



دانشکده مهندسی کامپیوتر

دانشگاه اصفهان

تکلیف چهارم درس بازیابی اطلاعات

استاد: دکتر محمد مهدی رضاپور

مهر والسادات نوحی

۹۹۳۶۱۳۰۶۱

بهار ۱۴۰۳

شاخص Cosine

صورت تمرین:

با سلام خدمت دانشجویان عزیز. پیرو طرح موضوع انجام شده در کلاس در رابطه با تکلیف جدید (شماره ۴)، شما می بایست دو سیستم بازیابی توسعه دهید:

۱. با استفاده از TF-IDF Score یک سیستم بازیابی روی اسناد جمع آوری شده در تکلیف قبل توسعه دهید.
۲. سیستم بازیابی دیگری نیز با استفاده از شاخص Cosine توسعه دهید که بر اساس وکتور وزن های TF-IDF کار کند. دقت داشته باشید که کدها می بایست با زبان پایتون نوشته شود. برنامه میبایست قابل تست باشد وگرنه نمره ای به تکلیف تعلق نمیگیرد. لذا توضیحات کافی به صورت تصویری در رابطه با برنامه توسعه داده شده در قالب یک فایل ورد پیوست تکلیف باشد. کمتر ➤

مانند گذشته تمرین را در چند گام تقسیم بندی کرده و جداگانه توضیح خواهیم داد. ترجیحا از همان ۲۰ مقاله دانلود شده در تمرین های قبل برای این تمرین و تمرین بعدی استفاده خواهد شد که نتایج به صورت واضح تر نشان داده شود.

نکته: برخی از گام ها کاملا مشابه tf.idf می باشد ۴ گام اول با بخش قبلی مشترک می باشد.

گام اول:

در گام اول از پوشه ای که ۲۰ مقاله مورد نظر در آن است باید از هر ۲۰ مقاله شروع به خواندن و استخراج متن کنیم.

```
def main():  
    pdf_directory = "arxiv_orgs_pdfs"  
    all_texts = read_pdf(pdf_directory)
```

```
def read_pdf(pdf_directory):  
    all_texts = {}  
    try:  
        pdf_files = [filename for filename in os.listdir(pdf_directory) if filename.endswith('.pdf')]  
        pdf_files.sort(key=lambda x: int(os.path.splitext(x)[0]))  
        for filename in pdf_files:  
            pdf_path = os.path.join(pdf_directory, filename)  
            text = extract_text_from_pdf(pdf_path)  
            tokens = tokenize_text(text)  
            filename_without_extension = os.path.splitext(filename)[0]  
            all_texts[filename_without_extension] = tokens  
    except Exception as e:  
        print(f"Error reading PDFs: {e}")  
    return all_texts
```

این تابع `read_pdf` فایل‌های PDF را از یک دایرکتوری می‌خواند و متن هر فایل را استخراج و توکنیزه می‌کند. سپس نتیجه را به صورت یک دیکشنری ذخیره می‌کند که در آن نام فایل به عنوان کلید و لیست توکن‌ها به عنوان مقدار است.

```
import os
import PyPDF2
import string
import math
from collections import Counter

def extract_text_from_pdf(pdf_path):
    text = ""
    try:
        with open(pdf_path, 'rb') as f:
            pdf_reader = PyPDF2.PdfReader(f)
            for page_number in range(len(pdf_reader.pages)):
                extracted_text = pdf_reader.pages[page_number].extract_text()
                if extracted_text:
                    text += extracted_text
    except Exception as e:
        print(f"Error extracting text from PDF: {e}")
    return text
```

این تابع `extract_text_from_pdf`، متن یک فایل PDF را استخراج می‌کند. ابتدا فایل PDF باز می‌شود و سپس متن هر صفحه استخراج و به یک رشته اضافه می‌شود.

با تعریف یک دیکشنری که کلید آن شماره داکيومنت و مقدار آن متن داخل فایل است .

تذکر: من اسم فایل‌ها را برای راحتی کار به مثلاً 1.pdf و.. تغییر دادم و برای استخراج فقط شماره فایل داریم.

