Министерство науки и высшего образования
Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

Факультет *«Информатика и системы управления»*

ОТЧЕТ
по лабораторной работе №3

по дисциплине

«Информационный поиск и извлечение информации из текстов»

Студент группы ИУ9-21М _____ *С.С. Погосян*
(подпись, дата)

Руководитель _____ *Н.В. Лукашевич*
(подпись, дата)

Москва — 2021

1. Постановка задачи

Задание на дом (на 3 недели)

- Запросы – это проанализированные вами факты из Википедии
- Коллекция текстов – это все упомянутые статьи из всех фактов (не менее 2 из каждого факта)
- Документы – это предложения из статей Википедии, указанных в этих фактах, т.е. объединенная коллекция предложений статей всех фактов
- Все должно быть обработано морфологическим анализатором
- Нужно найти наиболее релевантные предложения
  - По tf.idf (df в данном случае – это количество предложений, в которых встречалось слово)
  - Tf –
    - 1) это количество упоминаний слова в предложении (tf) или
    - 2) 0.4+0.6 (tf/tfmax)
  - Нормализация запроса и предложения
  - Т.е. выстроить все предложения из статей по мере сходства с запросом по векторной модели.
  - В отчете должны быть показаны веса выдаваемых предложений

2. Решение

За основу взяты следующие факты:

- К большому террору в революционной Франции привели восстания федералистов
- Ножницы превратили прозу Гюстава Леружа в стихи Блеза Сандрара.
- В годы правления Иди Амина из Уганды были изгнаны десятки тысяч азиатов

В приведенной в приложении программе было реализовано следующее:

1. Класс PrepareFile выполняет считывание текста из файла (в котором находятся все статьи, соответствующие своим фактам), токенизацию, текста и составление матрицы лемматизированных документов (предложений).
2. Класс WeightMatrix вычисляет матрицу терм-документ и нормализует ее
3. Класс VectorOfRequest вычисляет вектор запроса, на основе пространства состояний (набора лемм) и нормализует его

4. Класс RankingDocument рассчитывает косинусную близость между нормализованным запросом и нормализованной матрицей терм-документ и выдает ранжированный список вес-документ

5. Функция main() выполняет запрос

Примечание: в расчетах матрицы терм-документ использовалась следующая формула:

$$W_{td} = tf \cdot idf$$

Также использовались стоп слова при формировании запроса.
Код доступен по ссылке : `https://github.com/legion15q/sem2/tree/master/num3/py`

2.1. Результаты запросов

Запрос №1 : Ножницы превратили прозу Гюстава Леружа в стихи Блеза Сандрара.
Ответ: `https://github.com/legion15q/sem2/blob/master/num3/py/%D0%9E%D1%82%D0%B2%D0%B5%D1%821.txt`
Запрос №2: К большому террору в революционной Франции привели восстания федералистов
Ответ: `https://github.com/legion15q/sem2/blob/master/num3/py/%D0%9E%D1%82%D0%B2%D0%B5%D1%822.txt`
Запрос №3: В годы правления Иди Амина из Уганды были изгнаны десятки тысяч азиатов
Ответ `https://github.com/legion15q/sem2/blob/master/num3/py/%D0%9E%D1%82%D0%B2%D0%B5%D1%823.txt`

