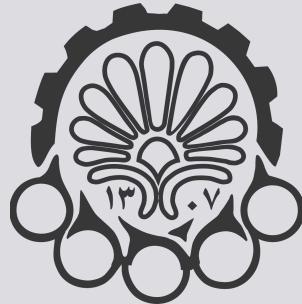


به نام خدا



بازیابی اطلاعات

گزارش پروژه

علی بازشوشتی

9923008

فهرست

5	1- ساخت شاخص مکانی
5	5- مجموعه داده
5	read_docs تابع
6	6- پیش پردازش اسناد
6	نرمال سازی متن
14	پیش پردازش متن - کلاس DataPreprocessing
19	19- ساخت شاخص مکانی
20	2- پاسخ دهی به پرسمن در فضای برداری
20	20- مدل سازی اسناد در فضای برداری و ساخت Champions lists
21	21- پاسخ دهی به پرسمن در فضای برداری و شباهت کسینوسی
25	25- گزارش چند پرسمن از موتور جستجو

1- ساخت شاخص مکانی

1.1- مجموعه داده

مجموعه داده شامل 12 هزار سند از اخبار خبرگزاری‌های مختلف می‌باشد. از عنوان، متن (content) و url خبر برای بازیابی استفاده می‌شود.

تابع read_docs

این تابع فایل json دیتاست را در برنامه باز می‌کند و آن را می‌خواند و اطلاعات مورد نیاز مرتبط را نظیر شماره سند، عنوان، متن و url را در ساختمان داده دیکشنری (map) ذخیره می‌کند که در آن key شماره سند و آن نیز خود یک دیکشنری دیگر است که عنوان، متن و url خبر در آن ذخیره شده است. در واقع همان فایل json را در قالب یک دیکشنری در برنامه داریم که تنها شامل اطلاعات مورد نیازمان است. همچنین یک لیست برای متن و urlها در نظر گرفته شده است که آنها را به صورت جداگانه نیز در یک ساختمان داده مجزا داشته باشیم.

```
1 def read_docs():
2     docs = {}
3     contents = []
4     urls = []
5     with open("../IR_data_news_12k.json", 'r') as file:
6         articles = json.load(file)
7         for doc_id in articles.keys():
8             # index of files
9             index = str(doc_id)
10            # extract and save url, title and content of each doc
11            docs[index] = {'title': articles[index]['title'],
12                         'content': articles[index]['content'],
13                         'url': articles[index]['url'],
14                         }
15
16            contents.append(docs[index]['content'])
17            urls.append(docs[index]['url'])
18
19     return docs, contents, urls
```

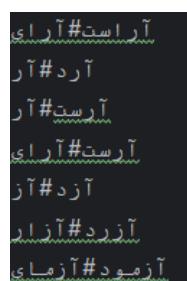
2.1- پیش پردازش اسناد

نرم‌ال سازی متن

کلاس DataNormalization برای نرم‌ال سازی متن پیاده شده است. در تابع `__init__` مقداردهی‌هایی جهت نرم‌ال سازی انجام می‌شود که شامل دو الگوی متنی در قالب یک tuple یک هستند و هدف آن است که در ادامه هرجای الگوی اول (قدیمی) مشاهده شد با الگوی دوم (جدید) جایگزین شود. یکی به عنوان مثال برخی از حروف عربی به همراه جایگزین مدنظرشان آورده شده است تا در نرم‌ال سازی مورد استفاده قرار گیرند. یک تابع پر استفاده در این کلاس، تابع `regex_replace` می‌باشد که دو `pattern` و یک رشته به عنوان ورودی می‌گیرد و در آن رشته، الگوی دوم (الگوی جدید و مدنظر) را با الگوی اول (الگوی قدیمی) جایگزین می‌کند.

```
@staticmethod
def regex_replace(patterns: list, text: str) -> str:
    for pattern, repl in patterns:
        text = re.sub(pattern, repl, text)
    return text
```

یک فایل `verbs.dat` نیز موجود است که شامل بن‌ماضی و مضارع افعال می‌باشد و برای پردازش افعال مانند درست کردن نیم فاصله "می" و "نمی" قبل از افعال به کار می‌رود. بن‌های ماضی و مضارع به وسیله `#` در فایل `verbs.dat` از یکدیگر جدا شده‌اند. نمونه‌ای از این فایل در تصویر زیر آمده است:



