Method	Nobel, NASA, History and Wikipedia	Average 5.90MB 1000000 texts	1. Long text, formal format, and clear semantics. 2. Some texts are short and some disorderly and have a lot of modern and old text.

4.3.3 Performance Measurement

This chapter reports on the performance of our system and the existing one during the crawl. In our analysis of web crawler performance, we contrast it with the performance of the Google, Internet archive crawlers, and the existing system. We make no attempt to adjust for different hardware configurations because the research's describing the two other crawlers do not contain enough information enabling us to do so. Our main intention in presenting this performance data is to convey the relative speeds of the various crawlers. The overall performance comparison of the proposed and existing methods is described in table 4.6. According to table 4.6, we can say the proposed method's HTTP request rate is much better than the existing method. Because the proposed method uses the Python programming language SciPy, Scrapy, and NLTK library. We know that SciPy, Scrapy, and NLTK library is faster for collecting data from the web. In this reason, in the proposed method HTTP data collection rate is higher.

Table 4.6 Performance area

Category		System Descriptions	
Index	**Title**	**Existing Method**	**Proposed Method**
1	Programming Language	Java	Python and Matlab
2	HTTP request in 7 days	75.6 million [11], [24]	**84.2 million**
3	Download rate	120 document/sec and 1682KB/sec	120 document/sec and 1872KB/sec

Chapter 5

Conclusion

5.1 Conclusion

In this thesis, topic classification based on deep learning is researched and the influence of different word embedding on classification accuracy is compared. An improved word embedding model Word2vec-Tiling is proposed based on word2vec and topic model. The experiments demonstrate that the model outperforms word2vec and Tiling using two different text classification datasets. The performance of our algorithm is quite satisfactory show in figure 5.1, represented proposed method and in figure 5.2, represented existing method, in particular, it yields the best results reported so far on the segmentation of this text collection. This thesis, the main topic of semantic relation based on Reinforcement Learning (RL) is studied and the influence of different word embedding on keyword accuracy is compared. In this thesis, we use two methods to reach our goal, one is Word2vec-Tiling and the other is RL algorithm, that is, we use Word2vec-Tiling to convert text to word vector and improve the RL algorithm from two aspects, and finally, we extract semantic relation with the improved method. Our proposed Word2vec-Tiling method advantage is, this method can differentiate the gap between text, following the tiling method. So, Word2vec-Tiling method assumes that a set of text items is in use during the course (text word) of a given subtopic discussion, and when that subtopic changes, a significant proportion of the vocabulary changes as well. As a result, users can find out easily the related information about searched keywords without any overloading, which provides a better result than the existing method. Our experimental results are compared with the five data sets. Experimental results suggest that our algorithm can show the semantic features of the web pages very quickly and properly.

