Final Project

"Opinosis: A Graph-Based Approach to Abstractive Summarization of Highly Redundant Opinions"

B07901093　EE3　蘇磐洋

B07901026　EE3　吳禎軒

工作分配：

蘇磐洋：讀論文、報告的 section1-6、程式架構設計、程式細部設計及 debug

吳禎軒：讀論文、報告的 section5、程式主體撰寫、程式細部設計及 debug

1. Abstract

We would like to implement the method in "Opinosis: A Graph-Based Approach to Abstractive Summarization of Highly Redundant Opinions", International Conference on Computational Linguistics, 2010. This is one of the references in chapter 11. We will refer to this paper as "this paper" and other references as [1], [2], ….

While reading and implementing this paper, we have gained a lot of insights. Therefore, in the final project, we not only write a program but also the literature survey.

The report is organized as follows. First, we will give the motivation for choosing this topic. Second, we will give an introduction to this paper and the concepts that are related to the course. Third, we will describe our program. Fourth, we will state some problems we have encountered during the final project. Finally, we draw our conclusions in the end.

2. Motivation

In the Algorithm course, we learned the basic ideas of graph theory. In the Introduction to Digital Speech Processing course, we found out that graph theory pervades chapter 11, in particular, abstractive summarization.

At first, we wanted to use simple graph construction methods and Dijkstra's algorithm to find the longest path, which is the combination of the most frequently used words in the graph. However, after second thoughts, we realized that this naïve method might induce some problems. For example, "I like this restaurant." may become "I don't like this restaurant." or the program may not be able to categorize "like," "enjoy," and "love" as in the same group. Therefore, we decided to look for a more robust yet not too complex graph-

based approach to abstractive summarization, and surprisingly, we found out a perfect one, which is our topic.

Although deep learning may have better performance over a lot of the traditional and delicate algorithms for abstractive summarization, we use traditional methods in that it gives us great insight into the concepts behind the architecture. (We are not sure if a paper from 2010 can be viewed as traditional.)

We are not pursuing high performance, but are trying to implement the various ideas in the course and this paper. Therefore, we chose to use Python instead of C++ due to the convenience of dealing with strings in Python.

3. Our findings in this paper

A lot of the content in this paper coincides with the material taught in class. We will go through this paper and point out the connections.

(1) Granularity

There are two types of summarizations: extractive summarization and abstractive summarization. As the title of this paper has shown, this paper uses abstractive summarization. If we construct the graph with each node denoting a “word” as this paper and the course, it is obvious that the output sentence may be different from every input sentence. (The left-hand graph is from chapter 11 and the right-hand one is from the paper.) However, as this paper has pointed out, their method is more extractive than abstractive in that it doesn’t use words that are not present in the data. Therefore, the paper suggests that their approach should be regarded as "word-level extractive summarization."

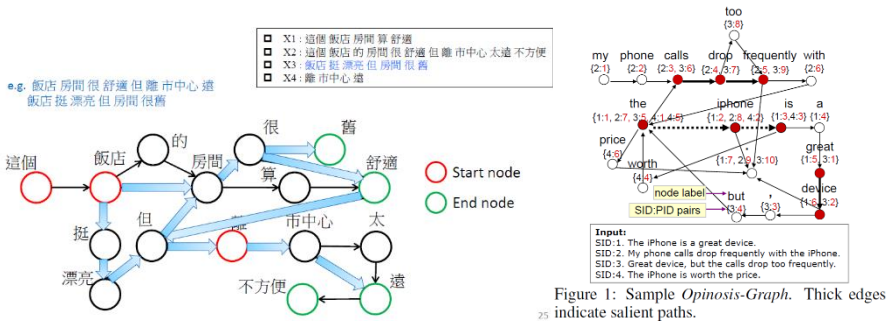


Figure 1: Sample *Opinosis-Graph*. Thick edges indicate salient paths.

As defined in [1], “extraction is the process of identifying important material in the text, abstraction the process of reformulating it in novel terms.” Thus, if we focus on "novel terms," the approach in this

paper is extractive. However, this paper indicates an interesting idea: granularity. In data mining, data granularity displays the trade-off between efficiency and efficacy. In acoustic modeling, the choice of a training unit displays the trade-off between accuracy, trainability, and generalizability as chapter 5 has shown.

