Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра интеллектуальных информационных технологий

ОТЧЁТ

по лабораторной работе №3 по дисциплине «ЕЯзИИС»

на тему: «Разработка системы автоматического реферирования документов»

Выполнили студенты группы 821701: Поживилко П.С.
Витушко Л. Д.

Проверил: Крапивин Ю.Б.

Минск, 2021

Цель работы: освоить на практике основные принципы автоматического реферирования документов с учетом функционала, предоставляемого технологией остис.

Задание

Вариант 1.

Язык текста: русский, английский.

Методика: Sentence Extraction + ML.

Реализуемый метод: научные статьи по computer science, сочинения по литературе.

Ход работы.

Описание системы.

Реферат в виде ключевых слов представляет собой список, возможно иерархический (в виде дерева) наиболее информативных слов и словосочетаний (именных групп) обрабатываемого документа. Для его реализации были подсчитаны частоты всех слов и были выведены 11 наиболее самых встречаемых слов.

Классический реферат – это набор наиболее информативных предложений текста, возможно трансформированных (удаление вводных конструкций, замена анафоричных местоимений и т.д. с целью улучшения связности реферата и уменьшения его объема).

Функции, характеризующие положение предложения в документе $\mathrm{Posd}(S_i)$ и положение в абзаце $\mathrm{Posp}(S_i)$:

$$\begin{aligned} \mathrm{Posd}(S_i) &= 1 - \frac{BD(S_i)}{|D|} \\ \mathrm{Posp}(S_i) &= 1 - \frac{BP(S_i)}{|P|}. \end{aligned}$$

где

|D|- число символов в документе D, содержащем предложение S_i;

$BD(S_i)$ – количество символов до S_i в $D(S_i)$;

|P| - количество символов в абзаце P, содержащем предложение S_i;

$BP(S_i)$ – количество символов до S_i в абзаце.

Модифицированная TFIDF функция:

$$\mathrm{Score}(S_i) = \sum_{t \in S_i} tf(t, S_i) \cdot w(t, D).$$

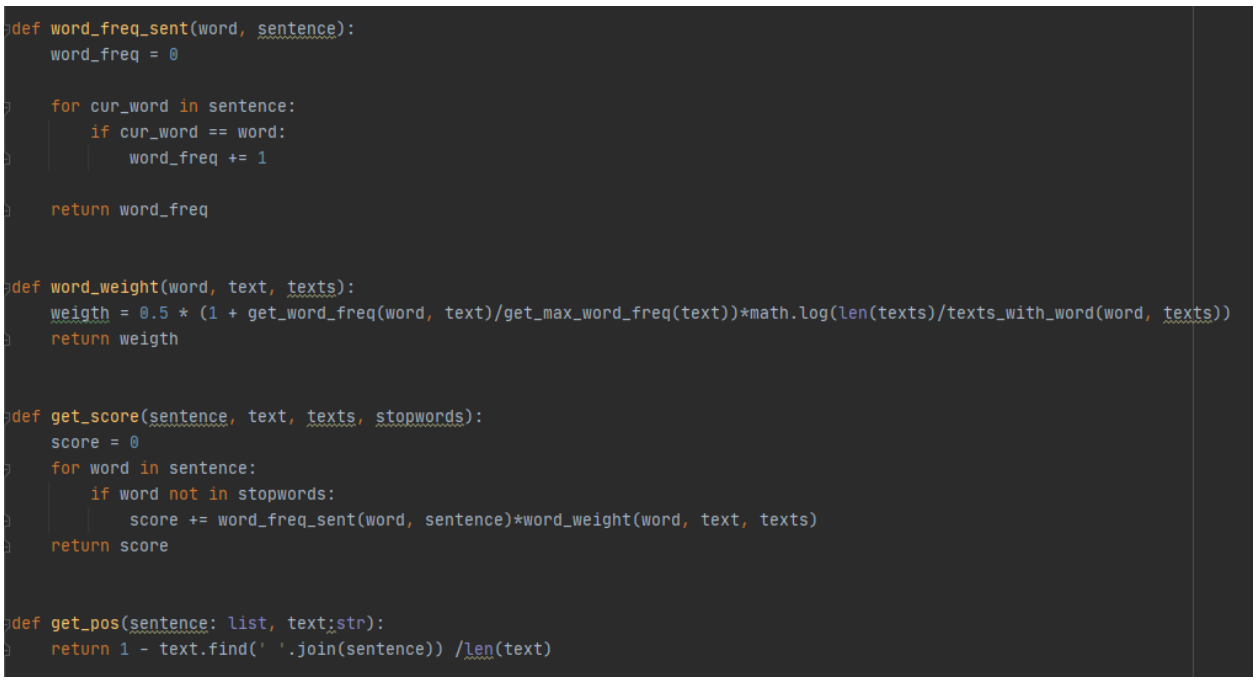
$tf(t, S_i)$ - частота термина t в предложении S_i;

$$w(t,D) = 0.5\left(1 + \frac{tf(t,D)}{tf_{max}(D)}\right) \cdot \log\left(\frac{|DB|}{df(t)}\right).$$