در قطعه کد زیر این بن‌ها از فایل `verbs.dat` جدا و ذخیره می‌شوند.

```
with Path('verbs.dat').open(encoding="utf8") as verbs_file:
    verbs = list(
        reversed([verb.strip() for verb in verbs_file if verb]),
    )
    self.present_bons = {verb[1:]:split("#")[0].strip() for verb in verbs[1:]}
    self.past_bons = {verb.split("#")[1] for verb in verbs}
```

کارهایی که توسط کلاس DataNormalization قابل انجام هستند به شرح زیر است:

- فاصله‌گذاری صحیح

در تابع spacing_correction تابع regex_replace سه بار فراخوانی می‌شود و هر بار به عنوان ورودی یک رشته به همراه الگویی که در __init__ تعریف شده است را دریافت می‌کند و الگوها را جایگزین می‌کند. این الگوها مرتبط با فاصله گذاری می‌باشند.

```
def spacing_correction(self, text: str) -> str:
    text = self.regex_replace(self.extra_space_patterns, text)
    text = self.regex_replace(self.punctuation_spacing_patterns, text)
    text = self.regex_replace(self.spacing_patterns, text)
    return text
```

- الگوهای مرتبط با فاصله اضافه:

```
# extra space patterns
self.extra_space_patterns = [
    (r"\s{2,}", " "),           # remove extra spaces
    (r"\n{3,}", "\n\n"),        # remove extra newlines
    (r"\u200c{2,}", "\u200c"),  # remove extra ZWNJs
    (r"\u200c{1,} ", " "),     # remove unneeded ZWNJs before space
    (r" \u200c{1,} ", " "),    # remove unneeded ZWNJs after space
    (r"\b\u200c*\B", " "),     # remove unneeded ZWNJs at the beginning of words
    (r"\B\u200c*\b", " "),     # remove unneeded ZWNJs at the end of words
    (r"\r", " "),              # remove keshire, carriage returns
]
```

- الگوهای مرتبط با فاصله اطراف علائم نگارشی و اعداد:

```
# punctuation marks
self.punc_after = r"\.!:?\)\]\)\}\"
self.punc_before = r"\{[\(\["
```

- الگوهای فاصله گذاری پیشوندها و پسوندها مانند می، نمی، تر، ترین و ...: مورد اول space را با non-breaking char جایگزین می‌کند.

مورد دوم "ی" ربط را با نیم فاصله به کلمه قبل متصل می‌کند.

مورد سوم "می" و "نمی" را با نیم فاصله به کلمه بعد متصل می‌کند.

مورد چهارم پسوندهای خاص فارسی را با یک کاراکتر zero-width non-joiner (ZWNJ) قبل از خود پسوند جایگزین می‌کند مثل "بزرگ تر" تبدیل به "بزرگتر" می‌شود. "تر" و "ترین" و "های" و "گری" از مواردی هستند که توسط این الگو هندل می‌شوند.

مورد پنجم در کلماتی مانند "خانه ام" کاربرد دارد که کلمه اصلی در انتهایش یک "ه" دارد و ضمیر متصل به آن با space قرار گرفته. در این شرایط ضمیر با نیم فاصله به کلمه قبل متصل خواهد شد.

مورد اخر برای کلماتی مثل "کلمهها" است که بین دو "ه" نیم فاصله قرار می‌دهد و توکن به "کلمهها" تبدیل خواهد شد.

```
self.spacing_patterns = [
    (r"\xa0", " "), # remove no-break char
    (r"([^\u0628]) \u0628", r"\1\u0628"), # fix 'ی' space
    (r"(\u0628| ) (\u0628|\u062a)", r"\1\2\u0628\u062a"), # fix 'می' and 'نمی' space. put ZWNJ after them
    # put zwnj before تر، تری، ترین، گر، گری، ها، های
    (r"(?<=[^\u0628\u062a\d] + self.punc_after + self.punc_before + r" ){2} (= [ \u0628 + self.punc_after + self.punc_before + r" ]|$)", r"\u0628\u062a\u0628\1"),
    # fix suffix spacing
    (r"([^\u0628\u062a\d]) (= [ \u0628 + self.punc_after + r" ]|$)", r"\1\u0628\u062a\u0628\2"), # fix verb conjugation spacing
    (r"(\u0628)(\u062a)", r"\1\u0628\u062a\2"),
]
```