In this paper, using a word as a unit seems like a good idea, but this choice limits their summarization to the level of n-gram but not a higher semantic level. The paper points out this defect in the conclusion. However, we should bear the concept of granularity in mind when reading the whole paper as the quality of the output sentence is highly dependent on the input data.

(2) Opinosis-Graph

The central idea in this paper is to use the structure of a graph to simulate the structure of language. The so-called Opinonsis-Graph only differs in the word graph in chapter 11 in the additional information associated with each node.

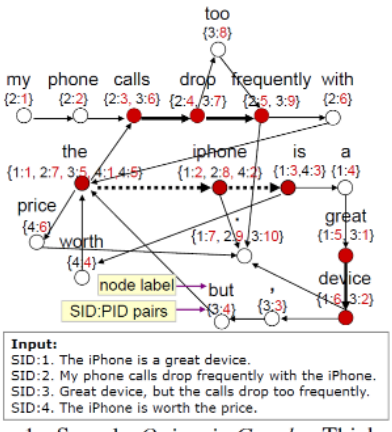


Figure 1: Sample *Opinosis-Graph*. Thick edges indicate salient paths.

Algorithm 1 (A1): $OpinosisGraph(Z)$

Input: Topic related sentences to be summarized: $Z = \{z_i\}_{i=1}^{n}$
Output: $G = (V, E)$
for $i = 1$ to n **do**
  $w \leftarrow Tokenize(z_i)$
  $sent\_size \leftarrow SizeOf(w)$
  for $j = 1$ to $sent\_size$ **do**
    $LABEL \leftarrow w_j$
    $PID \leftarrow j$
    $SID \leftarrow i$
    if $ExistsNode(G, LABEL)$ **then**
      $v_j \leftarrow GetExistingNode(G, LABEL)$
      $PRI_{v_j} \leftarrow PRI_{v_j} \cup (SID, PID)$
    else
      $v_j \leftarrow CreateNewNode(G, LABEL)$
      $PRI_{v_j} \leftarrow (SID, PID)$
    end if
    if not $ExistsEdge(v_{j-1} \rightarrow v_j, G)$ **then**
      $AddEdge(v_{j-1} \rightarrow v_j, G)$
    end if
  end for
end for

Algorithm 1 is the construction of an Opinosis-Graph. Each node is a word. Each node has some attributes. LABEL is the part-of-speech (POS) of the word. PRI is a set of two-tuples, (PID, SID). (PID, SID) = (i, j) means that the word occurs in the ith position of the jth sentence. Note that punctuations such as commas and periods are also represented as nodes, which are useful to indicate the end of a sentence.

The most difficult part is the POS. In general, POS tagging is context-dependent, requiring techniques such as conditional random field discussed in chapter 17 to analyze. However, in the

Opinosis-Graph, the POS of each word is unique, which is more similar to context-free grammar.

This approach leads to two issues. Is it reasonable? Can it be used in Chinese?

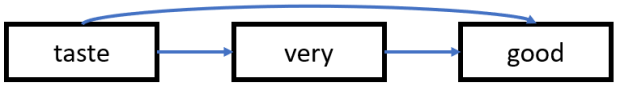
For the first question, we think that this method is not reasonable generally. However, we are dealing with opinions, reviews of a lot of people about a particular item. In this scenario, the POS of a word should be the same in the data. For example, if we are trying to summarize many reviews about a book, the word “book” is likely to be used as a noun instead of a verb such as “book a ticket”. Thus, we can conclude that when using this method on the summarization of reviews, the assumption that a word has a unique POS is reasonable. Even if a word has more than one POS in the graph, we can use more than one node to represent this word, treating the different POSs of the same word as different words.

For the second question, we think that it is difficult to use this method in Chinese directly. As chapter 5 has shown, when applying a word-based approach to Chinese, we need to have an accurate word segmentation program and a good POS annotation program. Chinese is a character-based language, but without word-level information, representing each character as a node may not be a good idea. Once again, we see the problem of granularity.