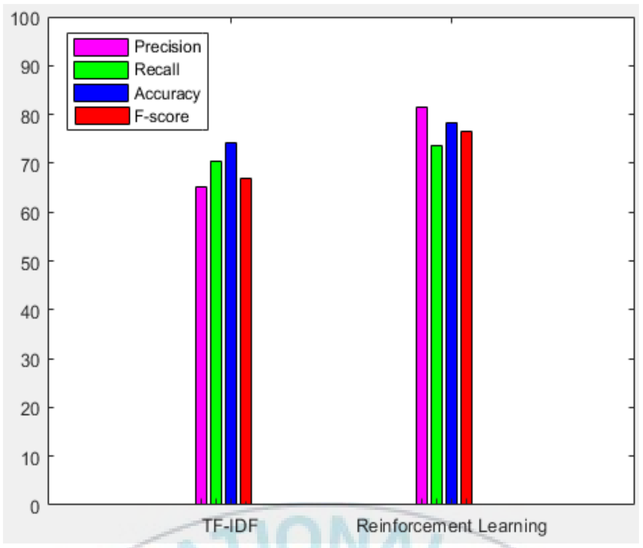


Figure 5.1 Proposed method performance overview

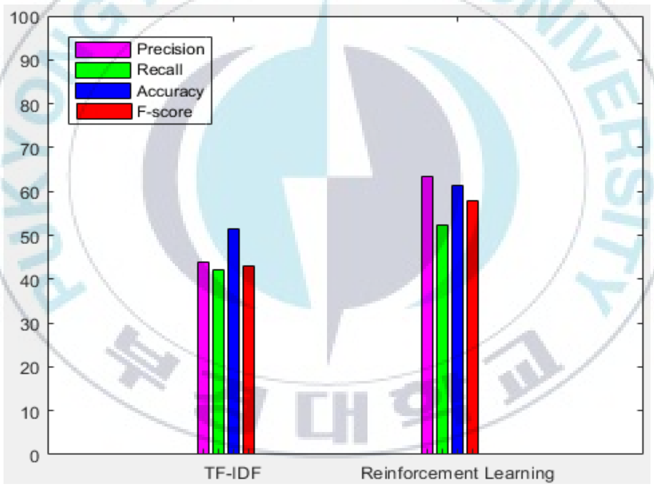


Figure 5.2 Existing method performance overview

5.1 Future Work

Although the initial results are encouraging, there is still a lot of work to do for improving the crawling efficiency. A major open issue for future work is to do extension test with large volume of web pages. In Word2vec-Tiling at this point, web services are solely modeled as their names of the service description files and mapped as “text or words”. It could be extended in the future by including more components of service description files. Moreover, service embedding is applied to generate documents for simplicity. Future work could be conducted such that more real-world service composition methods are used to generate different types of documents. The dependency of the proposed method is accuracy of the dictionary used in the method.

Appendix A

A.1 Practical Web Crawling Issues

When we tested our implementation, which is described in Chapter 4, we found that there were several problems of Web crawling that did not become evident until a large crawl was executed. Our experiences arise from several crawls of the Life, University, Love, War, and Sky Web carried out during this thesis.

We are interested in documenting these problems for two reasons:

- To help other crawler designers, because most of the problems we found are related to the characteristics of the Web, independent of the Web crawler architecture chosen.

-To encourage Web application developers to check their software and configurations for compliance to standards, as this can improve their visibility on search engine's results and attract more traffic to their Web sites.

A.2 Web Server Administrators Concerns

Web crawlers prompted suspicion from Web site administrators when they first appeared, mostly because of concerns about bandwidth usage and security, and some of those concerns are still in place today. In our experience, repeated access to a Web page can trigger some alarms on the Web server, and complaints from its administrator.

We consider that the two most important guideline given here:

- A crawler must identify itself, including an e-mail address for contact, or some Web site administrators will send complaints to the listed owner of the entire originating network segment.
- A crawler must wait between repeated accesses to the same Web site.

These guidelines are even more important if we consider that many host names point to the same IP, usually belonging to a Web hosting provider, and in general several Web sites are hosted by a few physical servers. Being unpolite with a Web site can result in being banned from all the Web sites hosted by the same ISP.

Recommendation or Our Solution: The crawler should avoid overloading Web sites, and it must provide an e-mail address in the from HTTP header, and/or a Web site address as a comment in the user-Agent HTTP header.

A.3 Wrong DNS Web records

Consider the scenario depicted in figure A.1

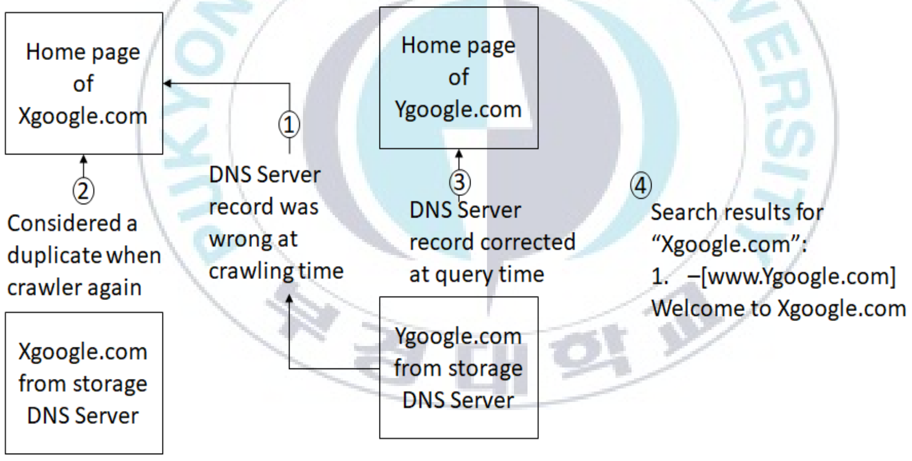

Home page of Xgoogle.com
①
DNS Server record was wrong at crawling time
Home page of Ygoogle.com
②
Considered a duplicate when crawler again
③
DNS Server record corrected at query time
④
Search results for "Xgoogle.com":
1. –[www.Ygoogle.com]
Welcome to Xgoogle.com
Xgoogle.com from storage DNS Server
Ygoogle.com from storage DNS Server

