POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Corso di Laurea Magistrale
in Ingegneria Informatica
/Computer Engineering

Tesi di Laurea Magistrale

Itemset-based document summarization of multilingual collections driven by pre-trained word vectors

Relatore

firma del relatore (dei relatori)
prof. Luca Cagliero
.....
.....

Candidato

firma del candidato
Yifu Zhao

Dicembre, 2018

Index

1. Introduction

With the gradual progress of information technology, people become much more easier to create, store and disseminate information in electronic form. Billions of users on Internet create quintillion of bytes everyday, even though the rich information is beneficial for human beings on several levels, the amount of information and knowledge are growing exponentially, which makes people difficult of find useful information. Taking into account the efficiency of information utilization, a viable solution for getting critical information from a large collection of document is to generate readable and concise summaries containing the most relevant information automatically. For example, news articles that focuses on the same issue may have many redundant fact and related point of view from different individuals, an automatic summary may collect the most relevant facts and common views in several sentences, avoiding getting lost in the large set of original tests.

The automatic summarization technology mainly includes two kinds, mechanical and comprehension. Mechanical summarization can be applied to unrestricted domains, which is consistent with the current trend of natural language processing technology for real corpus and practicality, while comprehension summarization sacrificing the width of the field in exchange for the depth of understanding, its value as a theoretical exploration is high, but hard for practice. In recent years, along with the dramatic increase in computer storage and computation, data mining and machine learning algorithms are rapidly evolving, it is possible to obtain information from a large amount of data to assist in the completion of tasks. For summary tasks, text mining is a solution.

Text mining refers to the acquisition of valuable information an knowledge from text data, which is a method in data mining. The most important and basic application in text mining is to realize the classification and clustering of texts. The former is a supervised mining algorithm and the latter is an unsupervised mining algorithm. The disciplines associated with text mining are very broad, with the combination of the knowledge in probability theory and statistical mathematical analysis, and the application of data mining and machine learning techniques, text mining is applied in natural language processing and information extraction, many studies are working to introdue new technologies to bring improvements to text mining.

So far, lots of research has been devoted to generating summaries that best summarize the content of the document in various ways. They mainly consist of two main categories, for keywords and for sentences. For analyzing and filtering sentences, there are several ways: (i) clustering, (ii) graph ranking, (iii) optimization strategies, and (iv) frequent itemset mining. And also studies linguistic or semantic analysis are used for benefits the generation of summary.

Although text mining can accomplish tasks well in many fields, in the summary task, it has limited information because the size of datasets are usually quite small(only a few kByte). On the other hand, machine learning algorithms for the text have sprung up. Word embedding technology makes it possible for computers to understand words. By converting words into vectors that can express word features, computers can better understand the relationship between words and words, words and paragraphs.

The goal of this thesis work is to exploit the powerfulness of word embedding to improve multilingual summarization performance. Specifically, it aims at integrating pre-trained word vector information into a state-of-the-art multilingual summarization approach. The summarizer generates a summary of a collection of textual documents consisting of a selection of the most significant sentences. It analyzes the correlation among multiple text words to drive sentence selection. Based on word embeddings, we are able to discriminate between significant sentences and not according to the relevance of the contained words.

The proposed methods were applied to DUC'04. TAC'11, MultiLing'13 corpus respectively, and the generated summaries are evaluated according to the corresponding standard, then compared with the scores of other summarizers state-of-the-art. The results show that the use of word vectors in pre-processing stage can effectively improve the results, but in the sentence selection stage does not achieve the results as expected. For the use of word vector in pre-processing stage, the results of experiment can reach the top level. This method is stable in multi-language environment and is not affected by language types.

The article is organized as follows. Section 2 introduce the previous approaches and posts the pros and cons. Section 3 introduce the previous researches on word embedding and toolkits with pre-trained vector models state-of-the-art. Section 4 proposes the main approaches for improve the summarizer. Section 5 lists our experiments, results and comparisons. And section 6 draws conclusions and put forward directions for future work.

2. Literature review

Document summarization is a general term for a class of tasks that generate concise text summary that describes the content of one ore more text documents. There are currently two main approaches for document summarization, sentence-based document summarizers and keyword-based document digests, depending on the type of generated summaries. The sentence-based approach generally splits the document into sentences and select the sentence that is most relevant to the content of document to form the summary[1,2]. The keyword-based approach detects the keywords that best represent the article, and then uses co-occurrence metrics or latent semantic analysis to generate document summaries. In order to generate a quality document summary, you must choose a sentence that is more suitable for generalizing the article. The summarizer structure presenting in this article is derived from MWI-Sum which is an sentence-based frequent itemset mining approach. Section 2.1 introduces the current strategies for sentence selection, and Section 2.2 introduces the main principle and structure of MWI-Sum, which the approach proposed in this article is based on.

2.1 Main categories of sentence-based summarization algorithms

There are currently several strategies for high quality sentence selection: (i) clustering, (ii) graph ranking, (iii) optimization strategies, and (iv) frequent itemset mining. In the field of summarization generation, lots of research has been done on these strategies, and a brief introduction will be made below.

2.1.1 Clustering based approaches

Clustering based approaches [1][2][3] are used to select sentences that meet the needs of the summarization. One idea is to find the subset of sentences that best represent the entire cluster from the set where sentence are represented in unique form[2], such as the sentence closest to the centroid or medoid of cluster. Another idea the MEAD summarizer[3] is to use the document set as a criterion for clustering evaluation. In this case, the centroid of the corresponding cluster is a pseudo-document consisting of a sentence selected by evaluating the tf-idf value[4] of the corresponding phrase. In addition, there are approaches to use the incremental clustering algorithm[5] or related algorithms to dynamically update the original summary[1] when adding or deleting some of the documents in the dataset without recalculating the entire cluster model. Clustering algorithms are especially useful for documents or sentences that could be divided into different topics.

2.1.2 Graph-based approaches

The general idea of the graph-based algorithm[6][7][8] is to construct a graph with the document sentences as nodes, whose edges of the connected nodes are weighted by comparing the similarities of the connected nodes, and pruned according to the minimum similarity threshold to reduce the complexity of subsequent program, and then use the indexing strategy[9] to extract the sentences that best fit the composition summary. Some studies[6][7] combine knowledge provided by user-generated content with graph-based

models to improve the quality of abstracts. Other studies[8] use the association rules between the term pairs to generate graphics,and then use the HITS[10] algorithm to sort the sentences.

2.1.3 Optimization strategies

Optimization strategies are also used to select the most appropriate sentence to form a subset. Some methods transform the sentence selection process into a min-max optimization problem, which is then solved by a combination optimization strategy[11]. Approach described in [12] uses a similar approach to summarize the differences between the collections. Other approach use Singular Value Decomposition(SVD)[13], Integer Linear Programming, and submodular function optimization techniques[14][15] to screen for appropriate sentences. The summarizer CLASSY[16][17] generates summaries for document sets by combining the optimization with the Markov model. Some approaches[13][17][18] that utilize an optimization strategy can be applied to multi-language file sets.

2.1.4 Frequent itemset mining

Frequent itemset mining is a very important research basis in data mining research[19]. It can tell us the objects that often appear together in the dataset and provide some support for possible decisions. Frequent itemsets have a wide range of applications, including document summaries. In [20], [21] and [22], the researchers initially studied the application summary of the application set. A document is considered to be a dataset consisting of terms. By researching the co-occurrence of itemsets, the approaches will find the itemsets which benefits more to the summary. And in order to distinguish more meaningful items according to relevance or attention ones, some studies propose approaches for weighting items by indexing strategies or known information. The weight of items was considered to help improve the quality of abstract generation in a multilingual environment[23]. The approach proposed in [20] abstracts all the distinct words in each document into a single transaction, while [21] and [22] treats each sentence of document as a transaction. The approach proposed in [22] measures the importance of each sentence by the association between document items, the disadvantage of this approach is the loss of high-order association between terms(combination of multiple terms more than 2). [23] has made improvements on the basis of [22], while using the weight itemset, term frequency – document frequency(tf-df) is also used instead of the original tf-idf as the relevance score. It is more suitable for processing file sets in related fields.

In the field of text processing, linguistic and semantic analysis has always had a place. Despite the rapid development of data mining technology in recent years, non-semantic methods have become popular, but the traditional methods still have the advantage that can not be ignored, that is, when technologies of artificial intelligence is not perfect, compared with the data mining algorithm based on word frequency, linguistic and semantic based analysis can better understand the meaning of text. In previous studies, [24][25] used ontology to find the meaning that was closest to what the user queried, while [26][27] separately studied the context used to generate the digest in the corresponding (mobile, business, disaster management) application domain, and [28][29][30] used the text argument structure. Another type of approach is the summarize generator with the lexical chain. The lexical chain usually comes from ontologies or lexical databases[31], the choice of language model will have a great impact on its effectiveness, which is largely influenced by the language and the type of document.

2.2 MWI-Sum[23]

MWI-Sum is a multilingual summarizer based on frequent weighted itemsets. It has shown outstanding results in collection of news documents in multilingual environment. The approaches proposed in this paper is based on this summarizer, improve the quality of the results by improve some of its stages. The summarizer consists of four main stages: document (i)pre-processing, (ii)sentence filtering, (iii)frequent weighted itemset mining and itemset-based model generation, (iv)sentence evaluation and selection. In pre-processing stage, the summarizer does stemming and stopword elimination for documents, and calculate tf-df value to establish a weight matrix, then the program filter sentences by the position of sentences and the total weight of words. After the sources are pruned, an itemset-based model is generated for frequent weighted text mining. At last the output of frequent itemsets will be used for evaluate sentences, the sentences with most itemsets covered and high relevance score will be selected greedily to form the required summary. Although this approach has already achieved good scores, it is still limited in the dataset to be summarized. Therefor, by introducing external information from pre-trained word vector model, it still as space for improvement.

3. Word embedding

The summarization approach mentioned in this article combines word embedding with the original one, resulting in better results for multi-language text summarization tasks. This section will introduce related content of word embedding. Section 3.1 will introduce what does word embedding do and the related technologies for word embedding, Section 3.2 introduces word embedding toolkits state-of-the-art, and Section 3.3 introduces the pre-trained vectors model used in the summarization approach exposed in this article.

3.1 Theory on word embedding

Word embedding is a technique for mapping word vectors in high-dimensional space into vector spaces with lower dimensions to obtain more characteristic and easier-to-handle word vectors. It allows people to convert the one-hot vectors whose original dimension is equivalent to the number of words(e.g. millions) into a very small dimension(100-300) by a certain means of generation(e.g. co-occurrence matrix, neural network, etc.) and make the vector easier to express some characteristics of words(e.g. similar relationship).

3.1.1 Vectors to represent words

In natural language processing (NLP) tasks, the first point is to consider how words are represented in the computer. Traditionally, rule-based or statistical-based natural semantic processing methods which treat words as an atomic symbol are called one-hot representation. One-hot representation is equivalent to assigning an id to each word, which could not showing the relationship between words. In addition, one-hot representation will result in a very large characteristic space, even though this makes many tasks linearly separable, it requires huge processing space and storage space. Word embedding technique could convert words into a distributed representation, which is called the word vectors. In this form, there is a concept of ‘distance’ between words, which is very helpful for computer to ‘understand’ the relationship between two words. For example, in a well-prepared model for words of countries and cities, the cosine distance between the vector(Rome) and vector(Italy) is bigger than word Rome to France. Furthermore, recently research[32] shows that the word vectors capture may linguistic regularities. For example, the value vector(‘Rome’) – vector(‘Italy’) is very close to the value vector(‘Paris’) - vector(‘Frence’). Therefor, the distributed representation of words after word embedding have more advantages in natural language processing. In addition, most of the algorithms state-of-the-art uses bag-of-words(BOW) model discussing the relationship between a word and its context, words due to the same root which may have similar context will receive vectors very close to each other, the word vectors could also make an effort on stemming.

Most of the current mainstream word vector generation methods are based on the relationship between the context and the word, that is, the word can be inferred according to its context, or its context can be inferred based on the word. Initially people used statistical methods such as co-occurrence matrix and singular value decomposition (SVD), and then methods of word embedding with latent semantic analysis and language models are

successively proposed[33][34][35][36]. In 2000, a research[37] presented a method based on feedforward neural network and nonlinear hidden layer, which proposed to use a “distributed representation” of words, that is, dimensionality reduction to obtain word vectors that are conducive to natural language analysis. In 2013, a team at Google led by Tomas Mikolov invented a toolkit called word2vec[38] for word embedding, which trains vector space models faster than ever. After word2vec, a lot of comparable toolkits[39][40] are developed.

3.1.2 Language model

The language model generates word vector by training the neural network language model(NNLM), where the word vector is the incidental output of the language model. The basic idea in NNLM is to predict the words that appear in the context. This prediction of the context is essentially a study of the co-occurrence statistics. Currently known methods for generating word vectors using the neural network language model are: Skip-gram, CBOW, GloVe[40] and so on. Since the word vector models used in this article is generated by word2vec[38] and FastText[39] which used Skip-gram and CBOW model, sections 3.1.2.1 and 3.1.2.2 will introduce these two methods briefly.

3.1.2.1 Continuous Bag-of-Words(CBOW) model

CBOW is a common model for word embedding, which uses word-of-bags model takes the context of words as input, and then outputs the distributed representation of the word. The main idea of CBOW is to predict the intermediate word by the context. After we specify the window size of the context, for each word CBOW will represents all words in its context window as one-hot format, with the same weight for the input of the neural network. So the input layer of the neural network has twice the window size. These vectors pass through the hidden layer and then passed to the output layer. The number of neuron in hidden layer is equivalent to the dimension of word vector we want to generate, i.e. each hidden layer represents a “feature”. The size of output layer is the same as the total number of words, and each neuron outputs a probability value. The goal of training is to expect the softmax probability of a particular word corresponding to the training sample to be the greatest, i.e., through its context, we can predict what the word is most likely to appear in the middle.

3.1.2.2 Skip-gram neural network model[41][42]

Skip-gram is another neural network model for training the vector representation of words. which seems reversed to CBOW, Skip-gram model receives a word as input, and the output is the context of the word, which has a window size to control the number of words in context. What the neural network needs to do is, when given a particular word in a sentence, it tells us the probability that each word in vocabulary whether it is the context for the word.

Before processing the data, the program first encodes the word as one-hot representation, constructs a vector space with corresponding dimensions according to the number of words. The words are represented by their vector corresponding to their unique number. Each vector is processed through the hidden layer and then passed to the output layer, whose numbers equals to the size of vocabulary. The hidden layer is a weight matrix,whose number of rows is equal to the size of the word list, and the number of columns is equal to the number of features to be learned, that is, the dimension of the resulting word vectors. In fact, this weight matrix is the ultimate goal of the word embedding: the dimension-reduced word vectors.