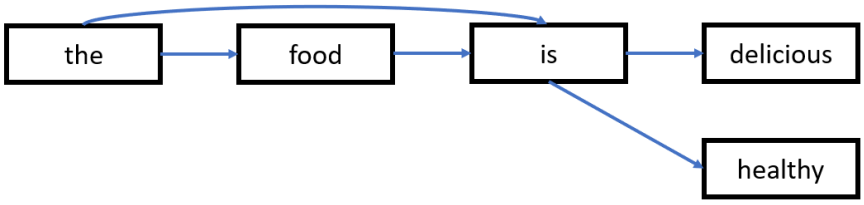
This graph has three properties: redundancy capture, gapped subsequence capture, and collapsible structures.

Redundancy is a measure of the information contained in the summary. We want the summary is concise but carries a lot of information. We call this “redundancy” and the higher the redundancy, the better. If a phrase appears in many sentences, the PRIs of the corresponding nodes contain a lot of (SID, PID) pairs. We can view these nodes as having heavy weights, thus capturing redundancy.

Gapped subsequence capture is a technique in extractive summarization. If we have two phrases: “taste good” and “taste very good,” the graph will be like the following. We can view “teste good” as a heavy path. When we allow for gapped subsequence in the summary, we will capture the more important parts of a sentence.



Collapsible structures help us combine different sentences into a new sentence. In this paper, their method only considers verbs to be candidates for collapse. Verbs such as “is” and “are” act like “hubs,” connecting to different adjectives. Take this graph as an example. We can combine them into one sentence: The food is delicious and healthy. The most difficult part is how to choose the conjunction, which will be discussed later.



(3) Valid Path

After the Opinosis-Graph is constructed, we are ready to find candidates for a summary. These candidates are called “valid path.”

A valid path should start with a “valid start node” and end with a “valid end node.” Also, between the start node and the end node, the whole sentence should follow some legitimate sentence structures.

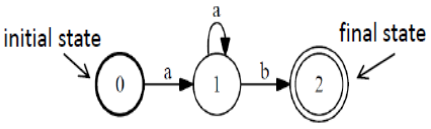
A node is a valid start node (VSN) if the average PID of the node is less than a threshold σ_{vsn}, meaning that, in average, the word appears in the first σ_{vsn} positions in a sentence.

A node is a valid end node (VEN) if it is a punctuation or coordinating conjunction. We can categorize some words as coordinating conjunctions based on the knowledge of the language.

Finally, we need to check whether the sentence is a legitimate one. This paper gives some examples, listed below. Finding a VSN requires no domain knowledge, and finding a VEN requires little domain knowledge. However, checking whether the sentence is legitimate is difficult.

We think that we can achieve this by the finite state machine in chapter 9. The initial state is a VSN and the final state is a VEN. If the sentence can go from the initial state to the final state, it is a valid sentence. This approach has a more flexible structure than that provided in this paper.

1. $.*(/nn)+.*(/vb)+.*(/jj)+.*$
2. $.*(/jj)+.*(/to)+.*(/vb).*$
3. $.*(/rb)*.*(/jj)+.*(/nn)+.*$
4. $.*(/rb)+.*(/in)+.*(/nn)+.*$



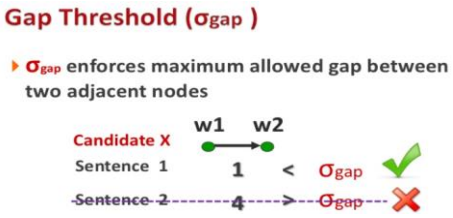
(4) Path Scoring

After finding a valid path, we need to give it a score. Later, we can use the scores to determine the best summary. As mentioned before, we want our summary to be concise but rich in information. Thus, “path redundancy” is a reasonable measure of the quality of a sentence.

