ISSN 2229-6956(ONLINE)
DOI: 10.21917/ijsc.2010.0004

MULTI-DOCUMENT TEXT SUMMARIZATION USING CLUSTERING TECHNIQUES AND LEXICAL CHAINING

S.Saraswathi[1] and R.Arti[2]
[1]*Department of Information Technology, Pondicherry Engineering College, Pondicherry, India*
E-mail: swathimuk@yahoo.com
[2]*Microsoft R&D India Private Limited, Hyderabad, India*
E-mail: arti_lakshmi@yahoo.co.in

Abstract
This paper investigates the use of clustering and lexical chains to produce coherent summaries of multiple documents in text format to generate an indicative, less redundant summary. The summary is designed as per user's requirement of conciseness i.e., the documents are summarized according to the percentage input by the user. For achieving the above, various clustering techniques are used. Clustering is done at two levels, one at single document level and then at multi-document level. The clustered sentences are scored based on five different methods and lexically linked to produce the final summary in a text document.

Keywords:
Hierarchical Clustering, Lexical Chaining, Precision, Recall

1. INTRODUCTION

Text summarization is the process of distilling the most important information from a source to produce an abridged version for a particular user or task. The rapid growth of the Internet has resulted in enormous amounts of information that has become more difficult to access efficiently. Internet users require tools to help and manage this vast quantity of information. This paper discusses on a method to create an efficient and effective tool that is able to summarize multiple documents with good efficiency. With the advent of efficient search engines and abundance of online information, it becomes necessary to give concise answers to user queries. Keyword search is the most popular information discovery method because the user does not need to know either a query language or the underlying structure of the data. The search engines available today provide keyword search on top of sets of documents. As the number of documents available on users' desktops and the Internet increases, so does the need to provide high-quality summaries in order to allow the user to quickly locate the desired information. Users are presented with vast information which suffers from redundancy and irrelevance. Also there may be situations where time and space may be of immense concern. Under such circumstances, summarized text comes very handy. This efficiency is necessary in Internet search applications where many large documents may need to be summarized at once, and where the response time to the end user is extremely important. Hence summarization is usually coupled with search engines to provide effective output. Text summaries can also be categorized into two types:

Query-relevant summaries: The summary is created based on the terms in the input query. As they are "query-biased", they do not provide an overall sense of the document content.

Generic summaries: A generic summary provides an overall sense of the document's contents and determines which category it belongs to. A good generic summary should contain the main topics of the document while keeping redundancy to a minimum.

MEAD a summarization tool uses Sentence Extraction concept. It involves assigning salience scores to some units–usually sentences or paragraphs–of a document or a set of documents and extracts these with the highest scores [1]. MEAD is a publicly available toolkit for multi-lingual summarization and evaluation. The toolkit implements multiple summarization algorithms (at arbitrary compression rates) such as position-based, TF*IDF, and query-based methods. Methods for evaluating the quality of the summaries include co-selection (precision/recall, kappa, and relative utility) and content-based measures (cosine, word overlap, bigram overlap). MEAD is written in Perl and requires several XML-related Perl modules and an external software package to run. Because of the inconsistencies in encodings, it has been tested in Mandarin Chinese, on certain versions of the Solaris operating system and some versions of Linux.

The goal of SUMMARIST, a summarization tool is to create summaries of arbitrary text in English and selected other languages [2]. By eschewing language-specific methods for the relatively surface-level processing, it is possible to create a multi-lingual summarizer fairly easily. Eventually, however, SUMMARIST will include language-specific techniques of parsing and semantic analysis, and will combine robust NLP processing (using Information Retrieval and statistical techniques) with symbolic world knowledge embodied in the concept thesaurus SENSUS [3,4], derived from WordNet [5] and augmented by dictionaries and similar resources, to overcome the problems endemic to either approach alone. These problems arise because existing robust NLP methods tend to operate at the word level, and hence miss concept-level generalizations (which are provided by symbolic world knowledge), while on the other hand symbolic knowledge is too difficult to acquire in large enough scale to provide adequate coverage and robustness.