تعویض یونیکد •

تبدیل "ی"ها و "ک"ها و "آ" و همچنین تبدیل ده کلمه که به دو شکل نوشته می‌شوند و نرمالسازی آنها

```
# replace special characters
def unicode_replacement(cls, text: str) -> str:
    for old, new in cls.unicode_replacements:
        text = re.sub(old, new, text)
    return text
```

لیست این کاراکترها و کلمات در تصویر زیر موجود است. به عنوان مثال حرف "ا" با حرف "ا" جایگزین می‌شود.

- حذف بعضی کاراکترها
حذف فتحه، کسره، ضمه، تنوین، تشدید، الف کشیده، سکون، همزه و همچنین تمام علائم نشانه‌گذاری مانند !، <، .. .
 - این کار در دو مرحله انجام می‌شود. ابتدا علائم نگارشی حذف می‌شوند و سپس برخی حروف عربی حذف می‌شوند.

لیست حروف عرب، حذف شده:

```
# fathe kasre ,....  
self.arabic_patterns = [  
    ("[\u064b\u064c\u064d\u064e\u064f\u0650\u0651\u0652]", ""),  
    ]
```

لیست علائم نگارشی حذف شده:

• تبدیل اعداد انگلیسی به فارسی

تابع `persian_number` ارقام غیرفارسی را به فارسی تبدیل می‌کند. برای این تبدیل از دو رشته هم طول استفاده می‌کنیم که کارکترهای هم‌مکان ترجمه یکدیگرند و باهم جایگزین می‌شوند. با استفاده از این دو رشته یک جدول ترجمه ساخته می‌شود و متن توسط این جدول ترجمه پردازش شده و در صورت `match` شدن یک حرف با این جدول، با ترجمه‌اش که عدد به فارسی است جایگزین می‌شود. رشته‌های استفاده شده در ساخت جدول نیز در تصویر زیر مشخص است.

```
self.number_not_persian = "0123456789%۰۱۲۳۴۵۶۷۸۹"
self.number_persian = "%۰۱۲۳۴۵۶۷۸۹۰۱۲۳۴۵۶۷۸۹"
```

```
# convert numbers to persian numbers
def persian_number(cls, text: str) -> str:
    translation_table = str.maketrans(
        cls.number_not_persian,
        cls.number_persian )
    translated_text = text.translate(translation_table)
    return translated_text
```

• جدا کردن "می" و "نمی" از افعال

در این تابع ابتدا تمام کلماتی که با "می" یا "نمی" شروع می‌شوند جدا می‌شوند. "می" و "نمی" از این کلمات حذف شده و بررسی می‌شود آیا بخش باقی مانده با یک بن فعل ماضی یا مضارع شروع می‌شود یا خیر. در صورتی که ادامه کلمه با بن فعل شروع شود یعنی با احتمال بالا آن کلمه فعل بوده که "می" یا "نمی" به شکل چسبیده به آن نوشته شده بوده، بنابراین بین "می" و ادامه‌ی کلمه نیم فاصله قرار می‌دهیم.

```
# separate mi in start of verbs
def separate_mi(cls, text:str) -> str:
    matches = re.findall(cls.mi_patterns, text)
    for m in matches:
        r = re.sub("(ن؟می)", "\1ZNJ", m)
        # remove mi from token to check it contains the bon of a verb or not
        x = re.sub("(ن؟می)", "", m)
        for verb in cls.present_bons:
            if verb in x:
                text = text.replace(m, r)
    for verb in cls.past_bons:
        if verb in x:
            text = text.replace(m, r)
    return text
```

normalize تابع •

در نهایت تابع normalize را داریم که متن ورودی را میگیرد و تمام پردازش‌های بالا را روی آن انجام می‌دهد و توابع لازم را روی آن صدا می‌زند.

```
def normalize(cls, text:str) -> str:
    text = cls.remove_special_chars(text)
    text = cls.seperate_mi(text)
    text = cls.persian_number(text)
    text = cls.unicode_replacement(text)
    text = cls.spacing_correction(text)
    return text
```

پیش پردازش متن - کلاس DataPreprocessing

کلاس DataPreprocessing وظیفه پیش پردازش متن را دارد و اقداماتی نظیر توکن کردن متن ورودی، نرم‌السازی، ریشه‌یابی و حذف کلمات پر تکرار را انجام می‌دهد.

