575 Report D4

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). The outputs of the content selection are then reordered using two information ordering (IO) methods. We applied the position ordering for summaries generated with TF-IDF content selection method and cosine similarity ordering for those generated with the ILP content selection method.

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 develop 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 0.34 for the TF-IDF content selection method and 0.36 for the ILP content selection method when measuring the matching unigrams; our system achieves F1-scores of around 0.079 for our former method and 0.099 for the latter method when measuring 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, content selection, and IO. 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 ILP. As for information ordering, we use position ordering for TF-IDF and use cosine similarity ordering for ILP.

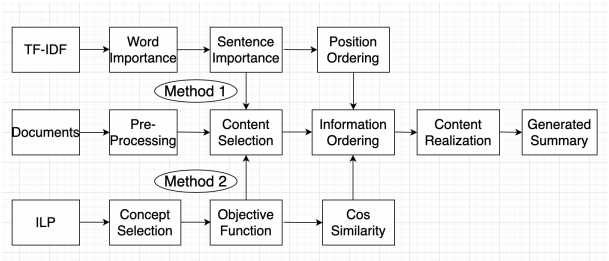

Figure 1: system overview

3 Approach

Our approach is comprised of four main steps. Preprocessing, Content Selection, Information Ordering, and Content Realization. We implemented two methods for content selection, TF-IDF and ILP, and

compared our implementation of four methods for IO.

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's[1] `sent_tokenize()` to break the paragraph and `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

For our summarization system, we implemented two content selection methods: TF-IDF and ILP. Descriptions of the two methods are detailed in the following subsections.

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.[2] 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 Pandas 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 including sentences into summary, we explored many hyperparameters to compare the model performance, as discussed in section 3.3.

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.[3] Thus, sentences (s_j) and concepts (c_i) were our two binary decision variables.

[1] https://www.nltk.org/

[2] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

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

Concepts could be defined as one of the following: named entities, stemmed unigrams, stemmed bigrams, or stemmed skipgrams. Skipgrams could take any size integer degree and could be at most two tokens. Additionally, all concepts, excluding named entities, could have punctuation and/or stop words removed. Unigrams always had stop words removed. For any bigram or skipgram, there could only be at most one stop word. We utilized NLTK's RegexpTokenizer to remove punctuation, SnowballStemmer (with `ignorestopwords=True`) to stem words, bigrams, and skipgrams functions. For named entities, we used the spaCy[4] NER pipeline component.

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 weighted (w_i) sum of each concept.

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

We had three required constraints: An overall 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.

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

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

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

Two constraints were optional: A minimum sentence length constraint (7) where each individual included sentence was required to be a minimum sentence length (m). And a theme constraint (8) where sentences could be clustered into themes, and each theme (t_k) could only be included at most 1 time among the selected sentences. p_{jk} denotes the binary presence of t_k in s_j. We used the python library `Scikit-Learn`'s `TfidfVectorizer` class and `KMeans`[5] clustering algorithm to denote each sentence with one of a set of k themes. Details of the k-means clustering algorithm can be found in Section 3.4.2.

$$l_j s_j \geq m s_j \tag{7}$$

$$\sum_j s_j p_{jk} \leq 1 \forall k \tag{8}$$

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 Hyperparameter Search

In this section, we describe the hyperparameters we experimented with for each content selection methods and how we decided the hyperparameters used in our final system for this deliverable.

3.3.1 TF-IDF

Abundant literature has addressed the selection of features when calculating the sentence score. In our TF-IDF model, we tested four hyperparameters: cosine similarity, stop words and punctuation, sentence length, and finally unigrams and bigrams. When adding sentences to the summary, we exclude sentences that are too similar and sentences that are too short. When calculate sentence length, we also explored the the model performance including and excluding stop words and punctuation. Lastly, we tested out both unigrams and bigrams to compare the performance.

