به نام خدا

تمرین ٤

درس بازیابی اطلاعات

استاد : دکتر رضاپور

کسری صمدی<۹۹۳۶۲۳۰۳۰>

بهمن ۱٤۰۲

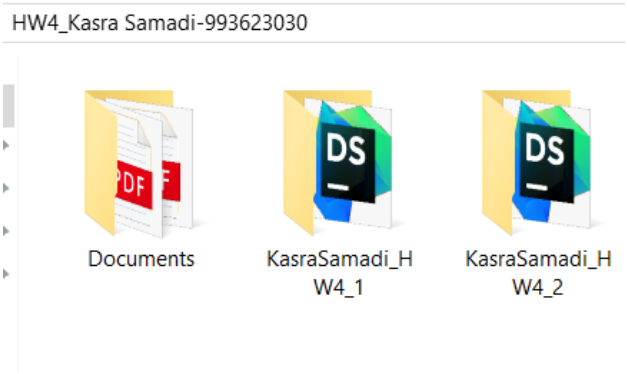
شما می بایست دو سیستم بازیابی توسعه دهید:

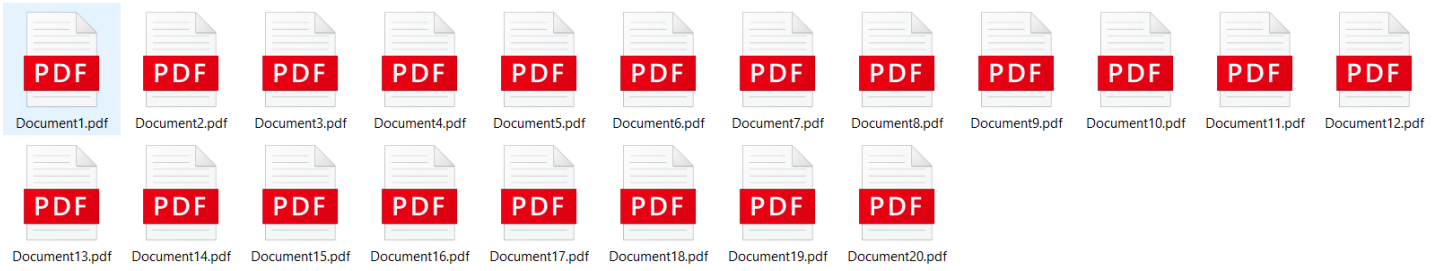
۱- با استفاده از TF-IDF Score یک سیستم بازیابی روی اسناد جمع آوری شده در تکلیف قبل توسعه دهید.

۲- سیستم بازیابی دیگری نیز با استفاده از شاخص Cosine توسعه دهید که بر اساس وکتور وزن‌های TF-IDF کار کند.

دقت داشته باشید که کدها می‌بایست با زبان پایتون نوشته شود. برنامه می‌بایست قابل تست باشد وگرنه نمره‌ای 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.

همانطور که در تمرین قبل خواسته شده بود، ابتدا ۲۰ مقاله با فرمت pdf با نام‌های Document1 تا Document20 دانلود و نام‌گذاری شده و در پوشه‌ای با نام Documents در کنار فایل اجرایی کد پایتون ذخیره شده است.





تمرین ۱:

حال به سراغ بخش کد پایتون می‌رویم :

در این بخش ابتدا کتابخانه‌های موردنیاز را ایمپورت می‌کنیم.

TF-IDF Score

```
In [1]: import nltk
        import PyPDF2
        import string
        import math
        from collections import defaultdict
        from ordered_set import OrderedSet
```

کتابخانه nltk : برای پردازش زبان طبیعی استفاده می‌شود و در این تمرین، برای tokenize کردن جملات موجود در مقالات استفاده شده است.

کتابخانه pyPDF2 : برای استخراج متن از مقالات کاربرد دارد.

کتابخانه string : برای کار با دیتاتایپ استرینگ استفاده می‌شود و در این تمرین، برای حذف علائم نگارشی (Remove punctuation) استفاده شده است.