The automatic text summarizer proposed in this paper discusses the use of multiple clustering techniques for reducing redundancy. Hierarchical Clustering at single document level and Fuzzy C Means Clustering for topic word identification at multi-document level have been employed. The application of semantic cosine similarity which takes care of the clustering based on the presence of words, meanings and related words ensures that clustering is more meaningful. The ranking of sentences within each cluster is done on five different dimensions which enable to retrieve the most deserving sentences into the summary. Finally the sentences with a higher rank within each cluster are picked and lexically chained based on the topic to which it corresponds according to an ordering in the topic word and their location in the sentence. The techniques

together produce a summary that captures the important sentences to a high precision level. Further it overcomes the prime drawbacks of other existing systems like redundancy, lack of cohesion, etc. The approach has been tested in Software Engineering domain and a high precision level in the resulting summary has been observed.

Section 2 gives the overall structure of the system and description about various modules comprising the system. Section 3 deals with the performance measures and their interpretation. Section 4 gives the conclusion and scope for future work.

2. PROPOSED SYSTEM

The overall architecture of the proposed system is depicted in Fig.1.

Pre-processing: The documents to be summarized are presented to the POS Tagger.

Automatic Text Summarizer: The following steps are involved in this module

1. The tagged documents are input to the Automatic Text Summarizer
2. Hierarchical Clustering is applied at single document level, The Hierarchical Clustering for the multiple documents is carried out parallel and the output is a set of clusters from all the documents. Semantic Cosine Similarity is used for Hierarchical Clustering i.e. clustering is accomplished by comparing the presence of words and their meanings and related words
3. The clusters from step 2 are subjected to Topic Word and Subtopic word identification using Fuzzy C Means Clustering. A sentence is categorized into a particular topic/subtopic based on the presence of certain words. The sentences are reclustered based on the topic/subtopic and hierarchical clustering index
4. The sentences in the newly formed clusters are ranked on 5 different dimensions and a rank is assigned for each sentence in the cluster
5. The sentences with a high rank are picked from each cluster according to the percentage of summarization specified by the user
6. The sentences which have been picked are lexically chained according to the order of topic words to which they have been categorized and according to their line number in the original document

The system is organized into the following phases:

1. Hierarchical Clustering using Semantic Cosine Similarity
2. Fuzzy C Means Clustering for topic word identification
3. Sentence Ranking
4. Lexical Chaining based on Topic Word order

The system has been decomposed into the following modules like cosine similarity measure, grouping of clusters and lexical chains as depicted in Fig.2, Fig.3 and Fig.4.

2.1 PRE PROCESSING

This work deals in summarizing text documents related to Software Engineering domain. The pre-processing involves collection of relevant documents.

2.2 PARTS OF SPEECH TAGGING

The documents collected are tagged using a POS tagger, namely the tree tagger [6]. For example, if the input document contains the sentence, "Users will have lots of questions and software problems which lead to the next phase of software." The tagged output will be as follows:

Users	NNS
Will	MD
Have	VH
Lots	NNS
Of	N
Questions	NNS
And	CC
Software	NN
Problems	NNS
Which	WDT
Leads	VVZ
To	TO
The	DT
Next	JJ
Phase	NN
Of	IN
Software	NN
.	SENT

2.3 COSINE SIMILARITY

Cosine similarity is a technique to find out the similarity between pairs of sentences in a document [7]. First of all, the keywords in each pair of sentences i.e. nouns; adjectives are extracted and stored separately. The presence of these words or their meanings in the considered pair of sentences is found out. The absence of the keyword is indicated by 0 and the presence is indicated by the number of occurrences, this account for giving more weightage to a word occurring more than once.

Let P_i and P_j be the vectors that indicate the presence of keywords.

P_i= (1, 1, 1, 1, 1, 1, 1, 0)

P_j= (1, 0, 1, 1, 0, 0, 0, 1)

Using the vectors P_i and P_j, the cosine similarity value is calculated using the following formula:

$$cos(P_i, P_j) = (P_i * P_j) / (|P_i| * |P_j|) \quad (1)$$

Where, $Pi * P_j$ is the vector dot product of vectors P_i and P_j.

The Fig.2 indicates the calculation of Cosine Similarity. The dictionary lookup was speeded up using a cache and initiating computations which can go on simultaneously in parallel. This was accomplished by leveraging Multi-Threading in Java.

