Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Информационный поиск»

Студент: А. К. Киреев
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. Индекс биграмм будем строить по готовому корпусу, по которому уже отработал nltk. Будем считать что слово нужно исправлять, если слова нет в словаре (словарь формируем из слов корпуса). 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. Выбираем из полученной выборки сто наиболее похожих слов (с наибольшим коэффициентом Жаккара), то есть делаем грубую оценку, а потом для них и исходного слова считаем расстояние Дамерау-Левенштейна и выбираем три слова с наименьшим расстоянием. Такая схема была выбрана, так как индекс биграмм быстро работает, но хуже умеет исправлять ошибки, чем алгоритм Дамерау-Левенштейна.

2 Исходный код

bigram_index.py

```python
from typing import List, Dict, Set
from logging import Logger

from db import get_words_by_bigrams, check_if_exists

from search_helper.common import get_bigrams
from search_helper.metrics import jaccard_coef, damerau_levenshtein_distance


class BigramIndex:
  def __init__(self, enriched_request: List[str],
        count_bound: int = 3, distance_bound: float = 3) -> None:
    self.req = enriched_request
    self.count_bound = count_bound
    self.distance_bound = distance_bound
    self.search_dict: Dict = dict()

  async def build(self, logger: Logger) -> None:
    for word in self.req:
      if await check_if_exists(word):
        logger.debug('Word "%s" exists', word)
        self.search_dict[word] = [word]
        continue

      self.search_dict[word] = []
      bigrams: Set[str] = get_bigrams(word)
      index: Dict = await get_words_by_bigrams(bigrams)

      words: Set[str] = set()
      for _, others in index.items():
        for other in others:
          words.add(other)

      coefs: List[List[float]] = []
      for other in words:
        coefs.append([jaccard_coef(word, other), other])
      coefs.sort(reverse=True)

      most_similar: List = [w for _, w in coefs[:100]]
      distances: List[List[float]] = []
      for other in most_similar:
```

```
        distances.append([damerau_levenshtein_distance(word, other), other])
      distances.sort()

      for d, supposed in distances[:self.count_bound]:
        logger.debug('Supposed word "%s" with LD-distance %s', supposed, d)
        if d <= self.distance_bound:
          self.search_dict[word].append(supposed)
        else:
          self.search_dict[word].append(word)

  def get_supposed(self, word: str) -> Set[str]:
    return self.search_dict[word]

  def get_search_dict(self) -> Dict:
    return self.search_dict

```

metrics.py

```
from typing import Set, Dict

from search_helper.common import get_bigrams


def damerau_levenshtein_distance(lhs: str, rhs: str) -> float:
  d: Dict = dict()
  for i in range(-1, len(lhs) + 1):
    d[(i, -1)] = i + 1
  for j in range(-1, len(rhs) + 1):
    d[(-1, j)] = j + 1

  for i in range(len(lhs)):
    for j in range(len(rhs)):
      if lhs[i] == rhs[j]:
        cost = 0
      else:
        cost = 0.9
      d[(i, j)] = min(
        d[(i - 1, j)] + 1,
        d[(i, j - 1)] + 1,
        d[(i - 1, j - 1)] + cost,
      )
      if i and j and lhs[i] == rhs[j - 1] and lhs[i - 1] == rhs[j]:
        d[(i, j)] = min(d[(i, j)], d[i - 2, j - 2] + 1)
```

```
26
27      return d[len(lhs) - 1, len(rhs) - 1]
28
29
30  def jaccard_coef(lhs: str, rhs: str):
31      lhs_bigrams: Set[str] = get_bigrams(lhs)
32      rhs_bigrams: Set[str] = get_bigrams(rhs)
33
34      coef: float = len(lhs_bigrams & rhs_bigrams) / len(lhs_bigrams | rhs_bigrams)
35      return coef
```

save_bigrams.py

```
1   import asyncio
2
3   from typing import List
4
5   from motor.motor_asyncio import AsyncIOMotorClient
6
7   client = AsyncIOMotorClient()
8
9
10  async def save_bigrams() -> None:
11      async for record in client.IR.WordsStorage.find():
12          word: str = record['word']
13          bigrams: List[str] = record['bigrams']
14
15          await asyncio.gather(
16              *(
17                  client.IR.BigramStorage.find_one_and_update(
18                      {'bigram': bigram},
19                      {'$push': {'words': word}},
20                      upsert=True
21                  )
22                  for bigram in bigrams
23              )
24          )
```

3 Выводы

Выполнив курсовую работу по курсу «Информационный поиск», познакомился с основной теорией по исправлению поискового запроса, коэффициентом Жаккара, вспомнил алгоритм подсчета расстояния Дамерау-Левенштейна. Следует заметить, что для коротких слов испраление через биграммы плохо подходит, так как лишком большое число биграмм относительно общего количества в слове поврежденно, поэтому после выбора ста наиболее подходящих слов я использую алгоритм Дамерау-Левенштейна.

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.))