کتابخانه math : برای انجام عملیات ریاضی استفاده می‌شود.

کتابخانه Defaultdict : نوعی از ساختمان داده دیکشنری است که مزیت‌های بیشتری نسبت به دیکشنری معمولی دارد.

کتابخانه ordered_set : یک داده ساختار ست (مجموعه) است که عناصر با ترتیب در آن ذخیره می‌شوند. همانطور که می‌دانید در مجموعه‌ها (Set)، عناصر تکراری ذخیره نمی‌شوند.

```
documents_folder_name = fr"..\Documents"
documents_name = "Document"
documents_count = 20
all_doc_ids = list()
```

حال در این قسمت، متغیرهایی را تعریف می‌کنیم که به ترتیب شامل نام فولدر ذخیره کننده مقالات (Documents)، نام ثابت برای هر مقاله (Document) و تعداد مقالات (۲۰) است.

متغیر all_doc_ids کل نام‌های داکیومنت‌ها را در خود ذخیره می‌کند.

در زیر سایر متغیرهای مورد نیاز این تمرین به همراه توضیحات کاربرد آن‌ها آورده شده است.

```
'''
dictionary to store the total words in each documents
# keys document names and values are the total words are there in the document
'''
documents_words_num = defaultdict(int)

'''
a dictionary to store the term frequencies (TF)
keys are words and values are defaultdict that keys are document name and
values are the word TF in the document
'''
tf = defaultdict(lambda: defaultdict(float))
```

```
'''
a dictionary to store the Inverse Document Frequency (IDF)
keys are words and values are the word idf in the collection
'''
idf = defaultdict(float)

'''
a dictionary to store the document frequencies (DF)
keys are words and values are the set of documents name that contain the word
'''
df = defaultdict(OrderedSet)

'''
a dictionary to store the TF-IDF scores
keys are words and values are defaultdict that keys are document name and
values are the word TF_IDF in the document
'''
tfidf = defaultdict(lambda: defaultdict(float))
```

```
for i in range(documents_count):
    doc_name = f"{documents_name}{i + 1}"
    doc_address = fr"{documents_folder_name}\{doc_name}.pdf"
    all_doc_ids.append(doc_name)

    number_of_words_in_doc = 0
    with open(doc_address, "rb") as pdf_file:
        read_pdf = PyPDF2.PdfReader(pdf_file)
        number_of_pages = len(read_pdf.pages)
```

در این قسمت، نام‌های مقالات (Document1 تا Document20) همچنین آدرس آن‌ها (..\Documents\Document1..20) به ازای هر مقاله ساخته می‌شود و با استفاده از کتابخانه PyPDF2 مقاله موردنظر خوانده شده و تعداد صفحات آن بدست می‌آید. همچنین نام‌های داکیومنت‌ها در لیست all_dic_ids ذخیره می‌شود. متغیر number_of_words_in_doc برای ذخیره تعداد کل کلمات موجود در هر داکیومنت تعریف شده است.

```
number_of_pages = len(read_pdf.pages)
for page_num in range(number_of_pages):
    page = read_pdf.pages[page_num]
    page_content = page.extract_text()

    lower_text = page_content.lower()
    text_without_punctuation_marks = lower_text.translate(str.maketrans('', '', string.punctuation))

    tokens = nltk.word_tokenize(text_without_punctuation_marks)
```

حال با استفاده از تعداد صفحات بدست آمده برای مقاله موردنظر، متن هر صفحه از آن استخراج می‌شود. سپس متن بدست آمده به حروف کوچک (lowercase) تبدیل شده و علائم نگارشی (punctuation marks) از آن حذف می‌شود.

