Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Информационный поиск»

Студент: И. Д. Недосеков
Преподаватель: А. А. Кухтичев
Группа: М8О-406Б

Дата:
Оценка:
Подпись:

Москва, 2022

Курсовая работа

Необходимо разработать систему исправления опечаток в поисковом запросе.

1 Описание

Для исправления опечаток будем использовать биграммы. N-грамма — последовательность из n элементов. С семантической точки зрения, это может быть последовательность звуков, слогов, слов или букв. Для нас подойдет биграмма из букв. Индекс биграмм будем строить по готовому корпусу. Будем ссчитать что слово нужно исправлять, если слова нет в словаре. Для слов с опечаткой будем находить все слова в которых содержаться некоторое биграммы из слова с опечаткой. Для слова и кандидата будем считать коэффициент Жаккара. Если коэффициент больше 0.8, то заменяем слова на кандидата. Иначе ищем слово с коэффициентом большим 0.5 и делаем запрос с этим словом и без него (поисковую выдачу смешиваем).

2 Исходный код

исправление опечаток

```
from helpers.mongodb_connector import mongo_client
from helpers.bigrams import get_bigrams_of_word
from config import Config



class Dimension:

def __init__(self, word: str) -> None:
  self.word = word
  self.words_bigrams = set(get_bigrams_of_word(word))

def distance_to(self, candidate: str) -> float:
  if abs(len(self.word) - len(candidate)) > 3:
    return 0
  words_bigrams = self.words_bigrams
  candidates_bigrams = set(get_bigrams_of_word(candidate))
  distance = abs(len(words_bigrams) - len(words_bigrams & candidates_bigrams))
  return abs(1 - distance / len(words_bigrams | candidates_bigrams))


async def get_fuzzy_words(word: str) -> list[str]:
  bigrams = get_bigrams_of_word(word)
  word_dimension = Dimension(word)
  fuzzy_words_rating: set[tuple[str, float]] = set()
  async for bigram_with_words in mongo_client.test.bigram_words.find({'bigram':
  ↪  {'$in': bigrams}}):
    fuzzy_words_rating.update(
      (candidate, word_dimension.distance_to(candidate))
      for candidate in bigram_with_words['words']
      if word_dimension.distance_to(candidate) >=
      ↪  Config.minimum_distance_for_fuzzy_words
    )

  return [
     (word, rank)
    for word, rank in reversed(sorted(fuzzy_words_rating, key=lambda row: row[1]))
  ]

async def is_dictionary_word(word: str) -> bool:
  return bool(
```

```
      await mongo_client.test.word_bigrams.count_documents({'word': word})
    )

  def get_transformed_queries(enrich_query: EnrichQuery) -> list[list[str]]:
    queries: list[list[str]] = [[]]

    for word in enrich_query.query:
      if word not in enrich_query.fuzzy_mapping:
        for query in queries:
          query.append(word)

      elif enrich_query.fuzzy_mapping[word][0][1] >
      ↪  Config.minimum_distance_for_fuzzy_words_for_replace:
        for query in queries:
          query.append(enrich_query.fuzzy_mapping[word][0][0])

      elif enrich_query.fuzzy_mapping[word][0][1] >
      ↪  Config.minimum_distance_for_fuzzy_words_for_combine_request:
        transformed = deepcopy(queries)
        for query in transformed:
          query.append(enrich_query.fuzzy_mapping[word][0][0])

        queries.extend(transformed)

    return queries
```

построение индекса биграмм

```
import asyncio
import asyncstdlib
from loguru import logger
from helpers.mongodb_connector import mongo_client


async def build_bigram_words_dictionary():
    total_documents = await mongo_client.test.word_bigrams.count_documents({})
    percentage = 0
    async for i, word_with_bigrams in
    ↪  asyncstdlib.enumerate(mongo_client.test.word_bigrams.find()):
        if i / total_documents * 100 > percentage:
            logger.debug('Process {:.0%} documents', i / total_documents)
            percentage += 10
```

```
14
15          word = word_with_bigrams['word']
16          await asyncio.gather(
17              *(
18                  mongo_client.test.bigram_words.find_one_and_update(
19                      {'bigram': bigram},
20                      {'$push': {'words': word}},
21                      upsert=True,
22
23                  )
24                  for bigram in word_with_bigrams['bigrams']
25              )
26          )
```

3 Выводы

Выполнив первую курсовую работу по курсу «Информационный поиск», познакомился с основной теорией по исправлению поискового запроса, коэффициентом Жаккара. Для коротких слов испраление через биграммы плохо подходит, так как лишком большое число биграмм относительно общего количества в слове поврежденно (для "малоко"повреждено 2 из 5 биграмм). Для длинных слов такой метод вполне применим.

Список литературы

[1] Маннинг, Рагхаван, Шютце *Введение в информационный поиск* — Издательский дом «Вильямс», 2011. Перевод с английского: доктор физ.-мат. наук Д. А. Клюшина — 528 с. (ISBN 978-5-8459-1623-4 (рус.))