计算机学院 社交媒体与舆情分析 课程实验报告

实验题目：2 Auto Summary		学号：201800130112
日期：2021.10.20	班级： 计科 18.3	姓名： 赵子涵
Email：zih_an@163.com		

实验方法介绍：

摘要生成的三种方式，本实验实现抽取式 TextRank，了解生成式：

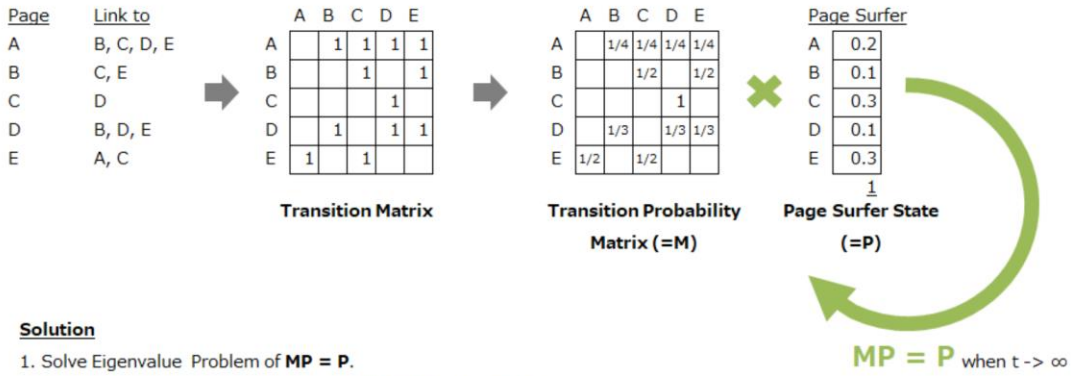
- 抽取式：TextRank
- 压缩式
- 生成式：Encoder-Decoder

一、TextRank

TextRank 算法基于 Google 的 PageRank 算法，将其应用在文本上。通过句子之间的关系（权重）计算出最重要的句子为该文本的摘要，本质是被指向的次数越多越重要，算法如下所示：

TextRank

TextRank is based on PageRank algorithm that is used on Google Search Engine. Its base concept is "The linked page is good, much more if it from many linked page". The links between the pages are expressed by matrix (like Round-robin table). We can convert this matrix to transition probability matrix by dividing the sum of links in each page. And the page surfer moves the page according to this matrix.



Solution

1. Solve Eigenvalue Problem of **MP = P**.
2. Repeat the transition until convergence(**MP - P < threshold**).

$$P_i' = (1-d) + d * M_i^T P_i \qquad \sum(P_i' - P_i) < threshold$$

*The page surfers randomly click the page with a probability of **1-d**. (d = usually 0.85)*

Page Rank Algorithm

TextRank regards words or sentences as pages on the PageRank. So when you use the TextRank, following points are important.

- Define the "text units" and add them as the nodes in the graph.
- Define the "relation" between the text units and add them as the edges in the graph.
  - You can set the weight of the edge also.

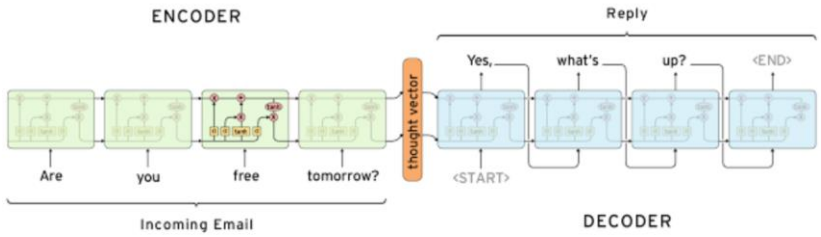
Then, solve the graph by PageRank algorithm. LexRank uses the sentence as node and the similarity as relation/weight (similarity is calculated by IDF-modified Cosine similarity).

If you want to use TextRank, following tools support TextRank.

二、Encoder-Decoder

Encoder-Decoder Model

The encoder-decoder model is composed of encoder and decoder like its name. The encoder converts an input document to a latent representation (vector), and the decoder generates a summary by using it.