Figure A.1 A misconfiguration in the server record for "Ygoogle.com" resulted in the wrong contents being assigned to its URL

1. At crawling time, the server record for Ygoogle.com pointed to the website of Xgoogle.com, so the contents of the later were indexed as if there URL was Ygoogle.com. This Domain Name Server (DNS) misconfiguration can be accidental or malicious.

2. When the home page of Xgoogle.com was downloaded, its contents were found to be a duplicate of Ygoogle.com, so the pages of Xgoogle.com were not downloaded again.

3. The wrong DNS server record of Ygoogle.com was fixed later.

4. In the search results, when users search for "Xgoogle", they can be mistakenly redirected to the Web site of "Ygoogle".

Recommendation or Our Solution: It is possible for Web site administrators to avoid these kind of problems by a careful configuration of virtual hosts. Any access to the IP address of the Web server that does not contain a known Host field in the request, should be redirected to the default virtual host, referencing the later by its canonical name.

A.4 Range Errors

To ensure a good coverage of the Web, we must limit the amount of data that is downloaded from every Web server. This can be done by limiting both the maximum page size, and the number of Web pages that are downloaded from a single Web site. We limit page size usualy to a default of 300-400 Kb per Web page. We consider that this should capture enough keywords to index each document. To inform the Web servers of the download limit, we use the HTTP Range header:

```
GET / page.html HTTP/1.1
Range: 0-400000
```

However, a few Web sites return a response code 416 (range error). We have found that these responses correspond to files that are smaller than the desired size. This is not correct, because the HTTP specification indicates that "if the [second] value is greater than or equal to the current length of the entity-body, last-bytepos is taken to be equal to one less than the current length of the entity- body in bytes".

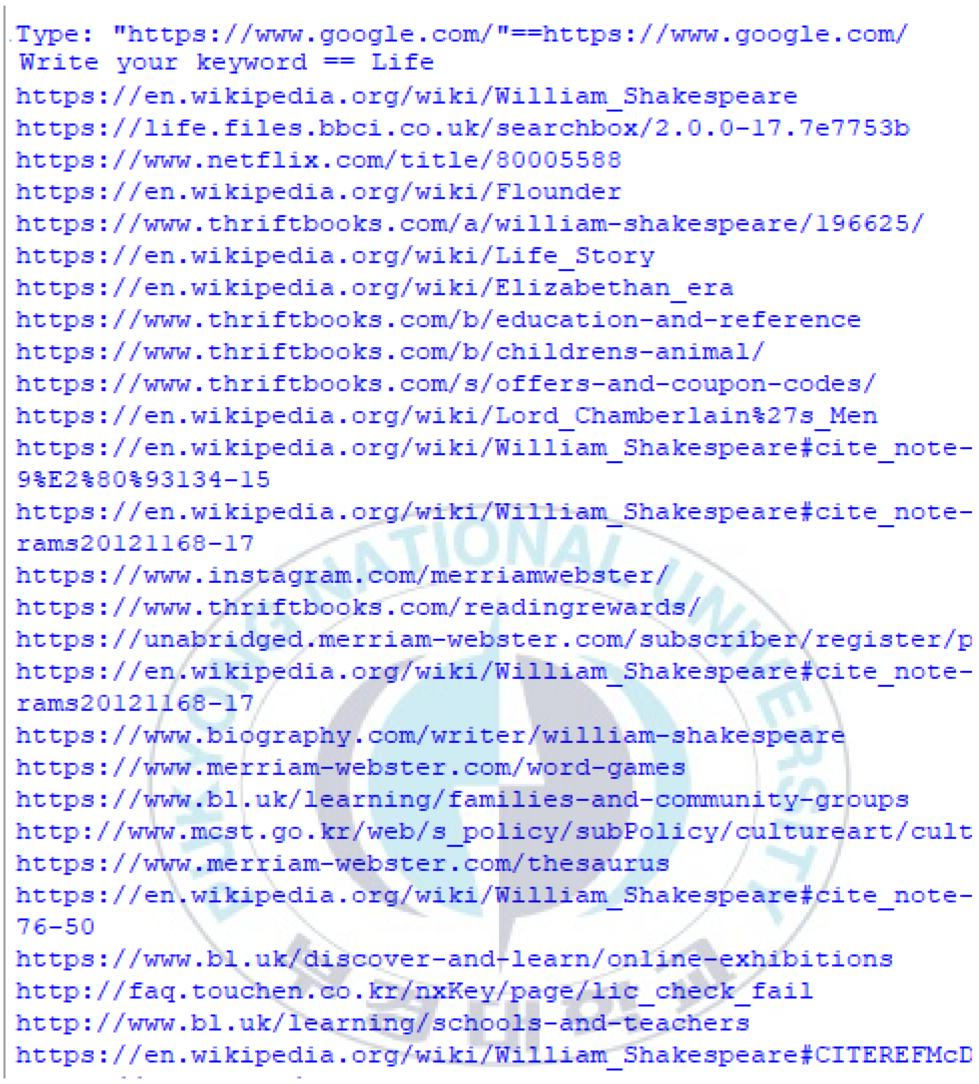
Recommendation or Our Solution: In the case of range errors, a second attempt for download could be made without the Range header. In all cases, the Web server may ignore the range, so the Web crawler must be prepared to disconnect from the Web server, or discard part of the contents, if the server sends a long document.