3. Приложение

```
import pymorphy2
from collections import Counter
import numpy as np
from math import log10
from math import sqrt
from math import pow


class PrepareFile(object):
    def __init__(self, file_name_):
        self.file_name = file_name_
        self.text = ''
        self.tokens_lst = []
        self.lemmatized_documents_matrix = []
        self.documents_matrix = []
        self.term_lemmas_lst = []
        self.morph = pymorphy2.MorphAnalyzer()
        self.read_file()
```

```
        self.tokenize()
        self.make_documents()
        self.make_lemmas()

    def read_file(self):
        with open(self.file_name, 'r') as f:
            self.text = f.read()

    def tokenize(self):
        signs = '-!$'"‘ % ^ & *() _ + | ~ =‘{}[]: ";<>?#-«»'
        self.text.lower()
        for i in signs:
            self.text = self.text.replace(i, ' ')
        self.text = self.text.replace('.', ' .')
        self.tokens_lst = list(filter(None, self.text.split()))

        self.tokens_lst.insert(0, '<s>')
        self.tokens_lst.append('</s>')

    def make_lemmas(self):
        for i in self.documents_matrix:
            temp_lst = []
            for j in i:
                temp_lst.append(self.morph.parse(j)[0].normal_form)
            self.lemmatized_documents_matrix.append(temp_lst)
        for i in self.tokens_lst:
            self.term_lemmas_lst.append(self.morph.parse(i)[0].normal_form)

    def make_documents(self):
        n = len(self.tokens_lst)
        temp = 0
        for i in range(n):
            if self.tokens_lst[i] == '.':
                self.documents_matrix.append(self.tokens_lst[temp:i + 1])
                temp = i + 1


class WeightMatrix(object):
    def __init__(self, term_lemmas_lst_, lemmatized_documents_matrix_, documents_ma
        self.documents_matrix = documents_matrix_
        self.lemmatized_documents_matrix = lemmatized_documents_matrix_
        self.term_lemmas_lst = term_lemmas_lst_
        self.tf = []
        self.df = []
        self.weight_matrix = []
        self.calc_tf()
        self.calc_df()
```

```
        self.calc_weight_matrix()
        self.normalize_weight_matrix()

    # в данном случае tf - это количество упоминаний слова в
     лемматизированном документе (предложении)
    def calc_tf(self):
        for i in self.lemmatized_documents_matrix:
            self.tf.append(Counter(i))

    def calc_df(self):
        for i in self.lemmatized_documents_matrix:
            temp_lst = []
            for j in i:
                k = 0
                for m in self.lemmatized_documents_matrix:
                    if m.count(j) > 0:
                        k += 1
                temp_lst.append({j: k})
            self.df.append(temp_lst)

    # матрица терм --> вес терма в каждом документе
    def calc_weight_matrix(self):
        for i in self.term_lemmas_lst:
            weight_for_doc = []
            for j in range(len(self.lemmatized_documents_matrix)):
                if self.lemmatized_documents_matrix[j].count(i) > 0:
                    weight_for_doc.append(
                        self.tf[j][i] * log10(len(self.term_lemmas_lst) /
                         self.getElement(self.df[j], i)))
                else:
                    weight_for_doc.append(0)
            self.weight_matrix.append(weight_for_doc)
            # self.weight_matrix.append({i: weight_for_doc})

    def getElement(self, lst, element):
        for i in lst:
            for k, v in i.items():
                if k == element:
                    return v

    def normalize_weight_matrix(self):
        weight_matrix = np.array(self.weight_matrix)
        N = len(self.weight_matrix)
        for i in range(len(weight_matrix[0])):
            length = calc_length_of_vector(weight_matrix[:, i])
            for j in range(N):
                self.weight_matrix[j][i] = self.weight_matrix[j][i] /
```

```
                length


class VectorOfRequest(object):
    def __init__(self, request, words_state_space_):
        self.request_vector = []
        self.normalized_request_vector = []
        request.lower()
        self.request_lst = request.split()
        self.morph = pymorphy2.MorphAnalyzer()
        stop_words_lst = ['в', 'и', 'не', 'к', 'или', 'из', 'на']
        for i in range(len(self.request_lst)):
            if stop_words_lst.count(self.request_lst[i]) == 0:
                self.request_vector.append(
                self.morph.parse(self.request_lst[i])[0].normal_form)
        self.words_state_space = words_state_space_

    def make_request_vector(self):

        for i in self.words_state_space:
            if self.request_vector.count(i) > 0:
                self.normalized_request_vector.append(1)
            else:
                self.normalized_request_vector.append(0)
        length = calc_length_of_vector(self.normalized_request_vector)
        for i in range(len(self.normalized_request_vector)):
            self.normalized_request_vector[i] = self.normalized_request_vector[i]/
            length
        return self.normalized_request_vector


class RankingDocument(object):
    def __init__(self, file_name_, request_):
        self.file_name = file_name_
        self.request = request_

    def RunSearch(self):
        file_collection = PrepareFile(self.file_name)
        lemmatized_documents_matrix = file_collection.lemmatized_documents_matrix
        docs_matrix = file_collection.documents_matrix
        term_lemmas_lst = file_collection.term_lemmas_lst
        weight_matrix = WeightMatrix(term_lemmas_lst,
        lemmatized_documents_matrix, docs_matrix)
        normalized_weight_matrix = weight_matrix.weight_matrix
        vr = VectorOfRequest(self.request, term_lemmas_lst)
        normalized_vector_of_request = vr.make_request_vector()
        ranking_lst = self.cos_similarity(normalized_vector_of_request,
```

```
         normalized_weight_matrix)
        sorted_map = self.make_ranking_docs_view(ranking_lst, docs_matrix)
        for i in sorted_map:
            print(i)

    def cos_similarity(self, normalized_request_vector,
    normalized_document_matrix):
        n = len(normalized_document_matrix)
        ranking_lst = []
        for i in range(len(normalized_document_matrix[0])):
            sum_ = 0
            for j in range(n):
                sum_ += normalized_request_vector[j] *
                 normalized_document_matrix[j][i]
            ranking_lst.append(sum_)
        return ranking_lst

    def make_ranking_docs_view(self, ranking_lst, documents_matrix):
        map_similarity_doc = {}
        for i in range(len(documents_matrix)):
            map_similarity_doc[ranking_lst[i]] = documents_matrix[i]
        sorted_map_similarity_doc = sorted(map_similarity_doc.items(),
         key=lambda x: x[0], reverse=True)
        return sorted_map_similarity_doc


def calc_length_of_vector(vector):
    sum = 0
    for i in vector:
        sum += pow(i, 2)
    return sqrt(sum)


def main():
    RD = RankingDocument('Статьи из фактов.txt', 'В годы правления
    Иди Амина из Уганды были изгнаны десятки тысяч азиатов')
    RD.RunSearch()


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