实验过程描述：（不要求罗列完整源代码）

一、加载数据（根据实验指导代码）

数据描述：

- Story：文本
- Highlights：该文本对应的摘要

二、分词、分句工具

使用 nltk.tokenize 工具进行分句操作：

```
import numpy as np
from nltk.tokenize import sent_tokenize
from nltk.tokenize import word_tokenize
# import nltk
# nltk.download('punkt')

text = "God is Great! I won a lottery."
print(sent_tokenize(text))
print(word_tokenize(text))
```

```
['God is Great!', 'I won a lottery.']
['God', 'is', 'Great', '!', 'I', 'won', 'a', 'lottery', '.']
```

三、TextRank

1. 句子之间的相似性矩阵计算公式：

$$Similarity(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \& w_k \in S_j\}|}{log(|S_i|)+log(|S_j|)}$$

编写代码时需注意 log(1)=0 导致分母为 0 的除 0 异常。

2. 运行 PageRank 的算法（算法如实验方法部分的图示，具体实现如下）

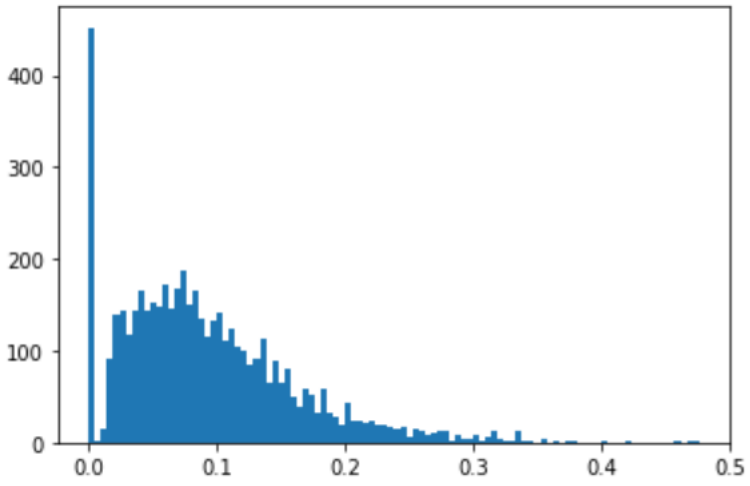
其中，对每个文档都做摘要+evaluation，将分数取平均值作为 TextRank 算法总体评估的 score，运行算法的循环如下：

```
from sumeval.metrics.rouge import RougeCalculator
rouge = RougeCalculator(stopwords=True, lang="en")

cnt = 0
# rouge2_sum, rougel_sum = [], []
rougel_sum = []
for story_dict in stories:
    if cnt % 200 == 0:
        print(cnt)
    cnt += 1
    story = story_dict["story"]
    highlight = story_dict["highlights"]
    tr = TextRank()
#     tr.run(story)
    chk = tr.initMatrix(story)
    if not chk:
        continue
    tr.calRank()

    summary = tr.getSummary(1)
#     rouge_2 = rouge.rouge_n(summary=summary[0], references=highlight, n=1)
    rouge_1 = rouge.rouge_l(summary=summary[0], references=highlight)
#     rouge2_sum.append(rouge_2)
    rougel_sum.append(rouge_1)
```

- **评估的分数如图：**

  直方图统计：

```
import matplotlib.pyplot as plt
plt.hist(rougel_sum, bins=100)
plt.show()
```



最小/最大值与平均值：

```
print("rouge_1 average: ", np.sum(rougel_sum) / cnt)
print("rouge_1 max: ", np.max(rougel_sum))
print("rouge_1 min: ", np.min(rougel_sum))
```

```
rouge_1 average:  0.09151555035403915
rouge_1 max:  0.47619047619047616
rouge_1 min:  0.0
```

- **TextRank 具体实现：**

```
class TextRank():
    def initMatrix(self, text, simThreshold=1e-4):
        self.sentence_list = sent_tokenize(text)
        self.matSize = len(self.sentence_list)
        if self.matSize <= 0:
            return False

        self.simMatrix = np.zeros((self.matSize, self.matSize), dtype=np.float64)
        for i in range(self.matSize):
            for j in range(self.matSize):
                if i==j:
                    continue
                self.simMatrix[i][j] = self.__calSim(self.sentence_list[i], self.sentence_list[j], simThreshold)
        self.weightSum = np.sum(np.sum(self.simMatrix))
        return True

    def calRank(self, threshold=1e-7, d=0.85):
        self.pg = np.random.rand(self.matSize, 1)
        self.pg /= np.linalg.norm(self.pg, 1)
        while True:
            pg_hat = (1-d) + d * np.dot(self.simMatrix.T, self.pg) / self.weightSum  ## 注意除以权重和weightSum
            if np.sum(np.abs(pg_hat-self.pg)) < threshold:
                break
            self.pg = pg_hat

    def run(self, text):
        self.initMatrix(text)
        self.calRank()

    def getSummary(self, top_n=3):
        idx = np.argsort(tr.pg, axis=0)
        idx = idx[-top_n:]
        idx = [item[0] for item in idx]
        res = []
#         print(idx)
#         print(len(self.sentence_list))
        for i in range(top_n):
            j = top_n - i - 1
            res.append(self.sentence_list[idx[j]])
        return res

    def __calSim(self, senA, senB, threshold):
        lenA, lenB = len(senA), len(senB)
        if lenA <= 1 and lenB <= 1:
            return 0
        intersection_set = set(word_tokenize(senA)) & set(word_tokenize(senB))
        sim = len(intersection_set) / (np.log(lenA) + np.log(lenB))
        if sim >= threshold:
            return sim
        return 0
```