در واقع به شکل زیرشده است:

```
{
  "Filename1": text1,
  "Filename2": text2,
  ...}
```

گام دوم:

بعد از اینکه متن‌ها استخراج شده است حال باید توکنایز شود و به اصطلاح هر توکن یک کلمه در نظر بگیریم برای درخواست‌های کاربر.

```
def tokenize_text(text):
    text = text.lower()
    text = ''.join([char if char not in string.punctuation else ' ' for char in text])
    tokens = text.split()
    return tokens
```

این تابع `tokenize_text` متن ورودی را به لیستی از توکن‌ها تبدیل می‌کند. ابتدا متن به حروف کوچک تبدیل می‌شود، سپس علائم نگارشی با فاصله جایگزین می‌شوند و در نهایت متن به لیستی از کلمات تقسیم می‌شود.

گام سوم:

```
def build_tf_dictionary(all_texts):
    term_document_tf_dict = {}
    for doc_name, tokens in all_texts.items():
        tf_counter = Counter(tokens)
        total_tokens = len(tokens)
        tf = {term: tf_counter[term] / total_tokens for term in tf_counter}
        for term, tf_value in tf.items():
            if term not in term_document_tf_dict:
                term_document_tf_dict[term] = []
            term_document_tf_dict[term].append((doc_name, tf_value))
    return term_document_tf_dict
```

این تابع `build_tf_dictionary`، دیکشنری TF را می‌سازد. برای هر داکيومنت، فراوانی نسبی هر توکن (TF) محاسبه می‌شود و در دیکشنری ذخیره می‌شود. به ازای هر داکيومنت، برای همه توکن‌ها در تمام متن استخراج شده یک دیکشنری ساخته و به عنوان مثال کلید آن ترم یا توکن و مقدار آن مقدار `tf` آن به ازای داکيومنت، و شماره داکيومنت است.

Dict {

Term1 → Document_ID → tf #for that document find tf

Term2 → Document_ID → tf #for that document find tf

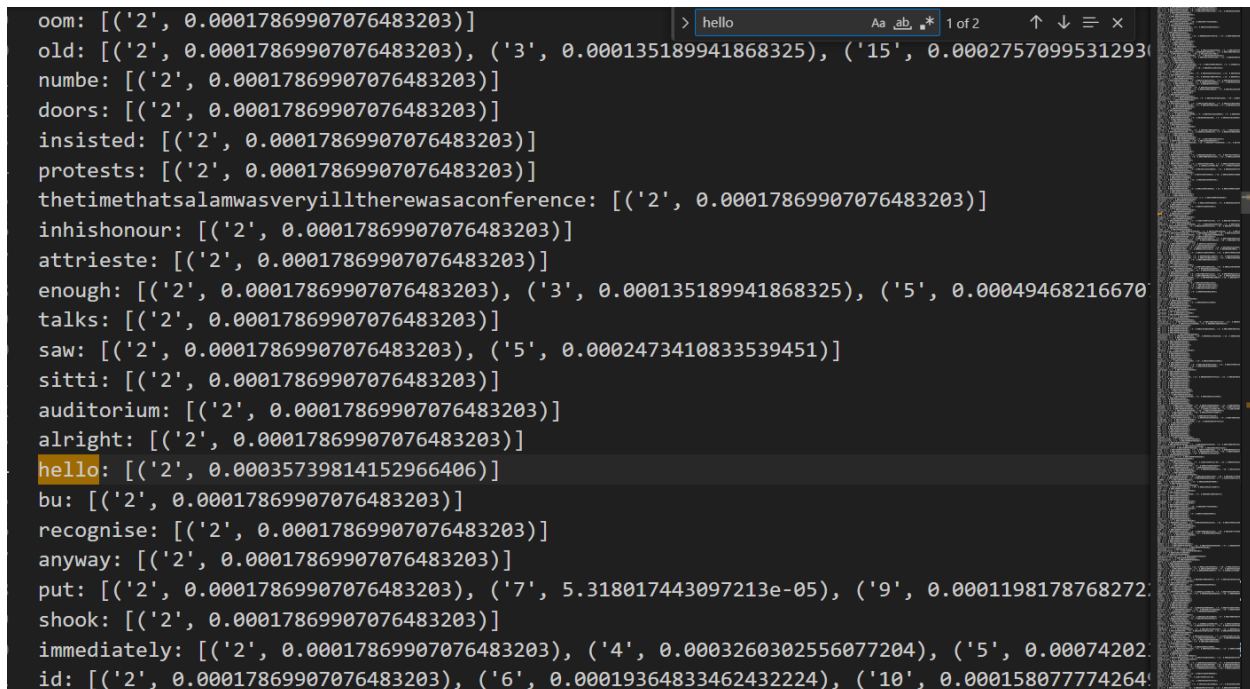
...}

مثال‌ها و توضیحات در بخش قبلی نوشته شده است.

