D3

Anna Batra, Sam Briggs, Junyin Chen, Hilly Steinmetz
Department of Linguistics, University of Washington
{batraa, briggs3, junyinc, hsteinm}@uw.edu

Abstract

This deliverable contains a skeleton of the paper and a list of team members. This is an example citation (Jurafsky and Martin, 2022). According to Jurafsky and Martin (2022), this is another example citation.

1 Introduction

2 Engines

3 System Overview

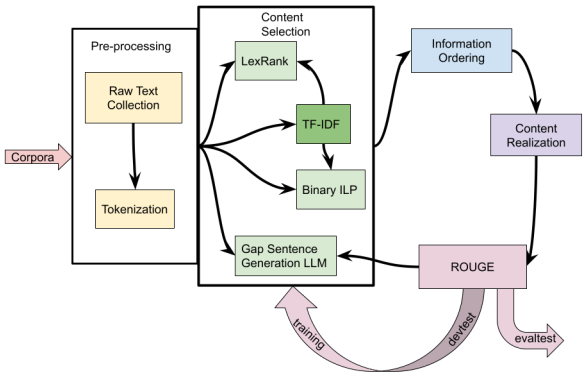

Figure 1: System Architecture

4 Approach

4.1 Data Pre-processing

4.1.1 Collecting raw text

We used the data from the TAC 2009, 2010, 2011 Shared Task data. To process the given XML files to retrieve the set of document names in docSetA, we used xml.etree.ElementTree. We then figured out which corpus data path matched each document in docSetA.

In order to read in these AQUAINT and AQUIANT2 files that are organized differently, we used lxml.etree as this can parse non-XML compliant files. We found there are three different organization methods that were used, and we made sure to process each kind uniquely. From these files, we parsed the the headline, time, and raw text paragraphs.

4.1.2 Tokenization

After getting the raw text, we used two different tokenization methods: spaCy and NLTK.

From spaCy 2.0 we used the English model tokenizer “en_core_web_sm". We decided on spaCy 2.0 since it fits the python version (Python 3.6) on Patas (the virtual machine we used). We decided not to tokenize the raw text into sentences first. Instead, we just ran a word tokenizer on the raw text. The spaCy tokenizer utilizes a transformer under the hood. We then realized that there was a sentence tokenizer, but since this tokenization method uses transformers on the CPU, it actually takes a really long time to run. Therefore, we decided to leave it as it is and turn to a rule-based tokenizer explained in the next method.

For NLTK, we used the English model tokenizer “tokenizers/punkt/english.pickle". We first ran a sentence tokenizer, and then for each tokenized sentence, we then ran a word tokenizer. This gave us a list of sentences, where each sentence contains tokenized words. The NLTK tokenizer is significantly faster than spaCy’s tokenizer and gives us the correct output we need per D2.

4.2 Content Selection

4.2.1 TF-IDF

We used a logarithmically scaled, add δ_1 smoothed tf, and we used an add δ_2 smoothed idf to weight each term in the document set (Seki, 2003). Given a set of documents D, a term t, and a document $d \in D$, we calculated the term-frequency, inverse

document frequency ($tf \cdot idf$) as follows:

$$
\begin{aligned}
tf \cdot idf(t, d, D) &= tf(t, d) \cdot idf(t, d, D) \\
tf(t, d) &= \log(\delta_1 + f_{t,d}) \\
&= \log(\delta_1 + |\{t \mid t \in d, d \in D\}|) \\
idf(t, D) &= \delta_2 + \log\left(\frac{N}{\delta_2 + n_t}\right) \\
&= \delta_2 + \log\left(\frac{|D|}{\delta_2 + |\{d \mid t \in d, d \in D\}|}\right)
\end{aligned}
$$

4.2.2 Binary Linear Programming

We used Luo et al. (2018) as a guide to design our summarizer of a document set. For notation, we use y_j for sentence j and z_i for concept i in the document set. We use $A_{i,j}$ to denote the indicator function $\mathbb{1}_{z_i \subseteq y_j}$, i.e. $A_{i,j}$ = 1 if concept z_i appears in sentence y_j, 0 otherwise. We use the weight w_i where weight i is the corresponding weight for "concept" i. We also have a maximum term summary length L.

