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

Лабораторная работа №3
по курсу «Информационный поиск»

«Ранжирование документов по запросу. Практическая часть»

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 векторной модели и TF-IDF.

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 осажде¨ннои˘ 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.

3 Текст программы

```
1  import math
2  from collections import defaultdict
3
4  import nltk.data
5  import pymorphy2
6  import re
7
8  morph = pymorphy2.MorphAnalyzer()
9  nltk.download('punkt')   # required to split text into sentences
10
11 ARTICLES = [
12     '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 осаждённой шведами русской крепости немцы побили
    ↪  шотландцев за пиво.',
    'Есть версия, что Джек Потрошитель был женщиной.'
]

VOCABULARY = set()

RESULTS_TO_OUTPUT = 5

class Vectorizable:

    def __init__(self, sentence: str):
        self.sentence = sentence
        self.words = sentence.split(' ')
        self.vector = {}
        self.tfidf_vector = {}

    # weight of term in the document is its frequency
    def vectorize(self):
        for term in VOCABULARY:
            self.vector[term] = self.sentence.count(term)  # count ==
            ↪  TermFrequency

    def __hash__(self):
        return hash(self.sentence)

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return self.sentence


def normalize_sentence(sentence: str) -> str:
    tags_to_remove = ['NPRO', 'PRED', 'PREP', 'CONJ', 'PRCL', 'INTJ']
    normalized = []
```

```
    for w in sentence.split(' '):
        parsed = morph.parse(w)[0]
        if (parsed.tag.POS not in tags_to_remove) and 3 <=
        ↪  len(parsed.normal_form):
            normalized.append(parsed.normal_form)
    return ' '.join(normalized)


def clean_up_sentence(s: str) -> str:
    no_punct = re.sub(r'[^а-яё]', ' ', s.casefold())
    no_duplicate_spaces = re.sub(r'\s+', ' ', no_punct)
    return no_duplicate_spaces.strip()


# remove punct -> split -> clean up -> remove short -> normalize
def read_sentences(filename: str) -> list:
    text = open(filename, 'r').read()
    preprocessed = re.sub(r'[,:;-]', ' ', text)
    sentences = nltk.sent_tokenize(preprocessed)
    clean_sents = list(map(lambda s: clean_up_sentence(s), sentences))
    no_short_sents = list(filter(lambda s: len(s.split(' ')) >= 3,
    ↪  clean_sents))
    normalized = list(map(lambda s: normalize_sentence(s),
    ↪  no_short_sents))
    return normalized


def scalar_product(v1: dict, v2: dict):
    p = 0
    for w in v1.keys():
        p += v1[w] * v2[w]
    return p


def norm(vect: dict) -> float:
    n = 0
    for word in vect.keys():
        n += vect[word] ** 2
    return math.sqrt(n)


def vector_space_model(query: Vectorizable, docs: list) -> list:
    cos_similarity = {}
    for doc in docs:
        multiplied_norms = norm(query.vector) * norm(doc.vector)
```

```
97          cos_similarity[doc] = scalar_product(query.vector, doc.vector) /
            ↪ multiplied_norms
98
99      sorted_by_weight = sorted(cos_similarity.items(), key=lambda doc:
        ↪ doc[1], reverse=True)
100     return sorted_by_weight[:RESULTS_TO_OUTPUT]
101
102
103 def tf_idf(query: Vectorizable, docs: list) -> list:
104     def tf(term: str, vect: Vectorizable) -> int:
105         return vect.words.count(term)
106     def idf(term: str) -> float:
107         containing_term = list(filter(lambda doc: term in doc.words,
            ↪ docs))
108         df = len(containing_term)
109         if 0 == df: df = 1
110         return math.log(len(docs) / df)
111
112     # count TFIDF weight for each document
113     for doc in docs:
114         weight = defaultdict(int)
115         for word in doc.words:
116             weight[word] = tf(word, doc) * idf(word)
117         doc.tfidf_vector = weight
118
119     # count TFIDF for a query
120     weight = defaultdict(int)
121     for word in query.words:
122         weight[word] = tf(word, query) * idf(word)
123     query.tfidf_vector = weight
124
125     # now just apply cosine similarity like for vector-space model
126     cos_similarity = {}
127     for doc in docs:
128         multiplied_norms = norm(query.tfidf_vector) *
            ↪ norm(doc.tfidf_vector)
129         cos_similarity[doc] = scalar_product(query.tfidf_vector,
            ↪ doc.tfidf_vector) / multiplied_norms
130
131     sorted_by_weight = sorted(cos_similarity.items(), key=lambda doc:
        ↪ doc[1], reverse=True)
132     return sorted_by_weight[:RESULTS_TO_OUTPUT]
133
134
135 def main():
```

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

    # create vocabulary from words of all sentences
    for sentence in sentences:
        VOCABULARY.update(sentence.split(' '))

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

    clean_queries = list(map(lambda s: clean_up_sentence(s), FACTS))
    norm_queries = list(map(lambda s: normalize_sentence(s),
     ↪  clean_queries))