Since the initial input is a unit vector, the hidden layer actually plays the role of acquiring the current word vector and the recorder of word vector. The output layer is a softmax classifier. Each output neuron produce their output value in 0-1 and the sum of all output value is 1. The system does not accept the offset of the output word from the input, it does not learn a different set of probabilities for words before and after the input word.

In this article, we use word vectors generated by Skip-gram.

3.1.3 Measure quality of the word vectors

For the word vector models generated by word embedding methods, we care whether it can reflect the characteristics of words and the relationship between words correctly. In general, vectors of similar words should exhibit stronger cosine correlation, and the cosine distance of two words with specific aspects should show specificity, and same type of words(such as “apple” and “banana” are all “fruit”) should be more inclined to present aggregation. Therefore, many word embedding toolkits provide testable data for these features. Take word2vec as an example, it provides a large number of analogy word pairs, such as “man”-“king” and “woman”-“queen”, after calculate the cosine distance in each word pair, compare the difference of the values, if the difference is small enough, it is considered as passing. Finally the pass rate is calculated to measure the quality of the generated word vector model.

3.1.4 Limitations of word embedding and advanced technology

Although the word vector is very helpful for the computer to understand the meaning of the word, it still has its limitations, that is, a word may have many meanings, but they are mixed and represented by a single vector. The solution so far is in the ongoing research of sense embeddings[43], which recognize the different individual meanings of a same word, separate them and represent them as different vectors in space.

3.2 Word embedding toolkits

There are many word embedding toolkits shared by different teams since the word2vec toolkit was presented in 2013. Sections 3.2.1 to 3.2.3 introduces the toolkits among the most popular ones state-of-the-art that offer at least one pre-trained vector model.

3.2.1 Word2vec[38]

Word2vec, which was created by a team of researchers led by Tomas Mikolov at Google, is a group of related models for producing word embeddings to generate word vectors. With the benefits of performance improvements, word2vec could train large amount of text(e.g. several gigas) and generate vectors for millions of words. The vectors, with hundreds of dimensions could represent the characteristics of words well, that is ,words with similar contexts are close in vector space.

Word2vec is well developed to process huge size of data. In a huge sample set, the most frequent words will appear in millions of times, such as: in, the,a, etc. These words tend to carry a small amount of information relative to rare words. For example, skip-gram has a

much higher degree of benefit from observing the combination of ‘France’ and ‘Paris’ than from the observation of the combination of ‘France’ and ‘the’, because the combination with ‘the’ is common appear in most of the sentences. In response to this phenomenon, word2vec provides a frequent word sampling mechanism that makes the frequency of a word appear to be approximately inversely proportional to the probability of the retaining the word, and also provides a parameter ‘sample’ to control the threshold of sampling, which makes it different to retain words that appear too frequent.

There are two main methods for training word vectors in word2vec, CBOW and Skip-gram. The offset of context words does no effort on the prediction. General CBOW or Skip-gram neural networks may complete the same training tasks as word2vec, but in case of large amount of data and in vector space of large dimension, the time it takes to process all the data will increase exponentially. To avoid these problem, word2vec does not use the traditional DNN model. Firstly, data structure optimized for use is to replace the hidden layer and the output layer with binary tree of Huffman coding. Moreover, word2vec introduced other algorithm like Hierarchical Softmax[44] and Negative Sampling , which could be used as optional add-ons to speed up the program as will as make the result more precise.

Negative sampling is a unique method used by word2vec to increase program speed. When a large number of sample s are used to train a neural network, the size of the vocabulary will leads to a large amount of weight data. If all the weights are updated when training each sample, the program will consume a lot of time and resources. Negative sampling solves this problem by having each training sample modify only a small portion of the weight instead of the full. When a network is trained using a word, the correct output of the network is the neuron output 1 corresponding to its context-dependent words, which are regarded as “positive” words, and the neuron output 0 for all other unrelated words, which are regarded as “negative” words. In the case of negative sampling, word2vec will randomly select a small number of “negative” words to update the weights while updating all “positive” words. Each word is given a probability associated with its frequency, from which frequent words are more likely to be choose as the negative sample. The number of negative samples for smaller datasets is 5-20 words, and for large datasets only 2-5 words, which reduces the complexity of each training update from O(n) to O(1). Despite this, the value of updates is still large, because the number of weights each word needs to update is equal to the dimension of the features.

Hierarchical Softmax is another algorithm used by word2vec to greatly improve computational efficiency. It uses a binary tree to represent the output layer and a word matrix to represent leaf nodes. For each node, the probability of its child nodes is clearly indicated. In the initial research[44], some methods for constructing tree structures were proposed, as well as the impact on training time and model accuracy. In word2vec, this method is combined with binary Huffman coding because it assigns short codes to frequent vocabularies, making training faster.

In addition, word2vec believes that a phrase consisting of multiple words should be treated as an independent unit and should have a unique word vector, for example the phrase ‘New_York’ should be independent of ‘New’ or ‘York’. Also because word2vec has improved the processing speed of the program, they are able to train vectors for a huge amount of vocabulary. The toolkit counts the times of word combination appears in the text, and then uses those counts in the equation to determine which words are combined into a phrase. This design avoids some common combinations such as ‘this is’, ‘and a’, etc. As it shows in the model they published, there are 3 million words and phrases in the model, which cover most common words and phrases. At the same time, the toolkit provides a stitching mechanism for

phrases that are not presented in the word list. The vector of the phrase can be calculated by splitting at the hyphen and based on the vector of all words that make up the phrase.

The final quality of the results depend on the parameters such as sub-sampling which filter words with frequency above a certain threshold, dimensionality of the features observed from word(the size of word vector to produce), context window which delimit the size of context, and the optional algorithms selected to use for training the model.

3.2.2 FastText[39]

FastText is a word vector calculation and text categorization toolkit that Facebook opened up in 2016. It is not much innovative in academics, but its advantages are also very obvious. In the text classification task, FastText(shallow network) can often achieve the accuracy comparable to the deep network, but it is many orders of magnitude faster than the deep network in training time. The main function of the fasttext model is that when a word stream is given, it classifies it and outputs its probability of belonging to different category. The model extracts information from words and phrases, constructs feature vectors, and maps them to labels through linear transformations. Unlike word2vec, its prediction object is label, not the middle word of the context corresponding to word2vec. And inherits the idea of word2vec, using Hierarchical softmax classifiers to speed up the program.

Since the word-of-bag model does not take use of the position of words, it loses the information expressed by relative position of the words. So FastText adds n-gram features for the words to take advantage of them to improve accuracy. In addition, for a string with more public characters, word2vec regard them as one-hot representation in the front, the correlation between them can only be kept by similar context. In FastText, there is a n-gram feature of the character lever simultaneously adopted, word is split into sub-strings having n characters, and vectors which can represent these sub-strings is obtained through training, and is used to represent the words composed thereof. This approach allows us to generate word vector for new word based on the substrings that make up them, even after the model has finished training.

3.2.3 GloVe[40]

GloVe is another word embedding toolkit derived from word2vec, using an unsupervised learning algorithm for obtaining vector representations for words. It improves the system by solving the missing part of relationship information in the word2vec due to the negative sampling. In addition, the Skip-gram algorithm in word2vec makes it easy to get too much weight for high-exposure words. GloVe with Global Vector combines the global statistical information of the matrix decomposition Latent Semantic Analysis(LSA) and the advantages of local context window, and integrates the global prior statistical information. These measures can control the relative weight of words while speeding up the training of the model.

3.3 Pre-trained vector models

In this article, we chose two models provided by word2vec and FastText. This is because they are currently available on the network, with the largest amount of

training data and best quality compared to others. Word2vec provides an model training by news corpus in English. Which is consistent with the type of our test set; while FastText provides a model that covers the widest variety of languages, wich corresponds to the multilingual test we use.

3.3.1 GoogleNews-Vectors-negative300

This pre-trained vectors model is offered by the word2vec team led by Tomas Mikolov at google[38], it using a simple data-driven approach of the skip-gram model with negative sampling. The training dataset is from Google News with about 100 billion word, and the generated vectors are 300-dimensional for 3 million words and phrases, conclude most English words, abbreviations, and some of the words in other languages that appeared in original English news. The word2vec group offers the model in both binary format and text format.

3.3.2 FastText Pre-trained vectors model

This model set is published by the Facebook FastText team generated by training Wikipedia data on Fasttext. It contains vectors in dimension 300 for 294 languages. The vectors is obtained using a Skip-gram model[39] with default parameters. These are published as both binary format and text formats, for the binary format, it has the unique form which could be interpreted by FastText into a trained model, then the program can train new datasets on the model, or generate word vectors for a list of words, even the words not originally appeared in the model.

4. THE PROPOSED SUMMARIZER

In this study we considered two different methods to integrate word embeddings into the itemset-based summarization process: pre-processing stage driven by word vectors and sentence evaluation stage driven by word vectors, and try to improve the performance of the summarizer through the word embedding based pre-trained word vector model. Since the vector set is produced by a way of data mining to make computer “understand” the meaning of vocabulary, this program makes it possible to get more benefits from the document without having understand the complex linguistic or grammatical analysis in advance. Benefiting from the feature, the program can be applied to a wide range of languages and yields better results in most languages. The following content will introduce the main ideas and working methods of the two approaches.

4.1 Document preprocessing with pre-trained vectors

Since our approach is based on weighted itemsets, the weight of the items is critical to the outcome. In MWI-Sum, the program uses tf-idf/tf-df to calculate the weight matrix of the terms, and removes the sentences that does not contain the term with higher weight by prune, then it performs frequent weighted itemset mining. In the approach proposed in this paper, we use the word vector to generate the weight matrix. The word vector contains far more features than tf-idf/tf-df, which makes it easier for us to study the meaning relationship between each term and each document. The key of this approach is to build the feature vector for each document according to the terms concluded, and then generate a weight matrix for terms according to the cosine distance between the feature vector of term and document. Since the contexts of terms in the same document or document set describing the same event are relatively close, their feature vectors should be close to each other, and the summation can be used yo remove their heterogeneous components and strengthen their homogeneous components. Thereby generating a feature vector of document. Because the feature vector expresses the homogeneity of the terms contained in the document, whether a term is related to the meaning of the document can be expressed by how close their feature vectors are, and the higher the value of cosine distance, the more likely it is to qualify for the document, and it should be given a higher weight.

In practice, first we get a Bag-of-words representation for each sentence by stemming, which is a collection of several word prototypes. Inherited from MWI-Sum, we also use prune, keeping the first N sentences of the document as the most meaningful part. Then we generate feature vector for each document based on term, which is the normalization of the sum of all term vectors. Then with the calculating of cosine distance value of the feature vectors between each term and each document, a weight matrix of D×T size is generated(where T represents the number of terms and D represents the number of documents). Each column of the matrix corresponds to a term that indicates the weight of the term for each document. The sum of the values of the column gives the total weight of the term on all documents, we select the k terms with the most top total weight, further prune the sentences that do not contain these terms. After the pruning, recalculate the weight matrix. This matrix is then used to perform frequent itemset mining. The following is the main process of the implementation of the approach:

a. make dictionary: for each term in document, if not in dictionary, add term into dictionary. A dictionary is a collection of term which contains all the words(after stemming) as a list. It may contains a number of total words, or just display the words one by one.

Figure 4.1 Example 1 of dictionary, first line is the total number of words and then the list of words

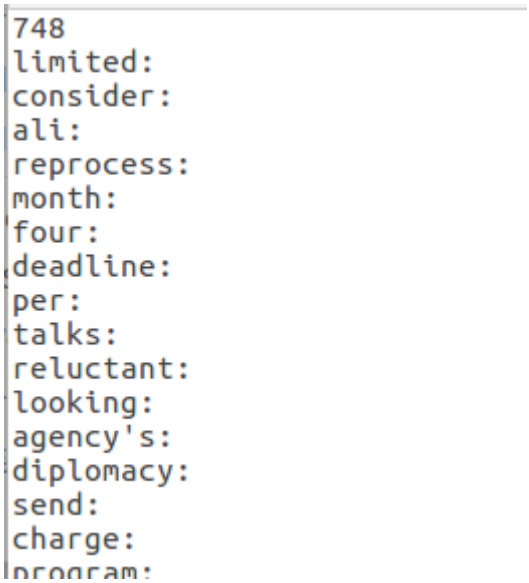


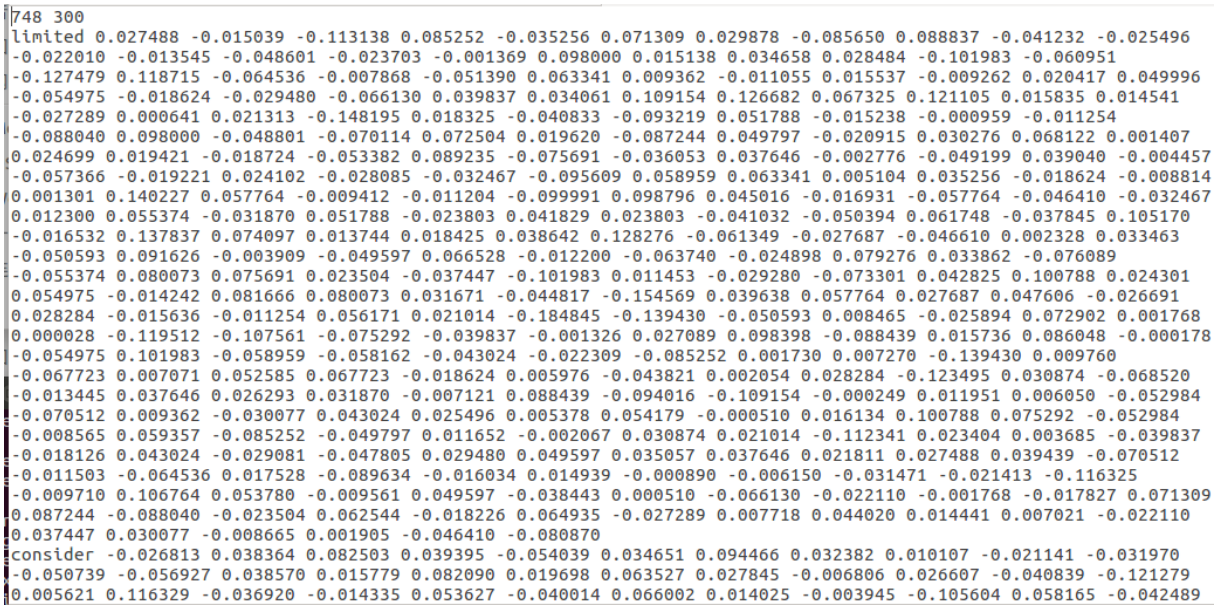
Figure 4.2 Example 2 of dictionary, the displayment of words, especially for Fasttext

limited consider ali reprocess month four deadline per talks reluctant looking agency's diplomacy send charge
program include sent case gholam-hossein co-operation far subject choice putting anonymous vastly vast forth try
team small inspection economically programme governing revolution force leaders ten sign go negotiator persuade
assembly akbar even reconciliation hide appear unsc anniversary spokesman consistently conduct international
officially public exercise body proliferation iran stance exchange soltanieh never french nation's behaviour
following china pursuit others active path along standard supplying convert engage technical compliance action
sirous country's diameter regardless russia thursday salehi sanction total crisis sermon use attendance spoke
working remains positive two angela next camera call outgo tell today israel parliament holy disarming warn hold
must enrichment-related join organization pursue work remain achieve install growing meet dmitry conformity claim
crazy want give process india accept involve way condescend council feasible guarantee ceremony tube end goal

b. load vectors from pre-trained package: for each term in dictionary, if term in package, get vector of the word, otherwise remove word from dictionary and document (the term which could not be found in package is not considered for summarization). For the GoogleNews pre-trained vectors model, the program does a reduce work by read each word and its vector from the model package, and find if the word is appeared in the dictionary, and for the Fasttext Wiki pre-trained vectors model, the binary-form model package must be resolved by Fasttext, and the program will generate a reduced vectors file. When encountering a word which is appeared in the dictionary but not found in the pre-trained vector model, we will try to split it into several pieces and find if all the pieces could be found in the vector model. Fasttext has its own set of methods for generating vectors for words that have never been seen through the pre-trained models. Each vector will be normalized into unit vector.

