LING 573 Deliverable #3 Report

Haobo Gu
University of Washington
haobogu@uw.edu

Yuanhe Tian
University of Washington
yhtian@uw.edu

Weifeng Jin
University of Washington
wjin@uw.edu

Haotian Zhu
University of Washington
haz060@uw.edu

Abstract

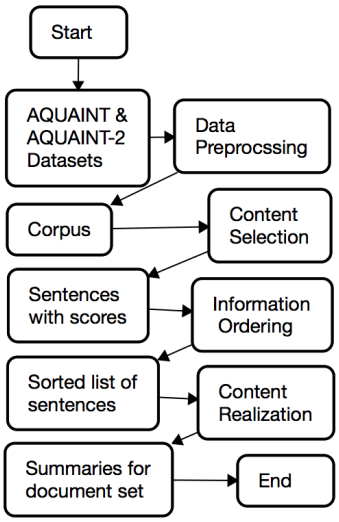
This document discusses the automatic summarization system we built based on various algorithms. Our system takes input as a set of document sets and generates the summarization for each document set. The system has four major parts: data preprocessing, content selection, information ordering and content realization.

1 Introduction

Automatic summarization is a shared natural language processing task, whose aim is to shorten the input text without losing too much information in the context and create a abbreviated, informative and consistent summary. Generally speaking, there are two approaches for this problem, extraction-based summarization and abstraction-based summarization. The extraction-based method keeps the original sentences while the abstraction-based method generates new sentences for summarization. In this system, we built a system upon the extraction-based method, which extracts sentences from the input document set without modifying the content of the sentences. We used both unsupervised and supervised models for our summarization task and thus prior training on the corpus is required. We used Python3 to implement our system.

2 System Overview

We employed the classic system structure for the automatic summarization task, which consists three major parts: content selection, information ordering and content realization. Since the AQUAINT and AQUAINT-2 Corpora have inconsistent data formatting, we add a data preprocessing part to prepare the data. The system flowchart is:



The data preprocessing part takes one document specification (one `.xml` file) and the two corpora as input, and generates a corpus, which includes the list of document sets and a token dictionary to record the frequency of each word occurrence in the input dataset. For each document set in the list, it includes the document set ID, a token dictionary, a topic ID, a collection of summary texts generated by human beings and a list of documents. For each document in the list, it records the document ID, document content, topic ID, a token dictionary, the time of generation and a list of sentences. For each sentence, we record the information of the ID of the document it comes from, the content of the sentence, its position in the document, its score (initialized as 0), sentence length, the time of generation (same as the document) and a token dictionary.

The content selection part takes a document set as an input, and generates a list of sentences with their scores, computed using a weighted average of several different models. For each document set in the corpus, the part would generate a list of sentences individually.

The information ordering part takes a list of sentences, and returns a sorted list of sentences, using

standards of chronology and other measures.

The content realization part takes the sorted list of sentences as input, and removes the unnecessary sentences based on the sentence length and low-rank sentences to generate an output summary for the corresponding document set. The output is a text file and for each document set in the corpus, the system would generate an individual summary text file for it.

3 Approach

As mentioned in the Introduction section, our system employs an extraction-based approach. The detailed approach implementation for the three major parts will be discussed below and this approach is targeted to a single document set input.

3.1 Content Selection

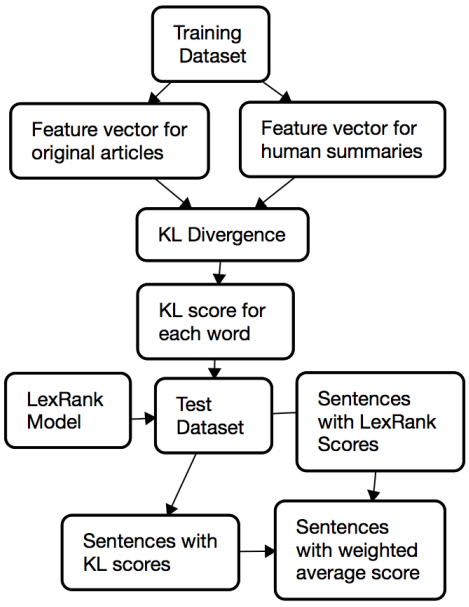
Initially, we considered using the method of Log Likelihood Ratio (LLR) and Hidden Markov Model (HMM) to predict if the sentence should be included in the summary based on John Conroy's paper (2004). The paper uses LLR as the HMM feature, and calculates the in-topic frequency and out-topic frequency for each word. In order to make a meaningful training, the training corpus' topics should be at least partially consistent with the test corpus. Unfortunately, the dataset we are using has different distribution of topics in the test and training corpus.

