575 Report D3

Rachel Hantz, Yi-Chien Lin, Yian Wang, Tashi Tsering, Chenxi Li

Department of Linguistics
University of Washington
Seattle, WA USA
{hantz1rk,yichlin,wangyian,tashi0,cl91}@uw.edu

Abstract

In this project, we focus on multi-document summarization. The input is a set of documents with the same topic, and the output is a summary. Considering the documents we use for train, devtest and evaltest are from different corpora with very different formats, we first preprocess them. Then, we begin summarizing and divide the process of summarization into three steps: content selection, information ordering, and content realization. We adopt two methods for content selection: Term Frequency-Inverse Document Frequency (TF-IDF) and Integer Linear Programming (ILP). For information ordering and content realization, we didn't implement them for D3 and will implement them in later delivery.

1 Introduction

Text summarization has been one of the major tasks in the field of natural language processing (NLP). Automatic text summarization is the process of selecting the most crucial part from a text to create a shortened version. The application of text summarization can be particularly helpful for extracting essential information from large amounts of data.

For our current text summarization system, we focus on applying methods for content selection. Approaches incorporating TF-IDF and ILP are implemented in this system. The datasets used in our system are the AQUAINT (Graff, 2002) and AQUAINT-2 (Vorhees and Graff, 2008) corpora. The datasets are divided into training, development, and evaluation data. For the development of our current system, we develoop our algorithm with the training data and evaluate the system performance on the development data. The evaluation data is held out and will only be used to evaluate the performance of our final system. Our current system achieves F1-scores of around 0.334 for our first method and 0.327 for our second method when measuring the matching unigrams; our system achieves F1-scores of around 0.077 for our first method and 0.08 for our second method when measuing the matching bigrams.

The rest of this report is organized as follows. Section 2 and Section 3 presents a overview of our system architecture and the details our approaches used for data preprocessing and content selection. Section 4 shows the results from our current system, and Section 5 provides an analysis of the results. Lastly, a conclusion is provided in Section 6.

2 System Overview

Given a document set, our method extracts sentences as a summary in four steps: preprocessing, content selection, information ordering, and content realization, as is showned in figure 1. We compare two methods for content selection: TF-IDF and Integer Linear Programming.

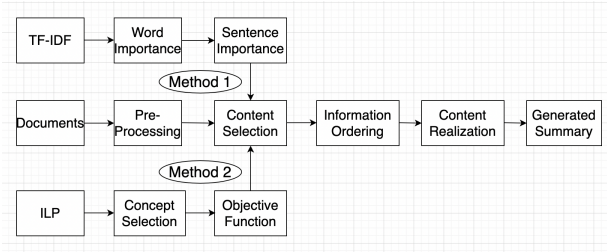

Figure 1: system overview

3 Approach

Our approach is comprised of four main steps. Preprocessing, Content Selection, Information Ordering, and Content Realization. We evaluate two methods for Content Selection: TF-IDF and Integer Linear Programming.

3.1 Preprocessing

In order to use the articles that we will summarize, we first needed to pre-process them. We accomplished this with a script that implemented two overall steps: process and tokenization.

Processing takes in the path of input xml file and extracts the document ID. Specifically, we imported xml.dom.minidom to parse the path of input xml file: we called getElementsByTagName() to obtain the elements under docsetA and called getAttribute() to obtain each document ID.

Tokenization takes in a document ID and output a file of desired format needed for our later tasks. After locating the xml document in corpora, we used xml.etree.ElementTree to create a tree for the xml document. On the root node, we obtained text content of a part by matching the tag name (e.g., node.tag == HEADLINE). For tokenization, we just used nltk.sent_tokenize() to break the paragraph and and nltk.word_tokenize() to break each sentence. The tokenization schema produces output files for each article. Each file starts with headline, date, etc. and has a single tokenized sentence per line. Paragraphs are separated with a single blank line.

3.2 Content Selection

3.2.1 Method 1: TF-IDF

The first method we implemented was Term Frequency-Inverse Document Frequency (TF-IDF). The objective of TF-IDF is to weigh the importance of tokens based on their frequency in the foreground and background. A token that occurs frequently in the foreground and less so in the background will have a high TF-IDF score; the higher the score is, the more important a token is.

Our model of TF-IDF consists of two parts: calculating TF-IDF scores of all the tokens and selecting sentences for summary based on the scores. For this particular task, we set all the documents in development test directory as background and the ten articles to be summarized under the same topic as foreground. For TF, we used the raw term frequency, the number of times a token has appeared in the foreground. Note that for the term here, we decided to remove the punctuation and stop words in NLTK library. The formula of IDF is (1), (N) stand for the total number of document and $(df(w))$ is the number of documents containing term (w). L2 normalization (Euclidean normalization) is applied when calculating the final score. Note that for the term here, we decided to remove the punctuation and stop words in NLTK library.