بعد از آن، با استفاده از کتابخانه nltk، متن موردنظر tokenize (کلمه کلمه) می‌شود و کلمات به دست آمده از متن هر صفحه، در آرایه tokens ذخیره می‌شود.

```
        tokens = nltk.word_tokenize(text_without_punctuation_marks)

        for term in tokens:
            number_of_words_in_doc += 1
            tf[term][doc_name] += 1
            df[term].append(doc_name)

 documents_words_num[doc_name] = number_of_words_in_doc
```

حال به ازای هر کلمه موجود در آرایه tokens ، متغیر number_of_words_in_doc و دیکشنری‌های tf و df مقداردهی می‌شوند. در نهایت تعداد کل کلمات موجود در داکومنت، در دیکشنری documents_word_num ذخیره می‌شود.

تعداد کل کلمات موجود در هر داکیومنت به شرح زیر است:

```
for iden, words in documents_words_num.items():
    print(iden, words)
```

```
Document1 7621
Document2 1394
Document3 7380
Document4 4005
Document5 3434
Document6 4474
Document7 3351
Document8 4009
Document9 7129
Document10 4107
Document11 10184
Document12 7497
Document13 58356
Document14 4468
Document15 7689
Document16 1883
Document17 3012
Document18 2986
Document19 6825
Document20 3643
```

```
# calculate tf and idf for each words
for term, doc_tf in tf.items():
    for doc_id, term_freq in doc_tf.items():
        # calculate tf
        tf[term][doc_id] /= documents_words_num[doc_id]

        # calculate idf
        idf[term] = math.log(documents_count / (len(df[term]) + 1))
        print(term, doc_id, tf[term][doc_id], idf[term])
```

حال به ازای هر ترمی که در دیکشنری tf است، مقادیر tf و idf محاسبه می‌شود و به ترتیب در دیکشنری‌های tf و idf با کلیدهای مربوطه ذخیره می‌شوند.

در تصویر زیر می‌توان قسمتی از مقادیر tf و idf را به ازای هر ترم در داکیومنت‌هایی که شامل ترم موردنظر هستند، مشاهده کرد :

TF IDF

```
efficient Document1 0.0030179766434851 07 0.7985076962177716
efficient Document3 0.00027100271002710027 0.7985076962177716
efficient Document4 0.00024968789013732833 0.7985076962177716
efficient Document5 0.0037856726849155504 0.7985076962177716
efficient Document8 0.0002494387627837366 0.7985076962177716
efficient Document13 0.00022277058057440537 0.7985076962177716
efficient Document14 0.00022381378692927484 0.7985076962177716
efficient Document15 0.00013005592404734037 0.7985076962177716
video Document1 0.012203122949744127 1.3862943611198906
video Document2 0.0014347202295552368 1.3862943611198906
video Document3 0.0008130081300813008 1.3862943611198906
video Document6 0.0049172999552972736 1.3862943611198906
object Document1 0.009841228185277523 1.0498221244986776
object Document3 0.0004065040650406504 1.0498221244986776
object Document6 0.00022351363433169424 1.0498221244986776
object Document8 0.0004988775255674732 1.0498221244986776
object Document9 0.00014027212792818068 1.0498221244986776
object Document11 9.819324430479184e-05 1.0498221244986776
```

```
df["efficient"]
```

```
OrderedSet(['Document1', 'Document3', 'Document4', 'Document5', 'Document8', 'D
ocument13', 'Document14', 'Document15'])
```

در دیکشنری df کلیدها ترم‌ها هستند و مقادیر یک orderedSet است که شامل نام مقالاتی است که ترم مورد نظر در آن مقالات وجود دارد.

```
# Calculate the TF-IDF scores
for term, doc_tf in tf.items():
    for doc_id, term_freq in doc_tf.items():
        tfidf[term][doc_id] = term_freq * idf[term]
```

حال با توجه به مقادیر موجود در دیکشنری‌های tf و idf ، مقادیر tf_idf به ازای در ترم محاسبه شده و در دیکشنری tfidf ذخیره می‌شود.