For cosine similarity, .R and Arutchelvan (2022) chose 0.75 as their criterion when eliminating similar sentences before clustering; we used it as our starting point and tested out different cosine similarities with fixed sentence length 10, including stop words and punctuation. The best performance is when cosine similarity is set to 0.25. Next, our test results of stop words and punctuation revealed that when punctuation is excluded, including or excluding stop words yield similar results, both better than other combinations. We decided to include stop words; otherwise the sentences would be long and do not resemble a typical summary. As for sentence length, it is represented in some literature as the ratio of current sentence length to the longest sentence in the article (Neto et al., 2002). We used the raw count of tokens, similar to Teufel and Moens (2002)'s method of penalizing sentences shorter than 12 words. Finally, we

[4] https://spacy.io/

[5] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

found that using bigrams in TF-IDF model, as expected, tend to have higher ROUGE2 score and lower ROUGE1 score.

When looking at the summaries generated by the TF-IDF model, we found that they summaries contain sentences that begin with linking words. These sentences are incomplete in meaning on their own. One way to solve this issue is to also include the preceding sentence, regardless of the sentence score, into the summary (Mohd et al., 2020). However, our experimentation with this proposed solution demonstrated that the inclusion of the previous sentence slightly reduced the rouge score. This result suggests that further investigation is needed to determine the optimal strategy for addressing incomplete sentences in summary generation.

3.3.2 ILP

Our manual hyperparameter search for ILP took place in three steps: (1) Stop Words and Punctuation, (2) Minimum Sentence Length and Concepts, (3) Number of Themes. We describe below why we elect to run ILP under the following configuration: Remove only punctuation, minimum sentence length of 5, degree 1 skipgrams, with no theme constraint.

F1, precision, and recall are highest when stop words are retained and punctuation is removed. This can be seen in Table 1, where bigrams are used as concepts and minimum sentence length is set to 0.

N		-SW+P	+SW-P	-SW-P	+SW+P
1	R	.36314	**.37443**	.36749	.37307
	P	.30704	**.33893**	.3188	.318
	F1	.33249	**.3556**	.34005	.34301
2	R	.07588	**.09304**	.08339	.08521
	P	.08999	**.1029**	.09633	.10019
	F1	.08226	**.09766**	.08908	.09201

Table 1: ROUGE-N scores of the ILP content selection based on stop word (SW) and punctuation (P) removal. (bigrams, minimum sentence length = 0)

ROUGE-1 F1 is highest when using degree 1 skipgrams as concepts with a minimum sentence length of 5. A minimum sentence length of 5 is also used by Gillick and Favre (2009) in their ILP pre-processing. Degree 2 skipgrams also perform similarly here, but the ROUGE-2 F1 scores are strongest in degree 1 skipgrams. Named entities perform worst in all cases. This can be seen in Table 2 where stop words are retained and punctuation is removed.

N	m	NE	Uni	Bi	SG-1	SG-2
1	0	.3205	.3388	.3556	.3603	.3611
	5	.3216	.3412	.3543	**.3620**	.3608
	8	.3196	.3394	.3545	.3430	.3597
2	0	.0764	.0816	.0977	.0988	.0972
	5	.0761	.0810	.0959	**.0994**	.0974
	8	.0757	.0813	.0964	.0991	.0974

Table 2: ROUGE-N F1 scores of the ILP content selection based on minimum sentence length (m) and concept type. (stop words retained, punctuation removed)

F1 is highest when we do not apply the theme constraint. Although, including the theme constraint does not drastically decrease F1, and should still be considered if sentence redundancy remains an issue in selected sentences. This can be seen in Table 3 where we evaluate the theme constraint when using degree 1 skipgrams.