- متغیرهای کلاس و تابع __init__
 - کلاس after_verbs و before_verbs که شامل افعال پیشوندی و افعال پسوندی برای افعال مرکب هستند و در واقع برای بخش اضافی به کار رفته‌اند.

```
before_verbs = {
    "خواهیم",
    "خواهی",
    "خواهد",
    "خواهیم",
    "خواهیم",
    "خواهید",
    "خواهند",
    "نخواهیم",
```

```
after_verbs = {
    "ام",
    "ای",
    "است",
    "ایم",
    "اید",
    "اند",
    "یکندیم",
```

تابع __init__ نیز همانطور که از کامنت‌ش مشخص است با استفاده از بن‌های ماضی موجود در verbs.dat، افعالی که به صورت بن‌ماضی + "ه" هستند را تولید می‌کند. مانند "گذشته"

```
def __init__(self):
    # save terms like گفتگه، خوردگی which are bon mazi + .
    with Path('verbs.dat').open(encoding="utf8") as verbs_file:
        verbs = list(
            reversed([verb.strip() for verb in verbs_file if verb]),
        )
        DataPreprocessing.verbe = {(verb.split("#")[0] + 'ه') for verb in verbs}
```

• **تابع preprocess**

این تابع وظیفه پیش پردازش متن را دارد و تمامی اقدامات لازم که ذکر شدند را با فراخوانی توابع متناظر انجام می‌دهد. در ادامه به توضیح هر یک از توابعی که در preprocess فراخوانی می‌شوند پرداخته می‌شود.

این تابع کل مجموعه اسناد را می‌گیرد و سپس متن (content) هر خبر را پیش پردازش می‌کند. این کار را به ترتیب با حذف علائم نگارشی، نرمال سازی، توکنایز کردن و ریشه یابی انجام می‌دهد و درنهایت توکن‌های پردازش شده را به جای متن خبر قرار می‌دهد.

همچنین توکن‌های پیش پردازش شده را در لیست tokens ذخیره می‌کنیم تا در ادامه برای حذف k کلمه با بیشترین تکرار (stop words) مورد استفاده قرار گیرد. تابع Top_K_Frequent فراخوانی می‌شود تا K کلمه پرتکرار استخراج شوند و سپس در ادامه حذف شوند.

```
# preprocess all given docs
def preprocess(self, docs):
    tokens = []
    self.print_top_k()
    counter = 0
    for idx in docs.keys():
        content = docs[str(idx)]['content']
        punctuated_content = self.Remove_Punctuations(content)
        normalized_content = self.Normalization(punctuated_content)
        all_tokens = self.Tokenization(normalized_content)
        stemmed_tokens = self.Stemming(all_tokens)
        docs[str(idx)]['content'] = stemmed_tokens
        tokens += stemmed_tokens
        counter += 1
        # print progress
        if counter % 1000 == 0:
            print(counter, ' docs processed')
    # save top k frequent
    self.top_k = self.Top_K_Frequent(tokens, 20)
    # remove stop words from doc tokens
    for doc_id, doc_content in docs.items():
        docs[doc_id]['content'] = [token for token in doc_content['content'] if token not in
                                  self.top_k]
    return docs
```

- تابع `:remove_punctuation` ●
وظیفه‌ی تابع حذف general علائم نگارشی از متن ورودی است. در پایتون، کتابخانه `string` مجموعه‌ای از علائم نگارشی را در متغیر `punctuation` تعریف میکند. با استفاده از این متغیر، میتوان به طور خودکار علائم نگارشی را در یک متن شناسایی و حذف کرد.