Another difficulty in employing the method is that the HMM requires us to label the sentences in the training dataset as "in the summary" and "not in the summary", which requires the training dataset's human summaries to be based on extraction rather than abstraction. Since the summaries we have are all based on abstraction, the approach is very difficult to implement (the only way to achieve that is we label these sentences by hand as the author mentioned in the paper) and thus we abandoned this method.

The first step of our content selection part is to calculate the scores generated by the LexRank model (Erkan and Radev, 2004) and this part can be done without training. Suppose we have m sentences and n tokens in the document set. According to the occurrence of the token, we can build a feature vector of length n for each sentence and generate a $m \times n$ matrix for the document set, in which each row represents a sentence's feature vector. In addition to that, we compute the cosine distance (based on the LexRank model) for each sentence pair and generate a new $m \times m$ matrix, in which the ij^{th} entry represents the cosine similarity between the i^{th} sentence and the j^{th} sentence. Next, we have to convert the matrix to be a transition probability matrix, in which each row of the matrix would be normalized, and mark this matrix as M. The results (after normalization) is as follows:

$$\begin{bmatrix} 0.61411 & 0.62993 & 0.41656 & \dots & 0.34192 \end{bmatrix}$$

After inspecting our results, the LexRank model turns to assign higher scores on long sentences. Thus, we decided to combined the unsupervised LexRank model and the supervised model based on Kullback-Leibler(KL) divergence (Hong and Nenkova, 2014) to calculate the score for each sentence in the document set, using a weighted average of both models. The flowchart of the whole process is:



The KL model needs a training stage before test. We build two 1-D feature vectors for words in the article and word in summaries. Suppose the feature vector representing original articles as M, summaries as N. The i^{th} entry in the vector M is $Pr_A(w_i)$, while the j^{th} entry in the vector N is $Pr_S(w_j)$.*w* represents the word to be calculated, $Pr_S(w)$ represents the probability of *w* appearing in the summaries and $Pr_A(w)$ represents the probability of *w* appearing in the articles. The probabilities would be logged later to prevent underflow of the system and both vectors would

be calculated for the training dataset. The feature vector (before being logged) would be:

$$\begin{bmatrix} 6.30\text{e}{-5} & 7.00\text{e}{-6} & 7.00\text{e}{-6} & \dots & 2.45\text{e}{-4} \end{bmatrix}$$

The next step is to calculate the KL score for every word appeared in both articles and summaries in the training data. According to Hong's (2014) research , we calculate both $KL(S||A)$ and$KL(A||S)$ based on the formulas below:

$$KL(S||A)(w) = Pr_S(w) \ln \frac{Pr_S(w)}{Pr_A(w)} \quad (1)$$

$$KL(A||S)(w) = Pr_A(w) \ln \frac{Pr_A(w)}{Pr_S(w)} \quad (2)$$

In addition, the value of both $KL(S||A)(w)$ and $KL(A||S)(w)$ will be zero, if either $Pr_S(w)$ is zero or $Pr_A(w)$ is zero.

Words with higher $KL(S||A)$ value are more likely to appear in the summary, because Pr_S(w) is higher than Pr_A(w). And words with higher $KL(A||S)$ value are less likely to appear in the summary, because Pr_A(w) is higher than Pr_S(w). Therefore, we define the KL score of a word to be the difference between $KL(S||A)$ and $KL(A||S)$, which is shown as following:

$$KL(w) = \frac{(KL(S||A)(w) - KL(A||S)(w))}{2} \quad (3)$$

We store the KL score of every word appearing in the training articles and summaries in a dictionary. Hence, for every word in a given sentence s from test articles, we can get the word's KL score from the dictionary. Then, we will calculate the KL score of that sentence by averaging the KL score of all words.

$$KL(s) = \frac{\sum_{w_i \in s} KL(w_i)}{length(s)} \quad (4)$$

Now, we find a way to calculate the KL score of a sentence, and the bigger the KL score is, the more likely these words in the sentence will appear in the summary, so that the more important the sentence will be. Therefore, for every sentence in the input document set, we can calculate their KL score and normalize them by the highest KL score of the sentence. The normalized KL score will be the importance score for that sentence.

Finally, we generate a 1-D vector P, of length m to record the score of each sentence, in which the i^{th} entry represents the score for the i^{th} sentence. The sentence score is determined by a weighted average of both the LexRank score and KL score. By several trials, we settled down using the weights as 0.25 LexRank Score : 0.75 KL score. In order to avoid short sentences, we remove sentences with length less than 8 words from our output list.

3.2 Information Ordering

We implement the method of experts, with a slight modification, to provide an ordered list of selected sentences with more readability and meaningfulness. Based on the work by Bollegala et al., we choose the chronological expert as the ordering strategy(Bollegala, Okazaki and Ishizuka, 2012). This sentence ordering procedure proposed in this paper contains five experts in total, but we only adopt the chronological expert for now, and we will consider implementing the rest in the next deliverable.