Concept	k	ROUGE-1	ROUGE-2
SG-1	0	**0.36202**	**0.09936**
	5	0.35208	0.09384
	10	0.35438	0.09464
	15	0.35702	0.09482
	20	0.35737	0.09639
	25	0.35793	0.09621

Table 3: ROUGE-N F1 scores of the ILP content selection based on number of themes (k) used in the theme constraint. (stop words retained, punctuation removed, minimum sentence length = 5)

3.4 Information Ordering

For the development of this system, we implemented four methods IO. *Position ordering* is implemented only within the frame of TF-IDF model; *majority ordering* is implemented in ILP model; IO with *cosine similarity* and IO with *jaccard similarity* are used in the outcome of both models. The following subsections describe the IO methods we implemented.

3.4.1 Position Ordering

Position ordering is implemented within the TF-IDF model. Each sentence in the summary is hashed with its index in the original article. The position of the sentence is the ratio of the index(s_i) to the total number of sentences (n) in the article.

$$p_s = s_i/n \quad (9)$$

The results of position ordering is the sorted outcome of the positions of selected sentences in their original article.

3.4.2 Majority Ordering

The idea of majority ordering is to classify sentences into themes before content selection and to create a final order of the themes based on the order of those themes in the original articles. This IO method consists of two steps: (1) classifying sentences into themes by clustering and (2) constructing a theme graph to get the final order of themes.

First, sentences under the same topic are classified into themes by using the k-means clustering algorithm implemented by the `sklearn` library. We clustered into 10 themes for the present implementation of majority ordering. The algorithm divides the total sentences into clusters by grouping the nearest data points together until the number of clusters reaches the designated number of clusters.

Once each sentence under a topic has it corresponding theme, we constructed a weighted directed graph. The weights on the arcs are represented by the number of input texts which have sentences from a theme occurring before other themes. For example, the arc weight from theme 1 to theme 2 is the number of sentences from theme 1 occurring before sentences from theme 2. On the other hand, the arc weight from theme 2 to theme 1 is the number of sentence from theme 2 occurring before sentences from theme 1. To decide the final order of the themes, we calculate the weight of all themes iteratively. In each iteration, the weight for each theme is obtained by subtracting the total incoming arc weights of the theme from its total outgoing arc weights. The most weighted theme in each iteration is the theme with the highest priority. This theme with the highest priority along with its relevant arcs are then removed from the graph, and the next iteration proceed. Therefore, once the iterations end, we get the final order of the themes in terms of the priority of occurrence.

This final order is then used to reorder the sentences in the output summaries. Sentences belongs to the themes with higher order are prioritized in the reordering process.

3.4.3 Cosine Similarity Ordering

For the selected sentences in a summary, we obtained permutations of these sentences to generate all possible orderings of a summary. Then, we calculated the cosine similarities between consecutive sentences. To calculate cosine similarities, we first remove stop-words in sentences, then, we use sentence-bert to covert sentences to vectors, and compute cosine similarities between two vectors. The score of each ordering is the sum of the added cosine similarities. The ordering with the highest score is the output of this method. Suppose the original summary contains n sentences. There would be n! permutations. Calculating the scores of all possible ordering is an NP hard problem. However, since there is a word limit of 100 words, the original summaries usually contain 3-6 sentences, which makes the permutation number small, so exhaustive search is feasible.

3.4.4 Jaccard Similarity Ordering

Much similar to cosine similarity, we looked at the sum of each order, with similarity between sentences represented by the ratio of the intersection of two neighbour sentences to the their union, and output the ordering with the highest score. Stopwords are also removed before the calculation of Jaccard simi

3.5 Content Realization

For this deliverable, we focused on content selection and information ordering. No content realization methods are implemented for our current system.

4 Results

In this section, we present the results for the summaries generated with the content selection and IO methods used in this system. We also show the results of the survey we conducted for deciding which IO method to use in our current system.

4.1 Content Selection

Our summarization system is evaluated on the Recall-Oriented Understudy for Gisting Evaluation metrics (Lin, 2004, ROUGE) with the ROUGE python package.[6] 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

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

process are: recall, precision, and F1 scores.[7] The results of the two content selection methods are presented respectively in the following.

