USING GENETIC ALGORITHMS WITH LEXICAL CHAINS FOR AUTOMATIC TEXT SUMMARIZATION

Mine Berker [1] and Tunga Güngör [1,2]
[1] *Boğaziçi University, Computer Engineering Dept., Bebek 34342, Istanbul, Turkey*

[2] *Universitat Politècnica de Catalunya, TALP Research Center, Barcelona, Spain*

Keywords: Automatic text summarization, Lexical chains, Genetic algorithms, Lexical cohesion.

Abstract: Automatic text summarization takes an input text and extracts the most important content in the text. Determining the importance depends on several factors. In this paper, we combine two different approaches that have been used in text summarization. The first one is using genetic algorithms to learn the patterns in the documents that lead to the summaries. The other one is using lexical chains as a representation of the lexical cohesion that exists in the text. We propose a novel approach that incorporates lexical chains into the model as a feature and learns the feature weights by genetic algorithms. The experiments showed that combining different types of features and also including lexical chains outperform the classical approaches.

1 INTRODUCTION

With the rapid increase in the amount of online text information, it became more important to have tools that help users distinguish the important content. Automatic text summarization (ATS) is a process that addresses this need, where a computer produces a summary of a text that contains the most important information. Text summarization studies mostly use sentence scoring methods (Mani, 2001). Following the work of Edmundson (1969), several text features were introduced in text summarization studies. Paice and Jones (1993) used stylistic clues and constructs. Kupiec, Pedersen and Chen (1995) checked the presence of proper names. Statistical measures of term prominence derived from word frequencies were used by Brandow, Mitze and Rau (1994).

Generally, a number of features drawn from different levels of analysis may contribute to the salience of a sentence. A summarization system must have an automatic way of finding out how to combine different text features. A solution is to use machine learning methods. In some studies, genetic algorithms (GA) were employed to learn the importance of different features for summarization (Mani and Bloedorn, 1998; Kiani and Akbarzadeh, 2006; Dehkordi, Khosravi and Kumarci, 2009; Fattah and Ren, 2009).

Barzilay and Elhadad (1997) proposed *lexical chains* to identify the cohesion in the text. A lexical chain can be defined as a sequence of words that are related to each other (Barzilay, 1997). The semantic relations between words were found using WordNet. Once the chains are built, the concepts represented by strong chains are used to select the sentences.

After this work, many researchers followed the lexical chain approach. Silber and McCoy (2000) proposed an algorithm to compute lexical chains that is linear in space and time. Brunn, Chali and Pinchak (2001) used the degree of connectiveness among the chosen text portions to identify the most important parts of the text which are topically more salient. Li, Sun, Kit and Webster (2007) proposed a model for a query-focused summarizer. Fuentes and Rodriguez (2002) proposed a system that combined lexical chains, coreference chains, and NamedEntity chains.

In this work, we combine the two approaches of sentence scoring and lexical chain computing to generate summaries by using genetic algorithms. In addition to shallow, syntactic text features, we use lexical chains as a feature to score sentences in a deeper and semantic manner. One novelty of this study is incorporating the lexical chain concept into a sentence scoring system as a new type of feature. These chains are expected to identify the cohesion that exists in the text and assign higher scores to sentences that are semantically related to each other.

Berker M. and Güngör T..
USING GENETIC ALGORITHMS WITH LEXICAL CHAINS FOR AUTOMATIC TEXT SUMMARIZATION.
DOI: 10.5220/0003882405950600
In *Proceedings of the 4th International Conference on Agents and Artificial Intelligence* (SSML-2012), pages 595-600
ISBN: 978-989-8425-95-9

2 PROPOSED APPROACH

In this work, we use a sentence extraction approach that makes use of different properties of the text to weight the sentences. Each sentence is given a score calculated using the scores of different features. The system first goes through a training phase, where the weights of the text features are learned using machine learning methods. Then, in the testing phase, the sentence score is calculated for each sentence in a newly-introduced document using the feature scores for that sentence and their respective score weights. Then the sentences are sorted in descending order of their scores and the highest scored sentences are selected to form the summary.

2.1 Text Features

In this study, we represent each sentence as a feature vector formed of 12 features extracted from the text. For each sentence in a document, a sentence score is calculated using the feature scores of these text features. Each feature score is normalized to the range [0,1]. We group the text features used in this study into three classes according to their level of text analysis. Table 1 shows the features and the classes. The features are explained below.

Table 1: Features used by the summarizer.