For example, if the word “pre-trained” is not seen in the GoogleNews pre-trained vectors model, it will be split into “pre” and “trained”, then try to find vectrs vector(“pre”) and vector(“trained”) in the model. After the two vectors are found, the vector of “pre-trained” will be calculated as the sum of the two vectors. Finally, the vector will be normalized into unit vector: vector(“pre-trained”) = norm(vector(“pre”) + vector(“trained”)).

Figure 4.3 the word vectors, after reduce, all the words appeared in this file are also in the documents for summary. The first line records the number of words(748 in total) and the dimension of vectors(300 features)



```
748 300
limited 0.027488 -0.015039 -0.113138 0.085252 -0.035256 0.071309 0.029878 -0.085650 0.088837 -0.041232 -0.025496
-0.022010 -0.013545 -0.048601 -0.023703 -0.001369 0.098000 0.015138 0.034658 0.028484 -0.101983 -0.060951
-0.127479 0.118715 -0.064536 -0.007868 -0.051390 0.063341 0.009362 -0.011055 0.015537 -0.009262 0.020417 0.049996
-0.054975 -0.018624 -0.029480 -0.066130 0.039837 0.034061 0.109154 0.126682 0.067325 0.121105 0.015835 0.014541
-0.027289 0.000641 0.021313 -0.148195 0.018325 -0.040833 -0.093219 0.051788 -0.015238 -0.000959 -0.011254
-0.088040 0.098000 -0.048801 -0.070114 0.072504 0.019620 -0.087244 0.049797 -0.020915 0.030276 0.068122 0.001407
0.024699 0.019421 -0.018724 -0.053382 0.089235 -0.075691 -0.036053 0.037646 -0.002776 -0.049199 0.039040 -0.004457
-0.057366 -0.019221 0.024102 -0.028085 -0.032467 -0.095609 0.058959 0.063341 0.005104 0.035256 -0.018624 -0.008814
0.001301 0.140227 0.057764 -0.009412 -0.011204 -0.099991 0.098796 0.045016 -0.016931 -0.057764 -0.046410 -0.032467
0.012300 0.055374 -0.031870 0.051788 -0.023803 0.041829 0.023803 -0.041032 -0.050394 0.061748 -0.037845 0.105170
-0.016532 0.137837 0.074097 0.013744 0.018425 0.038642 0.128276 -0.061349 -0.027687 -0.046610 0.002328 0.033463
-0.050593 0.091626 -0.003909 -0.049597 0.066528 -0.012200 -0.063740 -0.024898 0.079276 0.033862 -0.076089
-0.055374 0.080073 0.075691 0.023504 -0.037447 -0.101983 0.011453 -0.029280 -0.073301 0.042825 0.100788 0.024301
0.054975 -0.014242 0.081666 0.080073 0.031671 -0.044817 -0.154569 0.039638 0.057764 0.027687 0.047606 -0.026691
0.028284 -0.015636 -0.011254 0.056171 0.021014 -0.184845 -0.139430 -0.050593 0.008465 -0.025894 0.072902 0.001768
0.000028 -0.119512 -0.107561 -0.075292 -0.039837 -0.001326 0.027089 0.098398 -0.088439 0.015736 0.086048 -0.000178
-0.054975 0.101983 -0.058959 -0.058162 -0.043024 -0.022309 -0.085252 0.001730 0.007270 -0.139430 0.009760
-0.067723 0.007071 0.052585 0.067723 -0.018624 0.005976 -0.043821 0.002054 0.028284 -0.123495 0.030874 -0.068520
-0.013445 0.037646 0.026293 0.031870 -0.007121 0.088439 -0.094016 -0.109154 -0.000249 0.011951 0.006050 -0.052984
-0.070512 0.009362 -0.030077 0.043024 0.025496 0.005378 0.054179 -0.000510 0.016134 0.100788 0.075292 -0.052984
-0.008565 0.059357 -0.085252 -0.049797 0.011652 -0.002067 0.030874 0.021014 -0.112341 0.023404 0.003685 -0.039837
-0.018126 0.043024 -0.029081 -0.047805 0.029480 0.049597 0.035057 0.037646 0.021811 0.027488 0.039439 -0.070512
-0.011503 -0.064536 0.017528 -0.089634 -0.016034 0.014939 -0.000890 -0.006150 -0.031471 -0.021413 -0.116325
-0.009710 0.106764 0.053780 -0.009561 0.049597 -0.038443 0.000510 -0.066130 -0.022110 -0.001768 -0.017827 0.071309
0.087244 -0.088040 -0.023504 0.062544 -0.018226 0.064935 -0.027289 0.007718 0.044020 0.014441 0.007021 -0.022110
0.037447 0.030077 -0.008665 0.001905 -0.046410 -0.080870
consider -0.026813 0.038364 0.082503 0.039395 -0.054039 0.034651 0.094466 0.032382 0.010107 -0.021141 -0.031970
-0.050739 -0.056927 0.038570 0.015779 0.082090 0.019698 0.063527 0.027845 -0.006806 0.026607 -0.040839 -0.121279
0.005621 0.116329 -0.036920 -0.014335 0.053627 -0.040014 0.066002 0.014025 -0.003945 -0.105604 0.058165 -0.042489
```

c. select the first n-sentences of each document, take away the follow-up. For example if the parameter nSentence = 3, only the first 3 sentences will be considered to generate the summary.

d. calculate the vector of each document as follows:

$$\vec{D}_i = \frac{\sum_{j=1}^{LD_i} \vec{T}_{ij}}{\left| \sum_{j=1}^{LD_i} \vec{T}_{ij} \right|}$$

where $\vec{D}_i$ is the vector of the i-th document, LD_i is the number of terms in D_i , $\vec{T}_{ij}$ is the vector the j-th term in D_i

In this way, the vector to represent each document is made as the normalized sum of all the terms in the document, expressing features similar to terms.

e. for each document and each term, make a matrix which has documents numbers of rows, and term numbers of column as follow:

$$M_{ij} = \vec{D}_i \cdot \vec{T}_j$$

where M_{ij} is the cell at i-th row and j-th column in the matrix, $\vec{D}_i$ is the vector of the i-th document, $\vec{T}_i$ is the vector the j-th term in dictionary.

Each cell in the matrix is a value of cosine distance between the word and the document corresponding to the column and row, which could be considered as the similarity of the word to the document, this value is used to measure whether the word can represent the main meaning of the document.

f. select k-top words:

for each term, calculate the sum of matrix value of the column as total weight:

$$Weight_j = \sum_{i=1}^{LD_i} M_{ij}$$

where $Weight_j$ is the total weight of j-th term in dictionary, LD_i is the number of terms in D_i , M_{ij} is the cell at i-th row and j-th column in the matrix.

The weight value represent the total representable of the word to all the documents in the same topic.Select the top k words which has the highest weight, and remove the sentences which do not have any word in the top-k word list.

g. make the matrix again as the step-e did.

4.2 Sentence evaluation and selection with pre-trained vectors

Sentence evaluation and selection is another important step, based on the results given by the frequent weighted itemset, in order to find fewer sentences to cover more high-weight itemsets. In MWI-Sum, this work is done by a greedy algorithm that assigns a relevance value to each sentence by the weight of the frequently weighted itemset, and then selects the sentence with the highest relevance value between the sentences covering the most uncovered itemsets each time into the final summary. Here we adopt the concept of the coverage of sentences relative to the itemset, that is, a single sentence may not totally cover the meaning of single itemset, on the other side, even though the itemset is not covered by the sentence, they may have some common meanings innerly. Therefore, we use the coverage ratio to represent the coverage of a single sentence set by a single sentence, the value of which is expressed as the cosine distance between the feature vector of the sentence and the feature vector of the itemset. As with the generation method of the feature vector of document in section 4.1, the feature vectors of sentence and itemset canbe represented by the normalization of the sum of the term vectors. Similarly, sentences with the highest total coverage of all frequent itemsets that are not fully covered will be greedily selected to form the final summary.

The flow of this approach is as follows:

a. for each sentence in document, calculate the relevance score as follow:

$$SentScore_{ki} = \frac{\sum_{j=1}^{LS_i} M_{kj}}{LS_i}$$

where $SentScore_{ki}$ is the score of i-th sentence in k-th document, LS_i is the number of terms contains in the sentence, M_{kj} is the cell at k-th row and j-th column in the matrix, which measure the represent ability degree of the j-th term to the k-th document.

b. calculate vector for each itemset and each sentence:

$$\vec{It}_i = \frac{\sum_{j=1}^{LIt_i} \vec{T}_{ij}}{\left| \sum_{j=1}^{LIt_i} \vec{T}_{ij} \right|}$$

where $\overrightarrow{It_i}$ is the vector of the i-th itemset, LIt_i is the number of terms in It_i , $\overrightarrow{T_{ij}}$ is the vector the j-th term in It_i

$$\vec{S}_i = \frac{\sum_{j=1}^{LS_i} \vec{T}_{ij}}{\left| \sum_{j=1}^{LS_i} \vec{T}_{ij} \right|}$$

where $\vec{S}_i$ is the vector of the i-th sentence, LS_i is the number of terms in S_i , $\overrightarrow{T_{ij}}$ is the vector of the j-th term in S_i

c. for each sentence, calculate the coverage vectors and cover percent of the whole itemsets:

$$\overrightarrow{SC}_i = It\,\vec{S}_i^T$$

where $\overrightarrow{SC_i}$ is the coverage vector of the i-th sentence on the itemsets, It is the matrix of itemset vectors where i-th row is the vector of the i-th itemset, $\vec{S}_i$ is the vector of the i-th sentence

$$Cover_i = \frac{\sum_{j=1}^{n} \left(\vec{S}_i \cdot \overrightarrow{It_j} \right)}{n}$$

where $Cover_i$ is the cover percent of the i-th sentence, $\vec{S}_i$ is the vector of the i-th sentence, $\overrightarrow{It_i}$ is the vector of the i-th itemset, n is the total number of itemsets.

d. use a greedy method, to choose sentences, which covers most itemsets:

ALGORITHM: Greedy sentence selection with itemset vector and sentence vector

---

Require: vector of sentences SV
Require: vector of itemsets IV
Require: set of sentence coverage vectors SC
Require: set of sentence relevance scores SR
Ensure: summary SU
SU = ∅
SC∗ = set to all zeros() /*initialize the summary coverage vector with only zeros */
/* Cycle until SC∗ contains only values not less than 1 (i.e., until the generated summary covers all the itemsets
of the model) */
while SC∗ contains at least one value less than 1 do
MaxCoverageSentences = max coverage sentences(S, SC) /* Select the sentences with the highest coverage percent */
if MaxCoverageSentences is not empty then
S_{best} = arg $max_{s_j \in MaxCoverageSentences}$ SR(S_j) /* Select the sentence with maximum relevance score among the ones in MaxCoverageSentences */
SU = SU ∪ S_{best} /* Add the best sentence to the summary */
/* Update the summary coverage vector SC∗ . SC(S_{best}) ∈ SC is the sentence coverage vectorassociated with the best sentence S_{best} */
SC∗ = SC∗ + SC(S_{best}) /* Set the values associated with the itemsets covered by S_{best} to the total coverage */
/* Update the sentence coverage vectors in SC */
for all SC_{ij} in SC do

if $SC_{ij} > 1 - SC_j *$ then
$SC_{ij} = 1 - SC_j *$ /* if the coverage of sentence i to itemset j is larger than the part need to cover, it is set to the coverage of the required part*/
end for
end if
end while
return SU

---

5. Experiment results

In this section, some datasets of English-written and multilingual news documents are selected to generate summaries by vector driven MWI-Sum (Section 5.1), the generated summaries are scored by ROUGE system (Section 5.2). We compare the performance of vector driven MWI-Sum and other state-of-the-art summarizers on these datasets(Sections 5.3 and 5.4), and analyze the impact of algorithm parameters on the scores of summaries(Section 5.5). We performed the experiments on a 64-bit PC with 2.80GHz Intel Core i7-7700HQ CPU, 16GB memory and the system is Ubuntu 16.04 LTS (kernel 4.15.0-36-generic).

5.1 Textual Document Collections

The performance evaluation of vector driven MWI-Sum is tested on (i) DUC'04 English news clusters of the 2nd task [45], it is an English-written document collection of news which used as a competition for DUC'04, (ii) the TAC'11 datasets[46], a multi.lingual document collection of news which also used as a competition, and (iii) the MultiLing'13 datasets[47], which are the reference collection of the MultiLing'13 multilingual summarization task[48] in ACL 2013[Association for Computational Linguistics 2013]. The following will give a brief introduction to these collections.