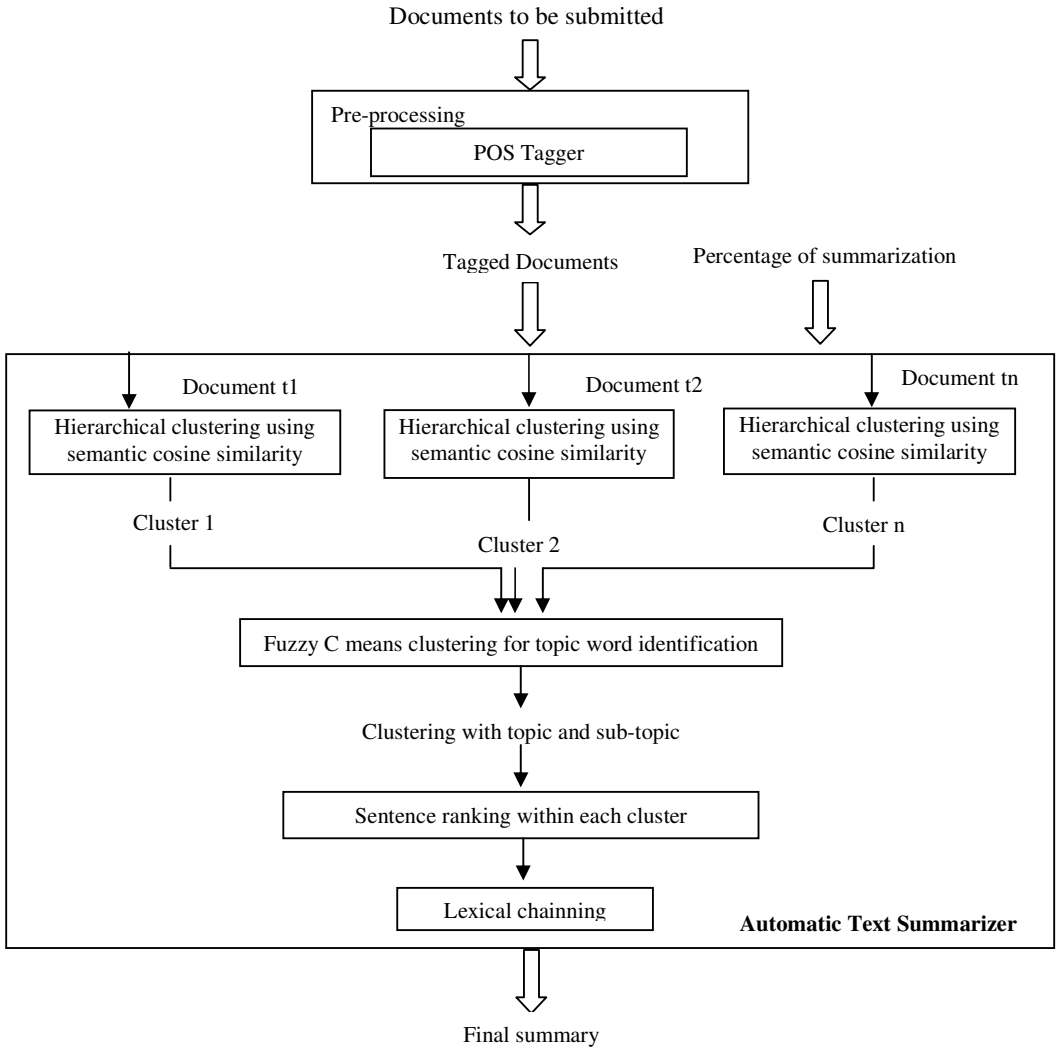

Fig.1. Overall system flow represented in diagrammatic form

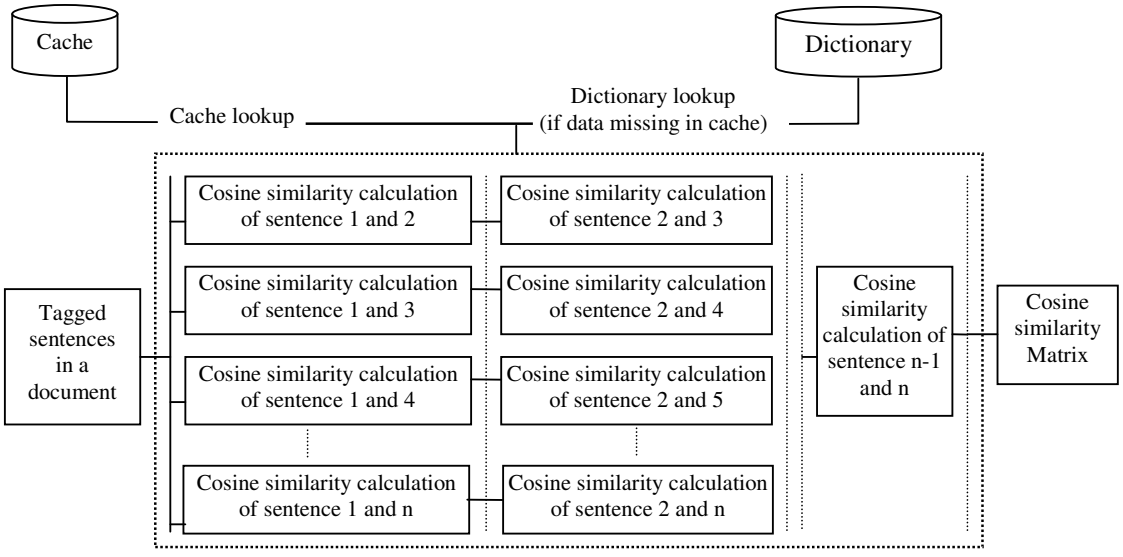

Fig.2. Semantic Cosine Similarity Calculation using Cache and Multithreading

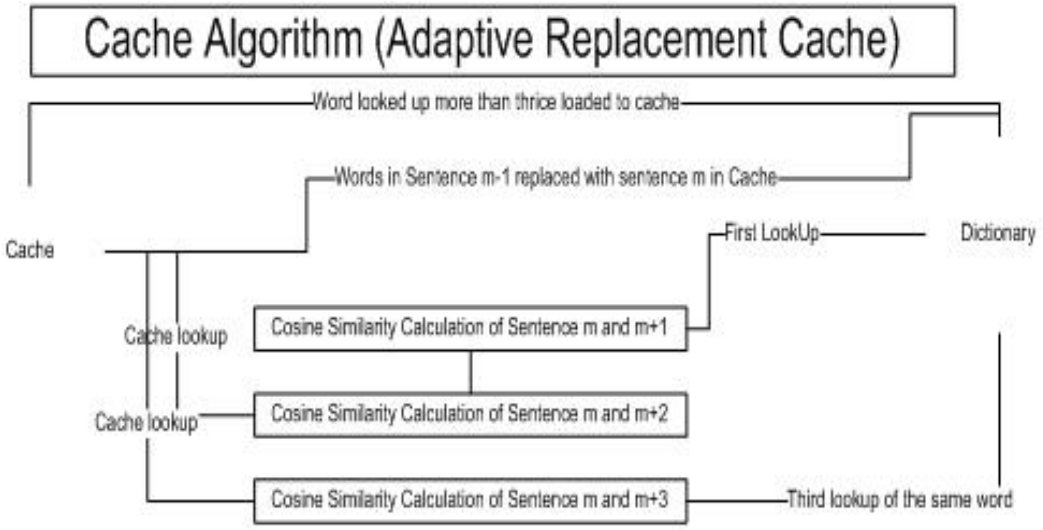

Fig.3. Cache Algorithm

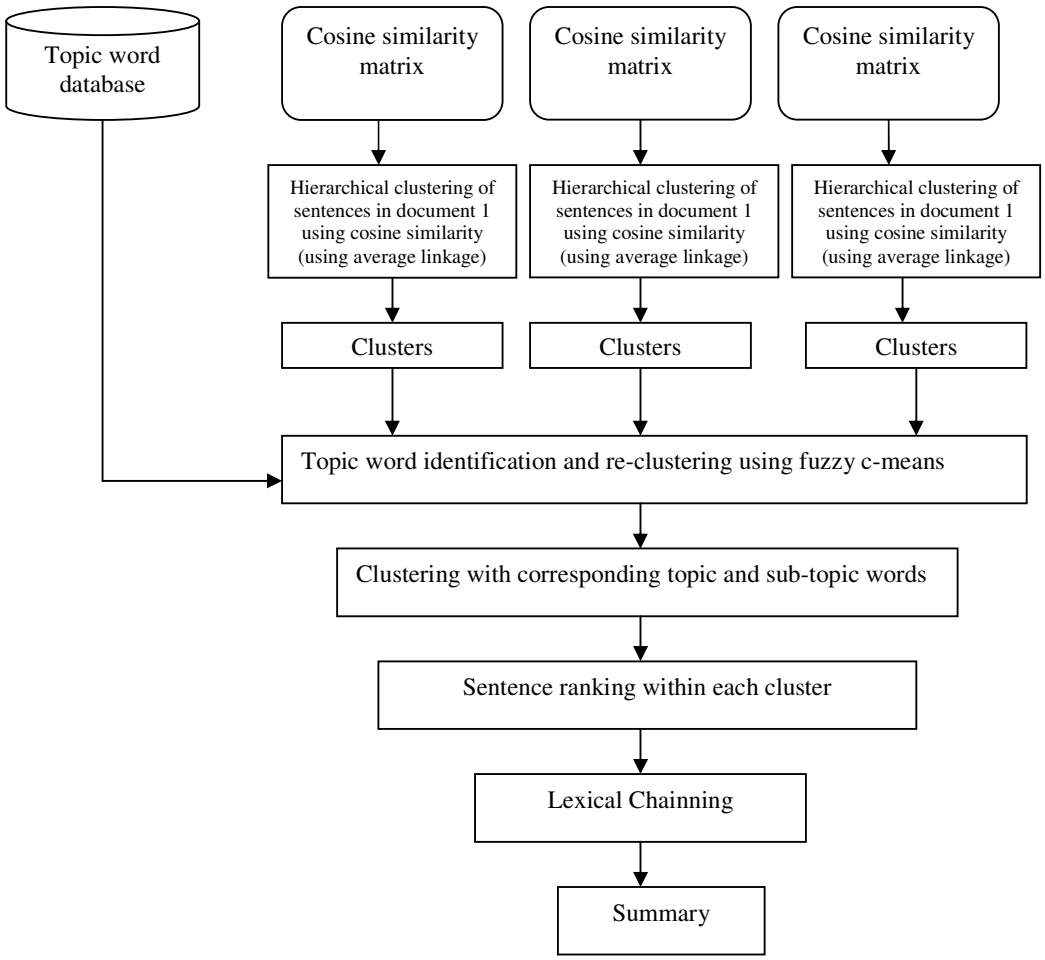

Fig.4. Grouping of clusters and Sentence Ranking

The Fig.3 indicates the cache algorithm which was employed for calculating Cosine Similarity of m*m+1 to m*n which occur in parallel and the words in sentence m which will be used frequently in near future will be stored in cache. Subsequently, when Cosine Similarity calculation of m+1*m+2 to m*n starts, words in cache corresponding to m will be replaced by words of sentence m+1. Also every third lookup of the cache for the same word causes it to be loaded into cache.

2.4 HIERARCHICAL CLUSTERING

Using the cosine similarity values, the sentences in the document are clustered [8]. The following steps are involved:

- Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item. Let the distances (similarities) between the clusters be the same as the distances (similarities) between the items they contain.
- Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one cluster less.
- Compute distances (similarities) between the new cluster and each of the old clusters.
- Repeat 2nd and 3rd steps until all items are clustered into a single cluster of size N.

3rd step can be done in different ways, viz. *single-linkage, complete-linkage* and *average-linkage* clustering. We have used Av*erage-linkage* clustering; we consider the distance between one cluster and another cluster to be equal to the average distance from any member of one cluster to any member of the other cluster.

2.5 TOPIC AND SUBTOPIC IDENTIFICATION

A fuzzy algorithm is used for identifying the topic and subtopic of each cluster generated by hierarchical clustering. An extensive topic database was designed. This database consisted of three tables namely, topic, subtopic and word. The topic table contains all the topics in the order of lexical connectivity. The subtopic table contains the subtopic numbers, subtopic names and also which topic they come under. In word table, the important words under each subtopic are listed along with a weight which indicates the importance of the word under the topic. The clusters so formed in the previous stage are assigned topic and subtopic after comparing the presence of words from the word database. For doing the same, the most frequently occurring words in the cluster are identified along with their number of occurrences, and each word is compared with the words under each subtopic. A weight is generated by using the equation (2):

$$WC\left(cluster_i\right) = \sum_{j=0}^{k} freq\left(W_j\right) * weight\left(W_j\right) \qquad (2)$$

where, W_j : word ϵ ($cluster_i \cap word_t$)

$freq(W_j)$: number of occurrences of W_j in cluster i.