قسمتی از مقادیر ذخیره شده در دیکشنری tfidf در تصویر زیر آمده است:

```
for term, doc_id in tfidf.items():
    print(term+ " => ",tfidf[term])
```

```
efficient =>  defaultdict(<class 'float'>, {'Document1': 0.002409877576828336,
'Document3': 0.00021639774965251263, 'Document4': 0.00019937770192703411, 'Doc
ument5': 0.003022888774266462, 'Document8': 0.00019917877181785276, 'Document1
3': 0.0001778840230795639, 'Document14': 0.00017871703138267047, 'Document15':
0.00010385065629051524})
video =>  defaultdict(<class 'float'>, {'Document1': 0.01691712053328301, 'Doc
ument2': 0.00198894456401706, 'Document3': 0.0011270685862763339, 'Document6':
0.0068168251999637})
object =>  defaultdict(<class 'float'>, {'Document1': 0.010331539081144314, 'D
ocument3': 0.00042675696117832425, 'Document6': 0.00023464955844851982, 'Docum
ent8': 0.0005237326637558881, 'Document9': 0.00014726078334951294, 'Document11
': 0.00010308544034747424})
segmentation =>  defaultdict(<class 'float'>, {'Document1': 0.011848571096240023, 'Document4': 0.00030061742929486543, 'Document5': 0.0007012072244181341, '
Document6': 0.004843878068365411, 'Document7': 0.009700765657057677})
```

برای محاسبه نتیجه و بازیابی نام داکیومنت‌ها با ورودی کوئری مورد نظر، تابع query_index با ورودی query مطابق تصویر زیر تعریف می‌شود:

```
def query_index(query):

    query_terms = query.lower().split() # Tokenize the query

    '''
    a dictionary to store the document score
    keys are document names and values are the document score
    '''
    doc_scores = defaultdict(float)

    '''
    a dictionary to store TF for each term are there in query
    keys are terms in query and values are term TF
    '''
    tf_query_term = defaultdict(float)
```

```
'''
a dictionary to store count duplicate terms are there in query
keys are terms in query and values are counts
'''
term_count = defaultdict(float)

'''
a dictionary to store TF_IDF for each term are there in query
keys are terms in query and values are term TF_IDF
'''
tfidf_query_term = defaultdict(float)
```

حال مطابق تصویر زیر tf_idf برای هر ترم موجود در query محاسبه می‌شود و در دیکشنری tfidf_query_term ذخیره می‌شود :

```
# calculate tf_idf for query tokens
for term in query_terms:
    term_count[term] += 1
for term, count in term_count.items():
    tf_query_term[term] = count / len(query_terms)
for term, tf_value in tf_query_term.items():
    idf_value = idf[term]
    tfidf_query_term[term] = tf_value * idf_value
```

در نهایت به ازای هر داکیومنت و به ازای ترم‌های موجود در query ، امتیاز کوئری برای هر داکیومنت محاسبه شده و در دیکشنری doc_scores به صورت نام داکیومنت و امتیاز محاسبه شده کوئری در آن داکیومنت، ذخیره می‌شود.
داکیومنت‌ها با توجه به امتیازهایشان، مرتب شده و نتیجه ریترن می‌شود. کد این عملیات در تصویر زیر آمده است:

```
# calculate score of each document for user query
query_terms = set(query_terms) # ignore same and duplicate words
document_scores = []

for document_name in all_doc_ids:
    document_score = 0
    for term in query_terms:
        query_term_idf = idf.get(term, 0)
        query_term_tf = tf.get(term, dict()).get(document_name, 0)
        query_term_tfidf_in_document = query_term_tf * query_term_idf
        new_document_score = query_term_tfidf_in_document * tfidf_query_term[term]
        document_score += new_document_score
    doc_scores[document_name] = document_score

# Sort the documents by their relevance scores
sorted_docs = sorted(doc_scores.items(), key=lambda x: x[1], reverse=True)
return sorted_docs
```

نتایج به ازای بعضی از کوئری‌های ورودی به شرح زیر است:

کوئری ورودی : the

```
n = 20 # return Top n of documents
query = "the"
# query = "kasra"
# query = "γSF"

results = query_index(query)
print(f"Top {n} most relevant documents:")
for doc_id, score in results[:n]:
    if score !=0:
        print(f"{doc_id}: {score}")
```

```
Top 20 most relevant documents:
Document11: 0.0002232284479865009 3
Document20: 0.0001875371381685967
Document8: 0.0001721973646064450 7
Document16: 0.0001706663920110556 5
Document18: 0.0001618343818804644
Document14: 0.0001609008496292300 2
Document12: 0.0001590797038761832 6
Document15: 0.0001523210064875308 6
Document17: 0.0001477920924237000 7
Document4: 0.0001474055105320031 8
Document9: 0.0001435834551076527 6
Document19: 0.0001419568364409983 3
Document13: 0.0001317185945994781 2
Document10: 0.0001240376785029335 7
Document2: 0.0001161209814478113
Document1: 0.0001130735865797412
Document6: 0.0001117346502308510 7
Document3: 0.0001022509753304338 4
Document5: 9.77424859857014e-05
Document7: 8.382472519315295e-05
```

کوئری ورودی : kasra

```
n = 20 # return Top n of documents
# query = "the"
query = "kasra"
# query = "γSF"

results = query_index(query)
print(f"Top {n} most relevant documents:")
for doc_id, score in results[:n]:
    if score !=0:
        print(f"{doc_id}: {score}")
```

```
Top 20 most relevant documents:
```

داکیومنتی شامل کلمه kasra وجود ندارد.

کوئری ورودی : γSF

```
n = 20 # return Top n of documents
# query = "the"
# query = "kasra"
query = "γSF"

results = query_index(query)
print(f"Top {n} most relevant documents:")
for doc_id, score in results[:n]:
    if score !=0:
        print(f"{doc_id}: {score}")
```

```
Top 20 most relevant documents:
Document17: 0.012321808357685522
```

در ۲۰ داکیومنت، کلمه γSF فقط در Document17 وجود دارد.

تمرین ۲:

در این بخش از تمرین، می‌خواهیم سیستم بازیابی دیگری با استفاده از شاخص Cosine توسعه دهیم که بر اساس وکتور وزن‌های TF-IDF کار کند.

در این تمرین کل عملیات ساخت دیکشنری tfidf عینا مشابه تمرین ۱ است که توضیحات آن از صفحه ۲ تا صفحه‌ی ۸ این گزارش آورده شده است، لذا از تکرار توضیحات آن صرفه نظر می‌کنیم.

در ادامه تابع get_query_tfidf با ورودی query تعریف می‌شود که مقدار tf_idf به ازای هر ترم موجود در کوئری محاسبه شده و حاصل به صورت یک دیکشنری که کلیدهای آن، ترم‌های کوئری و مقادیر آن امتیاز tf_idf آن ترم است، ریترن می‌شود.

```
def get_query_tfidf(query):
    query_terms = query.lower().split() # Tokenize the query

    '''
    a dictionary to store TF for each term are there in query
    keys are terms in query and values are term TF
    '''
    tf_query_term = defaultdict(float)

    '''
    a dictionary to store count duplicate terms are there in query
    keys are terms in query and values are counts
    '''
    term_count = defaultdict(float)

    '''
    a dictionary to store TF_IDF for each term are there in query
    keys are terms in query and values are term TF_IDF
    '''
    tfidf_query_term = defaultdict(float)

    '''
    a dictionary to store TF_IDF score for each query term
    keys are query terms and values are terms TF_IDF
    '''
    result = defaultdict(float)
```

```
# calculate tf_idf for query tokens
for term in query_terms:
    term_count[term] += 1
for term, count in term_count.items():
    tf_query_term[term] = count / len(query_terms)
for term, tf_value in tf_query_term.items():
    idf_value = idf[term]
    tfidf_query_term[term] = tf_value * idf_value

query_terms = set(query_terms) # ignore same and duplicate words
for term in query_terms:
    result[term] = tfidf_query_term[term]
return result
```