میتوانیم خروجی این گام را در فایل ذخیره کنیم تابع به صورت زیر است:

```
def save_tf_dictionary_to_file(term_document_tf_dict, output_file):
    try:
        with open(output_file, 'w', encoding='utf-8') as f:
            for term, doc_tf_list in term_document_tf_dict.items():
                f.write(f"{term}: {doc_tf_list}\n")
    except Exception as e:
        print(f"Error writing to file: {e}")
```

خروجی:



```
oom: [('2', 0.00017869907076483203)]
old: [('2', 0.00017869907076483203), ('3', 0.000135189941868325), ('15', 0.0002757099531293)]
numbe: [('2', 0.00017869907076483203)]
doors: [('2', 0.00017869907076483203)]
insisted: [('2', 0.00017869907076483203)]
protests: [('2', 0.00017869907076483203)]
thetimethatsalamwasveryilltherewasacference: [('2', 0.00017869907076483203)]
inhishonour: [('2', 0.00017869907076483203)]
attrieste: [('2', 0.00017869907076483203)]
enough: [('2', 0.00017869907076483203), ('3', 0.000135189941868325), ('5', 0.00049468216670)]
talks: [('2', 0.00017869907076483203)]
saw: [('2', 0.00017869907076483203), ('5', 0.0002473410833539451)]
sitti: [('2', 0.00017869907076483203)]
auditorium: [('2', 0.00017869907076483203)]
alright: [('2', 0.00017869907076483203)]
hello: [('2', 0.00035739814152966406)]
bu: [('2', 0.00017869907076483203)]
recognise: [('2', 0.00017869907076483203)]
anyway: [('2', 0.00017869907076483203)]
put: [('2', 0.00017869907076483203), ('7', 5.318017443097213e-05), ('9', 0.0001198178768272)]
shook: [('2', 0.00017869907076483203)]
immediately: [('2', 0.00017869907076483203), ('4', 0.0003260302556077204), ('5', 0.00074202)]
id: [('2', 0.00017869907076483203), ('6', 0.00019364833462432224), ('10', 0.000158077774264)]
```

گام چهارم:

در برنامه یک دیکشنری دیگری تعریف کردم که در آن به ازای هر ترم از هر داکيومنت، idf آن محاسبه شده است که در آینده از آن استفاده کنیم. در واقع این تابع calculate_idf، مقادیر IDF را برای هر توکن محاسبه می‌کند. فراوانی مستندات برای هر توکن محاسبه می‌شود و سپس IDF محاسبه و در دیکشنری ذخیره می‌شود.

$$\text{idf}_t = \log_{10} (N/\text{df}_t)$$

```
def calculate_idf(all_texts):
    total_documents = len(all_texts)
    term_document_frequency = {}
    for doc_tokens in all_texts.values():
        unique_tokens = set(doc_tokens)
        for term in unique_tokens:
            if term not in term_document_frequency:
                term_document_frequency[term] = 0
            term_document_frequency[term] += 1

    idf_dict = {}
    for term, document_frequency in term_document_frequency.items():
        if(document_frequency==0):
            idf = math.log10(total_documents / (1 + document_frequency))
        else:
            idf = math.log10(total_documents / (document_frequency))
        idf_dict[term] = idf

    return idf_dict
```

میتوانیم خروجی این گام را در فایل ذخیره کنیم تابع به صورت زیر است:

```
def save_idf_dictionary_to_file(idf_dict, output_file):
    try:
        with open(output_file, 'w', encoding='utf-8') as f:
            for term, idf in idf_dict.items():
                f.write(f"{term}: {idf}\n")
    except Exception as e:
        print(f"Error writing to file: {e}")
```

