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 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 unsupervised extraction approach and mainly uses the algorithms mentioned in the class. We used several different data structures, including list, dictionary and set to store the information extracted from the corpora and the detailed approach implementation for the three major parts would be discussed below and these approach is targeted to a single document set input.

3.1 Content Selection

Our team used the unsupervised LexRank model to calculate the score for each sentence in the document set. 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 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.

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 final step is trying to converge the vector P using the update rule $P_t = M^T P_{t-1}$ to converge. In our experiment, we set the tolerance to be 0.001 and the convergence speed very fast.

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:

$$PREF_e(u, v, Q) \in [0, 1] \tag{1}$$

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:

$$PREF_{chro}(u, v, Q) = \begin{cases} 1 & T(u) < T(v) \\ 1 & [D(u) = D(v)] \wedge [N(u) < N(v)] \\ 0.5 & [T(u) = T(v)] \wedge [D(u) \neq D(v)] \\ 0 & otherwise \end{cases} \tag{2}$$

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:

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 $t \in X$

```
1:  procedure ORDERING
2:      $V \leftarrow X$
3:      $Q \leftarrow \emptyset$
4:      for each $v \in V$ do
5:          $\pi(v) = \sum_{u \in V} PREF_{chro}(v, u, Q) - \sum_{u \in V} PREF_{chro}(u, v, Q)$
6:      while $V \neq Q$ do
7:          $t = argmax_{u \in V} \pi(u)$
8:          $\hat{\rho}(t) = |V|$
9:          $V = V - \{t\}$
10:         $Q = Q + \{t\}$
11:         for each $v \in V$ do
12:             $\pi(v) = \pi(v) + PREF_{chro}(t, v, Q) - PREF_{chro}(v, t, Q)$
    return $\hat{\rho}$
```

Basically, this ordering algorithm regards the preference function as a directed weighted graph where initially the set of vertices V is equal to the set of sentences X, and each edge $u \to v$ 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 and Favre, 2009). Different from the ILP algorithm we implemented in D2, we use a new objective function which maximizes the weighted summary of all presence bigrams:

$$\sum_i w_i c_i \tag{3}$$

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:

$$\sum_j l_j s_j \leq L \tag{4}$$

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:

$$s_j Occ_{ij} \leq c_i \ \forall i, j \tag{5}$$

$$\sum_j s_j Occ_{ij} \geq c_i \ \forall i \tag{6}$$

Constraint (3) 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 (4) 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 (4) can be always satisfied as well.

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

$$\begin{aligned} \text{Maximize} : & \sum_i w_i c_i \\ \text{Subject to} : & \sum_j l_j s_j \leq L \\ & s_j Occ_{ij} \leq c_i \ \forall i, j \\ & \sum_j s_j Occ_{ij} \geq c_i \ \forall i \\ & c_i \in \{0, 1\} \ \forall i \\ & s_j \in \{0, 1\} \ \forall j \end{aligned} \tag{7}$$

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.

Besides ILP programming, we also implemented a sentence compression module by removing unnecessary elements in the sentence. Three types of elements are removed from sentences: clauses starting with an adverb, contents in a pair of parenthesis and appositions. A regular expression is used to remove contents in a pair of parenthesis. For the other two constituents, we first parse the sentence using spacy dependency parser. If a transition is recognized as "advcl", which indicates the leading word of this clause is an adverb, the subtree under this transition is removed. Similarly, if a transition is recognized as "appo", which indicates that the subtree under this transition represents an apposition, then this subtree is pruned.

4 Results

We tested our code on the `training` and `devtest` dataset and generated two different sets of summary texts. Although our current baseline model does not require additional time for model training, it still takes a while to compute the score matrices and optimize the target function in the Content Selection and Content Realization module. 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	0.20038	0.25005	0.22145
ROUGE-2	0.04347	0.05310	0.04759
ROUGE-3	0.01218	0.01495	0.01336
ROUGE-4	0.00356	0.00433	0.00389

Table 2: ROUGE Results for `training`

ROUGE	Recall	Precision	F-score
ROUGE-1	0.20231	0.28003	0.23382
ROUGE-2	0.05260	0.07219	0.06062
ROUGE-3	0.01885	0.02640	0.02191
ROUGE-4	0.00847	0.01224	0.00997

Table 3: 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

5 Discussion

Frankly speaking, from the results shown in the above tables, this is not a satisfying system for the task. The performance indicators, namely the recall, precision and F-score for each `ROUGE` categories, are not good enough to deliver a readable and comprehensible machine-generated summary. As stated in the specification of this deliverable, this is a baseline automatic document summarization system focused primarily on connectivity and efficiency and we think our project delivers the goal and makes the system work without incurring significant problems.

For the baseline system, we employed an unsupervised, extraction-based algorithm in all three major parts of the summary generation. It does help our team to generate the summaries relatively quick and without training model, there is no risk for overfitting the training dataset. However, without a proper training process, our model might lose lots of its accuracy and credibility. In the future, our team might want to use a supervised algorithm to perform the training process first and then uses a trained model to generate summaries from the `devtest` and `evaltest` datasets.

Unfortunately, all three compression methods have negative affect, based on ROUGE score. It seems that the removed constituents still contain useful information, simply removing them is not a good choice for sentence compression.

6 Conclusion

This deliverable is our first team project for this course, and also our first trial towards building an automatic document summarization system. All of our four team members work well together, with each member focusing on a specific module and we connect the system together in the end to make it work. Our project delivers a baseline system focused on unsupervised, extraction-based and connectivity-focused summarization system. The performance of our system does left a lot improving space for us to work in the future and we want to introduce several supervised learning algorithms to make the system better.

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.