Feature class	*Feature*
Location features	Sentence location
	Sentence relative length
Thematic features	Average TF
	Average TF-IDF
	Similarity to title
	Cue words
	Named entities
	Numerical data
	Sentence centrality
Cohesion features	Synonym links
	Co-occurrence links
	Lexical chains

F1-Sentence location: Usually the initial sentences in a document are the most important ones. We score the first sentence of the document with 1.0, the second sentence with 0.8, etc., and the sentences past the fifth sentence get a score of 0.

F2–Sentence relative length: We assume that longer sentences contain more information. For a sentence *s* in a document *d*, the feature score is calculated as follows, where *ns* denotes the number of sentences in the document:

$$F2(d,s) = \frac{sentencelength(s)}{\max_{i=1,\dots,ns} sentencelength(s_i)} \quad (1)$$

F3–Average TF: The term frequency (TF) metric is based on two assumptions: i) The importance of a term for a document is directly proportional to its number of occurrences in the document, ii) The length of the document does not affect the importance of the terms. The TF score for a term *t* in a document *d* is calculated as follows, where *nt* denotes the number of terms in the document:

$$F3(d,t) = \frac{termfrequency(d,t)}{\max_{i=1,\dots,nt} termfrequency(d,t_i)} \quad (2)$$

For a sentence *s* in the document *d*, the feature score is the average of the TF scores of all the terms in *s*.

F4–Average TF-IDF: The term frequency-inverse document frequency (TF-IDF) metric makes one more assumption: iii) A term in a document that occurs rarely in the document collection is more important than a term that occurs in most of the documents. For a term *t* in a document *d* given a corpus *c*, the TF-IDF score is calculated as follows, where *nd* is the total number of documents in the corpus and the document frequency *df* denotes the number of documents in which the term occurs:

$$F4(c,d,t) = F3(d,t) * \log\left(\frac{nd}{df(c,t)}\right) \quad (3)$$

For a sentence *s* in document *d*, the feature score is the average of TF-IDF scores of all the terms in *s*.

F5–Similarity to title: This feature considers the vocabulary overlap between a sentence and the document title. It is calculated as follows:

$$F5(d,s) = \frac{|s\text{-}terms \cap t\text{-}terms|}{|s\text{-}terms \cup t\text{-}terms|} \quad (4)$$

where *s-terms* and *t-terms* are the set of terms that occur, respectively, in sentence *s* and in the title.

F6–Cue words: This and the next two features assume that sentences that include some types of special items contain salient information about the document. Thus the scores of these sentences are increased depending on the number of such entities. This feature counts the number of cue words (such as *especially*, *certainly*) in a sentence:

$$F6(d,s) = \frac{cuewords(s)}{sentencelength(s)} \quad (5)$$

F7–Named entities: This feature counts the number of named entities (such as proper nouns) in a sentence. In this work, named entities are recognized using the University of Illinois Named Entity Tagger (http://cogcomp.cs.illinois.edu/page/software/). The

feature score is calculated as follows:

$$F7(d,s) = \frac{namedentities(s)}{sentencelength(s)} \quad (6)$$

F8–Numerical data: Terms that are written in numerical form sometimes convey key information about a document. We test the usefullness of such terms using this feature. This feature counts the number of numerical terms in a sentence:

$$F8(d,s) = \frac{numericalterms(s)}{sentencelength(s)} \quad (7)$$

F9–Sentence centrality: This feature measures the vocabulary overlap between a sentence and the other sentences in the document. This is an indication of the importance of a sentence for a document. For a sentence *s* in the document *d*, the feature score is calculated as follows:

$$F9(d,s) = \frac{c\text{-}terms}{nt} \quad (8)$$

where *c-terms* is the number of common terms that occur both in *s* and in a sentence *d* other than s, and *nt* is the number of terms in the document.

F10–Synonym links: This feature is another form of sentence centrality and attempts to measure the centrality of a sentence using the number of common synonymous words in the sentences. We consider nouns only and we extract the nouns in sentences using the LingPipe part-of-speech (PoS) tagger (http://alias-i.com/lingpipe/). The synonymy relation between two nouns is determined by looking whether they have a synset in common in WordNet. The feature score is calculated as follows:

$$F10(d,s) = \frac{s\text{-}links}{ns} \quad (9)$$

where *s-links* is the number of synonym links between *s* and other sentences in the document, and *ns* is the number of sentences in the document.