5.1.1 DUC2004 English news clusters

This benchmark collection contains 50 event topics of news chosen by NIST, for each topic, 10 documents are chosen from the AP newswire and New York Times newswire. The 2nd task is to create a short multi-document summary (<= 665 bytes) of each cluster within fixed amount of time and space while the topic is unknown, and the summary will be truncated over the target length if it is beyond the limit. For each cluster, a short summary (<= 665 bytes) are created by NIST assessors manually, which could be regarded as the golden summary for ROUGE evaluation.

5.1.2 TAC2011 Multilingual news clusters

This benchmark collection is a human created multi-lingual documents set for news which are freely available from Internet. It contains 10 topics and each topic contains 10 documents. All of the documents are translated into seven languages (Arabic, Czech, English, French, Greek, Hebrew, Hindi) by native speakers. The task aims to create a short multi-document summary for each topic in each language, and a golden summary is offered for ROUGE evaluation. In this article, we test Arabic, Czech, English, French, Greek and Hindi languages for each documents cluster.

5.1.3 MultiLing2013 dataset

This dataset is a subset of a 2010 corpus which is created from Wikipedias originally used to measure the performance of summarizer on non-English documents outside the news

domain. Each language in this collection has no less than 30 articles in suitable size. In this article, we extract 15 clusters of Arabic, Czech, English, French, Greek, Hindi, Romanian and Spanish languages for generating summaries to be evaluated.

5.2 ROUGE[49] toolkit

ROUGE, which is short for Recall-Oriented Understudy for Gisting Evaluation, is a benchmark toolkit originally developed for DUC2002 to score the generated summary with a model summary for reference. The toolkit contains several methods such as ROUGE-N(N-gram based co-occurrence statics), ROUGE-L(LCS-based statistics), ROUGE-W(Weighted LCS-based statistics that favors consecutive LCSes), ROUGE-S(Skip-bigram-based co-occurrence statistics), ROUGE-SU(Skip-bigram plus unigram-based co-occurrence statistics). ROUGE-4 is a ROUGE-N method which calculate 4-gram co-occurrences between reference and the summary to be evaluated. It offers scores of Recall, Precision and F1-measure, by which the merits of the summary can be measured. In this article, we mainly use the Recall score of ROUGE-4 to evaluate the summaries generated by the summarizer.

Since the original ROUGE toolkit recognize tokens by regular expressions using English alphabet, it is applicable to English texts only. For Multilingual datasets, we use JRouge[50], which is a ROUGE metric implementation in JAVA using Unicode regular expressions to match tokens and punctuations, which making JRouge available through a variety of languages.

5.3 Performance of vector driven MWI-Sum compare with other state-of-the-art summarizers on English-written datasets

In order to verify whether the vector-driven MWI-Sum system improves the quality of the summarization, firstly we test the program on DUC’04 dataset and compare the score with other state-of-the-art summarizers. The comparison includes the following summarizers: ICSIsumm[51], OTS[52], TexLenAn[53], itemSum[20] and some summarizers involved in DUC’04 competition. In accordance with the requirements of the DUC’04 competition, we truncated all the results into 665 bytes to eliminate the impact of lenths on the score. The scores are shown in Table 5.1. The data of other summarizers are derived from previous studies[23] and DUC’04 reports[45].

From table 5.1, we can see that in the approaches described in this paper, the differences in the models(possibly training methods or training materials) have a certain impact on the quality of the automatic summary. The appropriate model(GoogleNews pre-trained vectors model) has a significant improvement over the original system(MWI-Sum) with the vector driven preprocessing approach, while the Fasttext pre-trained model did not significantly improve the results. On the other side, the approach on sentence selection stage could not bring a positive influence, and may even cause a drop in quality.

Table 5.1 Experiment on **English-written DUC'04 Collections**, comparison between the vector driven MWI-Sum(for googlenews vector driven preprocessing, support threshold = 3%, kTop = 26, nSentence = 8; for googlenews vector driven sent-selection, support threshold = 3%, kTop = 20, nSentence = 4; for fasttext driven preprocessing, support threshold = 3.5%, kTop = 24, nSentence = 6; for fasttext driven sent-selection, support threshold = 3%, kTop=24, nSentence = 3) and the other approaches

DUC2004 – English	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
ELSA	0.0170	0.0860
ICSISumm Tuned [Hong et al. 2014]	0.0165	0.0905
TextRank	0.0161	0.0896
DPP [Hong et al. 2014]	0.0158	0.0952
Classy-peer67	0.0157	0.0854
RegSum [Hong et al. 2014]	0.0152	0.0914
Submodular	0.0148	0.0935
CLASSY04 [Hong et al. 2014]	0.0146	0.0860
Classy-peer65	0.0146	0.0863
googlenews_vec_driven_preprocessing_MWI_Sum	**0.0140**	**0.0928**
ClusterCMRW	0.0139	0.0889
CLASSY11 [Hong et al. 2014]	0.0137	0.0852
Submodular [Hong et al. 2014]	0.0135	0.0907
Classy-peer66	0.0134	0.0833
Centroid	0.0130	0.0806
MWI-sum	**0.0128**	**0.0860**
OCCAMS V [Hong et al. 2014]	0.0127	0.0884
fasttext_vec_driven_preprocessing_MWI_Sum	**0.0127**	**0.0933**
googlenews_vec_driven_sent-selection_MWI_Sum	**0.0125**	**0.0794**
ICSISumm Standard	0.0124	0.0829
Peer81	0.0123	0.0763
GeedyKL [Hong et al. 2014]	0.0122	0.0820
LexPageRank	0.0121	0.0822
peer124	0.0117	0.0780
fasttext_vec_driven_sent-selection_MWI_Sum	**0.0116**	**0.0789**
Coverage	0.0116	0.0739
Centroid [Hong et al. 2014]	0.0115	0.0771
FreqSum [Hong et al. 2014]	0.0103	0.0781
TsSum [Hong et al. 2014]	0.0101	0.0763
peer102	0.0099	0.0795
peer104	0.0099	0.0803
peer19	0.0099	0.0751
peer35	0.0099	0.0777
ILP	0.0090	0.0680
LexRank [Hong et al. 2014]	0.0086	0.0741
peer34	0.0081	0.0716
AMTS	0.0078	0.0598
Lead	0.0063	0.0527

Table 5.2 Experment on **Arabic-written** collections of **TAC'11**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 7.5%, kTop = 10, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.6%, kTop = 18, nSentence = 3, minimum 15 sentences) and the other approaches

Arabic	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.2060	0.2750
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.1707**	**0.2370**
ELSA	0.1520	0.2167
fasttext vect-driven sent_selection MWI-Sum	**0.1469**	**0.2090**
fasttext vect-driven preprocessing MWI-Sum	**0.1453**	**0.2030**
JRC	0.1340	0.1931
LIF	0.0889	0.1579
CLASSY	0.0800	0.1577
UoEssex	0.0697	0.1213
ItemSum	0.0678	0.1195
MWI-Sum	**0.0671**	**0.1172**
TALN_UPF	0.0600	0.1125
CIST	0.0502	0.1010
AMTS	0.0482	0.0768
UBSummarizer	0.0374	0.0761
ICSIsumm	0.0216	0.0334

Table 5.3 Experment on **Czech-written** collections of **TAC'11**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 5.5%, kTop = 12, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.4%, kTop = 12, nSentence = 4, minimum 15 sentences) and the other approaches

Czech	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.0885	0.1775
ELSA	0.0744	0.1610
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0678**	**0.1513**
CIST	0.0660	0.1265
JRC	0.0646	0.1476
CLASSY	0.0613	0.1430
MWI-Sum	**0.0592**	**0.1392**
fasttext vect-driven preprocessing MWI-Sum	**0.0583**	**0.1305**
LIF	0.0583	0.1280
fasttext vect-driven sent_selection MWI-Sum	**0.0539**	**0.1384**
ICSIsumm	0.0536	0.1257
UBSummarizer	0.0447	0.0926
OTS	0.0411	0.1033
AMTS	0.0214	0.0557

Table 5.4 Experment on **English-written** collections of **TAC'11**, comparison between the vector driven MWI-Sum(for googlenews vector driven preprocessing, support threshold = 4%, kTop = 12, nSentence = 6, minimum 15 sentences for extend mode; for googlenews vector driven sent-selection, support threshold = 0.2%, kTop = 22, nSentence = 3, minimum 15 sentences; for fasttext driven preprocessing, support threshold = 4%, kTop = 20, nSentence = 6, minimum 15 sentences for extend mode; for fasttext driven sent-selection, support threshold = 0.2%, kTop=22, nSentence = 3, minimum 15 sentences) and the other approaches

English	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
fasttext vect-driven sent_selection MWI-Sum	**0.0792**	**0.1926**
GoogleNews vect-driven preprocessing MWI-Sum(extended)	**0.0791**	**0.1924**
GoogleNews vect-driven sent_selection MWI-Sum	**0.0787**	**0.1878**
GoogleNews vect-driven preprocessing MWI-Sum	**0.0785**	**0.1905**
fasttext vect-driven preprocessing MWI-Sum	**0.0775**	**0.1922**
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0775**	**0.1922**
Tuned ELSA	0.0751	0.1906
ICSIsumm	0.0728	0.1977
JRC	0.0680	0.1736
MWI-Sum	**0.0628**	**0.1733**
ELSA	0.0628	0.1742
CLASSY	0.0541	0.1707
TexLexAn	0.0531	0.1550
LIF	0.0522	0.1517
Coverage	0.0503	0.1287
OTS	0.0470	0.1407
ItemSum	0.0441	0.1211
SIEL_IIITH	0.0432	0.1362
ClusterCMRW	0.0404	0.1161
UoEssex	0.0377	0.1212
CIST	0.0374	0.1216
TextRank	0.0371	0.1006
Centroid	0.0362	0.0991
Submodular	0.0355	0.1111
TALN_UPF	0.0350	0.1068
AMTS	0.0348	0.1094
LexPageRank	0.0316	0.0922
ILP	0.0306	0.1048
Lead	0.0192	0.0745
UBSummarizer	0.0178	0.0967

Table 5.5 Experment on **French-written** collections of **TAC'11**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 8%, kTop = 10, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.4%, kTop = 10, nSentence = 3, minimum 15 sentences) and the other approaches

French	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.1126**	**0.2267**
Tuned ELSA	0.1078	0.2232
ELSA	0.0996	0.2109
fasttext vect-driven sent_selection MWI-Sum	**0.0991**	**0.2055**
ICSIsumm	0.0952	0.2201
fasttext vect-driven preprocessing MWI-Sum	**0.0860**	**0.1768**
JRC	0.0824	0.1913
MWI-Sum	**0.0753**	**0.1865**
OTS	0.0612	0.1637
CLASSY	0.0565	0.1848
Coverage	0.0535	0.1362
Submodular	0.0518	0.1339
LIF	0.0517	0.1531
UBSummarizer	0.0474	0.1410
ClusterCMRW	0.0461	0.1311
ILP	0.0451	0.1367
CIST	0.0436	0.1409
AMTS	0.0434	0.0992
TextRank	0.0377	0.1037
SIEL_IIITH	0.0346	0.1165
LexPageRank	0.0337	0.1048
TexLexAn	0.0316	0.1221
Centroid	0.0278	0.0875
Lead	0.0260	0.0865
TALN_UPF	0.0208	0.0977

5.4 Performance of vector driven MWI-Sum compare with other state-of-the-art summarizers on Multi-lingual datasets

Experiments on multilingual datasets are concentrated in TAC'11 and MultiLing'13. For the English part, we use both GoogleNews-Vectors-negative300 and FastTest vector models, and for other languages we only use the vector model provided by FastText.

In order to ensure fairness, we use the evaluation criteria of MultiLing'13 to measure the results, that is, the summaries are truncated to 250 words and then scores through the Jrouge system, and the quality of the generated summaries are measured according to the Recall value in the score.

Table 5.6 Experment on **Greek-written** collections of **TAC'11**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 6.5%, kTop = 18, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.8%, kTop = 10, nSentence = 6, minimum 15 sentences) and the other approaches

Greek	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0380**	**0.1379**
Tuned ELSA	0.0368	0.1268
fasttext vect-driven sent_selection MWI-Sum	**0.0361**	**0.1265**
CLASSY	0.0352	0.1354
MWI-Sum	**0.0351**	**0.1207**
fasttext vect-driven preprocessing MWI-Sum	**0.0349**	**0.1271**
ELSA	0.0311	0.1210
JRC	0.0228	0.1103
CIST	0.0218	0.0883
LIF	0.0186	0.1062
OTS	0.0182	0.0805
AMTS	0.0173	0.0594
ICSIsumm	0.0168	0.0636
UBSummarizer	0.0149	0.0695

Table 5.7 Experment on **hindi-written** collections of **TAC'11**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 12%, kTop = 15, nSentence = 4, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 1.5%, kTop = 28, nSentence = 7, minimum 15 sentences) and the other approaches

Hindi	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.1144	0.3680
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0988**	**0.3446**
ELSA	0.0961	0.3454
fasttext vect-driven sent_selection MWI-Sum	**0.0948**	**0.3350**
fasttext vect-driven preprocessing MWI-Sum	**0.0903**	**0.3083**
CLASSY	0.0847	0.3332
SIEL_IIITH	0.0794	0.3048
JRC	0.0767	0.2938
MWI-Sum	**0.0730**	**0.2887**
CIST	0.0712	0.3023
LIF	0.0484	0.2178
TALN_UPF	0.0476	0.2597
UBSummarizer	0.0474	0.2824
AMTS	0.0464	0.1603
ICSIsumm	0.0148	0.0580

Table 5.8 Experment on **Arabic-written** collections of MultiLing**'13**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 7.5%, kTop = 10, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.6%, kTop = 18, nSentence = 3, minimum 15 sentences) and the other approaches

Arabic	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.1479	0.2085
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.1285**	**0.1911**
fasttext vect-driven preprocessing MWI-Sum	**0.1113**	**0.1653**
fasttext vect-driven sent_selection MWI-Sum	**0.1111**	**0.1657**
ELSA	0.1097	0.1723
WBU	0.0997	0.1668
MWI-Sum	**0.0894**	**0.1476**
Maryland peer1	0.0715	0.1320
Maryland peer11	0.0673	0.1326
Shamoon peer5	0.0596	0.1002
CIST	0.0573	0.1188
Maryland peer21	0.0492	0.1059
Shamoon peer51	0.0367	0.0856
AMTS	0.0364	0.0608
Lancaster	0.0201	0.0522
ICSIsumm	0.0147	0.0236