A.5 Found where you mean error

It is hard to build a Web site without internal broken links, and the message shown by Web servers when a page is not found, such as, when the Web server returns a 404 (not found) response, is considered by many Web site administrators as too annoying for users.
Indeed, the default message looses the context of the Web site, so the Web site administrators of some Web sites prefer to build error pages that maintain visual and navigational consistency with the rest of their Web sites.
The problem is that in many cases the response for a page that does not exists is just a normal redirect to a custom-built error page, without the response header signaling the error condition. D. Zhao et al. refer to these error pages as “soft-404”, and observe that about 29% of dead links point to them.
The indexing process could consider a redirect to a “soft-404” error page as a link between the URL in which the page was not found and the error page, and this can increase the score of the later.
Recommendation or Our Solution: Web site administrators should configure their Web servers in such a way that the error messages have the correct response codes signaling the error conditions. Servers can be tested by Web crawlers issuing a request for a known non-existent page (such as, an existent URL concatenated with a random string) and checking the result code.

A.6 Data collection from Web pages

By solving the problem described above, we have collected data about our keyword’s from web pages. We have described our data collection process from web pages in figure A.2.



```
Type: "https://www.google.com/"==https://www.google.com/
 Write your keyword == Life
https://en.wikipedia.org/wiki/William_Shakespeare
https://life.files.bbci.co.uk/searchbox/2.0.0-17.7e7753b
https://www.netflix.com/title/80005588
https://en.wikipedia.org/wiki/Flounder
https://www.thriftbooks.com/a/william-shakespeare/196625/
https://en.wikipedia.org/wiki/Life_Story
https://en.wikipedia.org/wiki/Elizabethan_era
https://www.thriftbooks.com/b/education-and-reference
https://www.thriftbooks.com/b/childrens-animal/
https://www.thriftbooks.com/s/offers-and-coupon-codes/
https://en.wikipedia.org/wiki/Lord_Chamberlain%27s_Men
https://en.wikipedia.org/wiki/William_Shakespeare#cite_note-
9%E2%80%93134-15
https://en.wikipedia.org/wiki/William_Shakespeare#cite_note-
rams20121168-17
https://www.instagram.com/merriamwebster/
https://www.thriftbooks.com/readingrewards/
https://unabridged.merriam-webster.com/subscriber/register/p
https://en.wikipedia.org/wiki/William_Shakespeare#cite_note-
rams20121168-17
https://www.biography.com/writer/william-shakespeare
https://www.merriam-webster.com/word-games
https://www.bl.uk/learning/families-and-community-groups
http://www.mcst.go.kr/web/s_policy/subPolicy/cultureart/cult
https://www.merriam-webster.com/thesaurus
https://en.wikipedia.org/wiki/William_Shakespeare#cite_note-
76-50
https://www.bl.uk/discover-and-learn/online-exhibitions
http://faq.touchen.co.kr/nxKey/page/lic_check_fail
http://www.bl.uk/learning/schools-and-teachers
https://en.wikipedia.org/wiki/William_Shakespeare#CITEREFMcD
```

Figure A.2 Data collect process from the web page using keyword

A.7 Data value different between Existing and Proposed method

The main and important thing on this thesis is that we recommend by Word2vec to Word2vec-Tiling. Through, Table A.1, we describe the difference between the proposed and the existing method.