F11–Co-occurrence links: The co-occurrence of two terms signals semantic proximity between these terms. A sentence whose terms have several co-occurrences with terms in other sentences can be deemed as important. To compute this feature, all the bigrams in the document are considered and their frequencies are calculated. If a bigram in a document has a frequency greater than one, then this bigram is assumed to be a collocation. Then, terms of the given sentence *s* are compared to the terms in other sentences. This comparison checks if a term in *s* forms a collocation with a term in another sentence. If so, there is a co-occurrence link between this sentence and s. The feature is calculated as follows:

$$F11(d,s) = \frac{c\text{-}links}{ns} \quad (10)$$

where *c-links* is the number of co-occurrence links of *s* and *ns* is the number of sentences in document.

2.2 Lexical Chains

A novel aspect of the proposed approach is using the lexical chain concept as a sentence feature in the system. We first compute the lexical chains for the document, give a score to each chain, and select the strongest chains. Then, we score the sentences according to their inclusion of strong chain words.

The lexical relations between words are extracted using WordNet. When lexical chains are computed, each word must belong to exactly one chain. There are two challenges here. First, there may be more than one sense for a word (ambiguous word) and the correct sense must be identified. Second, a word may be related to words in different chains. The aim is to find the best way of grouping the words that will result in the longest and strongest chains.

In this work, we consider only nouns as the candidate words and first determine the nouns using the LingPipe PoS tagger. Then, we use a novel method to disambiguate the candidate words. The nouns are sorted in ascending order of their number of senses. Hence, the least ambiguous words are treated first. For each word, we find an appropriate chain that the word can be added according to a relatedness criterion between the chain members and the word. This criterion compares each member of the chain to the candidate word to find out if

- the sense of the lexical chain word belongs to the same synset as the sense of the candidate word
- the synset of the lexical chain word has a hyponym/hypernym relation with the synset of the candidate word
- the synset of the lexical chain word shares the same parent with the synset of the candidate word in a hyponym/ hypernym relation.

The search process continues for every sense of the candidate word until an appropriate chain is found. If such a chain is found, the current sense of the candidate word is set to be the disambiguated sense and the word is added to the chain. Otherwise, a new chain is formed for every sense of the word. When a new candidate word is compared to these chains, it will be possible to find a relation between the new word and any of the senses of the previous word. The problem is that there may be more than one chain for the same word. This problem is solved by removing the word from the other chains as soon as a second word is related with a sense of the word in one of the chains. This is illustrated in Figure 1 where the word *flower* is related to the second sense

of the word *plant* and thus the other two senses of *plant* are deleted.

Step 1: No chains

Step 2 (processing the word *plant*):
Chain1 → *plant*: buildings for carrying on industrial labor
Chain2 → *plant*: a living organism lacking the power of locomotion
Chain3 → *plant*: something planted secretly for discovery by another

Step 3 (processing the word *flower*):
Chain2 → *plant*: a living organism lacking the power of locomotion; *flower*: a plant cultivated for its blooms or blossoms

Figure 1: Lexical chain management example.

As the lexical chains are formed, each chain is given a score. The score of a chain depends on both its length and its homogeneity, and is a product of these two measures. The length is the number of occurrences of the members of the chain. Its homogeneity is inversely proportional to diversity:

$$homogeneity = 1 - \frac{\#DistinctOccurrences}{length} \quad (11)$$

After the chain scores are obtained, strong chains are determined and selected. In this work, a strong lexical chain must satisfy the following two criteria:

$$score(chain) > average(chainScores) + 2 * standardDeviation(chainScores) \quad (12)$$
$$wordcount(chain) > 1$$

Finally, after the chains are constructed and scored for a document *d*, the lexical chain score of a sentence *s* is calculated as follows, where *frequency* denotes the term frequency of a term and *ns* is the number of sentences in the document:

$$F12(d,s) = \frac{\sum_i \begin{matrix} frequency(i) | i \in s \text{ and} \\ i \text{ is a word in a strong chain} \end{matrix}}{\max_{i=1,\ldots,ns} F12(d,s_i)} \quad (13)$$

2.3 Learning Feature Weights and Summary Generation

The weights of the features are learned using a genetic algorithm. The score of a sentence is a weighted sum of the feature scores for that sentence:

$$Score(s) = \sum_{i=1}^{12} w_i * F_i \quad (14)$$

where F_i denotes the score of the *i*th feature and w_i its weight. In this work, w_i's can range from 0 to 15.