N	Metric	TF-IDF	ILP
1	Recall	0.31028	**0.34492**
	Precision	0.37183	**0.38148**
	F1-score	0.33793	**0.36202**
2	Recall	0.07246	**0.09455**
	Precision	0.08654	**0.10488**
	F1-score	0.0788	**0.09936**

Table 4: ROUGE-N scores of the methods used in the current summarization system

As shown in Table 4, The ILP model outperforms the TF-IDF model in ROUGE-1 and ROUGE-2 score. However, both models only capture around a third of the unigram content occurring in the human summaries.

The recall scores of ILP model are 0.34492 and 0.09455 for measuring the unigrams and bigrams across the system-generated summaries and the human summaries, showing that this method captures around 34.5% of the unigrams and 9.5% of the bigrams occurring in the human summaries. The precision scores of ILP model are 0.38148 and 0.10488, showing that around 38% of the unigrams and 10.5% of the bigrams in the system-generated summaries are consistent to the human summaries.

This is an improvement from our previous system's performance from D3, which can be seen in Table 4. Both D4 and D3 results improve upon our baseline system which selects the the first sentences up to 100 whitespace delimited tokens from a random article in the input set. Baseline results can be seen in Table 6

N	Metric	TF-IDF	ILP
1	Recall	**0.30895**	0.30750
	Precision	**0.36750**	0.35340
	F1-score	**0.33475**	0.32734
2	Recall	0.07160	**0.07539**
	Precision	0.08529	**0.08723**
	F1-score	0.07761	**0.08049**

Table 5: ROUGE-N scores of the methods used in the previous D3 summarization system

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

N	Metric	Baseline
1	Recall	0.23824
	Precision	0.33036
	F1-score	0.27436
2	Recall	0.05136
	Precision	0.07092
	F1-score	0.05906

Table 6: ROUGE-N scores of the baseline system

4.2 Information Ordering

To decide the final IO methods used in our system, we conducted a survey. In the survey, we asked 14 human raters to rate 5 set of summaries for each content selection method (i.e., TF-IDF and ILP), 10 in total. Each set contains four summaries produced by different ordering methods. Specifically, for summaries generated with TF-IDF, the human raters were asked to choose the best one from summaries produced with the following IO methods: (1) no ordering (2) cosine similarity (3) jaccard similarity, and (4) position ordering. For summaries generated with ILP, the human raters were asked to choose the best one from summaries produced with the IO methods: (1) no ordering (2) cosine similarity (3) jaccard similarity, and (4) majority ordering. The results of the 14 responses are presented in table 7

	Method	Ave Vote	Best vote
TF-IDF	No IO	12.84	0
	PO	**45.72**	4/5
	Cos_simi	17.12	0
	Jac_simi	24.26	1/5
ILP	No IO	25.72	1/5
	MO	28.56	2/5
	Cos_simi	**32.82**	2/5
	Jac_simi	12.86	0

Table 7: Human rating result

For the TF-IDF model, position order has the highest average vote of 45.72%; out of the 5 sets, it produced 4 best summaries according to the response. For the ILP model, cosine similarity has the highest average vote of 32.82%; out of the 5 sets, it produced 2 best summaries, the same as majority ordering.

Looking at the performance across two models, cosine similarity ordering performed well with the results produced by the ILP model, whereas it has the lowest performance with the results produced

by the TF-IDF model.

5 Discussions

In this section, we provide analyses for the results of our current system.

5.1 Information Ordering

For TF-IDF model, position ordering has the highest human voting. We first assumed that it was due to the fact that many sentences in the summary were from one article, causing positioning to be the most effective ordering feature. However, when looking at the example where position ordering has the highest voting percentage, we found that five sentences in the summary were from four different articles. Hence, another explanation is needed to account for the results. After examine the data and the TF-IDF model, we conclude two reasons why position ordering significantly ourperform ordering with cosine similarity and jaccard similarity.