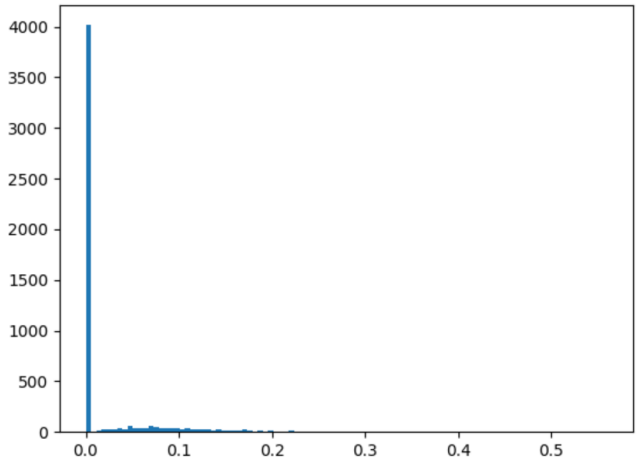
四、运行及测试验证

- Python 现成的生成摘要的工具：gensim 的 summarize。

  注：最新版的 gensim 因为摘要效果不好，已经移除了 summarize，需要使用 3.8.3 版本的 gensim 才有生成文本摘要的功能。

  代码运行结果（完整代码参见附录）：

```
rouge_1 average:  0.020202333962802594
rouge_1 max:  0.56
rouge_1 min:  0.0
```



- Python 验证正确率的工具说明：sumeval
  使用 from sumeval.metrics.rouge import RougeCalculator 进行验证：
  本实验均选择 rouge_l 计算

结论分析：

根据 PageRank 算法必定会收敛，因此必定会有解，不会出现死循环的情况。

运行手写的 TextRank，使用 sumeval 进行评估，发现分数并不高；同样，使用封装好的 gensim 进行自动摘要，分数也不高，甚至比自己手动实现的 TextRank 还要差。

因此，可以看出，抽取式生成摘要的 TextRank 效果并不是很理想。经查阅，gensim 库在 4 版本及之后（最新）已将 summarize 移除，理由为：因效果不好并不常用。

但是，gensim 做摘要的速度比手动实现的 TextRank 要快得多。

结论：

TextRank 抽取式的文本摘要一定能输出结果，但是并不理想，与人工专家的摘要仍相差较多。

核心代码

- **手动实现 TextRank：**

```
class TextRank():
    def initMatrix(self, text, simThreshold=1e-3):
        self.sentence_list = sent_tokenize(text)
        self.matSize = len(self.sentence_list)
        self.simMatrix = np.zeros((self.matSize, self.matSize), dtype=np.float64)
        for i in range(self.matSize):
            for j in range(self.matSize):
                if i==j:
                    continue
```

```
                    self.simMatrix[i][j] = self.__calSim(self.sentence_list[i], self.sentence_list[j],
simThreshold)
        self.weightSum = np.sum(np.sum(self.simMatrix))

    def calRank(self, threshold=1e-7, d=0.85):
        self.pg = np.random.rand(self.matSize, 1)
        self.pg /= np.linalg.norm(self.pg, 1)
        while True:
            pg_hat = (1-d) + d * np.dot(self.simMatrix.T, self.pg) / self.weightSum    ## 注意除以权重
和 weightSum
            if np.sum(np.abs(pg_hat-self.pg)) < threshold:
                break
            self.pg = pg_hat

    def run(self, text):
        self.initMatrix(text)
        self.calRank()

    def getSummary(self, top_n=3):
        if top_n > self.matSize:
            return []
        idx = np.argsort(tr.pg, axis=0)
        idx = idx[-top_n:]
        idx = [item[0] for item in idx]
        res = []
#           print(idx)
#           print(len(self.sentence_list))
        for i in range(top_n):
            j = top_n - i - 1
            res.append(self.sentence_list[idx[j]])
        return res

    def __calSim(self, senA, senB, threshold):
        lenA, lenB = len(senA), len(senB)
        intersection_set = set(word_tokenize(senA)) & set(word_tokenize(senB))
        sim = len(intersection_set) / (np.log(lenA) + np.log(lenB))
        if sim >= threshold:
            return sim
        return 0
```

```
# -*- coding: utf-8 -*-
from os import listdir
import tqdm
```

```
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, encoding='utf-8')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text
```