First, we calculate the path redundancy r(q, s) of a sentence from node q to node s as the following formula from this paper. In [2], a simple example is given to illustrate the idea. If σ_{gap} = 2, we can calculate the overlapping sentences of w1 and w2. Both w1 and w2 all appear in sentence 1, and their positions in sentence 1 differ in one word. 1 < 2, so sentence 1 is an overlapping sentence of w1 and w2. Both w1 and w2 all appear in sentence 2, and their positions in sentence 2 differ in four words. 4 > 2, so sentence 2 is not an overlapping sentence of w1 and w2.

$$r(q,s) = n_q \bar{\cap} n_{q+1} ... \bar{\cap} n_s,$$

where $n_i = PRI_{v_i}$ *and* $\bar{\cap}$ *is the intersection between two sets of SIDs such that the difference between the corresponding PIDs is no greater than* σ_{gap}, *and* $\sigma_{gap} > 0$ *is a parameter.*



Then, based on the path redundancy, this paper gives several scoring functions for a sentence. The first one only uses the path redundancy to score a path. The second and the third ones consider the path redundancy weighted by the path length in linear scale and log scale, respectively.

The logarithm is an interesting idea. In a lot of scenarios, taking a logarithm can have better performance than not taking a logarithm. In digital speech recognition, since humans hear relative pitch but not absolute pitch, taking a logarithm in the processing of MFCC is reasonable. However, in fields other than digital speech recognition, why don’t we use other methods such as taking square roots? We think this is just a trade-off between convenience and accuracy. In this paper, the third method performs the best.

1. $S_{basic}(W) = \frac{1}{|W|} \sum_{k=i+1,i}^{s} r(i,k)$
2. $S_{wt\_len}(W) = \frac{1}{|W|} \sum_{k=i+1,i}^{s} |v_i, v_k| * r(i,k)$
3. $S_{wt\_loglen}(W) = \frac{1}{|W|}(r(i,i+1) + \sum_{k=i+2,i+1}^{s} log_2|v_i, v_k| * r(i,k))$

(5) Collapsed Paths

By now, we have valid paths and the scores associated with them. Nonetheless, to have a better summary, we need one more technique: “collapsed paths.”

In this paper, a node is collapsible only if its POS is a verb. The subsequence before the node is called “C_{anchor}” and the subsequences after the node are called “collapsed candidates” (CC). If we can combine one C_{anchor} and two or more CCs into a sentence, this sentence is called a “stitched sentence.” The path score of the stitched sentence is the average of the original distinct path scores.

This is an example given in this paper. The most difficult part is how to choose the correct connector. This paper suggests that we can use the same input data to determine whether “and clear” or “but clear” is better by comparing the path redundancies of the sentences including “and clear” or “but clear”.

We think that this method is weird because when we stitch two sentences, it is likely that the connectors barely exist in the input data. That is, we only have “The sound quality is really good” and “The sound quality is clear.” If we want to find sentences containing “and clear,” why don’t we just choose this kind of sentence that already has a connector in it as the output sentence? If this kind of sentence is few, deciding the connector using these sentences may be problematic due to data sparsity.

In the experiments tested by this paper, sentences with weird connectors are indeed a problem.

C_{anchor}	CC	Connector
a. the sound quality is	cc_1 : really good cc_2 : clear	and
b. the iphone is	cc_1 : great cc_2 : expensive	but

(6) Summarization Algorithm

Finally, after we have picked all the candidates and give each of them a score, we can rank the candidate and pick the top few paths. However, there is an important procedure: eliminate duplicated paths. Besides selecting the paths with the highest scores, we need to make sure that the similarity between them is small. We will investigate how to measure similarity later. Now, let’s take a look at the summarization algorithm in this paper.

Algorithm 2 (A2): $OpinosisSummarization(Z)$

1: **Input:** Topic related sentences to be summarized: $Z = \{z_i\}_{i=1}^{n}$
2: **Output:** $\mathcal{O}$ ={Opinosis Summaries}
3: $g \leftarrow OpinosisGraph(Z)$
4: $node\_size \leftarrow SizeOf(g)$
5: **for** $j = 1$ to $node\_size$ **do**
6: **if** $VSN(v_j)$ **then**
7: $pathLen \leftarrow 1$
8: $score \leftarrow 0$
9: $cList \leftarrow CreateNewList()$
10: **Traverse**$(cList, v_j, score, PRI_{v_j}, label_{v_j}, pathLen)$
11: $candidates \leftarrow \{candidates \cup cList\}$
12: **end if**
13: **end for**
14: $\mathcal{C} \leftarrow EliminateDuplicates(candidates)$
15: $\mathcal{C} \leftarrow SortByPathScore(\mathcal{C})$
16: **for** $i = 1$ to σ_{ss} **do**
17: $\mathcal{O} = \{\mathcal{O} \cup PickNextBestCandidate(\mathcal{C})\}$
18: **end for**