خروجی:

```
3877 realisation: 0.8239087409443188
3878 orking: 1.0
3879 mensional: 0.8239087409443188
3880 noether: 1.0
3881 afterwards: 0.8239087409443188
3882 pale: 1.0
3883 cle: 0.6989700043360189
3884 examination: 1.0
3885 theoretical: 0.2596373105057561
3886 professors: 0.8239087409443188
3887 prize: 1.0
3888 claiming: 1.0
3889 spontaneously: 0.8239087409443188
3890 articl: 1.0
3891 voice: 1.0
3892 hello: 1.0
3893 spacetimes: 0.6989700043360189
3894 google: 0.8239087409443188
3895 note: 0.18708664335714445
3896 resi: 1.0
3897 endeavour: 1.0
3898 unless: 0.6020599913279624
```

گام پنجم

از این جا به بعد میریم سراغ محاسبه با شاخص Cosine:

```
def calculate_cosine_similarity(query_vector, doc_vector):
    intersection = set(query_vector.keys()) & set(doc_vector.keys())
    numerator = sum(query_vector[term] * doc_vector[term] for term in intersection)

    sum_query = sum(value**2 for value in query_vector.values())
    sum_doc = sum(value**2 for value in doc_vector.values())

    denominator = math.sqrt(sum_query) * math.sqrt(sum_doc)

    if not denominator:
        return 0.0
    else:
        return numerator / denominator
```

تابع `calculate_cosine_similarity` برای محاسبه شباهت کسینوسی بین دو بردار (وکتور) طراحی شده است. این دو بردار می‌توانند به عنوان نماینده‌ای از دو سند متنی در فضای برداری باشند. شباهت کسینوسی یک مقدار عددی بین ۰ و ۱ است که نشان‌دهنده میزان مشابهت دو بردار است. هرچه این مقدار به ۱ نزدیک‌تر باشد، شباهت بیشتر است و هرچه به ۰ نزدیک‌تر باشد، اختلاف بیشتر است.

محاسبه اشتراک کلیدها (واژگان)

```
intersection = set(query_vector.keys()) & set(doc_vector.keys())
```

این خط کد، واژگان مشترک بین دو بردار (سند) را پیدا می‌کند. به عبارتی دیگر، کلیدهایی که در هر دو query_vector و doc_vector وجود دارند، در مجموعه intersection قرار می‌گیرند.

```
numerator = sum(query_vector[term] * doc_vector[term] for term in intersection)
```

صورت کسر با استفاده از جمع حاصل ضرب مقادیر واژگان مشترک در هر دو بردار محاسبه می‌شود.

```
sum_query = sum(value**2 for value in query_vector.values())
sum_doc = sum(value**2 for value in doc_vector.values())
```

در این دو خط کد، مجموع مربعات مقادیر هر بردار محاسبه می‌شود. این مقادیر برای محاسبه قدر مطلق هر بردار نیاز است.

```
denominator = math.sqrt(sum_query) * math.sqrt(sum_doc)
```

مخرج کسر با استفاده از ضرب قدر مطلق (Norm) هر دو بردار محاسبه می‌شود. قدر مطلق هر بردار با جذر مجموع مربعات مقادیر آن بردار بدست می‌آید.

```
if not denominator:
    return 0.0
else:
    return numerator / denominator
```

در این بخش، ابتدا بررسی می‌شود که مخرج کسر صفر نباشد. اگر مخرج صفر باشد، شباهت کسینوسی برابر با صفر است. در غیر این صورت، صورت کسر تقسیم بر مخرج شده و شباهت کسینوسی محاسبه می‌شود.

به طور خلاصه، این تابع ابتدا واژگان مشترک بین دو بردار را پیدا کرده، سپس حاصل ضرب مقادیر این واژگان را جمع می‌کند (صورت کسر)، و در نهایت مجموع مربعات مقادیر هر بردار را محاسبه کرده و قدر مطلق هر بردار را بدست می‌آورد (مخرج کسر). با تقسیم صورت بر مخرج، شباهت کسینوسی بدست می‌آید.

cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \bullet \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\vec{q}}{|\vec{q}|} \bullet \frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

$$\cos(\vec{q}, \vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for q, d length-normalized.