We then formulated the optimization problem as follows:

$$
\begin{aligned}
\text{maxmimize} \quad & \sum_i w_i z_i \\
\text{Subject To} \quad & \sum_j A_{i,j} y_j \geq z_i \\
& A_{i,j} \leq z_i \\
& \sum_j l_j y_j \geq L
\end{aligned}
$$

For each "concept" z_i, we used unigrams. For the corresponding weight w_i, we used the tf-idf score of the unigram z_i as calculated in 4.2.1.

4.2.3 LexRank

We also implemented the LexRank algorithm described in Erkan and Radev (2004). LexRank is an adaption of the PageRank algorithm (Page et al., 1999), and was proposed as an alternative to centroid-based approaches. LexRank leverages relationships between documents by creating a weighted graph that connects sentences. Relating the sentences to one another has the advantage of (1) dampening the effect of high IDF scores of rare words and (2) formalizing a preference for more informative (or more connected) sentences.

Sentences are related to one another using a modified cosine similarity measure using term frequency and inverse document frequency values:

$$
\begin{aligned}
&sim(x, y) \\
&= \frac{\sum_{w \in x,y} \mathrm{tf}_{w,x} \mathrm{tf}_{w,y} (\mathrm{idf}_w)^2}{\sqrt{\sum_{x_i \in x} (\mathrm{tf}_{x_i,x} \mathrm{idf}_{x_i})^2} \cdot \sqrt{\sum_{x_i \in x} (\mathrm{tf}_{y_i,x} \mathrm{idf}_{y_i})^2}}
\end{aligned}
$$

Where, $\mathrm{tf}_{w,x}$ is the term frequency for word w in sentence x and idf_w is inverse document frequency for for word w. To calculate this similarity measure for multiple documents, we concatenated the documents in a document set and treated it as a single document.

Using this similarity measure, we created a similarity matrix between sentences in the document, which also functioned as a weighted graph. Per Erkan and Radev (2004), values with low similarities scores are discarded and self-connections between nodes. The matrix satisfies the properties of a stochastic matrix, allowing us to use the power method to eigenvalue of the matrix by applying the following update to a centrality vector $\boldsymbol{p}$:

$$\boldsymbol{p} = [d\boldsymbol{U} + (1 - d)\boldsymbol{B}]^T \boldsymbol{p}$$

where $\boldsymbol{U}$ is a square matrix with values equal to $1/\mathrm{count}(documents)$ and $\boldsymbol{B}$ is the adjacency matrix of the graph. After finding $\boldsymbol{p}$, the sentences with the highest values in $\boldsymbol{p}$ are extracted and added to the summary text until the summary reaches the maximum length.

4.2.4 Gap sentence generation

We used the gap sentence generation method introduced in Zhang et al. (2019). Based on the finding when Zhang et al. were training the Pegasus based model, we will selecting the top m sentences as gap sentences without replacement from a document based on importance score. The importance score is calculated based on the ROUGE score one sentence get comparing to the remaining sentences in one document as in Algorithm 1.

Algorithm 1 Independent sentence selection

```
D := {x_i}_n ← sentences in document
S := ∅
I ← list contains index from 0 to n
for j ← 1 to n do
    s_i := rouge(x_i, D \ {x_i})
    S := S ∪ {s_i}
I := sort(I) Based on the value in S
```

We only mask the top thirty percent of the sentences as Zhang et al. finds out achieves rela-

tively high performance without sacrifice training effiency.

4.3 Information Ordering

4.3.1 Binary Linear Programming

We currently do not order the sentences, and keep the sentences in the same order as they appear in order seen in the docsets. For example, in the docset file system, all the sentences from the first document will appear first, in the order they are seen in the actual document as well.