Initialization

Traverse each node

Generate summary

Algorithm 3 (A3): $Traverse(...)$

1: **Input:** $list, v_k \subseteq V, score, PRI_{overlap}, sentence, len$
2: **Output:** A set of candidate summaries
3: $redundancy \leftarrow SizeOf(PRI_{overlap})$
4: **if** $redundancy \geq \sigma_r$ **then**
5: **if** $VEN(v_k)$ **then**
6: **if** $ValidSentence(sentence)$ **then**
7: $finalScore \leftarrow \frac{score}{len}$
8: $AddCandidate(list, sentence, finalScore)$
9: **end if**
10: **end if**
11: **for** $v_n \in Neighbors_{v_k}$ **do**
12: $PRI_{new} \leftarrow PRI_{overlap} \ \bar{\cap} \ PRI_{v_n}$
13: $redundancy \leftarrow SizeOf(PRI_{new})$
14: $newSent \leftarrow Concat(sentence, label_{v_n})$
15: $L \leftarrow len + 1$
16: $newScore \leftarrow score + PathScore(redundancy, L)$
17: **if** $Collapsible(v_n)$ **then**
18: $C_{anchor} \leftarrow newSent$
19: $tmp \leftarrow CreateNewList()$
20: **for** $v_x \in Neighbors_{v_n}$ **do**
21: **Traverse**$(tmp, v_x, 0, PRI_{new}, label_{v_x}, L)$
22: $CC \leftarrow EliminateDuplicates(tmp)$
23: $CCPathScore \leftarrow AveragePathScore(CC)$
24: $finalScore \leftarrow newScore + CCPathScore$
25: $stitchedSent \leftarrow Stitch(C_{anchor}, CC)$
26: $AddCandidate(list, stitchedSent, finalScore)$
27: **end for**
28: **else**
29: **Traverse**$(list, v_n, newScore, PRI_{new}, newSent, L)$
30: **end if**
31: **end for**
32: **end if**

If VEN, output the path

If not, traverse all neighbors

If collapsible, stitch sentences

If not, no stitching

Algorithm 2 is the traversal of each node and will output the summary of the input data. The depth-first traversal algorithm is in Algorithm 3, which will take a node as input and output each valid

path starting from that node and the corresponding score associated with that path. Also, when encountering a collapsible node, Algorithm 3 will stitch the sentences starting from that node. The function of each section of the algorithm is annotated on the right-hand side of algorithms.

(7) Intricacies

Now, we want to discuss some intricacies in the algorithm. We will talk about how to eliminate duplicates and how to speed up the program.

This paper uses Jaccard to measure the similarity between two sentences. In [3], the Jaccard index of A and B is defined as the ratio between their intersection and their union. It is easy to calculate the Jaccard index of two sentences, but is this reasonable? We don't think so.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

First, a sentence is a sequential sequence, but the Jaccard index doesn't contain this kind of information. As chapter 11 has shown, we need positional encoding to consider the position information, and it is difficult to use positional encoding in the Jaccard index. However, maybe this won't be a big issue. Because each candidate sentence is a review of a particular item, the structure of them is likely to be similar. Therefore, it is more like the slot filling in chapter 14 than the sequence-to-sequence learning in chapter 11. Also, we can use dynamic time warping in chapter 8 to first match the two sentences, and then calculate their similarity. However, since we are only dealing with two short sentences, it may not be necessary to use dynamic time warping, which is better for two long sequences.

Second, in the Jaccard index, each word is equally important, but in reality, this is not the case. This can be easily solved by a weighted Jaccard index. The weight of each word can be determined by the corresponding POS. If the POS is an adjective, the weight should be higher. If the POS is punctuation, the weight should be lower. Moreover, if we can view the input data set as different documents, we can use term frequency and inverse document frequency in chapter 10 to determine the weight of each word. Based on this

weight, we can calculate the weighted Jaccard index.