Table 5.9 Experment on **Czech-written** collections of MultiLIng**'13**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 7.5%, kTop = 12, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.4%, kTop = 12, nSentence = 4, minimum 15 sentences) and the other approaches

Czech	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.1153	0.2046
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.1067**	**0.1901**
ELSA	0.1032	0.1884
fasttext vect-driven sent_selection MWI-Sum	**0.0948**	**0.1828**
WBU	0.0934	0.1788
fasttext vect-driven preprocessing MWI-Sum	**0.0841**	**0.1547**
Maryland peer11	0.0772	0.1662
Maryland peer21	0.0706	0.1514
Maryland peer1	0.0697	0.1551
ICSIsumm	0.0589	0.1347
CIST	0.0587	0.1232
AMTS	0.0202	0.0513
MWI-Sum	**0.0123**	**0.0341**

Table 5.11 Experment on **English-written** collections of MultiLIng**'13**, comparison between the vector driven MWI-Sum(for googlenews vector driven preprocessing, support threshold = 2%, kTop = 12, nSentence = 6, minimum 15 sentences for extend mode; for googlenews vector driven sent-selection, support threshold = 0.2%, kTop = 22, nSentence = 3, minimum 15 sentences; for fasttext driven preprocessing, support threshold = 4%, kTop = 20, nSentence = 6, minimum 15 sentences for extend mode; for fasttext driven sent-selection, support threshold = 0.2%, kTop=22, nSentence = 3, minimum 15 sentences) and the other approaches

English	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.0746	0.1862
GoogleNews vect-driven preprocessing MWI-Sum(extended)	**0.0719**	**0.1817**
GoogleNews vect-driven preprocessing MWI-Sum	**0.0719**	**0.1817**
ICSIsumm	0.0713	0.1938
ELSA	0.0692	0.1829
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0671**	**0.1711**
fasttext vect-driven preprocessing MWI-Sum	**0.0664**	**0.1702**
fasttext vect-driven sent_selection MWI-Sum	**0.0663**	**0.1693**
GoogleNews vect-driven sent_selection MWI-Sum	**0.0649**	**0.1650**
WBU	0.0640	0.1711
Maryland peer1	0.0502	0.1614
Maryland peer11	0.0492	0.1610
MWI-Sum	**0.0462**	**0.1389**
CIST	0.0447	0.1467
Maryland peer21	0.0415	0.1418
Coverage	0.0335	0.0931
Shamoon peer5	0.0315	0.1105
ClusterCMRW	0.0259	0.0794
TextRank	0.0248	0.0716
Centroid	0.0241	0.0704
Submodular	0.0237	0.0813
AMTS	0.0224	0.0567
Shamoon peer51	0.0224	0.0965
LexPageRank	0.0211	0.0690
ILP	0.0204	0.0763
Lancaster	0.0169	0.0828
Lead	0.0128	0.0565

The ROUGE system is rated by the n-gram comparison between generated summary and gold summary and calculate the coverage, the longer the summary, the higher the coverage, the higher the score. Therefore, length control of the generated summary is very necessary. Generally, in order to make a fair judgement, we should make each summary longer than the requirement and then truncate it, but for MWI-Sum, it is difficult to generate long summaries more than 250 words due to the explosive growth of runtime and memory required. In order to make the generated summary long enough, we introduce an extend mode here. The extend

Table 5.12 Experment on **French-written** collections of MultiLIng**'13**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 8%, kTop = 10, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.4%, kTop = 10, nSentence = 3, minimum 15 sentences) and the other approaches

French	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.1168	0.2309
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.1126**	**0.2267**
ELSA	0.1063	0.2167
ICSIsumm	0.1012	0.2243
WBU	0.1007	0.2134
fasttext vect-driven sent_selection MWI-Sum	**0.0999**	**0.2081**
fasttext vect-driven preprocessing MWI-Sum	**0.0860**	**0.1768**
Maryland peer11	0.0816	0.2013
Maryland peer1	0.0806	0.2008
MWI-Sum	**0.0768**	**0.1858**
Maryland peer21	0.0625	0.1772
Coverage	0.0585	0.1412
Submodular	0.0570	0.1383
ClusterCMRW	0.0564	0.1418
CIST	0.0560	0.1660
ILP	0.0480	0.1390
AMTS	0.0463	0.1006
TextRank	0.0403	0.1061
LexPageRank	0.0373	0.1086
Centroid	0.0311	0.0920
Lead	0.0265	0.0868

mode is to set a minimum number of sentences, when in the sentence selection stage, if the total number of sentences in the generated summary does meet the minimum value setted, and there are still sentences not selected, the program will fill the summary with the sentence which has the highest coverage until the summary has enough sentences. In the experiment, the minimum sentences number is set to 15.

Table 5.2 – 5.7 shows the performance of vector driven MWI-Sum in Arabic, Czech, English, French, Greek and Hindi in dataset TAC'11, and table 5.8-5.15 shows the performance of vector driven MWI-Sum in Arabic, Czech, English, French, Greek, Hindi, Romanian and Spanish in dataset MultiLing2013.

The experiment results show that in most cases, both approaches can improve the summary quality of the multi-language test set under the MultiLing'13 standard, and the vector driven preprocessing method is more obvious than the vector driven sentence selection method. In addition, although the MWI-Sum system can produce high quality summaries with the two approaches, we can still further improve the rating by extend mode since the generated summaries were not long enough. In the end, the overall perforamance of vector driven preprocessing MWI-Sum is best, and its performance is better than all the other state-of-the-art approaches except Tuned ELSA.

Table 5.10 Experment on **Greek-written** collections of MultiLIng**'13**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 6.5%, kTop = 18, nSentence = 3, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.8%, kTop = 10, nSentence = 6, minimum 15 sentences) and the other approaches

Greek	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0405**	**0.1368**
Tuned ELSA	0.0390	0.1294
fasttext vect-driven preprocessing MWI-Sum	**0.0352**	**0.1204**
ELSA	0.0338	0.1243
fasttext vect-driven sent_selection MWI-Sum	**0.0338**	**0.1209**
Maryland peer11	0.0329	0.1237
Maryland peer1	0.0313	0.1200
Maryland peer21	0.0312	0.1117
MWI-Sum	**0.0285**	**0.1074**
WBU	0.0260	0.1104
CIST	0.0195	0.1002
ICSIsumm	0.0142	0.0567
AMTS	0.0142	0.0512

Table 5.13 Experment on **Hindi-written** collections of MultiLIng**'13**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 12%, kTop = 15, nSentence = 4, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 1.5%, kTop = 28, nSentence = 7, minimum 15 sentences) and the other approaches

Hindi	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.1144	0.3680
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0988**	**0.3446**
ELSA	0.0961	0.3454
fasttext vect-driven sent_selection MWI-Sum	**0.0948**	**0.3355**
WBU	0.0934	0.3192
fasttext vect-driven preprocessing MWI-Sum	**0.0903**	**0.3083**
Maryland peer11	0.0874	0.3439
CIST	0.0809	0.3456
Maryland peer21	0.0800	0.3285
Maryland peer1	0.0795	0.3348
MWI-Sum	**0.0679**	**0.2425**
AMTS	0.0464	0.1603
ICSIsumm	0.0091	0.0482

Table 5.14 Experment on **Romanian-written** collections of MultiLIng**'13**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 6.5%, kTop = 15, nSentence = 4, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 2%, kTop = 20, nSentence = 4, minimum 15 sentences) and the other approaches

Romanian	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.0960	0.1910
fasttext vect-driven sent_selection MWI-Sum	**0.0919**	**0.1833**
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.0821**	**0.1777**
fasttext vect-driven preprocessing MWI-Sum	**0.0801**	**0.1654**
ELSA	0.0760	0.1637
WBU	0.0681	0.1655
Maryland peer21	0.0496	0.1381
Maryland peer1	0.0493	0.1489
ICSIsumm	0.0469	0.1504
Maryland peer11	0.0437	0.1392
CIST	0.0385	0.1202
AMTS	0.0332	0.0714
MWI-Sum	**0.0152**	**0.0461**

Table 5.15 Experment on **Spanish-written** collections of MultiLIng**'13**, comparison between the vector driven MWI-Sum(for Fasttext vector model driven preprocessing, support threshold = 4.5%, kTop = 10, nSentence = 4, minimum 15 sentences for extend mode; for Fasttext vector model driven sentence selection, support threshold = 0.6%, kTop = 16, nSentence = 5, minimum 15 sentences) and the other approaches

Spanish	ROUGE-4	ROUGE-2
Summarizer	Recall	Recall
Tuned ELSA	0.1408	0.2773
fasttext vect-driven sent_selection MWI-Sum	**0.1356**	**0.2597**
fasttext vect-driven preprocessing MWI-Sum(extended)	**0.1187**	**0.2500**
ELSA	0.1145	0.2458
fasttext vect-driven preprocessing MWI-Sum	**0.1138**	**0.2429**
MWI-Sum	**0.1133**	**0.2377**
ICSIsumm	0.1116	0.2565
WBU	0.1039	0.2274
Coverage	0.1008	0.1994
Maryland peer11	0.0877	0.2180
ILP	0.0716	0.1723
ClusterCMRW	0.0705	0.1702
Maryland peer1	0.0670	0.1975
CIST	0.0669	0.1801
Submodular	0.0625	0.1647
Maryland peer21	0.0596	0.1754
AMTS	0.0534	0.1011
Centroid	0.0488	0.1308
TextRank	0.0474	0.1357
Lead	0.0469	0.1198
LexPageRank	0.0428	0.1253

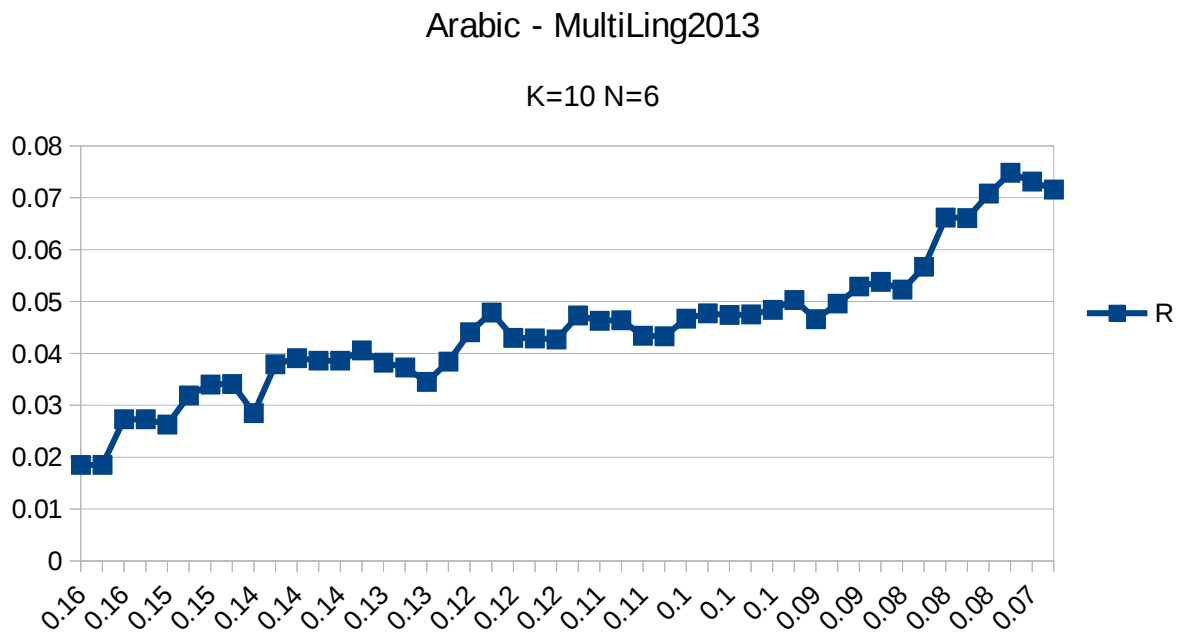
5.4 Analysis of Parameters

From the experiments of multi-language dataset, we can see that in the same language, the parameters that give the optimal solution have certain commonalities and rules to follow. We analyzed the effects of several parameters of the program, looked for a more suitable parameter combintion for the program by observing the influence of the Recall score judged by JROUGE with the change of the specific parameter. Inherited from MWI-Sum, there are three parameters worthy of our attention: support threshold, kTop an nSentence.

Support threshold represents the ratio between the minimum weight and the total weight of items that can be selected for text mining. In theory, too small an support threshold will cause the program to contain too many low-weight(inefficent) items, wich will put a lot of burden on the performance of the program, whilt too large the support threshold value will lead to a loss of some important items for the documetns, which will cause a loss of a lot of information. We set the values of the ramining two parameters kTop = 10 and nSentence = 6, then gradually reduce the st value and do experiment on MultiLing'13 dataset to observe the change of the Recall score of generated summaries with the driven of Fasttext model vectors

judged by JROUGE. The results are shown in figure 5.1 – 5.8, which shows the impact of results on Arabic, Czech, English, French, Greek, Hindi, Romanian and Spanish. The figures show that for all languages, when support threshold shrinks, the score tends to rise.When the support threshold value is to a certain small value, due to too many itemsets to be processed in the program, the time and space consumed by the program will increase exponentially, so it is difficult to obtain subsequent related results.

Figure 5.1 Experment on **Arabic-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.



The parameter kTop represents how many terms to choose as the essential ones for sentences to be kept for further mining in preprocessing stage. And nSentence represents the number of sentences the program extracts from each document to remove the subsequent sentences which mary contain less information. Since the frequency of words in different languages and the amount of information contained in each sentence are different, the influence of these two parameters on the results presents the specificity based on the type of language. With the support threshold in an appropriate value, we experimented the MultiLing'13 dataset using the FastText vector driven preprocessing MWI-Sum. The results are presented in Figure 5.9-5.14. Overall, the results of the experiments show that, in most cases, the program is more likely to achieve better results when the value of nSentence is smaller. The effect of kTop value on the results is different among languages. Some languages such as Arabic, Czech, French, Hindi tends to present good results when kTop is lower(10-15), while the others such as English and Greek show advantage when K value is higher(18-20).

Figure 5.2 Experment on **Czech-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.