$weight(W_j)$: weight of W_j under the t^{th} subtopic

$cluster_i$: words in i^{th} cluster

k : no of words that occur in $cluster_i$

The above calculation is repeated for all the subtopics. The degree to which a cluster is part of a subtopic is categorized by $WC(cluster_i)$. The topic with maximum number of subtopics will have weights greater than the average and it is decided to be the topic of the Cluster. The clusters which come under the same subtopic and topic are grouped and ordered in ascending order. The new clusters so formed are stored into the database.

2.6 SENTENCE RANKING

The sentences inside each cluster have to be scored for their relevance to identify the most important sentences in the document [9]. For this each sentence is scored based on 4 different metrics, namely,

1. Length
2. Location
3. Presence of Content Words
4. Lexical Connectivity score
5. Special Score based on presence of symbols like **“** , **’** , (,) , etc

2.7 LENGTH SCORE

The length score is computed by calculating the length of each sentence in each cluster and subjecting them to the sigmoid function for normalizing them. The length of the sentence is the number of words it contains, i.e., *l(S)*, normalized by sigmoid function:

$$L = \left(1 - e^{-\alpha}\right) / \left(1 + e^{-\alpha}\right) \qquad (3)$$

Where

$$\alpha = \left(l(S) - \mu\left(l(S)\right)\right) / std\ \left(l(S)\right) \qquad (4)$$

Where

$\mu\ (l(S))$ is the average length of sentences

$std(l(S))$ is the standard deviation of the sentence lengths.

2.8 LOCATION SCORE

The location of a sentence in a document can have significance in determining its importance. For example the first sentence introduces the topic and also last sentence presents some important conclusions hence they both are given highest score. Hence location is considered in scoring the sentence

$$S = X / N \qquad (5)$$

Where,

N is the total number of sentences in the paragraph;

X is the index of sentence *S*.

2.9 CONTENT-WORD SCORE

This score is based on the presence of standard content words which ought to be present in the sentence. For computing this we rely upon the previous phase of topic word identification. The topic to which the considered sentence belongs is found out and the most important words that ought to be in that topic are identified and checked if present in the sentence. Depending upon the presence a score is generated. To calculate the content word score, the cluster to which the

sentence belongs i.e. the topic and sub topic of the sentence is identified. Then depending on how many content words of that sub topic are present in the sentence, the sentence is given a score, which is calculated using (6).

$$F(S) == (1 - e^{-\alpha}) / (1 + e^{-\alpha}) \quad (6)$$

where, S = sentence in the document under consideration

$$\alpha = (CW(S) - \mu(CW(S))) / std(CW(S)) \quad (7)$$

$$CW(S) = -\sum_{i=0}^{k} \log[freq(W_i)], \quad where \quad W_i \in S \quad (8)$$

where, $freq(W_i)$ is the frequency of W_i in that document

$\mu(CW(S))$ is the mean of all the sentence scores

$std(CW(S))$ is the standard deviation

2.10 SPECIAL SCORE

Based on the presence of special characters like Bullets, Quotations, figures, brackets etc, a special score is generated.

$$Special(S) = (1 - e^{-\alpha}) / (1 + e^{-\alpha}) \quad (9)$$

Where

$$\alpha = (S(S) - \mu(S(S))) / std(S(S)) \quad (10)$$

Where,

$\mu(S(S))$ is the mean of special score of sentence S.

$std(S(S))$ is the standard deviation special score of sentence S.

$$S(S) = \sum_{i=0}^{k} \log[freq(W_i)], \quad where\ W_i \in S \quad (11)$$

2.11 AVERAGE LEXICAL CONNECTIVITY (ALC)

For lexical connectivity the number of terms that the sentence shares with other sentences is calculated and accordingly a score is given. The assumption is that a sentence that share more terms with other sentences is more important.

$$ALC(S) = (1 - e^{-\alpha}) / (1 + e^{-\alpha}) \quad (12)$$

Where

$$\alpha = (L(S) - \mu(L(S))) / std(L(S)) \quad (13)$$

where, $\mu(L(S))$ is the mean of lexical score of sentence S.