During training, for each training document, first the feature scores are computed for the sentences in the document. At each iteration of the genetic algorithm, feature weights are initialized randomly. Then the sentence scores are calculated and a summary is generated and evaluated for each document. The process repeats and the average of the precisions (Eqn. 15) gives the performance of that iteration. The best of all the iterations is selected by the genetic algorithm.

Each individual of the population is a vector of feature weights. The vector has a length of 48 bits since there are 12 features and each feature value (between 0 and 15) can be represented by four bits. There are 1000 individuals in the population. At each generation, the mating operator selects the fittest 50 individuals and carries them directly to the next generation. The other 950 individuals are produced by a selected pair of parents. Each individual is selected to be a parent according to a probability rate calculated from its fitness value. A child is produced by merging the first *n* bits of the vector of one parent and the last 48-*n* bits of the vector of the other parent, where *n* is random for each reproduction. After a child is produced, it is put through mutation with a predetermined probability. If it goes through mutation, one of its bits is set to a random value. Finally, after mutation, the produced child is added to population for the next generation.

The genetic algorithm is run for 100 generations to obtain a steady combination of weights. The best individual that is produced after these iterations is selected to be the set of feature weights. During testing, for each document, the sentence scores are calculated using the learned feature weights.

3 EXPERIMENTS AND RESULTS

The proposed approach was tested using the CAST (Computer-Aided Summarization Tool) corpus (http://www.clg.wlv.ac.uk/projects/CAST). We used 100 documents, of which 80 were used for training and 20 for testing. We performed a five-fold cross-validation. The results show the average of these five runs. We used precision as the performance measure defined as follows, where *T* is the manual summary and *S* is the machine generated summary:

$$P = |S \cap T| / |S| \quad (15)$$

3.1 Performance of Single Features

Before analyzing the performance of the proposed approach, we tested the effect of each feature on the

summarization task separately. For this purpose, we used the score function (Eqn. 14) with one feature weight being equal to 1 and the rest to 0. Table 2 shows the success rates.

We can see that using only sentence location gives one of the best results. The leading sentences in a document usually give a general understanding about the topic. Sentence centrality also yields good results. This feature favors sentences that mention many of the topics that appear throughout the text. Moreover, named entities feature shows high performance. This is a sign that sentences that give information about specific people or organizations are likely to be selected for the summary.

Table 2: Success rates of individual features.

Feature	*Average precision*
Sentence location	0.43
Sentence relative length	0.42
Average TF	0.32
Average TF-IDF	0.30
Similarity to title	0.39
Cue words	0.36
Named entities	0.43
Numerical data	0.29
Sentence centrality	0.43
Synonym links	0.42
Co-occurrence links	0.41
Lexical chains	0.40

The lexical chain feature is also among the high performance features. This can be regarded as a quite high success rate since it corresponds to using solely the lexical chain concept without any other clue important for summarization. This shows that lexical chains can be used as an intermediate representation of lexical cohesion that exists throughout the text to determine the importance of sentences in that text. This feature makes better predictions than many of the other text features.

3.2 Performance of the Proposed Model

In the next experiment, we measured the performance of the proposed approach which incorporates the lexical chain concept as a new type of feature and integrates this with genetic algorithm learning. Table 3 shows the weight of each feature calculated during training and the average precision of the documents summarized during testing.

When all the features are used in the algorithm, the performance of the system increased to 46%, which outperforms the success rate obtained by the best individual feature. Moreover, the system succeeds to distinguish the features whose effects on the summarization task are high, and rewards them by giving the highest weights to these features. Sentence location and sentence centrality are two such features that obtained the best individual success rates and the highest feature weights. The weight of the lexical chain feature does not seem to be as high as the weights of features like sentence location and sentence centrality. However, it supports and reinforces the results of the sentence centrality and co-occurrence link features as it analyzes the cohesion in the text from a different perspective. While the sentence centrality and co-occurrence link features analyze the cohesion at the sentence level, lexical chain feature goes deeper and analyzes the relations among the nouns that are spread throughout the text. Also, since the individual performance of the lexical chain feature is among the highest rates, it brings a valuable understanding about the salience of the sentences in the text.

Table 3: Feature weights and success rate of the model.

Feature	*Feature weight*
Sentence location	14
Sentence relative length	3
Average TF	1
Average TF-IDF	5
Similarity to title	4
Cue words	12
Named entities	11
Numerical data	2
Sentence centrality	13
Synonym links	10
Co-occurrence links	12
Lexical chains	5
Average precision	*0.46*

The results of the experiment show that using combination of different features increases the performance of summarization and genetic algorithms are successful to learn a set of weights for those features that would result in a better output.