- هدف این تابع نرمال کردن متن ورودی است. در پخش قبلي به تفصيل نحوه کارکرد اين تابع شرح داده شد.

```
@staticmethod
def Normalization(text):
    my_normalizer = DataNormalization()
    return my_normalizer.normalize(text)
```

- تابع Tokenization و tokenize در خط اول Tokenization دو کار انجام می‌شود. یکی اینکه tabها و new line‌های متن ورودی با فاصله (single space) جایگزین می‌شوند و دیگری آنکه با توجه به pattern موجود در کلاس که عکس آن نیز (single space) داده شده است، حروف و علائم موجود در این pattern از حروف قبل و بعد خود با فاصله (single space) جدا می‌شود. با توجه به محتویان pattern در واقع علائم نگارشی و اعداد با فاصله از حروف دیگر جدا می‌شوند تا به آنها نچسبیده باشند.

سپس در خط بعد متن بر حسب فاصله tokenize می‌شود.

در خط بعد پاکسازی توکن‌های حاصل از مرحله قبل از کاراکتر `0xa\` و حذف توکن‌های خالی صورت می‌گیرد.

بلک بعدی برای پردازش افعال مرکب است که در صورتی که یک فعل پیشوندی داشته باشیم یا ترکیب بن ماضی + "5" + فعل پیشوندی، این‌ها را با _ به یکدیگر می‌چسبانند.

```
pattern = re.compile(r'([?!?]+|[\d.:]+|[\.,,\»\|])\}«\[(\{/\\\})']
```

```
@staticmethod
def Tokenization(text):
    text = DataPreprocessing.pattern.sub(r" \1 ", text.replace("\n", " ").replace("\t", " "))
    tokens = [word for word in text.split(" ") if word]
    tokens_cleaned = [token.strip('\xa0') for token in tokens if len(token.strip()) != 0]

    result = [""]
    # merge multi term verbs like خواهیم_رفت to خواهیم_رفت
    for token in reversed(tokens_cleaned):
        if token in DataPreprocessing.before_verbs or (
            result[-1] in DataPreprocessing.after_verbs and token in DataPreprocessing.verbs):
            result[-1] = token + "_" + result[-1]
        else:
            result.append(token)
    return list(reversed(result[1:]))

# method to tokenize a text
def tokenize(self, text):
    return self.Tokenization(text)
```

• تابع Stemming

از کتابخانه parsivar پرای ریشه یا پر کلمات استفاده شده.

تابع Top_K_Frequent •

در این تابع روی توکن‌های تمامی اسناد پردازش صورت می‌گیرد و با استفاده از Counter تعداد تکرار هر توکن استخراج می‌شود. این لیست سورت شده و k تا توکن اول که بیشترین تکرار را دارند بازگردانده می‌شوند. تابعی برای print نتایج این بخش نیز به عنوان top_k_print تعریف شده است

```
@staticmethod
def Top_K_Frequent(tokens, k):
    token_counts = Counter(tokens)
    sorted_tokens = sorted(token_counts.items(), key=lambda x: x[1], reverse=True)
    stopwords_to_remove = [token for token, count in sorted_tokens[:k]]
    report = {token: count for token, count in sorted_tokens[:k]}
    return report
```

لیست کلمات حذف شده به شرح زیر است (به ترتیب از سمت چپ بالا به سمت راست پایین):