4.3.2 ROUGE score ranking

Due to the input size limitation for the majority of the language model, we have to truncate the input text to 1024 tokens. After we mask the important sentence, we then use the ROURGE score ranking calculated for the gap sentence generation to discard fifty percent of the sentences that are ranked in the low fifty percent. We keep the ordering of the remainder of the sentences. Discarding fifty percent of the sentences helps including more important sentences from multiple documents. When discarding, we calculate the token length for each added sentences and stop when adding additional sentence will cause the token size to exceed 1024 tokens. This make sure we have full sentences for the input sequence.

4.4 Content Realization

4.4.1 Binary Linear Programming

We currently do not order the sentences, and keep the sentences in the same order as they appear in order seen in the docsets. For example, in the docset file system, all the sentences from the first document will appear first, in the order they are seen in the actual document as well.

4.5 Large Language Model Training

We trained our model based on "google/pegasus-cnn_dailymail" model, as the "google/T5-small" model does not produce complete sentences. We trained the model with batch size of 8 and epoch of 12, since increasing batch size will cause CUDA run out of memory. We then use ROUGE-1 F1's score to determine the best model.

5 Results

	ROUGE1	ROUGE2
Binary ILP	0.12085	0.01533
LexRank	0.13720	0.02341
GSG LLM	0.21037	0.06214

Table 1: ROUGE Recall Scores

6 Discussion

We perform a casual error analysis for the summaries based on docset D1001, which are shown in Table 2. Binary ILP method indicates successfully in the first sentence that there is a school shooting and bombing. The method also mentions the place the school shooting occurred in the following sentences. For the rest of the sentences produced by the Binary ILP method, however, are involving around communities and songs wrote by two brother. The LexRank method also indicates that the story is about a mass shooting, and mentions it is a shool shooting occur in the following sentences. However, the LexRank method does not capture the place the shooting occur. Lastly, LexRank method produces "Littleton needs comfort" twice. The GSG LLM methods captures that the documents are about a massacre, but does not mention that it is about a school shooting. The two sentences are logically connected. Both the LexRank and GSG LLM methods mentions the death count.

These our our future steps for the next step of the project:

- Finish information ordering and content realization for TF-IDF.
- Use dev-test to pick good delta for smoothing in TF-IDF (TF-IDF & ILP Summarization)
- Add bigram concepts to ILP, dev-test to see if better than unigram
- Switch to BLOOM with AutoModelForQuestionAnswering class for training, since using AutoModelForCausalLM for BLOOM produce input size not match error.
- Improve LLM pre-processing methods

gold	In the worst school killing in U.S. history, two students at Columbine High School in Littleton, Colorado, a Denver suburb, entered their school on Tuesday, April 20, 1999, to shoot and bomb. At the end 15 were dead and dozens injured. The dead included the two students, Eric Harris and Dylan Klebold, who killed themselves. Harris and Klebold were enraged by what they considered taunts and insults from classmates and had planned the massacre for more than a year. The school is a sealed crime scene and Columbine students will complete the school year at a nearby high school.
Binary ILP	At one point , two bomb squad trucks sped to the school after a backpack scare . Phone : (888) 603-1036 Please comfort this town . " Many looked for it Saturday morning on top of Mt . But what community was it from ? There are the communities that existed already , like Columbine students and Columbine Valley residents . Brothers Jonathan and Stephen Cohen sang a tribute they wrote . " Columbine ! " " Love is stronger than death . " Some players said the donations and support will encourage them to play better .
LexRank	Sheriff John Stone said Tuesday afternoon that there could be as many as 25 dead. The arrival of two bomb squad trucks with sirens blaring further shook those inside. With photo. With photo. The New York Times plans two pages of stories, photos and graphics on the aftermath of the school shooting in a Denver suburb that left 15 dead. Herbert interjected:"I'm a little worried about putting all the kids in one place. Littleton needs comfort. Littleton needs comfort.
GSG LLM	Sheriff's initial estimate of as many as 25 dead in Columbine massacre was off the mark. discrepancy occurred because the SWAT teams that picked their way past bombs and bodies in an effort to secure building covered overlapping areas.