حال با توجه به قطعه کدی که در تصویر زیر آورده شده است، با تابع cosine_similarity تشابه بردار tfidf ترم‌های کوئری با بردار tfidf همین ترم‌ها که در کالکشن داکیومنت‌ها محاسبه شده است، بدست می‌آید و نتیجه به صورت دیکشنری similarities که کلیدهای آن، نام‌های داکیومنت‌ها و مقادیر آن، امتیاز شاخص cosine است، ریترن می‌شود. این دیکشنری بر اساس امتیازهای Cosine مرتب شده است.

```
def cosine_similarity(query_dic_tfidf, tfidf):
    similarities = defaultdict(float)
    query_magnitude = math.sqrt(sum(score ** 2 for score in query_dic_tfidf.values()))

    for word, doc_tfidf in tfidf.items():
        if word in query_dic_tfidf:
            for doc_name, doc_score in doc_tfidf.items():
                doc_magnitude = math.sqrt(sum(val ** 2 for val in doc_tfidf.values()))
                if doc_magnitude > 0:
                    similarities[doc_name] = \
                    query_dic_tfidf[word] * doc_score / (query_magnitude * doc_magnitude)
    similarities = sorted(similarities.items(), key=lambda x: x[1], reverse=True)
    return similarities
```

نتایج به ازای بعضی از کوئری‌های ورودی به شرح زیر است:

کوئری ورودی : the

```
n = 20 # return Top n of documents
query = "the"
# query = "kasra"
# query = "γSF"

query_tfidf_results = get_query_tfidf(query)
results = cosine_similarity(query_tfidf_results, tfidf)
if len(results) > 0:
    print(f"Top {n} most relevant documents:")
    for doc_id, score in results[:n]:
        if score != 0:
            print(f"{doc_id}: {score}")
else :
    print(f"'{query}' NOT FOUND!")
```

```
Top 20 most relevant documents:
Document11: 0.34152145868281575
Document20: 0.2869166432963464
Document8: 0.2634480312531049
Document16: 0.26110576708965727
Document18: 0.24759350639838848
Document14: 0.24616527760835408
Document12: 0.24337907199852002
Document15: 0.23303881200124632
Document17: 0.22610994002601248
Document4: 0.225518501019275
Document9: 0.2196710655536588
Document19: 0.21718247064218002
Document13: 0.20151878924492878
Document10: 0.18976760926328573
Document2: 0.17765570349767826
Document1: 0.17299343598691383
Document6: 0.1709449717383784
Document3: 0.15643571669101114
Document5: 0.14953809288294673
Document7: 0.12824504528823746
```

کوئری ورودی : kasra

```
n = 20 # return Top n of documents
# query = "the"
query = "kasra"
# query = "γSF"

query_tfidf_results = get_query_tfidf(query)
results = cosine_similarity(query_tfidf_results, tfidf)
if len(results) > 0:
    print(f"Top {n} most relevant documents:")
    for doc_id, score in results[:n]:
        if score != 0:
            print(f"{doc_id}: {score}")
else :
    print(f"'{query}' NOT FOUND!")
```

```
'kasra' NOT FOUND!
```

داکیومنتی شامل کلمه kasra وجود ندارد.

کوئری ورودی : γSF

```
n = 20 # return Top n of documents
# query = "the"
# query = "kasra"
query = "γSF"

query_tfidf_results = get_query_tfidf(query)
results = cosine_similarity(query_tfidf_results, tfidf)
if len(results) > 0:
    print(f"Top {n} most relevant documents:")
    for doc_id, score in results[:n]:
        if score != 0:
            print(f"{doc_id}: {score}")
else :
    print(f"'{query}' NOT FOUND!")
```

```
Top 20 most relevant documents:
Document17: 1.0
```

در ۲۰ داکیومنت، کلمه γSF فقط در Document17 وجود دارد.