Token: و, Count: 219205	Token: در, Count: 164334	Token: به, Count: 133277
Token: jl, Count: 92933	Token: اين, Count: 82977	Token: که, Count: 76240
Token: چ, Count: 68996	Token: ا, Count: 67490	Token: اس, Count: 45541
Token: کرد&کن, Count: 45104	Token: برای, Count: 31022	Token: داشت&دار, Count: 30052
Token: تیم, Count: 27692	Token: شد&شو, Count: 26688	Token: ان, Count: 26332
Token: کرد, Count: 22938	Token: هم, Count: 22393	Token: کشور, Count: 21730
Token: ما, Count: 19717	Token: یک, Count: 18733	Token: بود&باش, Count: 18275
Token: بازی, Count: 17796	Token: باید, Count: 16196	Token: تا, Count: 15870
Token: بر, Count: 15685	Token: شد, Count: 15641	Token: وی, Count: 15486
Token: داد&ده, Count: 14849	Token: خود, Count: 14552	Token: مجلس, Count: 14520
Token: اسلامی, Count: 14415	Token: گزارش, Count: 14040	Token: فارس, Count: 13969
Token: گفت, Count: 13773	Token: مردم, Count: 13201	Token: پیام, Count: 13112
Token: رییس, Count: 12793	Token: ایران, Count: 12717	Token: خبرگزاری, Count: 12429
Token: انتهای, Count: 12250	Token: دولت, Count: 12236	Token: اما, Count: 12211
Token: سال, Count: 11955	Token: گرفت&گیر, Count: 11845	
Token: توانست&توان, Count: 11059	Token: بازیکن, Count: 11023	Token: داشت&دارد, Count: 10782
Token: ملی, Count: 10235	Token: اینکه, Count: 10051	Token: کار, Count: 9738

3.1- ساخت شاخص مکانی

در تابع `positional_inverted_index` شاخص مکانی ساخته می‌شود. این تابع به عنوان ورودی اسناد پیش پردازش شده در قسمت‌های قبلی را دریافت می‌کند و شروع به ساخت شاخص مکانی برای هر `term` می‌کند. از ساختمان داده دیکشنری (همان `map`) برای شاخص استفاده شده است. `Key` در این شاخص `term` می‌باشد و `value` آن خود نیز یک دیکشنری دیگر است که شامل `document_frequency`, `collection_frequency` و اسنادی که آن کلمه در آنها حضور دارد می‌باشد. خود اسناد که در واقع همان `postings list` آن کلمه می‌باشد، خود مجدد یک دیکشنری است که در آن شماره اسنادی است که آن کلمه در آنها حضور داشته است و این شماره اسناد به لیستی از `key` در آن کلمه در آن سند و `term frequency` آن کلمه در آن سند نگاشت می‌شود. این تابع `term` های موجود در متن هر سند را دونه به دونه بررسی می‌کند. اگر `term` در شاخص وجود نداشته باشد یک `item` برایش ساخته می‌شود و مقادیر لازم پر می‌شوند. در صورتی که کلمه در شاخص نیز موجود باشد با توجه به اینکه آیا در آن سند در حال بررسی اولین بار است که حضور دارد یا قبلاً نیز وجود داشته و الان تکرار شده است، مقادیر پر و آپدیت می‌شوند.

```
def positional_inverted_index(docs):
    p_inverted_index = {}
    for index in docs:
        for position, term in enumerate(docs[index]['content']):
            # if its not a new term
            if term in p_inverted_index:
                # if the doc has already been in posting list of that term
                if index in p_inverted_index[term]['docs']:
                    p_inverted_index[term]['docs'][index]['positions'].append(position)
                    p_inverted_index[term]['docs'][index]['tf_td'] += 1 # tf_td is term frequency of t in d
                else:
                    p_inverted_index[term]['doc_freq'] += 1
                    p_inverted_index[term]['docs'][index] = {
                        'positions': [position],
                        'tf_td': 1
                    }
                    p_inverted_index[term]['collection_frequency'] += 1

            # add term to dictionary if its new
            else:
                p_inverted_index[term] = {
                    'doc_freq': 1,
                    'collection_frequency': 1,
                    'docs': {
                        index: {
                            'positions': [position],
                            'tf_td': 1
                        }
                    }
                }
    return p_inverted_index
```

2- پاسخ دهی به پرسمان در فضای برداری

1.2 مدل سازی اسناد در فضای برداری و ساخت Champions lists

در تابع calculate_doc_tf_idf که پس از ساخت کامل شاخص مکانی اجرا می شود، سه کار انجام می شود:

- محاسبه وزن هر کلمه در هر سند یا به عبارتی $tf \cdot idf$ و ذخیره آن در خود شاخص مکانی
- ساخت بردار هر سند که در آن هر term موجود در یک سند به مقدار $tf \cdot idf$ آن کلمه در آن سند نگاشت می شود. اینگونه بردار وزن های کلمات موجود در هر سند ساخته می شود.
- ساخت champions list برای هر کلمه. این champions list را در همان شاخص مکانی نگهداری و ذخیره می کنیم. برای هر کلمه، شماره اسنادی که در آنها وجود دارد بر حسب وزن آن کلمه در آن سند، مرتب می شود. سپس به تعداد `champ_len` سندی که بیشترین وزن را دارند، در champions list متناظر با آن کلمه قرار می گیرند.

```
# This function calculates tf_idf and add it to p_inverted_index, creates documents' champions lists
def calculate_doc_tf_idf(p_inverted_index, N, champ_len): # N is number of documents
    docs_vectors = {}

    for term in p_inverted_index:
        term_docs = dict(p_inverted_index[term]['docs']) # term_docs is actually postings list of the term
        df_t = p_inverted_index[term]['doc_freq']
        # calculating tf_idf for each doc in postings list
        for doc in p_inverted_index[term]['docs']:
            tf = p_inverted_index[term]['docs'][doc]['tf_td']
            w_td = (np.log10( N / df_t )) * (1 + np.log10(tf))
            p_inverted_index[term]['docs'][doc]['tf_idf'] = w_td

            # add weight of this term to docs vector for future usages in query processing
            if doc not in docs_vectors:
                docs_vectors[doc] = {}
            docs_vectors[doc][term] = {'tf_idf':w_td, 'tf':tf}

    # sort postings list and put it in champions list of each term
    sorted_term_docs = sorted(term_docs, key=lambda doc: term_docs[doc]['tf_td'], reverse=True)
    p_inverted_index[term]['champions_list'] = {}
    for doc_number in sorted_term_docs[:champ_len] if champ_len < df_t else sorted_term_docs:
        p_inverted_index[term]['champions_list'][doc_number] = {
            'tf_td': p_inverted_index[term]['docs'][doc_number]['tf_td'],
            'tf_idf' : p_inverted_index[term]['docs'][doc_number]['tf_idf']
        }

    return p_inverted_index, docs_vectors
```

2.2 پاسخ دهی به پرسمان در فضای برداری و شباهت کسینوسی

اکنون نوبت به پاسخ دهی به کوئری های کاربر، محاسبه شباهت کوئری با اسناد و در نهایت رتبه بندی اسناد و نمایش آنهاست.

ابتدا دوتابع کمکی استفاده شده را معرفی می کنیم و سپس به سراغ تابع محاسبه شباهت کسینوسی و پس از آن نمایش نتایج رتبه بندی شده می پردازیم.

- تابع calculate_query_tf_idf

این تابع همانطور که در تصویر نیز مشخص است، مقدار `tfidf` یک کلمه در کوئری را محاسبه می کند و بر می گردد.

```
def calculate_query_tf_idf(tf_tq, N, df_t):  
    tf = 1 + np.log10(tf_tq)  
    idf = np.log10(N / df_t)  
    return tf * idf
```

- تابع vector_length

این تابع برای محاسبه اندازه طول بردار به کار می رود که به جهت نرمال سازی بردار و استفاده در فرمول شباهت کسینوسی می باشد.

```
import math  
def vector_length(vector_dict):  
    length = math.sqrt(sum(tf_idf_value['tf_idf'] ** 2 for tf_idf_value in vector_dict.values()))  
    return length
```

• query_scoring

در این تابع ابتدا بردار هر کوئری ساخته شده و پس از آن میزان شباهت کسینوسی آن با دیگر اسناد محاسبه می شود. محاسبه شباهت کسینوسی به صورت term-at-a-time (TAAT) می باشد. بدین صورت که هر term موجود در کوئری، آن پیمایش (champions list) یا postings list می شود و وزن آن کلمه در کوئری و سند در یکدیگر ضرب شده و امتیاز آن سند اضافه می شود. اینگونه تنها اسناد مورد پردازش و محاسبه شباهت کسینوسی قرار می گیرند که با کوئری term مشترک دارند و آنها که عمل امتیازشان 0 است پردازش نمی شوند. در واقع از الگوریتم شبیه کد موجود در تصویر 1 استفاده شده است.