$std(L(S))$ is the standard deviation lexical score of sentence S.

$$L(S) = -\sum_{i=0}^{k} \log [freq (W_i)], \quad where \quad W_i \in S \quad (14)$$

2.12 LEXICAL CHAINING

The user specifies the percentage of summarization. According to the percentage specified, the number of sentences to be picked from each cluster in every document is computed. The required number of sentences is selected from each cluster according to their score. The sentences selected are subjected to hierarchical lexical chaining. Lexical chaining selects a set of candidate words, generally nouns. Then search through the list of chains and if a word satisfies the relatedness criteria with a chain word then the word is added to the chain, otherwise a new chain is created. The topics and subtopics are first ordered according to the sequence in which they occur, for example Software Requirements topic should occur in the beginning, and it should be followed by design and so on. The sentences selected for the final summary are ordered according to the topic/subtopic to which they belong to. [10, 11].

3. PERFORMANCE ANALYSIS

Documents related to the subject Software Engineering under the topics Requirement analysis, Project management, risk analysis and software development were collected. The performance measures used for the evaluation of the summary generated by the application are precision, recall and F-score as shown in formula (15), formula (16) and formula (17) respectively. Precision measures the percentage of correctness for the total number of summaries judged by the summary assessor to be relevant. Precision also measures the usefulness of the summarizer while recall is a measure of the completeness of the summarizer. Recall is a measure of how effective the system in including relevant sentences in the summary. It is 1.0 when all relevant sentences are retrieved. Precision is a measure of how effective the system in excluding irrelevant sentences from the summary. It is 1.0 when all documents returned to the system's users are relevant to the summary. Meanwhile, F-Score is a composite score that combines the precision and recall measures.

$$precision = \frac{relevant\ sentences \cap retreived\ sentences}{retreived\ sentences} \quad (15)$$

$$recall = \frac{relevant\ sentences \cap retreived\ sentences}{relevant\ sentences} \quad (16)$$

$$F - Score = \frac{2 * precision * recall}{precision + recall} \quad (17)$$

To obtain the results of all performance measures, a reference output should be at hand. This section of evaluation uses a human-generated summary. The individuals involved in this process are the experts in area in the Software Engineering. The summary generated by experts would be used as a reference in obtaining the number of relevant sentences in a particular summary. There are different summaries generated based on the percentage of requirement from the user. Fig.5 and Fig.6 depicts the results obtained for different percentage of summarisation.