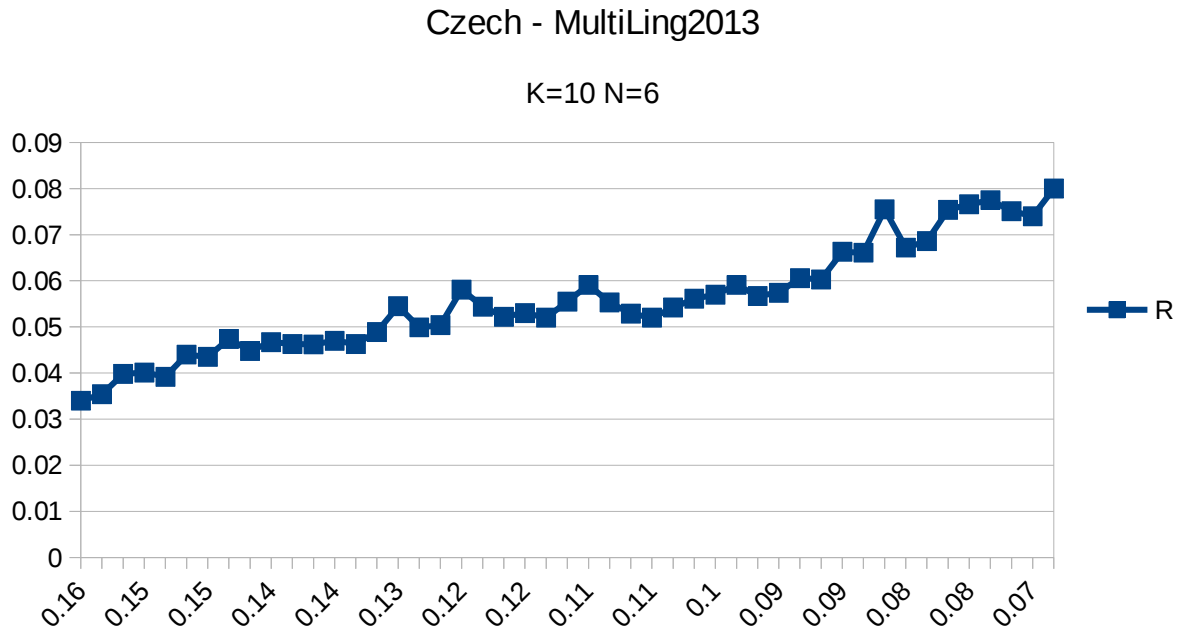


Figure 5.3 Experment on **English-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.

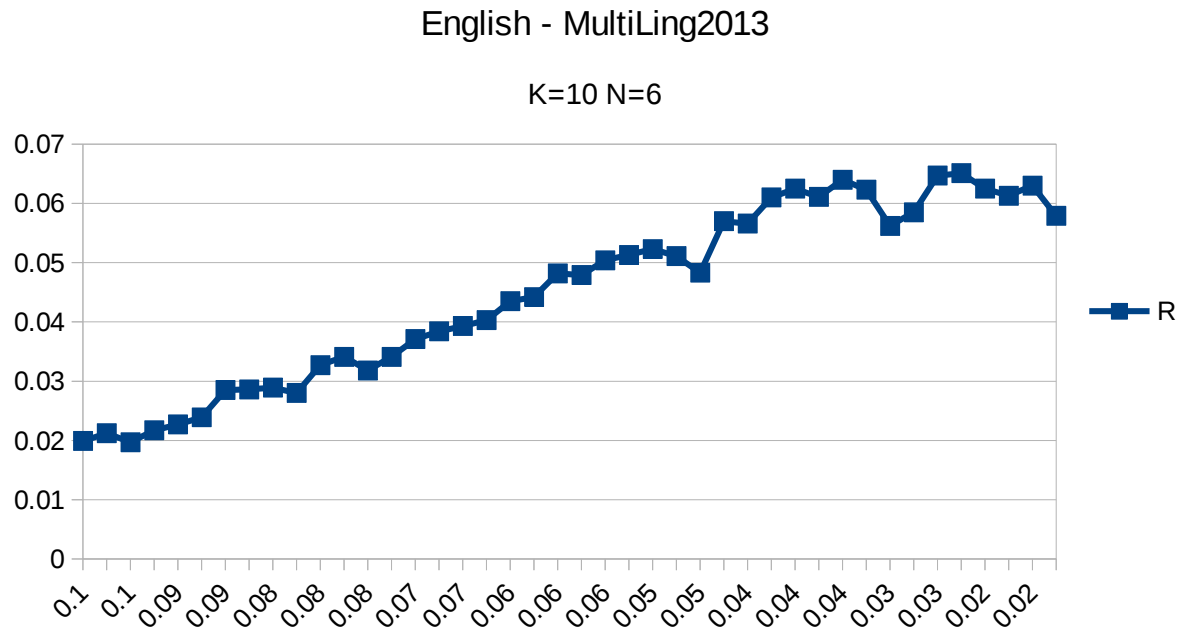


Figure 5.4 Experment on **French-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.

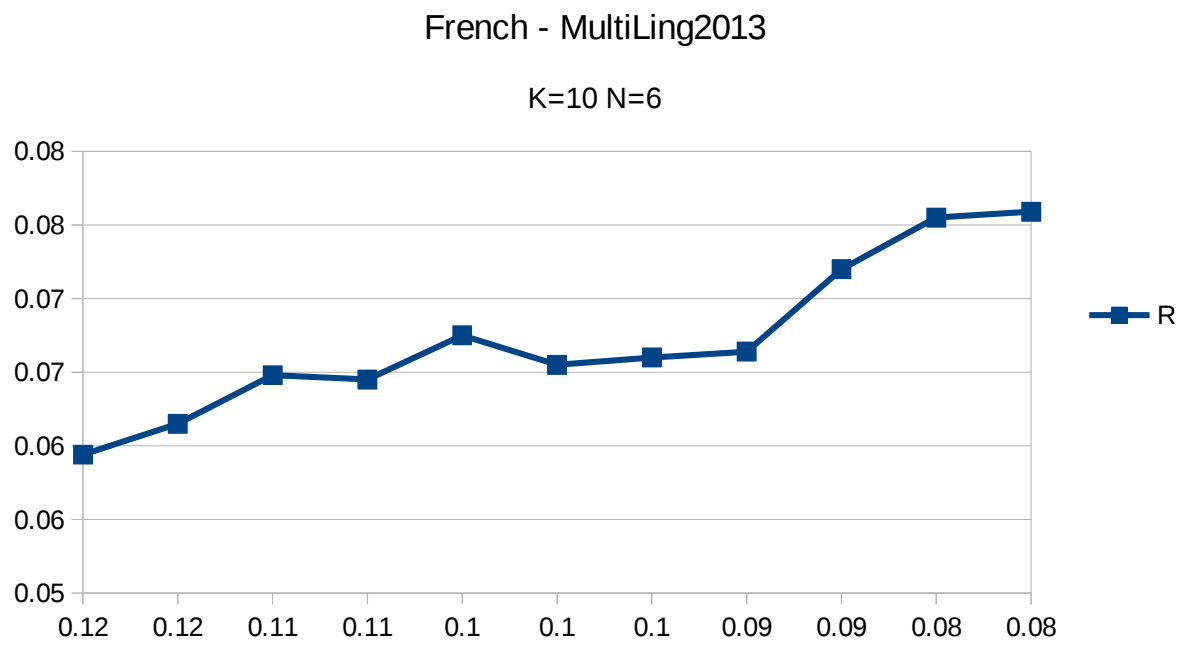


Figure 5.5 Experment on **Greek-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.

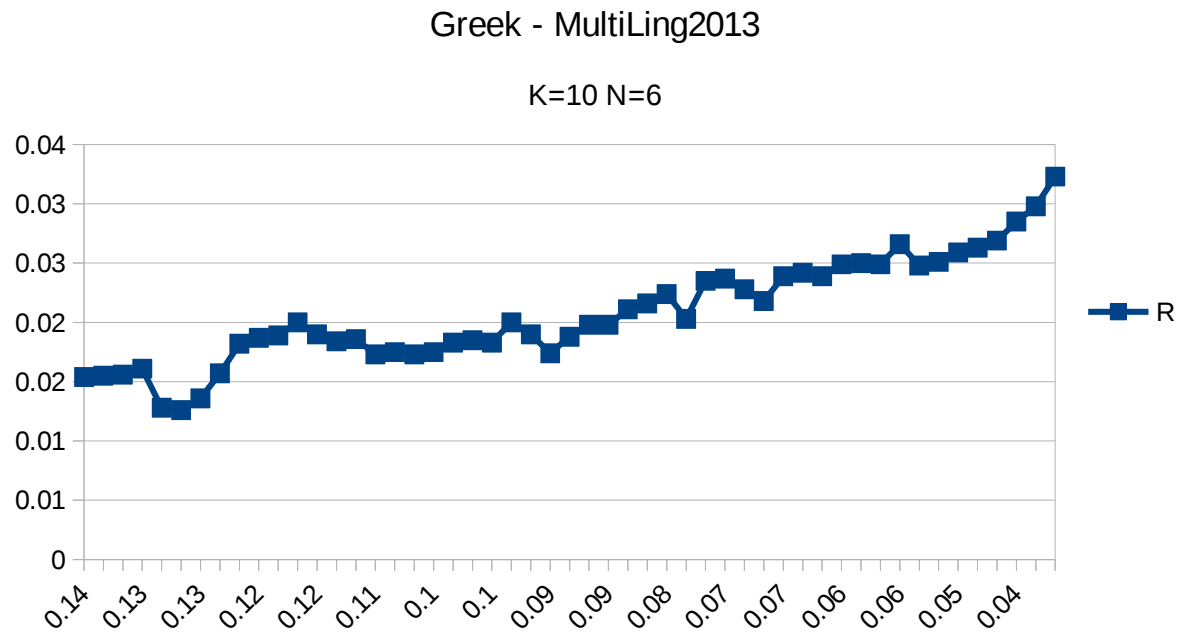


Figure 5.6 Experment on **Hindi-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.

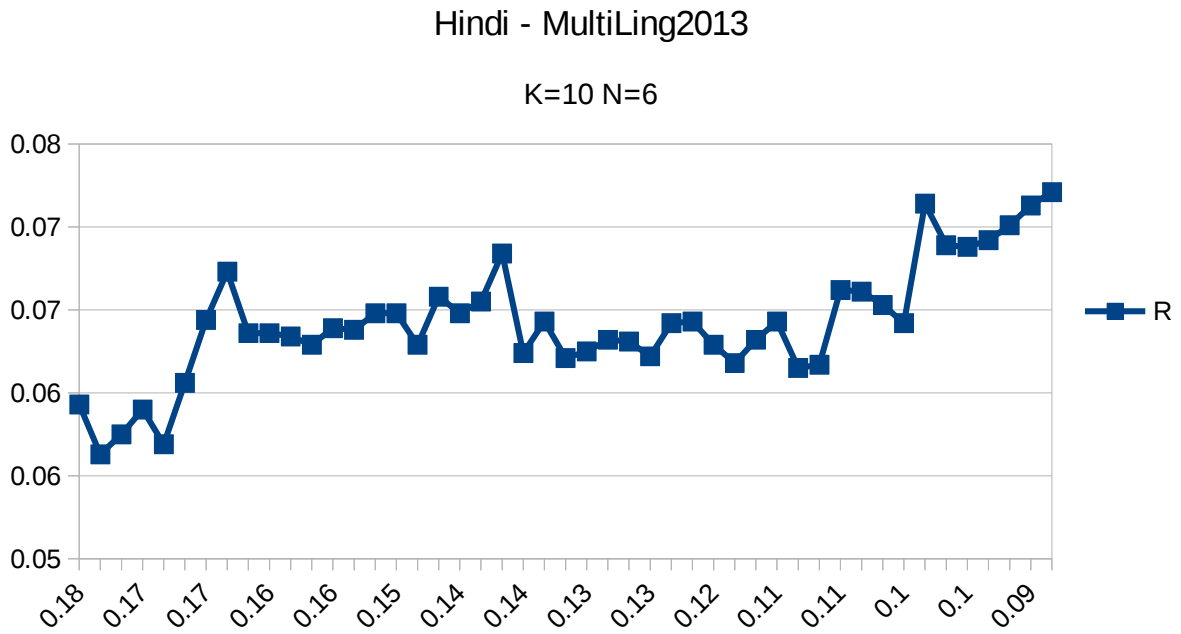


Figure 5.7 Experment on **Romanian-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.

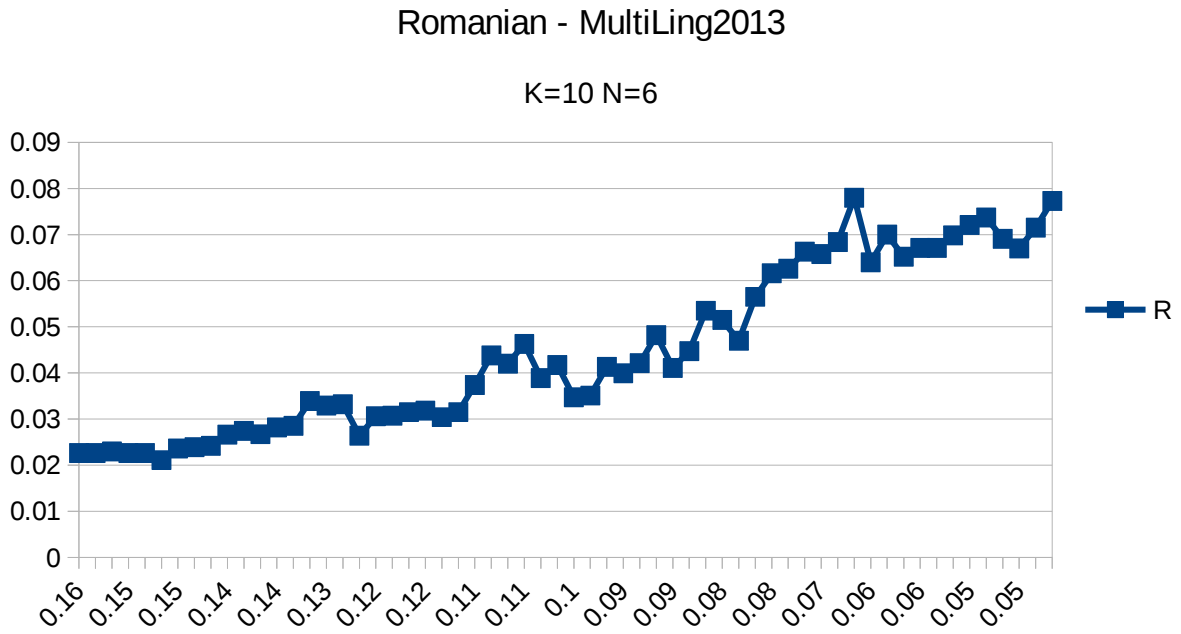


Figure 5.8 Experment on **Spanish-written** collections of MultiLIng**'13**, the impact of support threshold, with the other parameters ktop = 10, nSentence = 6.

