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 №5

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

- Реализовать языковую модель информационного поиска для поиска предложений в Википедии
  - лямбда=0.5 и лямбда=0.9

  – 1 апреля (3 недели)

Оценить NDCG языковой модели по вашим трем запросам и сравнить с предыдущими моделями

– 8 апреля (4 недели)

2. Решение

В ходе работы была реализована языковая модель для поиска предложений (документов) из Википедии. Набор предложений взят из предыдущей работы. Код, приведенный во вложении, также доступен по ссылке `https://github.com/legion15q/sem2/blob/master/num5-6/py/main.py` Для ранжирования результатов выдачи по запросу q построенной языковой модели использовалась следующая формула:

$$P(d|q) = \prod_{t \in q} \big((1-\lambda)P(t|M_c) + \lambda P(t|M_d)\big) \tag{2.1}$$

Здесь M_c – языковая модель коллекции, а M_d –языковая модель документа. Для второй части задания вручную были размечены результаты выдачи языковой и векторной моделей (из предыдущей работы). Проставлены оценки релевантности первых пятнадцати документов по шкале от 0 до 2, где 2 – предложение содержит полный факт, 1– предложение содержит часть факта. Размеченные выдачи доступны по ссылке `https://github.com/legion15q/sem2/tree/master/num5-6/py`

Приведенный в приложении код реализует следующий функционал:

- Класс PrepareFile парсит коллекцию документов (предложений) из Википедии, которая находится в .txt файле проекта. Разделяет документы (предложения) по точкам, токенизирует и лемматизирует их, и возвращает список полученных документов

- Класс VectorOfRequest формирует лемматизированный вектор запроса (не бинарный)

- Класс LanguageModel выполняет основные вычисления в программе, а именно: считает языковые модели документов и всей коллекции и вероятности $P(t|M_c), P(t|M_d)$. Замечание: Если слова из запроса нет в коллекции, то выполняется сглаживание по всей коллекции.
- Класс RankingDocument инициализирует работу алгоритма.

3. Приложение

```
import pymorphy2
from collections import Counter


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 LanguageModel(object):
    def __init__(self,  lemmatized_documents_matrix_, documents_matrix_,
     request_lst, lambda_):
        self.documents_matrix = documents_matrix_
        self.lemmatized_documents_matrix = lemmatized_documents_matrix_
        self.request_lst = request_lst
        self.tf = []
        self.df = []
        self.calc_tf()
        self.calc_df()
        self.p_2 = []
        self.p_1 = {}
        self.lambda_ = lambda_
        self.p = {}

    # 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))
        #print(self.tf)

    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)

    def calc_p_1(self):
```

```
        lemmas_collection = {}
        for i in self.lemmatized_documents_matrix:
            for j in i:
                if j in lemmas_collection:
                    lemmas_collection[j] = lemmas_collection[j] + 1
                else:
                    lemmas_collection[j] = 1
        for k,v in lemmas_collection.items():
            self.p_1[k] = v/len(lemmas_collection)

        #print(self.p_1)




    def calc_p_2(self):
        for i in self.tf:
            k = 0
            temp_lst = []
            for j in i:
                temp_lst.append({j : list(i.values())[k]/len(i) })
                k += 1
            self.p_2.append(temp_lst)

    def calc_p(self):
        k = 0
        for i in self.lemmatized_documents_matrix:
            value = 1
            for j in self.request_lst:
                try:
                    p_1 = self.p_1[j]
                except:
                    p_1 = 1/ (len(self.p_1)) # сглаживаем, если слова
                    нет в коллекции
                p_2 = p_1
                if i.count(j) > 0:
                    for m in range(len(self.p_2[k])):
                        if list(self.p_2[k][m].keys()).count(j) > 0:
                            p_2 = list(self.p_2[k][m].values())[0]
                value = value*((1-self.lambda_)*p_1 + self.lambda_*p_2)
            self.p[value] = tuple(self.documents_matrix[k])
            k += 1


    def find_prob_for_term(self, term):
        for i in self.p_2:
            for m in range(len(i)):
```

```
                if list(i[m].keys()).count(term) > 0:
                    return list(i[m].values())[0]



    def get_result(self):
        x = sorted(self.p.items(), key=lambda x: x[0], reverse=True)
        k = 0
        for i in x:
            k +=1
            print(k, i)




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

        # print(self.request_vector)



class RankingDocument(object):
    def __init__(self, file_name_, lambda_, request_):
        self.file_name = file_name_
        self.request_vector = VectorOfRequest(request_).request_vector
        self.lambda_ = lambda_

    def RunSearch(self):
        file_collection = PrepareFile(self.file_name)
        lemmatized_documents_matrix =
         file_collection.lemmatized_documents_matrix
        docs_matrix = file_collection.documents_matrix
        language_model = LanguageModel(lemmatized_documents_matrix,
```

```
         docs_matrix, self.request_vector, self.lambda_)
        language_model.calc_p_1()
        language_model.calc_p_2()
        language_model.calc_p()
        language_model.get_result()



def main():
    RD = RankingDocument('docs.txt', 0.5, 'Ножницы превратили прозу
     Гюстава Леружа в стихи Блеза Сандрара')
    RD.RunSearch()


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