Define an expert e by a pairwise preference function as follows:

$$PREF_e(u, v, Q) \in [0, 1] \quad (5)$$

where u and v are two sentences to order, and Q is the set of sentences that has been ordered so far by some ordering algorithm. Note that the cardinality of Q is 0 when all sentences are unordered. Also, it is worth noting that preference of u to v is not necessarily the same as the preference of v to u.

Chronological expert orders the sentences in chronological order of publication date, which is defined as follows:

$$PREF_{chro}(u, v, Q) = \begin{cases} 1 & T(u) < T(v) \\ 1 & [D(u) = D(v)] \wedge [N(u) < N(v)] \\ 0.5 & [T(u) = T(v)] \wedge [D(u) \neq D(v)] \\ 0 & otherwise \end{cases} \quad (6)$$

where $T(u)$ is the publication data of the sentence u, $D(u)$ is the unique identification code of the document to which the sentence u belongs and $N(u)$ is the in-text index of the sentence u in its document.

By computing chronological preference of every pair of sentences, we then use the sentence ordering algorithm presented in the same paper to produce the final order of sentences. The algorithm is given as follows:

Basically, this ordering algorithm regards the preference function as a directed weighted graph where initially the set of vertices V is equal to the

Algorithm 1 Sentence Ordering Algorithm.

Input: A set of X of unordered sentences and a preference function $PREF_{chro}(u, v, Q)$

Output: Ranking score $\hat{\rho}(t)$ of each sentence $t \in X$

procedure ORDERING
  $V \leftarrow X$
  $Q \leftarrow \emptyset$
  for each $v \in V$ **do**
    $\pi(v) = \sum_{u \in V} PREF_{chro}(v, u, Q) - \sum_{u \in V} PREF_{chro}(u, v, Q)$
  while $V \neq Q$ **do**
    $t = argmax_{u \in V} \pi(u)$
    $\hat{\rho}(t) = |V|$
    $V = V - \{t\}$
    $Q = Q + \{t\}$
    for each $v \in V$ **do**
      $\pi(v) = \pi(v) + PREF_{chro}(t, v, Q) - PREF_{chro}(v, t, Q)$
  return $\hat{\rho}$

set of sentences X, and each edge $u \rightarrow v$ has the weight equal to the preference. Then each vertex $v \in V$ is assigned a potential value $\pi(v)$, which is the sum of outgoing edges minus sum of ingoing edges. This ordering algorithm is a greedy algorithm that orders a certain node t with maximal potential and assigns a rank equal to the cardinality of the set V, which then is removed from the graph; also the potential value π of the remaining vertices are updated accordingly. This algorithm continues until the graph is empty, i.e., all sentences are ordered.

This paper also proposed an algorithm for learning weights for each experts. We will implement this alongside with all other experts in the next deliverable.

3.3 Content Realization

In our content realization module, we implemented an Integer Linear Programming (ILP) method which was proposed by Gillick and Farve Gillick. We use a new objective function which maximizes the weighted summary of all presence bigrams:

$$\sum_i w_i c_i \tag{7}$$

where c_i is the presence of bigram i and w_i is the weight. Each bigram has an associated weight, which is the number of sentence it appears in. Intuitively, the more bigrams appear in the summary, the better the summary is.

Besides presences of bigrams, we also have presences of sentences as variables, which are what we actually want by solving the ILP problem. We have three constraints in our ILP problem. The first constraint is the length constraint:

$$\sum_j l_j s_j \leq L \tag{8}$$

where s_j is the presence of the sentence j, l_j is the length of sentence j, and L is the maximum summary length.

The second and third constraints are about relationships between s_j and c_i. Here, we use Occ_{ij} to indicate the occurrence of bigram i in sentence j. Obviously, Occ_{ij} is a constant matrix which should be calculated before solving the ILP problem. With Occ_{ij}, we can formulate the following constraints:

$$s_j Occ_{ij} \leq c_i \ \ \forall i, j \tag{9}$$

$$\sum_j s_j Occ_{ij} \geq c_i \ \ \forall i \tag{10}$$

Constraint (9) means that if a sentence j is in the summary, then all the bigrams in this sentence would occur in the summary. If sentence j is not in the summary, or bigram i is not in this sentence, this constraint can be always satisfied. Constraint (10) means that if bigram i is in the summary, then at least one sentence in the summary has bigram i. Similarly, if bigram i is not in the summary, constraint (10) can be always satisfied as well.

Hence, the final ILP problem we established for content realization is:

$$\begin{aligned} \text{Maximize} &: \sum_i w_i c_i \\ \text{Subject to} &: \sum_j l_j s_j \leq L \\ & s_j Occ_{ij} \leq c_i \ \ \forall i, j \\ & \sum_j s_j Occ_{ij} \geq c_i \ \ \forall i \\ & c_i \in \{0, 1\} \ \ \forall i \\ & s_j \in \{0, 1\} \ \ \forall j \end{aligned} \tag{11}$$

By maximizing the target function, a set of sentences is generated. Finally, those generated sentences are organized to the summary using the order we got from information ordering module.