Next, we want to consider how to speed up the program. This paper uses the technique of pruning to remove paths with scores lower than a threshold. Can we use the beam search or A* search in chapter 8? Also, the algorithm only checks whether each sentence is a valid one when a VEN is reached. Can we remove the invalid paths before a VEN is reached? There are a lot more ways to speed up the program. However, when we want to terminate the searching of some paths earlier, we need to be careful that these paths are truly hopeless.

(8) Experimental Results

In the end, this paper gives us the experimental results by using ROUGE to assess the performance. In [4], Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is proposed as a way to quantitatively measure the performance of a machine-generated summary. The idea is similar to the vector space model in chapter 10. ROUGE-N counts the overlapping N-grams between a machine-generated summary and a human-generated one. However, if there are no human-generated summaries to be compared with, ROUGE-N can't be used.

In this paper, the baseline is MEAD. In [5], a multi-document summarizer based on cluster centroids (MEAD) is proposed. In this paper, their approach has better precision and F-score but lower recall than the baseline. This is similar to the receiver operating characteristics curve in chapter 13.

We won't discuss the details of their experimental results. We will test the algorithm using our program and our dataset.

4. Our Program

Our program follows the algorithms in this paper. We import nltk to label POS of words.

We have five versions of the program with different parameters. The algorithms follow the structures of this paper with some modifications.

We also write another program, which uses our summarization algorithm and interpolation with the original data to determine whether a review is positive or negative.

Moreover, we have written an extractive version of our program. We have an “extractive_version.py” and an “extractive_version.ipynb.”This is the most successful one, and can be viewed as our final version.

The py file can be run by "python extractive_version.py input_file number_of_summary(1 or 2)." For example, we test our program with “11_9.txt” and the result is as follows.

```
(base) C:\Users\su350\Desktop>python extractive_version.py 11_9.txt 1
python extractive_version.py input_file number_of_summary(1 or 2)
11_9.txt
summary1:  Characters were very convincing and felt like you could understand their feelings .

(base) C:\Users\su350\Desktop>python extractive_version.py 11_9.txt 2
python extractive_version.py input_file number_of_summary(1 or 2)
11_9.txt
summary1:  Characters were very convincing and felt like you could understand their feelings .
summary2:  The absence of violence and porno sex was refreshing .
```

5. Problems

We have encountered several problems when implementing this paper. We will give the problems we encountered and how we solved them in the following. We use the Large Movie Review Dataset v1.0 as our testing dataset, which is provided by [6] and can be downloaded on [7]. The data for a dictionary including the positive and negative lexicons are provided by [8]. Our testing data and results are all included in the folder.

(1) Cycles

After testing the three simple sentences in this paper, we tested our program on the database. We input a file to our program and our program didn’t terminate. We think that maybe the path went through too many nodes before reaching a VEN, so we added a threshold to our traverse function. When the path length is greater than 10 (10 can be changed into other numbers), stop traversal. Also, we modify our codes to deal with some extreme situations such as 1/0 = inf.

We run our program again. It gave this summary. How unbelievable! The affection is so strong, but it is not concise at all.

```
summary:  unbelievable how unbelievable how unbelievable how unbelievable all the music is
```

We read the original input and found out the sentence where “unbelievable” was from. Interestingly, “unbelievable” and “how” will form a cycle in the graph. We can use cycle-breaking techniques

to deal with it. However, how to break cycles in a weighted directed graph, the famous minimum feedback arc set problem, is an NP-hard problem.

It's unbelievable how unbelievable all the lines in the movie sound.