Table 2: Error analysis for docset D1001

7 Conclusion

8 Appendix A: Workload distribution

8.1 D1 Workload

- Anna Batra set up the Github repository, turned in D1
- Junyin Chen got the team together and set up a communication channel
- Sam Briggs set up the Overleaf file and sent out a when-to-meet to schedule weekly meetings
- Hilly Steinmetz edited the Overleaf file to prepare it for D1.

8.2 D2 Workload

- Anna Batra and Sam Briggs wrote test code to test the file structure of the output docSets, created the outline for the presentation, and updated the report.
- Junyin Chen wrote code for tokenizing documents in the docSets using spaCy, PR reviewed the code to merge with Hilly's, cleaned up the code, and created slides for the pre-processing section.
- Hilly Steinmetz wrote the code for the pre-processing steps before tokenization, such as locating paths for AQUAINT and AQUAINT2 files. Hilly also wrote code for tokenizing documents using NLTK.

8.3 D3 Workload

- Anna Batra and Sam Briggs wrote the code to create a json file to easily access our data for the rest of the project. They also wrote the code for the TF-IDF and Linear Programming content selection methods. The Linear Programming information ordering and content realization was also written by them. They also drew the system architecture.
- Junyin Chen wrote the code to create JSON file writer which contains doc_id, text for summarization, the gold standard summarization based on doc_id for both docsetA and docsetB. The writer help cache the JSON file for easier access. He also wrote the code for Gap sentences generation content selection method,

truncate the input text based on ROUGE score for information ordering, and write the code for training a large language model for content realization. He also performs quick error analysis.

- Hilly Steinmetz wrote the code for the LexRank method and debugged issues with the original XML document parser.

Everyone worked on the paper and presentation slides for the parts we explicitly worked on.

9 Appendix B: Code repository and additional software and data used in your system

9.1 Source Code

The repository for our project can be found on Github at github.com/LING-575-Summarization/Summarization.

9.2 Packages

We used the following packages for our system:

9.2.1 Pre-processing

- lxml.etree (for processing AQUAINT, AQUAINT2, TAC files)
- xml.etree.ElementTree (for processing docSetA file lists)
- spaCy 2.0 (for word tokenization on paragraphs)
  - English model “en_core_web_sm"
- NLTK (for sentence and word tokenization)
  - English model 'tokenizers/punkt/english.pickle'

9.3 Content Selection

- PuLP (for binary ILP)
- rouge-score
- Datasets
- Evaluate
- Pytorch

9.4 Future possible packages

We plan to use the following software for the project:

- Python
- Anaconda (for virtual environment)
- NLTK (for parsing, NER, and more)
- spaCy (for parsing, NER, and more)
- Scikit-learn (for machine learning models)
- Pytorch (for large LMs)
- Hugging Face's Transformer library (for auto-tokenization and find-tune trained model)

References

Günes Erkan and Dragomir R Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, 22:457–479.

Dan Jurafsky and James H. Martin. 2022. *Speech and Language Processing*, 3rd edition (draft) edition. Online.

Wencan Luo, Fei Liu, Zitao Liu, and Diane Litman. 2018. A novel ilp framework for summarizing content with high lexical variety. *Natural Language Engineering*, 24(6):887–920.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The pagerank citation ranking: Bringing order to the web.

Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. 2000. Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies. In *NAACL-ANLP 2000 Workshop: Automatic Summarization*.

Yohei Seki. 2003. Sentence extraction by tf/idf and position weighting from newspaper articles. *National Institute of Informatics: Proceedings of the Third NTCIR Workshop*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. 2019. PEGASUS: pre-training with extracted gap-sentences for abstractive summarization. *CoRR*, abs/1912.08777.