4 Results

We trained our model on the training dataset and test on the devtest dataset. After that, we used the ROUGE script to compare the generated summaries and the human summaries (as gold standard), and we get the following results.

Table 1: ROUGE Results for devtest

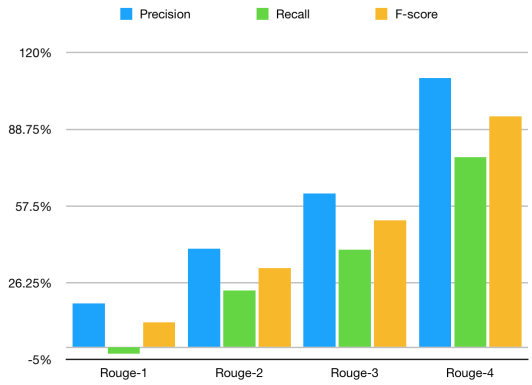
ROUGE	Recall	Precision	F-score
ROUGE-1	0.23619	0.24349	0.24384
ROUGE-2	0.06092	0.06541	0.06292
ROUGE-3	0.01981	0.02088	0.02028
ROUGE-4	0.00746	0.00768	0.00755

Also, we tried several different prune methods in the Content Realization module, including removing clauses starting with an adverb, removing appositions, removing contents in parenthesis. Although these methods do not improve the system performance significantly, we want to include the results below for future reference.

Table 2: ROUGE for different prune methods on `devtest`

Recall	advcl	apposition	parenthesis
ROUGE-1	0.24018	0.23915	0.23218
ROUGE-2	0.06411	0.06158	0.06145
ROUGE-3	0.01943	0.01881	0.01874
ROUGE-4	0.00697	0.00588	0.00607

Our improved system performs far better than our original system, except the Rouge-1 Recall score. The improvement rate is shown below:



5 Discussion

Based on the improvement chart above, our system improves significantly in Rouge-4 and Rouge-3 standards. However, it does not improve that much in Rouge-1 standard.

Although the improvements of our current system are significant, there are some system flaws that we currently cannot solve. The first problem is that for each time of summarization generation, our system would generate slightly different summaries for a small subset of the test dataset, resulting in a small fluctuation of the ROUGE scores. The problem resulted from the ILP algorithm we used in the Content Realization module. On some occasions, there might be more than 1 solution for the optimization problem and the system would randomly select one optimal sentence for construction. We expect to solve the randomness problem when we improve the Content Realization part. Another problem is that this improved system is still extraction-based and does not modify sentence content or combine different sentences. An abstraction-based system would be more plausible in such task and more similar to the thinking of human beings. We might want to modify our Content Realization part to see if we can build an abstraction-based system.

6 Conclusion

We employed both unsupervised and supervised models in this improved system. The Content Selection part combines LexRank model with KL divergence to calculate the sentence scores. The Information Ordering part uses a modified version of experts method to sort the sentences and the Content Realization part uses ILP to generate the final summary.

Our improved automatic summarization system performs significantly better than our baseline system, especially for longer word sequences since the improvements on Rouge-4 and Rouge-3 are more significant. We expect to solve the randomness problem in our next deliverable.

References

Danushka Bollegala, Naoaki Okazaki and Mitsuru Ishizuka 2012. *A Preference Learning Approach to Sentence Ordering for Multi-document Summarization.* Information Sciences 2012, 217:78-95.

Dimitrios Galanis, Gerasimos Lampouras and Ion Androutsopoulos. 2012. *Extractive Multi-Document Summarization with Integer Linear Programming and Support Vector Regression.* Proceedings of COLING 2012, 2012: 911-926.

Dan Gillick and Benoit Favre. 2009. *A Scalable Global Model for Summarization.* Proceedings of the NAACL HLT Workshop on Integer Linear Programming for Natural Language Processing, pages 1018, Boulder, Colorado, June 2009.

Ryan A.Georgi 2018. *Lecture of Content Selection and Information Ordering.* in the course LING 573 Spring 2018 in the University of Washington.

John M.Conroy, Judith D.Schlesinger, Jade Goldstein and Dianne P.O'Leary 2004. *Left-Brain/Right-Brain Multi-Document Summarization.* Proceedings of DUC, 2004

Kai Hong and Ani Nenkova 2014. *Improving the Estimation of Word Importance for News Multi-Document Summarization.* Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics 2004, 2004:712-721

Gunes Erkan and Dragomir R.Radev 2004. *LexRank: Graph-based Lexical Centrality as Salience in Text Summarization.* Journal of Artificial Intelligence Research 22 2004, 2004:457-479