و در انتها داکيومنت‌هایی با بیش‌ترین ضریب برای برگرداننده می‌شود.

```
while True:
    query = input("Enter your query (or type 'exit' to quit): ")
    if query.lower() == 'exit':
        break

    query_tokens = tokenize_text(query)
    query_tf = Counter(query_tokens)
    query_vector = {term: (query_tf[term] / len(query_tokens)) * idf_dict.get(term, 0) for term in query_tf}

    cosine_scores = {}
    for doc_name, doc_vector in tf_idf_dict.items():
        cosine_scores[doc_name] = calculate_cosine_similarity(query_vector, doc_vector)

    sorted_cosine_scores = sorted(cosine_scores.items(), key=lambda item: item[1], reverse=True)

    for doc, score in sorted_cosine_scores:
        print(f"{doc}: {score:.6f}")
```

گام ششم:

در این گام با نوشتن چند تست برنامه را تست می‌کنیم:

```
Enter your query (or type 'exit' to quit): hello
2: 0.019424
1: 0.000000
3: 0.000000
4: 0.000000
5: 0.000000
6: 0.000000
7: 0.000000
8: 0.000000
9: 0.000000
10: 0.000000
11: 0.000000
12: 0.000000
13: 0.000000
14: 0.000000
15: 0.000000
16: 0.000000
17: 0.000000
18: 0.000000
19: 0.000000
20: 0.000000
Enter your query (or type 'exit' to quit):
```

در بخش قبل نیز داکيومنت ۲ برگردانده شد.

مثال بعدی:



What an event is not: unravelling the identity of events in quantum theory and gravity

Anne-Catherine de la Hamette^{*,†}, Viktoria Kabel[†], and Ćaslav Brukner
University of Vienna, Faculty of Physics, Vienna Doctoral School in Physics,
and Vienna Center for Quantum Science and Technology (VCQT),
Boltzmannngasse 5, A-1090 Vienna, Austria and
Institute for Quantum Optics and Quantum Information (IQOQI),
Austrian Academy of Sciences, Boltzmannngasse 3, A-1090 Vienna, Austria

```
Enter your query (or type 'exit' to quit): what an event is not
10: 0.275442
20: 0.045069
2: 0.011555
18: 0.006119
11: 0.003701
14: 0.002753
3: 0.002241
16: 0.001203
8: 0.000914
7: 0.000785
9: 0.000501
1: 0.000000
4: 0.000000
5: 0.000000
6: 0.000000
12: 0.000000
13: 0.000000
15: 0.000000
17: 0.000000
19: 0.000000
Enter your query (or type 'exit' to quit):
```

دایکومنت ۱۰ به عنوان اولین دایکومنت آورده شده است.

مثال بعدی:



Diffusion based Zero-shot Medical Image-to-Image Translation for Cross Modality Segmentation

```
Enter your query (or type 'exit' to quit): Diffusion based
17: 0.218979
13: 0.008181
1: 0.003822
5: 0.003406
6: 0.002398
12: 0.001806
14: 0.001729
16: 0.001677
15: 0.000849
4: 0.000457
11: 0.000369
20: 0.000199
18: 0.000191
8: 0.000182
2: 0.000000
3: 0.000000
7: 0.000000
9: 0.000000
10: 0.000000
19: 0.000000
Enter your query (or type 'exit' to quit):
```

داکیومنت ۱۷ در اولین داکيومنت است.

مثال بعدی:



C-XGBOOST: A TREE BOOSTING MODEL FOR CAUSAL EFFECT ESTIMATION

```
23: 0.000000
Enter your query (or type 'exit' to quit): C-XGB00ST
13: 0.366416
1: 0.000000
2: 0.000000
3: 0.000000
4: 0.000000
5: 0.000000
6: 0.000000
7: 0.000000
8: 0.000000
9: 0.000000
10: 0.000000
11: 0.000000
12: 0.000000
14: 0.000000
15: 0.000000
16: 0.000000
17: 0.000000
18: 0.000000
19: 0.000000
20: 0.000000
```

چون کلمه یکم عجیب و ی جورایی یونیک بوده فقط داکيومنت ۱۳ بازگرداننده شده است. تست‌های دیگر نیز درست بازگرداننده شده است.