    # add words from query to dictionary
    for q in norm_queries:
        VOCABULARY.update(q.split(' '))

    queries = list(map(lambda q: Vectorizable(q), norm_queries))
    [q.vectorize() for q in queries]

    docs = list(map(lambda s: Vectorizable(s), sentences))
    [d.vectorize() for d in docs]

    print('Vector space model')
    for q in queries:
        q.vectorize()
        matched_docs = vector_space_model(q, docs)
        print(q)
        for match in matched_docs:
            print(f'\t{match[1]:.3f} {match[0]}')

    print('TF-IDF')
    for q in queries:
        matched_docs = tf_idf(q, docs)
        print(q)
        for match in matched_docs:
            print(f'\t{match[1]:.3f} {match[0]}')


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

Таблица 1: Векторная модель

рецензия компьютерный игра критик пожаловаться смерть заставлять начинать уровень заново
0.272 метр выпуск книга компьютерный игра быть указать игра хорошеть графика отличный звуковой сопровождение 0.258 заключение критик сообщить впечатлеть 0.250 потеря жизнь уровень запускаться заново жизнь последний игра заканчиваться 0.231 автор книга компьютерный мир посчитать игра отличный график очень неплохой музыка 0.224 разработать свой редактор игра позволять редактировать уровень график игра
стен осажда нной швед русской крепость немец побить шотландец пиво
0.240 неоднократный попытка швед совершить подкоп взорвать стена вовремя пресекаться защитник крепость 0.236 результат бойня погибнуть немец шотландец 0.224 шотландец бежать немец русский гарнизон везенберг быть поздний доставить москва 0.185 несколько год перемирие северный прибалтика вызвать русско литовский война год русский войско возобновить военный действие 0.174 март немец шотландец дело дошлый потасовка вызвать неоплаченный эль взаимный оскорбление
версия джек потрошитель быть женщиной
0.589 утверждать джек потрошитель быть льюис кэрролл 0.575 возвращение джек потрошитель героиня фильм молли считать потомок известный убийца джек потрошитель присутствовать фильм качество персонаж виртуальный реальность собиратель душа сериал сезон серия который представляться версия тот джек потрошитель быть женщина 0.533 корнуэлла заявить джек потрошитель быть британский художник уолтер сикерта 0.516 потрошитель эпизод потрошитель сериал грань возможный возвращение джек потрошитель слэшер который присутствовать аллюзия способ убийство джек потрошитель 0.490 джек потрошитель перерезать горло слева направо рана быть очень глубокий

4 Результаты работы

Результаты ранжирования представлены в таблицах 1 и 2. В таблицах представлены очищенные нормализованные данные, с которыми работал алгоритм, а также итоговые веса документов по отношению к запросу.

Можно заметить, что в целом результаты работы моделей схожие. В 2х из 3х случаев TFIDF показывала результаты чуть лучше векторной модели.

Оснвная проблема - слова в запросе и документах были сопоставлены «как есть». Если бу допускалось использование синонимов, а документы не вопринимались бы как «мешки слов», результаты были бы значительно лучше.

5 Выводы

В ходе работы были изучены 2 модели, позволяющие ранжировать документы по степени соответствия запросу. Наивные реализации этих методов далеки от идеала,

Таблица 2: TFIDF

рецензия компьютерный игра критик пожаловаться смерть заставлять начинать уровень заново
0.264 вскрывать брюшной полость джек потрошитель начинать уже смерть жертва 0.221 потеря жизнь уровень запускаться заново жизнь последний игра заканчиваться 0.164 критик отметить являться клон англ русск 0.159 заключение критик сообщить впечатлеть 0.136 метр выпуск книга компьютерный игра быть указать игра хорошеть графика отличный звуковой сопровождение
стен осажда нной швед русской крепость немец побить шотландец пиво
0.208 результат бойня погибнуть немец шотландец 0.193 неоднократный попытка швед совершить подкоп взорвать стена вовремя пресекаться защитник крепость 0.151 шотландец бежать немец русский гарнизон везенберг быть поздний доставить москва 0.132 март немец шотландец дело дошлый потасовка вызвать неоплаченный эль взаимный оскорбление 0.127 камень алмаз останавливать свой падение оказываться земля стен
версия джек потрошитель быть женщиной
0.216 основа быть взять женский версия убийца 0.202 придерживаться версия пять жертва 0.156 возвращение джек потрошитель героиня фильм молли считать потомок известный убийца джек потрошитель присутствовать фильм качество персонаж виртуальный реальность собиратель душа сериал сезон серия который представляться версия тот джек потрошитель быть женщина 0.149 один версия имя джек потрошитель скрываться душевнобольной польский еврей эмигрант аарон косминский 0.141 ставить этот версия сомнение однозначный доказательство тот жертва быть задушить существовать

однако несмотря на это они показали хорошие результаты. Ранжирование, построенное с помощью TFIDF показало результаты несколько лучше.