山东大学 计算机科学与技术 学院

信息检索 课程实验报告

学号:201800130036	姓名： 戎思宇	班级： 大数据
实验题目：Ranked retrieval model		
实验学时：2	实验日期：2020.10.25	
实验目的： 在 Experiment1 的基础上实现最基本的 Ranked retrieval model Use SMART notation: lnc.lt 改进 Inverted index		
软件环境： anaconda		
实验内容与设计： 1. 实验内容（题目内容，输入要求，输出要求） 在 Experiment1 的基础上实现最基本的 Ranked retrieval model Use SMART notation: lnc.lt 改进 Inverted index 2. 算法描述 （整体思路描述，所需要的数据结构与算法） 读取内容，处理文本，提取主要信息，记录 tweetid，求平均长度；对查询内容正则化，进行查询 3. 测试结果（测试输入，测试输出，结果分析） 4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）		
分析与体会： 初步学习了有序检索模型，了解了有序检索模型中如何进行不通过具有精确语义的逻辑表达式来构建查询，而采用一个或者多个词来构成自由文本查询，相对布尔查询能够有更加精确的结果		

附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

```
import sys
import math
from functools import reduce
from textblob import TextBlob
from textblob import Word
from collections import defaultdict

uselessTerm = ["username", "text", "tweetid"]
postings = defaultdict(dict)
document_frequency = defaultdict(int)
document_lengths = defaultdict(int)
document_numbers = len(document_lengths)
avdl = 0

def main():
    get_postings_dl()
    initialize_document_frequencies()
    initialize_avdl()

    while True:
        do_search()

def tokenize_tweet(document):
    global uselessTerm
    document = document.lower()
    a = document.index("username")
    b = document.index("clusterno")
    c = document.rindex("tweetid") - 1
    d = document.rindex("errorcode")
    e = document.index("text")
    f = document.index("timestr") - 3
    document = document[c:d] + document[a:b] + document[e:f]
    terms = TextBlob(document).words.singularize()

    result = []
    for word in terms:
        expected_str = Word(word)
        expected_str = expected_str.lemmatize("v")
        if expected_str not in uselessTerm:
            result.append(expected_str)

    return result

def get_postings_dl():
```

```
    global postings, document_lengths
    f = open(r"D:/777/tweets.txt")
    lines = f.readlines()  # 读取内容

    for line in lines:
        line = tokenize_tweet(line)
        tweetid = line[0]  # tweetid
        line.pop(0)
        document_lengths[tweetid] = len(line)  # 记录词数
        unique_terms = set(line)
        for te in unique_terms:
            postings[te][tweetid] = line.count(te)
    # 按字典序对 postings 进行升序排序,但返回的是列表，失去了键值的信息

def initialize_document_frequencies():
    global document_frequency, postings
    for term in postings:
        document_frequency[term] = len(postings[term])
    mylog2 = open(r"C:\Users\ASUS\Desktop\Inverteddf.txt", mode='a', encoding='utf-8')
    print(document_frequency,file=mylog2)

# 求平均文档长度
def initialize_avdl():
    global document_lengths, avdl
    count = 0
    for twid in document_lengths:
        count += document_lengths[twid]
    avdl = count / len(document_lengths)


# 对输入的查询内容正则化
def token(doc):
    doc = doc.lower()
    terms = TextBlob(doc).words.singularize()

    result = []
    for word in terms:
        expected_str = Word(word)
        expected_str = expected_str.lemmatize("v")
        result.append(expected_str)

    return result
```

```
# 这是对每件的每个单词进行查询
def get_result(file_name):
    with open(file_name, 'w', encoding='utf-8') as f_out:

        Quaries = get_queries()
        qkeys = Quaries.keys()
        # 这里取得是索引
        for key in qkeys:

            q_result = do_search(Quaries[key])
            # 想清楚 Quaries[key]的含义
            for tweetid in q_result:
                f_out.write(str(key) + ' ' + tweetid + '\n')


# queries 是建立的查询索引
def get_queries():
    # 输出为对于查询 token 后的结果:id + 查询字符串
    queries = {}
    keyid = 171
    fq = open(
        r"C:\Users\ASUS\Desktop\qrels2014.txt")
    lines = fq.readlines()
    for line in lines:
        index1 = line.find("<query>")
        if index1 >= 0:
            # 得到<query>和</query>之间的查询内容
            index1 += 8
            index2 = line.find("</query>")
            # 添加到 queries 字典
            queries[keyid] = line[index1:index2]
            keyid += 1
    return queries
#对给定的 qres 文件进行查询

def do_search():
    query = token(input("Search query >> "))
    result = []   # 返回对于 query 的所有 tweetid 排序后的列表

    if query == []:g
        sys.exit()

    unique_query = set(query)
    # 避免遍历所有的 tweet，可先提取出有相关性的 tweetid，tweet 中包含查询的关键词
```

之一便可认为相关

```
    relevant_tweetids = Union([set(postings[term].keys()) for term in unique_query])
    print ("<<<<<Score(PLN)--Tweeetid>>>>>")
    print("PLN 一共有"+str(len(relevant_tweetids))+"条相关 tweet！ ")
    if not relevant_tweetids:
        print("No tweets matched any query terms for")
        print(query)

    else:
        # PLN
        scores3 = sorted([(id,similarity_PLN(query,id))for id in relevant_tweetids],key=lambda x:
x[1],reverse=False)
        # PLN+BM25
        '''scores3 = sorted([(id, similarity_BM25(query, id) + similarity_PLN(query, id))
                             for id in relevant_tweetids],
                            key=lambda x: x[1],
                            reverse=False)'''
        i = 1
        for (id, score) in scores3:
            if i<=10:
                result.append(id)
                print(str(score) + ": " + id)
                i = i + 1
            else:
                break
        print("finished")


def similarity_PLN(query, id):
    global postings, avdl
    fenmu = 1 - 0.1 + 0.1 * (document_lengths[id] / avdl)
    similarity = 0.0
    unique_query = set(query)
    for term in unique_query:
        wtq=query.count(term)/len(query)

        if (term in postings) and (id in postings[term].keys()):
            wtd = (1 + math.log(postings[term][id]) * math.log((document_numbers + 1) /
document_frequency[term]))
            similarity = wtq*wtd
            # 使用 ln(1+ln(C(w,d)+1))后发现相关性的分数都为负数很小

    return similarity
```

```
def similarity_BM25(query, id):
    global postings, avdl
    fenmu = 1 - 0.2 + 0.2 * (document_lengths[id] / avdl)
    k = 1
    similarity = 0.00

    unique_query = set(query)
    for term in unique_query:
        if (term in postings) and (id in postings[term].keys()):
            C_wd = postings[term][id]
            similarity += (query.count(term) * (k + 1) * C_wd * math.log(
                (document_numbers + 1) / document_frequency[term])) / (k * fenmu + C_wd)

    return similarity


def Union(sets):
    return reduce(set.union, [s for s in sets])


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