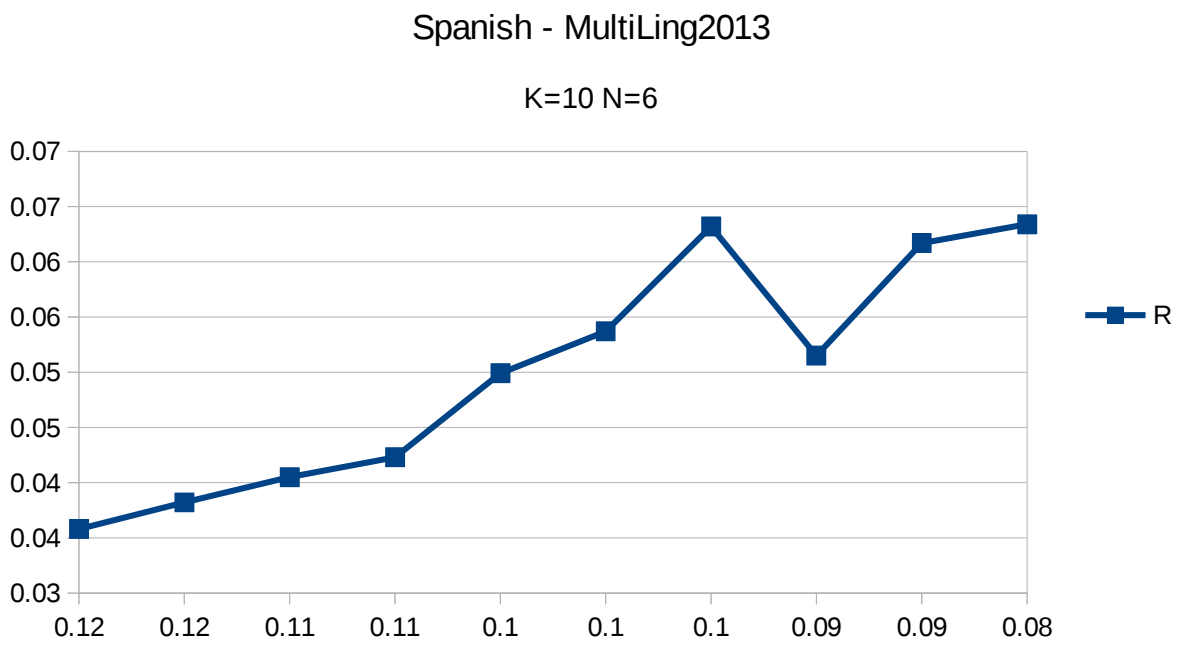


Figure 5.9 Experment on **Arabic-written** collections of MultiLIng**'13**, the impact of kTop and nSentence, with support threshold = 7.5%

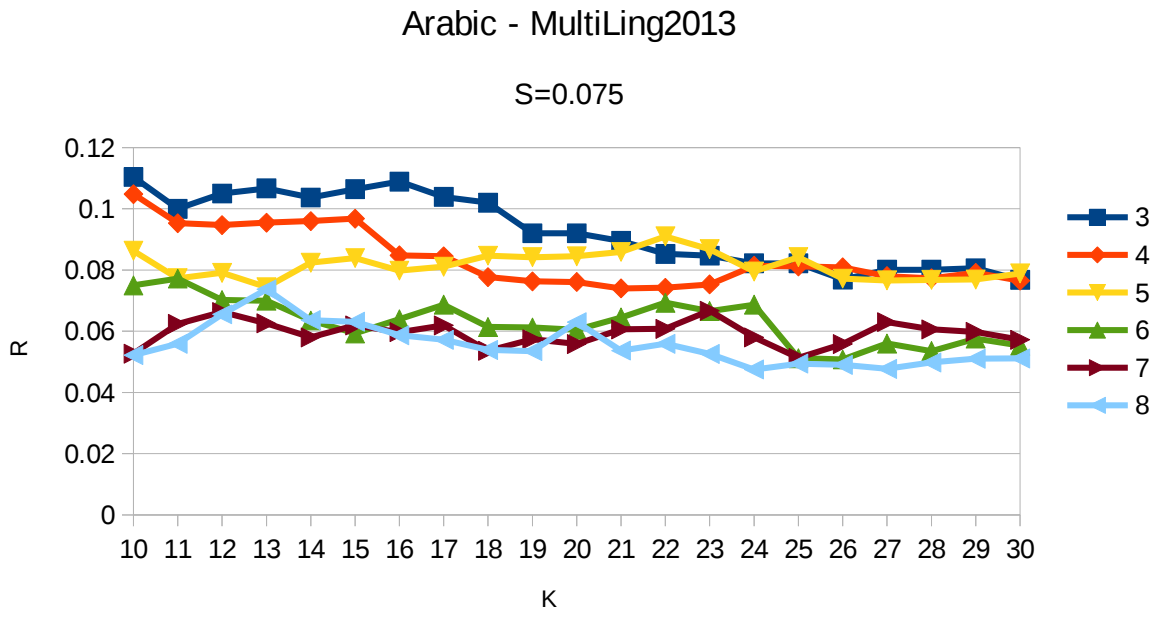


Figure 5.10 Experment on **Czech-written** collections of MultiLIng**'13**, the impact of kTop and nSentence, with support threshold = 7.5%

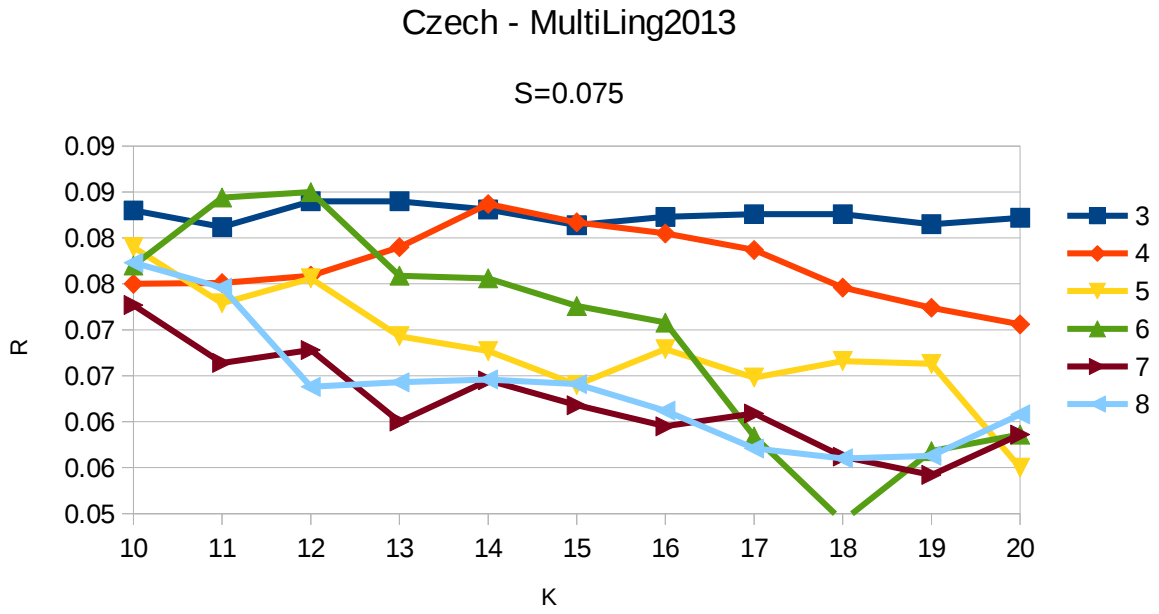


Figure 5.11 Experment on **English-written** collections of MultiLIng**'13**, the impact of kTop and nSentence, with support threshold = 4%

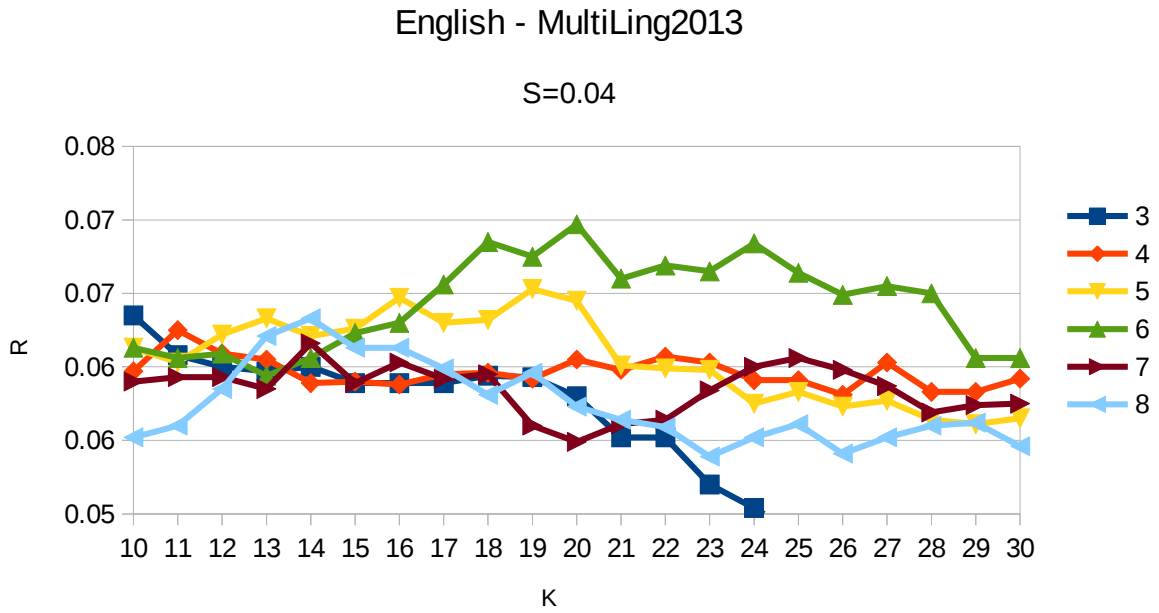


Figure 5.12 Experment on **French-written** collections of MultiLIng**'13**, the impact of kTop and nSentence, with support threshold = 8%

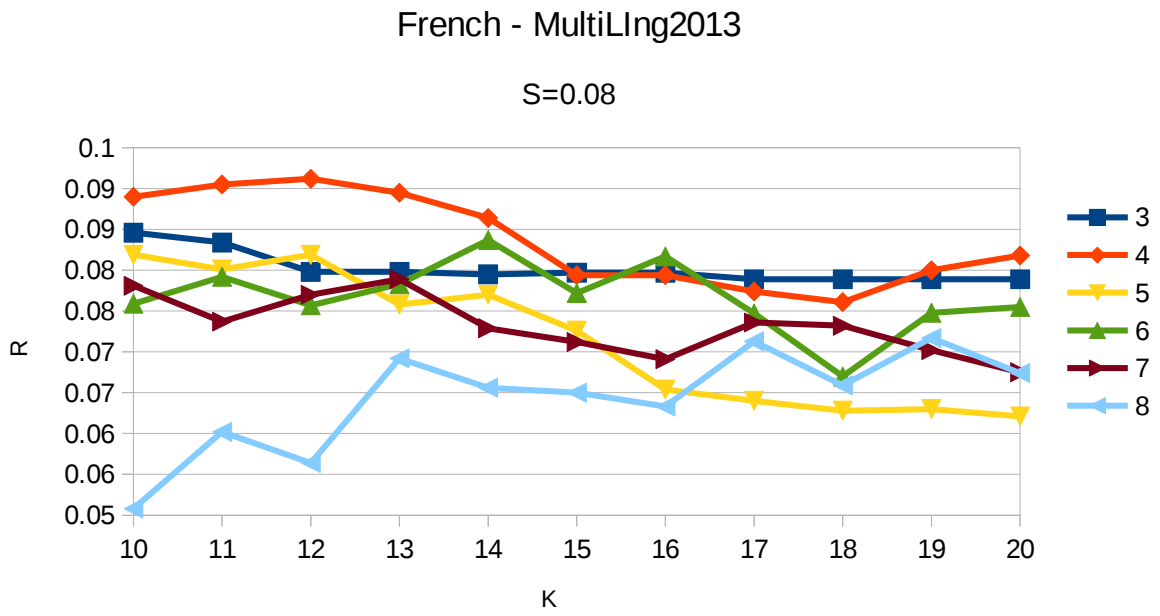


Figure 5.13 Experment on **Greek-written** collections of MultiLIng**'13**, the impact of kTop and nSentence, with support threshold = 6.5%