First, the previously mentioned four articles that produced five sentences roughly follow the same writing style and logic. Majority of them start out by the discovery of trace of pandas, then segue into a general discussion of pandas and their endangered situation, and end with the nation's protective measure to save them. Therefore, despite being drawn from different articles, the position of the sentence is still a strong indicator of its sequence in the summary. Second, since we have already used cosine similarity to remove similar sentences in the summary, the remaining ones do have have many words in common. This will cause jaccard similarity and cosine similarity between sentences to be very low, making ordering with these two methods not as informative as position ordering.

When applying information ordering to the ILP selected sentences, we saw a much more uniform distribution of which Information Ordering method was preferred on average (as compared to the TF-IDF selected sentences). We can infer some reasoning behind this. In the ILP objective function, each concept, regardless of frequency in the selected sentences, is only counted once towards the weighted sum. Thus, having repeated concepts across sentences, while certainly acceptable, is not ultimately beneficial to the ILP solution. Perhaps, this leads sentences to be quite conceptually distinct, and it is hard for sentences to flow smoothly together, regardless of the Information Ordering method chosen. If this is the case, we would expect a uniform distribution from averaged rater evaluations, which is exactly what we see.

Overall, the strength, or lack thereof, of content selection can impact the fluidity of a certain Information Ordering method as well. A recipe is only as good as the ingredients used. It is possible that Content Selection method acts as a confounding variable when interpreting Information Ordering method ratings.

Lastly, our rater sample is small at 14 people, and many survey items had results for each ordering option. While we are able to to see a majority choice for each survey option, it's important to note that proper ordering is a subjective choice that varies from human to human.

6 Conclusion

We have implemented and evaluated two methods for Content Selection: TF-IDF and Integer Linear Programming. Our current results show that ILP has the best performance for returning unigrams and bigrams that appear in human generated summaries. Additionally, our current results improve upon our baseline and our previous deliverables. Going forward, we will continue to shape our Content Selection to be stronger, keeping topic-focused summarization in mind.

We also implemented four different versions of Information Ordering methods. Through evaluation via a small sample of human raters, we learned that Position Ordering works best for TF-IDF and Cosine Similarity Ordering works best for ILP.

The last component of our system will tackle Content Realization in D5.

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.

Mudasir Mohd, Rafiya Jan, and Muzaffar Shah. 2020. Text document summarization using word embedding. *Expert Systems with Applications*, 143:112958.

Joel Neto, Alex Freitas, and Celso Kaestner. 2002. Automatic text summarization using a machine learning approach. volume 2507, pages 205–215.

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.

Senthamizh Selvan .R and K. Arutchelvan. 2022. Automatic text summarization using document clustering named entity recognition. *International Journal of Advanced Computer Science and Applications*, 13.

Simone Teufel and Marc Moens. 2002. Summarizing scientific articles: Experiments with relevance and rhetorical status. *Comput. Linguist.*, 28(4):409–445.

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 among everyone.

For D4: Rachel and Yi-Chien improved ILP method and implemented majority ordering within the ILP frame; Chenxi and Tashi improved TF-IDF method and implemented position ordering within the TF-IDF frame; Yian implemented Cosine similarity and Jaccard similarity ordering. Report, slides, and survey were contributed to 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 uni/bi/skipgramming
- `spaCy` for Named Entity Recognition
- `os` for system operation on Patas
- `PuLP` for Linear Programming
- `GLPK` as the ILP solver
- `TfidfVectorizer` from `sklearn` for TF-IDF matrix
- `KMeans` from `sklearn` for clustering sentences into themes for majority ordering
- `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

Problem 5:
Description: comparing cosine similarities between sentences for ILP would become a quadratic programming task.
Solution: Implement theme constraint that leverages clustering instead.