$tf(t,D)$ - частота термина t в документе D;
$df(t)$ - количество документов, с термином t;
$tf_{max}(D)$ - максимальная частота термина в документе D;
|DB| - количество документов.

```python
def generate_summary(file_name, texts, top_n=5):
    stop_words = stopwords.words('russian')
    summarize_text = []

    statistic = []

    # Step 1 - Read text anc split it
    sentences = read_article(file_name)
    print(sentences)

    text = ''
    for sentence in sentences:
        text += ' '.join(sentence) + '.'

    for sentence in sentences:
        statistic.append([sentence, get_score(sentence, sentences, texts, stop_words), get_pos(sentence, text)])

    statistic.sort(key=lambda x: x[1], reverse=True)
    most_weight = statistic[:10]
    most_weight.sort(key=lambda x: x[2], reverse=True)
    print(most_weight)

    text = ''
    for i in range(len(most_weight)):
        text += ' '.join(most_weight[i][0]) + '\n'

    print(text)
```

Рисунок 1. Метод sentence extraction



```
def word_freq_sent(word, sentence):
    word_freq = 0

    for cur_word in sentence:
        if cur_word == word:
            word_freq += 1

    return word_freq


def word_weight(word, text, texts):
    weigth = 0.5 * (1 + get_word_freq(word, text)/get_max_word_freq(text))*math.log(len(texts)/texts_with_word(word, texts))
    return weigth


def get_score(sentence, text, texts, stopwords):
    score = 0
    for word in sentence:
        if word not in stopwords:
            score += word_freq_sent(word, sentence)*word_weight(word, text, texts)
    return score


def get_pos(sentence: list, text:str):
    return 1 - text.find(' '.join(sentence)) /len(text)
```

Рисунок 2. Функции для подсчета веса предложения, веса слова и подсчета частоты встречаемости слова в предложении

```python
def get_word_freq(word, text):
    word_freq = 0

    for sentence in text:
        word_freq += word_freq_sent(word, sentence)

    return word_freq


def get_max_word_freq(text):
    words_freq = dict()
    max_freq = 0

    for sentence in text:
        for cur_word in sentence:
            if cur_word in words_freq:
                if words_freq[cur_word] > max_freq:
                    max_freq = words_freq[cur_word]
                words_freq[cur_word] += 1
            else:
                words_freq[cur_word] = 1

    return max_freq


def texts_with_word(word, texts):
    text_count = 0

    for text in texts:
        for sentence in text:
            # print(sentence)
            if word in sentence:
                text_count += 1
                break

    return text_count
```

Рисунок 3. Функции подсчета документов содержащих слово, подсчета частоты встречаемости слова в документе и подсчета максимальной частоты встречаемости слов в документе

В качестве тестовых данных использовалось краткое содержание романа "Война и Мир" в 3-х томах.

```
text_names = ["war_1_ru.txt", "war_2_ru.txt", "war_3_ru.txt"]
texts = []

for text_name in text_names:
    texts.append(read_article(text_name))

generate_summary("war_1_ru.txt", texts)
words = dict()
stop_words = stopwords.words('russian')

for sentence in texts[0]:
    for term in sentence:
        if term.lower() not in words and term.lower() not in stop_words and term != '':
            words[term] = get_word_freq(term, texts[0])

sorted_words = sorted(words.items(), key=lambda x:x[1], reverse=True)
for i in range(11):
    print(sorted_words[i])
```

Рисунок 4. Метод для реферата ключевых слов

```
import pysummarization
from pysummarization.nlpbase.auto_abstractor import AutoAbstractor
from pysummarization.tokenizabledoc.simple_tokenizer import SimpleTokenizer
from pysummarization.abstractabledoc.top_n_rank_abstractor import TopNRankAbstractor


f = open("war_1_ru.txt", "r", encoding='utf-8')
document = f.read()


auto_abstractor = AutoAbstractor()
auto_abstractor.tokenizable_doc = SimpleTokenizer()
auto_abstractor.delimiter_list = [".", "\n"]
abstractable_doc = TopNRankAbstractor()
result_dict = auto_abstractor.summarize(document, abstractable_doc)

for sentence in result_dict["summarize_result"]:
    print(sentence)
```

Рисунок 5. Метод ml для реферирования текста при помощи библиотеки pysummarization

В качестве метода ML был использована библиотека pysummarizer, которая использует seq2seq и lstm для оптимизации полученного результата. Seq2seq – это модель, состоящая из декодера и энкодера, т.е. двух рекурентных сетей, она пытается предугадать последовательность по предыдущей последовательности. Lstm – тоже рекурентная сеть, но она усложнена фильтром, который определяет, что запоминать, а что нет.