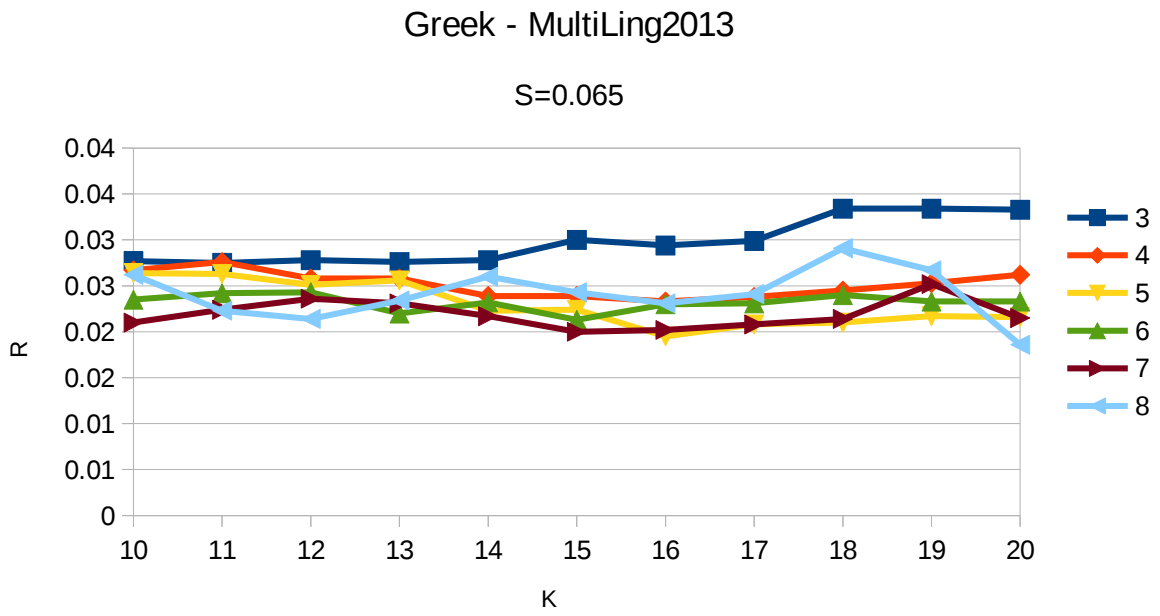
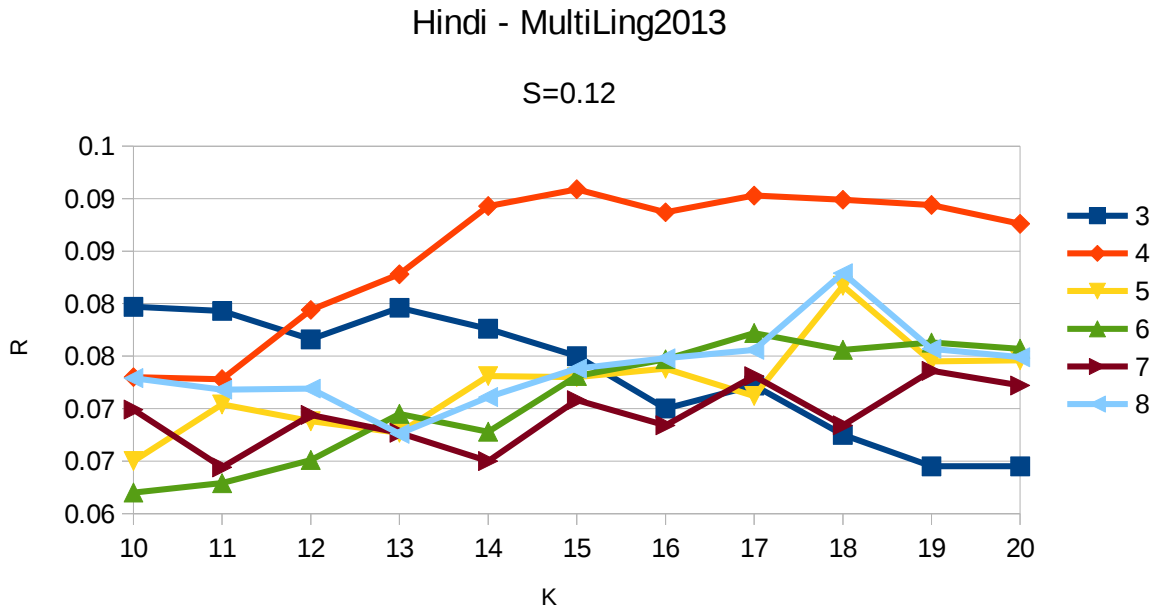


Figure 5.14 Experment on **Hindi-written** collections of MultiLIng**'13**, the impact of kTop and nSentence, with support threshold = 6.5%



6. Conclusions and future work

This paper proposes two novel methods to improve the performance of the original summarizer by combining word embedding technology with text mining based on weighted itemsets. These approach can be widely used in various languages and plays a very effective role in generating document summaries. The experimental results show that the proposed approach for pre-processing stage can effectively improve the quality of the generated summaries, and is favorably to the other current summarizers.

Since the information introduced from word embedding does improve the original system, we envision whether more information can be used(such as adding analysis at the phrase level), the approach proposed in this paper produces the feature vector of document and sentence by summation, it still lose many information which may be worthy for the summary task. Another idea is to improve the performance of the system. In the research of this paper, we find that MWI-Sum based on the weighted itemset is weak in the completion of the summary task requiring more words(e.g. MultiLIng2013 standard 250 words). In addition, due to the addition of the vector processing module on the basis of text mining, the program is slowed down and memory required increased significantly. Solving these issues will further improve the quality of the abstract. On the other side, the existing approach needs to set the parameters according to experience, but the values of parameters should be regular, or can be learned through the study of the corpus by the program, how to let the program automatically set the parameters remains to be studied.

For word embedding, we expect new progress in research on sense embedding technology, which will help our system understand more about the documents and words. In addition, in the research of this paper, the model trained through news data shows better results on the test set of news corpus. We consider whether the material type used to train the model has an influence on the performance of the summarizer.

The approaches described in this paper has only been tested on the test set of news content, but given the versatility of word vectors and itemset, we believe that it is likely to be applicable to the summary tasks of other stylistics.

References

[1] Dingding Wang and Tao Li. 2010. Document update summarization using incremental hierarchical clustering. In Proceedings of the 19th ACM International Conference on Information and Knowledge Management. 279–288.

[2] Dingding Wang, Shenghuo Zhu, Tao Li, Yun Chi, and Yihong Gong. 2011. Integrating document clustering and multidocument summarization. ACM Trans. Knowl. Discov. Data 5, 3, (August 2011), Article 14, 26 pages.

[3] Dragomir R. Radev, Hongyan Jing, Malgorzata Stys, and Daniel Tam. 2004. Centroid-based summarization of multiple documents. Inf. Process. Manag. 40, 6 (2004), 919–938.

[4] Chin-Yew Lin and Eduard Hovy. 2003. Automatic evaluation of summaries using N-gram co-occurrence statistics. In Proceedings of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1. 71–78.

[5] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. 2003. Clustering data streams: Theory and practice. IEEE Trans. on Knowl. and Data Eng. 15, 3 (March 2003), 515–528.

[6] Junyan Zhu, Can Wang, Xiaofei He, Jiajun Bu, Chun Chen, Shujie Shang, Mingcheng Qu, and Gang Lu. 2009. Tag-oriented document summarization. In Proceedings of the 18th International Conference on World Wide Web (WWW'09). ACM, New York, NY, 1195–1196.

[7] Zi Yang, Keke Cai, Jie Tang, Li Zhang, Zhong Su, and Juanzi Li. 2011. Social context summarization. In Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11). ACM, New York, NY, 255–264.

[8] Elena Baralis, Luca Cagliero, Naeem A. Mahoto, and Alessandro Fiori. 2013b. GraphSum: Discovering correlations among multiple terms for graph-based summarization. Inf. Sci. 249 (2013), 96–109.

[9] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. In Proceedings of the 7th International Conference on World Wide Web 7. 107–117.

[10] Jon M. Kleinberg. 1999. Authoritative sources in a hyperlinked environment. J. ACM 46, 5 (Sept. 1999), 604–632.

[11] Hiroya Takamura and Manabu Okumura. 2009. Text summarization model based on the budgeted median problem. In Proceeding of the 18th ACM Conference on Information and Knowledge Management. 1589–1592.

[12] Dingding Wang, Shenghuo Zhu, Tao Li, and Yihong Gong. 2013. Comparative document summarization via discriminative sentence selection. ACM Trans. Knowl. Discov. Data 7, 1, Article 2 (March 2013), 18 pages.

[13] Josef Steinberger, Mijail Kabadjov, Ralf Steinberger, Hristo Tanev, Marco Turchi, and Vanni Zavarella. 2011. JRC's participation at TAC 2011: Guided and multilingual summarization tasks. In Proceedings of the 2011 Text Analysis Conference (TAC'11).

[14] Dan Gillick, Benoit Favre, and Dilek Hakkani-Tur. 2008. The ICSI summarization system at TAC 2008. In Proceedings of the Text Analysis Conference (TAC'08).

[15] Hui Lin and Jeff Bilmes. 2011. A class of submodular functions for document summarization. In Proceedings of the 49th Annual Meeting of the Association for

Computational Linguistics: Human Language Technologies - Volume 1 (HLT'11). Association for Computational Linguistics, Stroudsburg, PA, 510–520.

[16] John M. Conroy, Jade Goldstein, Judith D. Schlesinger, and Dianne P. OLeary. 2004. Left-brain/right-brain multi-document summarization. In DUC 2004 Conference Proceedings.

[17] John Conroy, Judith Schlesinger, Jeff Kubina, Peter Rankel, and Dianne OLeary. 2011. CLASSY 2011 at TAC: Guided and multi-lingual summaries and evaluation metrics. In TAC'11: Proceedings of the the 2011 Text Analysis Conference (TAC'11).

[18] Josef Steinberger, Mijail Kabadjov, Ralf Steinberger, Hristo Tanev, Marco Turchi, and Vanni Zavarella. 2011. JRC's participation at TAC 2011: Guided and multilingual summarization tasks. In Proceedings of the 2011 Text Analysis Conference (TAC'11).

[19] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. 2007. Frequent pattern mining: Current status and future directions. Data Min. Knowl. Discov. 15, 1 (2007), 55–86.

[20] Elena Baralis, Luca Cagliero, Saima Jabeen, and Alessandro Fiori. 2012. Multi-document summarization exploiting frequent itemsets. In Proceedings of the ACM Symposium on Applied Computing (SAC'12). 782–786.

[21] Elena Maria Baralis, Luca Cagliero, Alessandro Fiori, and Saima Jabeen. 2011. PatTexSum: A pattern-based text summarizer. In Proceedings of the Mining Complex Patterns Workshop. 18–29.

[22] Oskar Gross, Antoine Doucet, and Hannu Toivonen. 2014. Document summarization based on word associations. In Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR'14). ACM, New York, NY, 1023–1026.

[23] Elena Baralis, Luca Gagliero, Alessandro Fiori, and Paolo Garza, 2015. A Multilingual Summarizer Based on Frequent Weighted Itemsets

[24] A. Kogilavani and B. Balasubramanie. 2009. Ontology enhanced clustering based summarization of medical documents. Int. J. Recent Trends Engin. 1, 1 (2009).

[25] Elena Baralis, Luca Cagliero, Saima Jabeen, Alessandro Fiori, and Sajid Shah. 2013a. Multi-document summarization based on the Yago ontology. Expert Syst. Appl. 40, 17 (2013), 6976–6984.

[26] Garcia Lus Fernando Fortes, de Lima Jos Valdeni, Stanley Loh, and Jos Palazzo Moreira de Oliveira. 2006. Using ontological modeling in a context-aware summarization system to adapt text for mobile devices. In Active Conceptual Modeling of Learning (Lecture Notes in Computer Science), Peter P. Chen and LeahY. Wong (Eds.), Vol. 4512. Springer, 144–154.

[27] Lei Li, Dingding Wang, Chao Shen, and Tao Li. 2010. Ontology-enriched multi-document summarization in disaster management. In SIGIR, Fabio Crestani, Stphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy (Eds.). ACM, 819–820.

[28] Mohsen Pourvali and Mohammad Saniee Abadeh. 2012. Automated text summarization base on lexicales chain and graph using of WordNet and Wikipedia knowledge base. CoRR abs/1203.3586 (2012).

[29] John Atkinson and Ricardo Munoz. 2013. Rhetorics-based multi-document summarization. Expert Syst.Appl. 40, 11 (2013), 4346–4352.

[30] Leonhard Hennig, Winfried Umbrath, and Robert Wetzker. 2008. An ontology-based approach to text summarization. In Web Intelligence/IAT Workshops. IEEE, 291–294.

[31] Wordnet Lexical Database. 2012. Homepage. Available at http://wordnet.princeton.edu.

[32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

[33] D. E. Rumelhart, G. E. Hinton, R. J. Williams. Learning internal representations by back-propagating errors. Nature, 323:533.536, 1986.

[34] G.E. Hinton, J.L. McClelland, D.E. Rumelhart. Distributed representations. In: Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations, MIT Press, 1986.

[35] J. Elman. Finding Structure in Time. Cognitive Science, 14, 179-211, 1990.

[36] Mikolov, Tomas; et al. "Efficient Estimation of Word Representations in Vector Space". ArXiv:1301.3781.

[37] "A Neural Probabilistic Language Model". Studies in Fuzziness and Soft Computing: 137–186. doi:10.1007/3-540-33486-6_6.

[38] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS, 2013.

[39]P. Bojanowski, E. Grave*, A. Joulin, T. Mikolov, Enriching Word Vectors with Subword Information.*

[40] Jeffrey Pennington, Richard Socher, Christopher D. Manning, GloVe: Global Vectors for Word Representation.

[41] Mikolov T, Chen K, Corrado G, et al. Efficient Estimation of Word Representations in Vector Space[J]. Computer Science, 2013.

[42] Mikolov T, Sutskever I, Chen K, et al. Distributed Representations of Words and Phrases and their Compositionality[J]. 2013, 26:3111-3119.

[43] Camacho-Collados, Jose; Pilehvar, Mohammad Taher (2018). From Word to Sense Embeddings: A Survey on Vector Representations of Meaning.

*[44] Frederic Morin and Yoshua Bengio. Hierarchical proba*bilistic neural network language model. In Proceedings of the international workshop on artificial intelligence and statistics, pages 246–252, 2005.

[45] Document Understanding Conference. 2004. HTL/NAACL Workshop on Text Summarization. Retreived from http://duc.nist.gov/pubs.html#2004.

[46] George Giannakopoulos, Mahmoud El-Haj, Benoit Favre, Marina Litvak, Josef Steinberger, and Vasudeva Varma. 2011. TAC2011 MultiLing Pilot Overview. In Proceedings of the TAC 2011 Workshop. NIST, Gaithersburg, MD.

[47] Jeff Kubind, John Conroy, Judith Schlesinger. 2013. ACL 2013 MultiLing Pilot Overview.

[48] George Giannakopoulos. 2013. Multi-document multilingual summarization and evaluation tracks in acl2013 multiling workshop.

[49] Lin, Chin-Yew and E.H. Hovy 2003. Automatic Evaluation of Summaries Using N-gram Co-occurrence Statistics. In Proceedings of 2003 Language Technology Conference (HLT-NAACL 2003), Edmonton, Canada, May 27 - June 1, 2003.

[50] Eugene Krapivin, Mark Last, and Marina Litvak. 2014. JRouge - Java ROUGE Implementation. Retrievedfrom https://bitbucket.org/nocgod/jrouge/wiki/Home/

[51] Dan Gillick, Benoit Favre, and Dilek Hakkani-Tur. 2008. The ICSI summarization system at TAC 2008. In Proceedings of the Text Analysis Conference (TAC'08).

[52] N. Rotem. 2011. Open Text Summarizer (OTS). Retrieved from http://libots.sourceforge.net/.

[53] TexLexAn. 2011. TexLexAn: An Open-Source Text Summarizer. Retrieved from http://texlexan.sourceforge.net/.

Acknowledgements

I am grateful to my parents for not only supporting my life, but also giving me confidence and courage to face every difficult I have encountered. Thanks to my aunts, who generously helped me when my family was in trouble, so that I could concentrate on the research. Thanks to my girlfriend Yang, she gave me enough trust and comforted me when I fell into the trough. Thanks to my friends Pan and Huang, they always offer me help and provided me with a lot of convenience during my research.