Table A.1 Same data Existing but Proposed method word2vec and word2vec-tiling value different.

Keyword's	**Category**	**Existing Method**	**Proposed Method**
Sky	atmosphere	0.421321	**0.521421**
	stellar	0.365421	**0.632145**
	breeze	0.362211	**0.614441**
War	struggle	0.452122	**0.623214**
	combat	0.012877	**0.321552**
	hostility	0.321132	**0.512211**
Love	mercy	0.366532	**0.463222**
	rapacity	0.623111	**0.652112**
	devotion	0.321452	**0.612212**
University	prospective	0.211221	**0.310011**
	intervention	0.312135	**0.312221**
	essential	0.421211	**0.481222**
Life	cognition	0.321122	**0.381222**
	longevity	0.341141	**0.361122**
	conduct	0.421121	**0.512211**

References

[1]S. Saranya, B.S.E. Zoraida, and P.V. Paul, "A Study on Competent Crawling Algorithm (CCA) for Web Search to Enhance Efficiency of Information Retrieval," *Proceeding of Artificial Intelligence and Evolutionary Algorithms in Engineering* Systems, Springer, New Delhi, pp. 9-16, 2015.

[2]K.S. Kim, K.Y. Kim, K.H. Lee, T.K. Kim, and W.S. Cho, "Design and implementation of web crawler based on dynamic web collection cycle," *Proceeding of The International Conference on Information Network*, IEEE, pp. 562-566, 2012.

[3]Y. Kim, H. Hong, and M. Chung, "Application of Cohesion Devices for Improvement of Distributional Representation," *Proceeding of The 14th International Conference on Multimedia Information Technology and Applications (MITA),* pp. 84-87, 2018.

[4]M.Y. Ivory and M.A. Hearst, "Improving web site design," *IEEE Internet Computing 2*, Vol. 6, No. 2, pp. 56-63, 2002.

[5]D. Debraj and P. Das, "Study of deep web and a new form based crawling technique," *International Journal of Computer Engineering and Technology (IJCET),* Vol. 7, No. 1, pp. 36-44, 2016.

[6]Z. Guojun, J. Wenchao, S. Jihui, S. Fan, Z. Hao, L. Jiang, et al., "Design and application of intelligent dynamic crawler for web data mining," *Proceeding of 2017 32nd Youth Academic Annual Conference of Chinese Association of Automation (YAC) IEEE*, pp. 1098-1105, 2017.

[7]K.A. Pakojwar, R.S. Mangrulkar, and V.G. Bhujade, "Web data extraction and alignment using tag and value similarity," *Proceeding of 2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS),* pp. 1-4, 2015.

[8]S. Kolhatkar, M.M. Pati, M.S. Kolhatkar, and M.S. Paranjape, "Emergence of Unstructured Data and Scope of Big Data in Indian Education," *International Journal of Advanced Computer Science and Applications (IJACSA),* Vol. 8, No. 1, pp. 150-157, 2017.