- **导入数据：**

```
# split a document into news story and highlights
def split_story(doc):
    # find first highlight
    index = doc.find('@highlight')
    # split into story and highlights
    story, highlights = doc[:index], doc[index:].split('@highlight')
    # strip extra white space around each highlight
    highlights = [h.strip() for h in highlights if len(h) > 0]
    return story, highlights


def load_stories(directory, num_stories=-1):
    """load stories

    Args:
        directory(str): the path of cnn_stories
        num_stories(int): NUM of stories to use

    Returns:
        all_stories(list): A list of dict, dict contains `story`(str) and `highlights`(a list of str)
    """
    all_stories = list()
    filenames = listdir(directory)
    if num_stories > -1:
        filenames = filenames[:num_stories]

    for name in tqdm.tqdm(filenames):
        filename = directory + '/' + name
        # load document
        doc = load_doc(filename)
        # split into story and highlights
        story, highlights = split_story(doc)
        # store
```

```
            all_stories.append({'story': story, 'highlights': highlights})

        return all_stories

# load stories
directory = 'data/cnn_stories_tokenized/'
stories = load_stories(directory, 10000)
print('Loaded Stories %d' % len(stories))

print(stories[4]['story'])
print(stories[4]['highlights'])
```

- 使用 gensim 生成摘要及评估的完整代码：

```
# -*- coding: utf-8 -*-
from os import listdir
import tqdm


# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, encoding='utf-8')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text


# split a document into news story and highlights
def split_story(doc):
    # find first highlight
    index = doc.find('@highlight')
    # split into story and highlights
    story, highlights = doc[:index], doc[index:].split('@highlight')
    # strip extra white space around each highlight
    highlights = [h.strip() for h in highlights if len(h) > 0]
    return story, highlights


def load_stories(directory, num_stories=-1):
    """load stories

    Args:
        directory(str): the path of cnn_stories
```

```
        num_stories(int): NUM of stories to use

    Returns:
        all_stories(list): A list of dict, dict contains `story`(str) and
`highlights`(a list of str)
    """
    all_stories = list()
    filenames = listdir(directory)
    if num_stories > -1:
        filenames = filenames[:num_stories]

    for name in tqdm.tqdm(filenames):
        filename = directory + '/' + name
        # load document
        doc = load_doc(filename)
        # split into story and highlights
        story, highlights = split_story(doc)
        # store
        all_stories.append({'story': story, 'highlights': highlights})

    return all_stories


from gensim.summarization.summarizer import summarize
from sumeval.metrics.rouge import RougeCalculator
import numpy as np
import matplotlib.pyplot as plt


def main():
    # load stories
    directory = 'data/cnn_stories_tokenized/'
    stories = load_stories(directory, 10000)
    print('Loaded Stories %d' % len(stories))
    rouge = RougeCalculator(stopwords=True, lang="en")

    cnt = 0
    rougel_sum = []
    for story_item in stories:
        story = story_item["story"]
        highlight = story_item["highlights"]
        gen_sum = summarize(story, ratio=0.02)
        rouge_1 = rouge.rouge_l(summary=gen_sum, references=highlight)
        rougel_sum.append(rouge_1)
```

```
        if cnt % 200 == 0:
            print(cnt, rouge_1)
        cnt += 1

    print("rouge_1 average: ", np.sum(rougel_sum) / cnt)
    print("rouge_1 max: ", np.max(rougel_sum))
    print("rouge_1 min: ", np.min(rougel_sum))
    ## 作图观察各个文本摘要的分数情况
    plt.hist(rougel_sum, bins=100)
    plt.show()

if __name__ == '__main__':
    main()
```