3.3 Alternative Models

In order to compare the results of the proposed approach with similar methods, we built two other models by changing some of the parameters of the system. In the first model, we used a smaller set of features. Instead of using a single feature or combining all the features, we considered the first three features that scored best on their own, together with the lexical chain feature, and analyzed the

performance using only these four features.

The second column in Table 4 shows the weights of the features learned during training and the success rate of the model on the test corpus. We can see that the result is almost the same as the performance of the system when all the features are considered.

Table 4: Feature weights and success rates for the alternative models.

Feature	*Feature weight (top features)*	*Feature weight (modified criterion)*
Sentence location	14	15
Named entities	3	3
Sentence centrality	14	9
Lexical chains	10	9
Average precision	*0.45*	*0.46*

Another alternative model is decreasing the threshold for determining strong lexical chains. Assuming that an increase in the lexical chain scores might affect the performance of the system, we changed Eqn. 12 in such a way that chains whose scores are more than one standard deviation from the average are accepted as strong chains. The four features in the previous experiment were used in this model also. The results are shown in the last column of Table 4. The success is a bit higher, but the difference is not statistically significant. In order to observe the effect of the threshold more clearly, the experiments should be repeated with other thresholds on corpora of different sizes.

4 CONCLUSIONS

In this work, we combined two approaches used in automatic text summarization: lexical chains and genetic algorithms. Different from previous works, this paper combines information from different levels of text analysis. The lexical chain concept is included as a feature in the proposed model. We also make use of machine learning to determine the coefficients of the feature combinations. The results showed that the combination of the features yields better success rates than any individual feature. Also, incorporating lexical chains into the model as a feature increases the success of the overall model.

ACKNOWLEDGEMENTS

This work was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) BİDEB under the programme 2219.

REFERENCES

Barzilay, R., 1997. Lexical Chains for Summarization. M.Sc. Thesis, *Ben-Gurion University of the Negev, Department of Mathematics and Computer Science*.

Barzilay, R., Elhadad, M., 1997. Using Lexical Chains for Text Summarization. *In ACL/EACL Workshop on Intelligent Scalable Text Summarization*, pp. 10-17.

Brandow, R., Mitze, K., Rau, L., 1994. Automatic Condensation of Electronic Publications by Sentence Selection. *Information Processing and Management*, 31(5), 675-685.

Brunn, M., Chali, Y., Pinchak, C. J., 2001. Text Summarization Using Lexical Chains. *In Document Understanding Conference*, pp. 135-140.

Dehkordi, P. K., Khosravi, H., Kumarci, F., 2009. Text Summarization Based on Genetic Programming. *International Journal of Computing and ICT Research*, 3(1), 57-64.

Edmundson, H. P., 1969. New Methods in Automatic Abstracting. *Journal of the Association for Computing Machinery*, 16(2), 264-285.

Fattah, M. A., Ren, F., 2009. GA, MR, FFNN, PNN and GMM Based Models for Automatic Text Summarization. *Computer Science and Language*, 23, 126-144.

Fuentes, M., Rodriguez, H., 2002. Using Cohesive Properties of Text for Automatic Summarization. *In Workshop on Processing and Information Retrieval*.

Illinois Named Entity Tagger. http://cogcomp.cs.illinois.edu/page/software/

Kiani, A., Akbarzadeh, M. R., 2006. Automatic Text Summarization Using Hybrid Fuzzy GA-GP. *In IEEE International Conference on Fuzzy Systems*, pp. 5465-5471.

Kupiec, J., Pedersen, J., Chen, F., 1995. A Trainable Document Summarizer. *In ACM-SIGIR*.

Li, J., Sun, L., Kit, C., Webster, J., 2007. A Query-Focused Multi-Document Summarizer Based on Lexical Chains. *In Document Understanding Conference*.

LingPipe. http://alias-i.com/lingpipe/

Mani, I., 2001. Automatic Summarization. John Benjamins, Amsterdam.

Mani, I., Bloedorn, E., 1998. Machine Learning of Generic and User-Focused Summarization. *In 15th National Conference on Artificial Intelligence*, pp. 821-826.

Paice, C., Jones, P., 1993. The Identification of Important Concepts in Highly Structured Technical Papers. *In ACM-SIGIR*.

Silber, H. G., McCoy, K. F., 2000. Efficient Text Summarization Using Lexical Chains. *In 5th International Conference on Intelligent User Interfaces*, pp. 252-255.