Therefore, we use an easier method that limits that an adjective or adverb can only occur in a sentence once. The output is the following. The reason that it doesn't seem like a complete sentence is that we haven't figure out how to construct a legitimate sentence.

the 1 it a 3 instead of the 1 it deserves

(2) Valid paths

So, the next step is to define a function that can decide whether a sentence is valid or not. We use two methods. The first one is naïve, simply comparing if the bigrams are legitimate. We define the following dictionary, which is the left-hand figure, to determine the legitimacy of a bi-gram. We also use another method, which is based on the concept of a finite state machine. The first kind o valid sentence is displayed on the right-hand figure.

```
prohibit = {'VERB':['VERB'],        #We check t
    'NOUN':['ADJ','PRON','NOUN'],
    'ADJ':['VERB','PRON','ADV'],
    'ADV':['ADJ','NOUN','PRON','ADV'],
    'PRON':['ADJ','PRON','NOUN'], #I,she
    'CONJ':['CONJ','PRT','ADP'], #and
    'PRT':['CONJ','PRT','ADP'], # 's
    '.':['VERB','.','ADJ','PRON','NOUN','ADV'
    'ADP':['CONJ','PRT','ADP'], #prep+介詞
    'NUM':[],
    'DET':['CONJ','PRT','ADP'], #to
    'X':[]
  }
```

```
#the first kind of valid sentence
state = 0
for i in sent:
    if state == 0 and i[1] == 'NOUN':
        state = 1
    if state == 1 and i[1] == 'VERB':
        state = 2
    if state == 2 and i[1] == 'ADJ':
        state = 3
if state == 3:
    return True
```

We have found that the two methods are not enough. We think that we should have a large corpus instead of some simple rules. Since the size of the input data is small, relying on the structure of the input data and some simple rules is far from enough. Therefore, we later use extractive summarization to deal with the problem of data sparsity.

(3) Repetitiveness Between Two Summaries

In our first generation, two summaries in the two cases are similar. The main reason is that we only delete the "identical" duplicates within the candidates. Thus, we try to implement

duplicate removal according to Jaccard similarity. If two sentences have Jaccard similarity that is higher than a threshold, then the one with a higher path score is kept. Notice that selecting an appropriate threshold is quite relevant to the results.

```
05_10.txt
summary1:  In the middle of the score of the positive side regarding
summary2:  Of the middle of the score of the positive side regarding
```

(4) Fragmentation

In the third generation, many summaries are composed of meaningless and incomplete sentence fragments. The reason is that we set VEN incorrectly. We have tried to set different VEN, such as punctuations and conjunctions.

(5) Weird Punctuations

Sometimes, the punctuations are weird. We think that this problem is also caused by the setting of VEN.

```
16_9.txt
summary1:  Is this , when a short is this good , who 's
summary2:  A bit of true whodunit mystery in this , to meet
```

There are a lot more problems. The use of extractive summarization can solve all of them. That's the reason why we chose to implement the extractive version of this paper as well as the abstractive version.

6. Conclusions

We have gone through a lot of try and error during the final project. We found out that theory and implementation are far away. The concepts in this paper are simple and reasonable, but we are stuck in the construction of a valid sentence.

When using extractive summarization, the summary looks good. However, it is a pity that we lose a lot of beautiful ideas from abstractive summarization, such as stitched sentences. This a trade-off.

In general, we think that we need a more complete language model and a larger dataset to complete the abstractive summarization. As for the extractive summarization, we think maybe we can test it with

more data to see if it gives a concise and relevant summary.

The digital speech recognition course has come to an end, but we believe that, in the future, we will surely use a lot of the concepts from this course.

7. References

[1] Radev, Dragomir R., Eduard Hovy, and Kathleen McKeown. 2002. Introduction to the special issue on summarization

[2] https://kavita-ganesan.com/opinosis/#.YATwY-gzY2x

[3] https://en.wikipedia.org/wiki/Jaccard_index

[4] Lin, Chin-Yew. 2004b. Rouge: a package for automatic evaluation of summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004), Barcelona, Spain.

[5] Radev, Dragomir, Hongyan Jing, and Malgorzata Budzikowska. 2000. Centroid-based summarization of multiple documents: Sentence extraction, utility-based evaluation, and user studies. In In ANLP/NAACL Workshop on Summarization, pages 21–29.

[6] Potts, Christopher. 2011. On the negativity of negation. In Nan Li and David Lutz, eds., Proceedings of Semantics and Linguistic Theory 20, 636-659.

[7] http://ai.stanford.edu/~amaas/data/sentiment/

[8] Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews."; Proceedings of the ACM SIGKDD International Conference on Knowledge; Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, Washington, USA