МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА
Факультет информатики и систем управления
Кафедра теоретической информатики и компьютерных технологий

Лабораторная работа №8
по курсу «Информационный поиск»

«Ранжирование документов по запросу. Языковая модель»

Выполнил:
студент группы ИУ9-21М
Беляев А. В.

Проверила:
Лукашевич Н. В.

Москва 2019

1 Цель работы

Для фактов из Л.Р.1 и соответствующих статей, необходимо среди этих статей найти предложения, содержащие факты. Поиск осуществляется с помощью языковой модели

2 Ход работы

В Л.Р.1 были предложены следующие факты:

- В рецензии на компьютерную игру критик пожаловался на то, что смерть заставляет начинать уровень заново.
- Под стенами осажденнои шведами русскои крепости немцы побили шотландцев за пиво.
- Есть версия, что Джек Потрошитель был женщинои.

В этих фактах содержались ссылки на следующие статьи:

- Earth Shaker (игра)
- Осада Везенберга 1574 года
- Пиво
- Джек Потрошитель
- Мэри Пирси

Перед началом ранжирования документы и запросы были предобработаны: были убраны разделители и статьи были разбиты на предложения с помощью NLTK, убрана пунктуация, убраны слишком короткие предложения (менее трех слов), произведена нормализация предложений.

Функции чтения файлов, а также «очистки» и нормализации предложений были взяты из Л.Р. 3.

3 Текст программы

```
from lab3 import read_sentences, clean_up_sentence, normalize_sentence

ARTICLES = [
    'art1-game.txt',
    'art2-siege.txt',
    'art3-beer.txt',
    'art4-jack.txt',
    'art5-jack-rule-63.txt'
]

```

```
FACTS = [
    'В рецензии на компьютерную игру критик пожаловался на то, что смерть заставляет начинать уровень заново.',
    'Под стенами осажденнои шведами русскои крепости немцы побили шотландцев за пиво.',
    'Есть версия, что Джек Потрошитель был женщинои.'
]

COLLECTION = []

RESULTS_TO_OUTPUT = 5

LAMBDA = 0.5


def lang_model(query: str, docs: list) -> list:
    def collection_model (term: str) -> float:
        return COLLECTION.count(term) / len(COLLECTION)

    def document_model(term: str, document: list) -> float:
        return document.count(term) / len(document)  # tf(t,d) / len(d)

    similarity = {}
    for doc in docs:
        p = 1
        for term in query.split(' '):
            p *= (1 - LAMBDA) * collection_model(term) + LAMBDA * document_model(term, doc)
            assert 0 != p    # make sure smoothing works
        similarity[doc] = p

    sorted_by_weight = sorted(similarity.items(), key=lambda doc: doc[1], reverse=True)
    return sorted_by_weight[:RESULTS_TO_OUTPUT]


def main():
    sentences = []
    for article_filename in ARTICLES:
        sentences.extend(read_sentences(article_filename))

    # create collection from words of all sentences
    for sentence in sentences:
        COLLECTION.extend(sentence.split(' '))

```

```
    # remove empty word that could appear by mistake :)
    if '' in COLLECTION:
        COLLECTION.remove('')

    clean_queries = [clean_up_sentence(s) for s in FACTS]
    queries = [normalize_sentence(s) for s in clean_queries]

# smoothing: add words from query to collection to avoid 'p(term)==0'
    for q in queries:
        COLLECTION.extend(q.split(' '))

    for q in queries:
        matched_docs = lang_model(q, sentences)
        print(q)
        for match in matched_docs:
            print(f'\t{match[1]} {match[0]}')


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

4 Результаты работы

Результаты ранжирования представлены в таблице 1. В таблице представлены очищенные нормализованные данные, с которыми работал алгоритм, а также итоговые веса документов по отношению к запросу.

Оснвная проблема - слова в запросе и документах были сопоставлены «как есть». Если бы допускалось использование синонимов, результаты были бы значительно лучше.

Также заметно тяготение алгоритма к более коротким предложениям. В целом же результат ранжирования получился схожим с векторонй моделью и TFIDF. Точно так же, как и в других моделях, документы были выданы верно в 2х из 3х случаев, с незначительными отличиями.

В качестве отличительной черты стоит отметить простоту алгоритма (в том числе при нехватке данных (сглаживание)) по сравнению с двумя предыдущими.

5 Выводы

В ходе работы были изучена языковая модель, позволяющая ранжировать документы по степени соответствия запросу. Наивная реализация этого метода далека от идеала. Тем не менее, результат получился схожим с предыдущими реализациями.

Таблица 1: Языковая модель

	рецензия компьютерный игра критик пожаловаться смерть заставлять начинать уровень заново
8.34e-33	вскрывать брюшной полость джек потрошитель начинать уже смерть жертва
7.75e-33	потеря жизнь уровень запускаться заново жизнь последний игра заканчиваться
8.24e-34	метр выпуск книга компьютерный игра быть указать игра хорошеть графика отличный звуковой сопровождение
7.74e-34	заключение критик сообщить впечатлеть
7.46e-34	автор книга компьютерный мир посчитать игра отличный график очень неплохой музыка
	стен осажда нной швед русской крепость немец побить шотландец пиво
7.87e-33	неоднократный попытка швед совершить подкоп взорвать стена вовремя пресекаться защитник крепость
4.35e-34	результат бойня погибнуть немец шотландец
1.78e-34	шведский финский солдат присоединиться шотландский преимущественно пехота немецкий преимущественно конница артиллерия наёмник
1.40e-34	почти девятимесячный осада ревелеть оплот шведский владычество прибалтика увенчаться успех войско иван грозный удаться взять несколько меньший крепость частность вейсенштейн
1.28e-34	шотландец бежать немец русский гарнизон везенберг быть поздний доставить москва
	версия джек потрошитель быть женщиной
4.32e-12	основа быть взять женский версия убийца
2.19e-12	придерживаться версия пять жертва
1.71e-12	один версия имя джек потрошитель скрываться душевнобольной польский еврей эмигрант аарон косминский
1.36e-12	ставить этот версия сомнение однозначный доказательство тот жертва быть задушить существовать
1.22e-12	утверждать джек потрошитель быть льюис кэрролл