$$idf(w) = 1 + ln[(N+1)/(df(w)+1)] \quad (1)$$

We used python library Sicikit-Learn, which provides a simple implementation of the TF-IDF method through the TfidfVectorizer class. We store Topic ID, sentence, word count (excluding stop words), word count (including stop word) and TF-IDF score each as corresponding column and populated a Panda dataframe. Then we sort TF-IDF score in ascending order for sentences within each document set to select the sentences with with highest score.

$$s_w = (\sum_i w_i)/n \quad (2)$$

In the second part of sentence selection, the score of each sentence (s_w) is the average TF-IDF score of the tokens (w_i) in the sentence. When choosing sentences, we set two parameters to compare the performance of the model. The first parameter is the number of tokens in a sentence, excluding stop words: if the sentence length is below the threshold, the sentence will not be selected. The second parameter is cosine similarity between sentences. When adding sentence to the summary, the average cosine similarity between the new sentence and currently selected sentence is calculated: if cosine similarity is above the threshold, meaning that the new sentence is too similar to the current sentences, the new sentence will not be selected.

In the remainder of this paper, 'TF-IDF: X, Y' denotes TF-IDF model with X as the sentence length threshold and Y as the cosine similarity threshold.

3.2.2 Method 2: Integer Linear Programming

We also leveraged ILP for our multi-document summarization system, akin to previous implementations (Gillick and Favre, 2009). ILP optimizes a function that is subject to various constraints. This function and these constraints are comprised of decision variables that must be integer valued.

Using the python package PuLP (Mitchell et al., 2011, PuLP)–a python linear programming API– we selected sentences that contained the most important (highest weighted) concepts from a set of multiple documents[1].

Sentences (s_j) and concepts (c_i) were our two binary decision variables. Each concept was a stemmed bigram that could not contain any punctuation or only stopwords. We utilized nltk's RegexpTokenizer to remove punctuation, SnowballStemmer (with ignorestopwords=true), and bigrams.

[1] https://coin-or.github.io/pulp/

Each concept had a weight equaling the number of articles it appeared in. Only concepts found in at least three documents were retained. Using each concept, we maximized our objective function (3): the wieghted (w_i) sum of each concept.

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

We had three constraints: A length constraint (4) where the sum of each included sentence's length (l_j) could not be more than 100 whitespace delimited tokens. A coverage constraint (5) where each included concept was found in at least one included sentence. And a second coverage constraint (6) where no concepts that weren't included could be found in any included sentences. o_{ij} denotes the binary presence of c_i in s_j.

$$\sum_j s_j l_j \leq 100 \tag{4}$$

$$\sum_j s_j o_{ij} \geq c_i \forall i \tag{5}$$

$$s_j o_{ij} \leq c_i \forall i, j \tag{6}$$

Using the GLPK solver within PuLP, the ILP model returned which sentences should be selected to maximize the weighted sum of the concepts included. Each included sentence was then printed to a single output file.

3.3 Information Ordering

Our system's current information ordering schema is heurstic based. We order sentences first by their position in an article and then by the article date of publication.

3.4 Content Realization

Our system's current content realization is to not modify the selected sentences just yet.

4 Results

Our summarization system is evaluated on the Recall-Oriented Understudy for Gisting Evaluation metrics (Lin, 2004, ROUGE) with the ROUGE python package[2]. Specifically, we use the ROUGE-N to measure the number of n-grams occurring in both system-generated summaries and the human summaries. For the evaluation of our system, we evaluated matching unigrams and bigrams between the system-generated summaries and the human summaries. The metrics included in the evaluation process are: recall, precision, and F1 scores[3]. The results of the two content selection methods are presented respectively in the following.

ROUGE-N	Metric	Method1	Method2
ROUGE-1	Recall	0.30895	0.30750
	Precision	0.36750	0.35340
	F1-score	0.33475	0.32734
ROUGE-2	Recall	0.07160	0.07539
	Precision	0.08529	0.08723
	F1-score	0.07761	0.08049

Table 1: ROUGE scores of the methods used in the summarization system; TF-IDF: 10, 0.1

For the first method, when the two parameters are chosen appropriately, ROUGE-1 scores are slightly higher than the second method, as shown for TF-IDF:10, 0.1 in the table. However, ROUGE-2 scores are consistently lower than those of the second method in all combinations of parameters tested. When we change the number of tokens to a higher number, 15 or 20, the performance show a minor imporvement. For example: TF-IDF: 15 (token number), 0.3 (cosine similarity) yields ROUGE-1 score: recall, 0.31097; precision, 0.37256; F-score, 0.33777 which are slightly better than the scores reported in table 1

As shown in Table 1, the recall scores of the second method are 0.3075 and 0.07539 for measuring the unigrams and bigrams across the system-generated summaries and the human summaries, showing that this method captures around 31% of the unigrams and 8% of the bigrams occurring in the human summaries. The precision scores of this method are 0.35340 and 0.08723, showing that around 35% of the unigrams and 9% of the bigrams in the system-generated summaries are consistent to the human summaries. For our second method, the F1 score is respectively 0.32734 and 0.08049.