[9]M. Afsharizadeh, H. Ebrahimpour-Komleh, and A. Bagheri, "Query-oriented text summarization using sentence extraction technique," *Proceeding of 4th International Conference on Web Research (ICWR),* pp. 128-132, 2018.
[10]S. Ringe, N. Francis, and A.H.S.A. Palanawala, "Ontology Based Web Crawler," *International Journal of Computer Applications in Engineering Sciences*, Vol. 2, No. 3, pp. 194-197, 2012.
[11] L. Jiang, Z. Wu, Q. Feng, J. Liu, and Q. Zheng, "Efficient deep web crawling using reinforcement learning," *Proceeding of Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Berlin, Heidelberg*, pp. 428-439, 2010.
[12] Y. Kim, B. Kim, and M. Chung, "Unstructured data analysis and multi-pattern storage technique for traffic information inference," *The Journal of Multimedia Information System,* Vol. 21, No. 2, pp. 211-223, 2018.
[13] R. Jason and A. McCallum, "Using reinforcement learning to spider the web efficiently," *Proceeding of International Conference on Machine Learning (ICML),* Vol. 99, 1999.
[14] M. Liu, Y. Jiangang, "An improvement of TFIDF weighting in text categorization", *International computer science and information technology (ICCTS),* Vol. 47 No. 9, pp. 44-47, 2012.
[15] M. Shi, J. Liu, D. ZHou, M. Tang, B. Cao, "WE-LDA: A Word Embeddings Augmented LDA Model for Web Services Clustering", *Processing of IEEE International Conference on Web Services (ICWS),* pp. 9-16 2017.
[16] F. Harrag, A. Hamdi-Cherif, A. S. Al-Salman, "Comparative study of topic segmentation Algorithms based on lexical cohesion: Experimental results on Arabic language", *Arabian Journal for Science and Engineering*, Vol. 35, No.2, pp.183-202, 2010.
[17] A. M. Tanvir, M. Chung, "Design and Implementation of an Efficient Web Crawling Using Neural Network", *Proceeding of The International Conference on Computer Science and its Applications (CSA),* December 17-19, Kuala Lumpur, Malaysia, 2018.
[18] X. Wang, L. Xie, B. Ma, E. S. Chng, H. Li, "Phoneme lattice based TextTiling towards multilingual story segmentation", *Proceeding of Eleventh Annual*

Conference of the International Speech Communication Association Makuhari, Chiba, Japan, pp. 1305-1308, 2010.

[19] M. Riedl and C. Biemann "TopicTiling: a text segmentation algorithm based on LDA", *Proceedings of ACL 2012 Student Research Workshop. Association for Computational Linguistics*, 2012.

[20] A.M. Tanvir and M. Chung, “Design and Implementation of Web Crawler utilizing Unstructured data”, *Journal of Korea Multimedia Society*, Vol. 22, No. 3, pp. 374-385, 2019.

[21] D. Zhao, N. Du, Z. Chang and Y. Li, "Keyword extraction for social media short text." *Procedding of 2017 14th Web Information Systems and Applications Conference (WISA),* IEEE, pp. 251-256, 2017.

[22] G. Saurabh. and V. Khare, "Blazingtext: Scaling and accelerating word2vec using multiple gpus" *Proceedings of the Machine Learning on HPC Environments*, ACM, 2017.

[23] C.D.C. Ta and T.P. Thi, “Identifying the semantic relations on unstructured data”, *International Journal of Information Sciences and Techniques (IJIST),* Vol.4, No.3, pp. 1-10, 2014.

[24] X. Zhang and M. Lapata, "Sentence Simplification with Deep Reinforcement Learning", *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 584-594, 2017.

[25] S. Qaiser and R. Ali, "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents", *International Journal of Computer Applications (0975 - 8887),* Vol. 181, No. 1, pp.25-29, 2018.

Acknowledgements

First of all, I am very pleased to convey my sincere gratefulness and indebtedness to Professor Mokdong Chung, my advisor, for giving me all the necessary research facilities, valuable suggestion, guidance, and financial support during my study. My advisor is the man who saved me in Korea as well as my study so that I am really grateful to him. No words are enough to express gratitude to him. I express my regards to all committee members Prof. Jeong-Mo Yeo and Prof. Kyung-Ryong Seo for their valuable comments and suggestions. I would also like to thank the faculty and staff members in the Department of Computer Engineering for their necessary help throughout my study.

I also express my gratitude to my elder lab mates, Ph.D. candidate Yonghoon Kim, my study would not be possible without his recommendation. I really appreciated that the numerous help is done by my labmates Master degree friend, Hyungi Hong, with generous help, encouragement and assisted me to complete my research.

Special thanks to all my friends and lab mates especially to senior Ph.D. candidate Yonghoon Kim, Master classmate Hyungi Hong, and Jaehyeok Choi and undergraduate students Seoyoung Jang, Seunghyeon Baek, Park JaeWan and Kim SangYul for their care, help and support.

Finally, I would like to convey my deepest gratefulness to my parents for their great sacrifices and it was nearly impossible to complete my work without their support during my study in South Korea. I would not be capable to come here and finish work without their encouragement and sacrifice.