LING 573 Deliverable #2 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 baseline automatic summarization system our team built for the Deliverable #2 of LING 573 in Spring 2018. The system takes input as a set of document sets and generates the summarization for each document set respectively. Our system has four major parts: data preprocessing, content selection, information ordering and content realization. These four parts were implemented separately and assembled together for submission.

1 Introduction

Automatic summarization is a traditional 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 widely-used approaches for this problem, extraction-based summarization and abstraction-based summarization. In this baseline system, our team built a system upon the extraction-based method, which extracts sentences from the input document set without modifying the content of the sentences. Since this deliverable is a baseline system, we made our decision to use an unsupervised approach to assist with out extraction summary. Hence, there is no training and model construction required in this deliverable. Given the dataset documentation, our system can output the summary files individually without training process. We used `Python3` to implement our system.

2 System Overview

We employed the classic system structure mentioned in the lecture 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 generate a `corpus` object, which includes `docsetList` and a token dictionary to record the frequency of each word occurrence in the input dataset. The `docsetList` object includes the document set ID, a token dictionary, a topic ID, a collection of summary texts generated by human beings and `documentCluster`. The `documentCluster` object is a list of document with its essential information, and each document has a list of sentences recorded.

The `content_selection` part takes a document set as an input, and generates a list of sentences with their scores, computed by the LexRank algorithms. 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 truncates the unnecessary sentences 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

Given a list of sentences selected by Content Selection procedure and salient scores associated, the baseline model of Information Ordering takes inter-document chronology and intra-document cohesion factors into account to generate a sorted list. Firstly, our model sorts the sentences by their chronological order, the dates of publication of the document where a particular sentence is extracted. The chronology information has been recorded in the data preprocessing part and would sort the sentences from different documents. Secondly, within the same document, sentences are ordered linearly by their positions in the document to impose cohesion. The position information has also been recorded using indices.

Finally, the Information Ordering part would produce a sorted list of sentences with respect to the chronology information of individual documents and the position information of the sentence in the context.

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:

$$\sum_i w_i c_i \tag{1}$$

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:

$$\sum_j l_j s_j \leq L \tag{2}$$

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:

$$s_j Occ_{ij} \leq c_i \ \ \forall i, j \tag{3}$$

$$\sum_j s_j Occ_{ij} \geq c_i \ \ \forall i \tag{4}$$

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:

$$\begin{aligned} \text{Maximize}: & \sum_i w_i c_i \\ \text{Subject to}: & \sum_j l_j s_j \leq L \\ & s_j Occ_{ij} \leq c_i \ \forall i, j \\ & \sum_j s_j Occ_{ij} \geq c_i \ \forall i \\ & c_i \in \{0, 1\} \ \forall i \\ & s_j \in \{0, 1\} \ \forall j \end{aligned} \quad (5)$$

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 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

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.

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

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.