Natural Language Processing Homework1 Report

1. Code Modular

   The code contains four functions:

   1) splitToPara:

      This function receive an article as a parameter, and split it into paragraphs.

      Since between 2 paragraphs, there must be\t or \n or multiple of them, so we could easily split the article to paragraphs by the regular expression '[\t\n]*' .

      ```
      def splitToPara(article):
          return re.split(r'[\t\n]+', article)
      ```

   2) splitToSentence

      This function receives a paragraph as a parameter, and split it into paragraphs. Since every sentence should end with a "." or "?" or "!", the regular expression for this is [.?!] .

      Some sentences is quoted, and should have a quotation mark at the end. Some sentences is enclosed in () as annotation, so there should be a bracket at the end. To meet these 2 conditions, add ["）]* in the regular expression.

      Every sentence should be followed by a whitespace or a Newline character, so we use [\s]+ to match this. Adding the [\s]+ can also help to separate the sentence from initials like B.P.M., since the first 2 periods in this initial don't have a whitespace or a newline character after the period.

      But there're still problems. Phrases like "Dr. Who" have a period followed by a whitespace; this can be considered as a sentence splitter. So we use regex *(?<!Dr)(?<!Ms)(?<!Mr)* to avoid splitting at Dr. Mr. Ms. Miss., and *(?<![A-Z].[A-Z])* to avoid splitting at the end of initials like B.P.M. or D.J.

      One more problem is the initials like B.P.M. could be at the end of an sentence with no additional period. I add a "or" regex at the end *|[A-Z].[A-Z].[\s]+[A-Z][a-z]* to check if the next character after *[A-Z].[A-Z].[\s]* is *[A-Z][a-z]*,if so, the initials are at the end of an sentence thus should be split here.

      ```
      def splitToSentence(para):
           return re.split(r'(?<!Dr)(?<!Ms)(?<!Mr)(?<!Miss)(?<![A-Z].[A-
           Z])[.?!]["）]*[\s]+|[A-Z].[A-Z].[\s]+[A-Z][a-z]', para)
      ```

   3) splitToToken

      This function receive a sentence as a parameter and split it into tokens. We could use whitespace to split the tokens, but after tokenized, there may be punctuations grouped with letters. We use *'["):,;-]*[\s]*["):,;-]*[.?!]*[\s]+[-"(]*'* to avoid common punctuations grouping with letters. But there still may be some tokens listed as words grouped with punctuations. Since we only need to calculate the number of tokens, so the punctuations wouldn’t affect the result. In further development, we should consider how to find the tokens without punctuation attached to them.

```
def splitToToken(sentence):
        return re.split(r'["):,;-]*[\s]*["):,;-]*[.?!]*[\s]+[-
"(]*',sentence)
```

4) AnalyzeArticle

This function calls the 3 functions above to analysis an article, and prints out the result.

We could easily use this function to analyze any article in the future. If we only want to analyze a sentence or a paragraph, we could use the 3 functions above. By separating these functions, we could easily call any of them to meet our needs in the future. This structure is also more convenient and clearer when we need to update functions.

2. Results

After running my code on the articles, I get the results:

Development article:

8 paragraphs, 21 sentences and 646 tokens.

3. How to evaluate this kind of system

1) Right Result.

The first thing to evaluate is if this system can get us the right result.

The right result requires the system to deal with ambiguous situations like we use period in the initials instead of at the end of a sentence.

There’re lots of this kind of ambiguous situations in English, a good system should have the ability to handle the common ones.

2) Modular

A good system should have a well-designed structure. With a better structure, the system could have scalability, portability and maintainability, and could also be easy to test.

3) Efficiency

A good system should analysis fast, especially when the article is very long, this requires a good algorithm. Also, it shouldn’t use too much storage.

4) Function Variety

It’s better to have more functions to meet the users’ need. Like supporting multiple language or error correction.