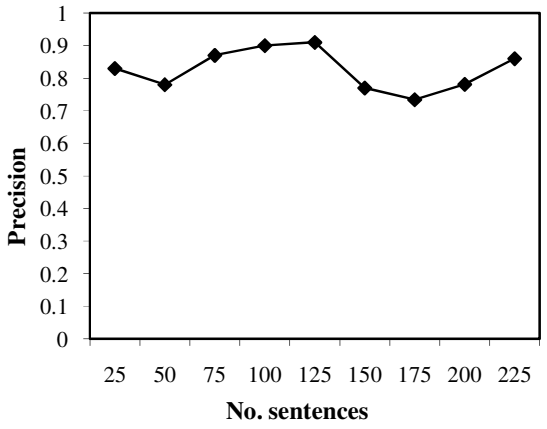

Fig.5. Precision graph for 80% summarization

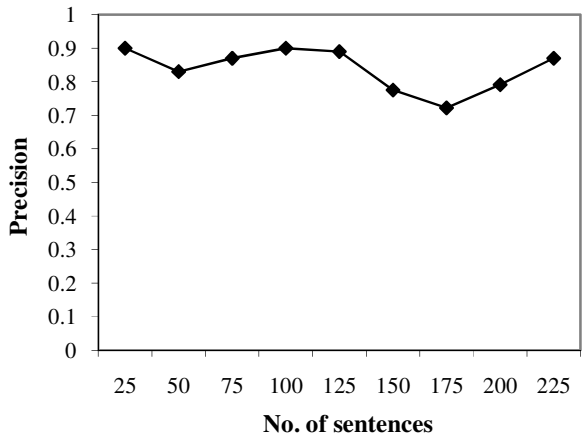

Fig.6. Precision graph for 90% summarization

Table.1 Evaluation on Software Engineering Domain

Percentage of summarization	Precision	Recall	F-score
10	**1**	**1**	**1**
20	0.94	0.93	0.93
30	0.9	0.91	0.9
40	0.88	0.84	0.86
50	0.88	0.82	0.85
60	0.85	0.84	0.84
70	0.81	0.84	0.82
80	0.79	0.8	0.79
90	0.75	0.79	0.77
Average			0.86

Based on the results shown in Table I, the average F-score for all articles is 0.86. The summary generated for Software Engineering domain by using machine learning algorithm shows a similarity with the summaries generated by the expert (human-generated summaries). Therefore, the conclusion which have arisen from the results, suggest that this technique is suitable for a specific topic corpus.

4. CONCLUSIONS

The project has high demand in today's world due to the problem of information overload. There is an abundance of information available to the user and very less time to go through all the available information. This project summarizes Software Engineering documents in an efficient way considering the importance of each sentence. Previous methods extract only the most highly ranked sentences which would often lead to redundancy in the final summary. Clustering the similar sentences and choosing the best among them helped to reduce redundancy by a significant amount. Also the clustering was on the basis of semantic cosine similarity which provided a more meaningful and effective clustering. An extensive dictionary was developed and the lookup in dictionary was speeded up using multithreading. Also to achieve logical coherence lexical chaining was employed. The lexical chaining was designed in such a way to cater the needs of the domain which improved the readability of the summary. The system can be enhanced to summarize not only text documents but also other type of documents like word, PDF, HTML, etc. One other enhancement that can be made is to extend it for other domains. The Dictionary can be expanded to include antonyms, homonyms and hyponyms, abbreviations. Using this expanded dictionary the efficiency of the text summarizer tool can be improved.

REFERENCES

[1] A public domain portable multi-document summarization system, webpage: http://www.summarization.com/mead/

[2] Information Sciences Institute, The University of South California, website: http://www.isi.edu/naturallanguage/projects/

[3] Kevin Knight and S. Luk, 2000,"Building a large-scale knowledge base for machine translation", in *Proc. of AAAI'94*

[4] Hovy E.H., 1998, "Combining and Standardizing Large-Scale, Practical Ontologies for Machine Translation and Other Uses", in *Proc. of the 1st International Conference on Language Resources and Evaluation* (LREC). Granada, Spain.

[5] Miller, G., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. 1990, "WordNet: An on-line lexical database", *International journal of lexicography*, Vol. 3, No. 4: pp. 235-244.

[6] Horacio Saggion., 2007, "SHEF: Semantic Tagging and Summarization Techniques Applied to Cross-document Co reference", in *Proc.of SemEval 2007.*

[7] Soe-Tsyr Yuan and Jerry Sun., 2004, "Ontology Based Structured Cosine Similarity in Speech Document Summarization", in *Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence.*

[8] Huang-Cheng Kuo, Tsung-Han Tsai and Jen-Peng Huang., 2004, "Building a Concept Hierarchy by Heirarchical Clustering with Join/Merge Decision", in *Proc. Of ISI Conference, LNCS* 3073, 100-133.

[9] Zhuli Xie, Xin Li, Barbara Di Eugenio, Weimin Xiao, Thomas M. Tirpak, and Peter C. Nelson., 2004, "Using Gene Expression Programming to Construct Sentence Ranking Functions for Text Summarization.", in *Proc. of the 20th International Conference on Computational Linguistics (COLING-2004). Geneva, Switzerland.*

[10] H. Gregory Silber, Kathleen F. McCoy, 2000, "Efficient Text Summarization Using Lexical Chains", in *Proc. of the Intelligent Scalable Text Summarization Workshop ACL Madrid.*

[11] Meru Brunn, Yllias Chali, Christopher J. Pinchak., 2000, "Text Summarization Using Lexical Chains", in *Proc. Of ACL/EACL Workshop on Intelligent Scalable Text Summarization*, pp. 10-17.