5 Discussions

The two methods we implement for content selection yield similar ROUGE scores, as is shown in Table 1. However, the content of the summaries they generated were very different. Take D1046 as an example. There are 10 documents in this document set, with 42 sentences in total. The summary generated by TF-IDF contains 3 sentences, and the

[2] https://pypi.org/project/rouge-score/

[3] The scores reported here are the average scores of all system-generated summaries

one generated by ILP contains 4 sentences. None of these sentences overlap with each other.

The articles in D1046 mainly mention four points: (i) An earthquake unleashed a tsunami, killing 150,000 people in 11 nations. (ii) Indonesia was the hardest hit, suffering great damage. (iii) Countries including the United States, China, Australia, Japan, Bangladesh, Uzbekistan, Denmark, and Switzerland, offer help to Indonesia. (iv) Indonesia would host a summit of world leaders from 23 countries to discuss the aftermath of the disaster.

The summary generated by TF-IDF is: The killer tsunami that was touched off by an 8.7-magnitude earthquake off Indonesia 's Sumatra island on Dec. 26 , has climbed to over 145,000 so far , with over 90,000 deaths reported in Indonesia . Uzbekistan on Monday sent a plane with 35 metric tons (39 short tons) of humanitarian aid to tsunami-hit Indonesia , the Foreign Ministry said . Nearly 500,000 people were made homeless after tsunami swept Aceh province , in Indonesia on Dec. 26 , ministry of information and communication said on Wednesday .

We observe that this summary covers point (i), (ii), and (iii). The overlap between the three sentences is small, which is good. This can be attributed to the calculation of cosine similarities between sentences, as is described in section 3.2.1. The three sentences selected are long, so there is no space for a fourth sentence due to the word limitation of 100. This is a drawback we want to solve in the future. If we can simplify the sentences selected, we may have space to write more sentences and cover more points.

The summary generated by ILP is: The Japanese government is committed to increase its aid for tsunami-hit Aceh and North Sumatra provinces to 146 million US dollars , Japanese Ambassador to Indonesia Yutaka Iimura said on Monday . Japan has provided some 500 million US dollar fund for tsunami- affected countries including Indonesia , the Maldives and Sri Lanka . The Ilyushin-76 cargo plane headed for Medan , the main city on Indonesia's Sumatra island , one of the areas hardest hit by the Dec. 26 tsunami , the ministry said in a statement . They will leave for Aceh Province later in the day .

We observe that this summary only covers point (iii). The reason causing redundancy here is ILP doesn't check similarities between sentences, and it doesn't assign a new weight to a concept if the sentence it is in is already selected.

6 Conclusion

Summarize the main points and look ahead. What would your next steps be?

References

Dan Gillick and Benoit Favre. 2009. A scalable global model for summarization. In *Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*, pages 10–18, Boulder, Colorado. Association for Computational Linguistics.

D Graff. 2002. The aquaint corpus of english news text. philadelphia, pa: Linguistic data consortium.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Stuart Mitchell, Michael J. O'Sullivan, and Iain Dunning. 2011. Pulp : A linear programming toolkit for python.

Michael Paul, ChengXiang Zhai, and Roxana Girju. 2010. Summarizing contrastive viewpoints in opinionated text. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 66–76.

Ellen Vorhees and David Graff. 2008. *AQUAINT-2 information-retrieval text: research collection*. Linguistic Data Consortium.

Appendix A

For D1: Rachel went through the tutorial of git; Yi-Chien and Yian showed the basics of overleaf; Tashi and Chenxi wrote up the submission pdf.

For D2: Yian did the coding part, with some help from Rachel and Yi-Chien; Yi-Chien posted tutorial on setting up Anaconda environment on Patas for the group; Tashi made the slides for presentation; Chenxi wrote up the report D2. Rachel will be presenting in class.

For D3: Rachel implemented ILP method; Chenxi and Tashi implemented TF-IDF method; Yi-Chien implemented evaluation script and the pipeline. Yian tested and analyzed the result. Report and slides were contributed by everyone.

Appendix B

Link to the code repository on github:
https://github.com/rhantz/575_summarization
Off-the-shell tools used in code:
• xml to parse the path of input file and the xml document in corpora
• nltk for sentence and word tokenization, stemming, and bigramming
• os for system operation on Patas
• PuLP for Linear Programming
• GLPK as the ILP solver
• TfidfVectorizer from sklearn for TF-IDF matrix
• Pandas for building dataframe
• rouge_score for evaluation

Appendix C

Problem 1:
Description: documents in AQUAINT corpora are not rooted, causing parsing to fail.
Solution: created a root node by inserteing <tag> at the start and appending <\tag> in the end.

Problem 2:
Description: code fails to run on Patas for some group members
Solution: set Anaconda on Patas to ensure people have the same environment.

Problem 3:
Description: Anaconda environment with multiple dependencies became tricky to export and re-install from other's branches
Solution: Create new environment file from scratch with clear dependencies

Problem 4:
Description: ROUGE implementation on patas was inaccessible due to missing perl file.
Solution: Use python rouge-score package instead