تابع به این صورت کار می کند که ابتدا بر روی کوئری مشابه با اسناد پیش پردازش انجام می شود و سپس توکن های آن به همراه تعداد تکرارشان در قالب یک دیکشنری ذخیره می شود. سپس به ازای هر term لیست مدنظر (postings list یا champions list) فراخوانی می شود. ابتدا مقدار tf.idf آن کلمه در کوئری به کمک تابع calculate_query_tf_idf محاسبه می شود. پس از آن روی لیست آن کلمه بر روی تمامی شماره اسناد پیمایش می شود تا مقدار tf.idf آن کلمه در آن سند که قبلا ذخیره کرده بودیم را بازیابی کنیم و در tf.idf کلمه در کوئری ضرب کنیم و حاصل را به امتیاز آن سند اضافه کنیم (در دیکشنری cosine_scores که هر شماره سند را به امتیاز آن نگاشت می کند).

پس از انجام این کار برای تمامی term های موجود در کوئری و محاسب امتیاز اسناد، نوبت به نرمال سازی طول سند و تقسیم بر طول آن است. در اینجا به ازای هر سندی که در cosine_scores امتیاز آن ذخیره شده، امتیاز آن بر طولش تقسیم می شود. محاسبه اندازه سند به کمک تابع vector_length صورت می پذیرد.

در نهایت نیز cosine_scores بر حسب امتیازها مرتب شده و k سند برتر به صورت مرتب شده در قالب (doc_id, score) برگردانده می شوند.

COSINESCORE(q)

```
1 float Scores[N] = 0
2 float Length[N]
3 for each query term  $t$ 
4 do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5 for each pair( $d, tf_{t,d}$ ) in postings list
6 do Scores[d] +=  $w_{t,d} \times w_{t,q}$ 
7 Read the array Length
8 for each  $d$ 
9 do Scores[d] = Scores[d] / Length[d]
10 return Top K components of Scores[]
```

تصویر 1: شبیه کد شباهت کسینوسی

```
def query_scoring(query, total_number_of_docs, p_inverted_index, k, champions_list = False):
    cosine_scores = {}
    query_tokens = preprocessor.simple_preprocess(query)
    query_tokens_count = dict(Counter(query_tokens))
    print(query_tokens_count)

    for term in query_tokens_count:
        if term in p_inverted_index:
            if champions_list:
                term_docs = p_inverted_index[term]['champions_list']
            else:
                term_docs = p_inverted_index[term]['docs']
            df_t = p_inverted_index[term]['doc_freq']
            w_tq = calculate_query_tf_idf(query_tokens_count[term], total_number_of_docs, df_t)
            for doc in term_docs:
                w_td = term_docs[doc]['tf_idf']
                # update doc scores for cosines similarity
                if int(doc) in cosine_scores:
                    # update doc scores for cosines similarity
                    cosine_scores[int(doc)] += w_td * w_tq
                else:
                    cosine_scores[int(doc)] = w_td * w_tq
            # calculate cosine score by dividing by doc vector length
            for doc_number in cosine_scores:
                cosine_scores[doc_number] /= vector_length(docs_vectors[str(doc_number)])
            # sort scores for top k
            sorted_doc_cosine = sorted(cosine_scores.items(), key=lambda x:x[1], reverse=True)
            return sorted_doc_cosine[:k]
```

تصویر 2: تابع query scoring

● **تابع query_search**

این تابع در واقع کاربر برای کار با موتور جستجو می باشد. کوئری را از کاربر می گیرد و اسناد مرتبه را به صورت مرتب شده برمی گرداند. تعداد اسناد برتر (همان top_k) و اینکه آیا می خواهیم جستجو در صورت بگیرد یا در postings lists از آرگومان champion lists قابل تنظیم است.

```
def query_search(query, result_numbers = 5, champion_list = False):
    results = query_scoring(query, N, p_inverted_index, result_numbers, champion_list)
    if len(results) == 0:
        print("نتیجه ای یافتن نشد")
    else:
        print("Cosine Scores:")
        print(100*'=')
        return print_results(results)
```

تابع print_results •

این تابع یک تابع کمکی و به جهت چاپ و نمایش خبرها می‌باشد. ورودی آن که results می‌باشد در واقع همان k سند برتر است که به صورت لیستی از دو تایی‌های (doc_id, score) (doc_id, score) می‌باشد. از enumerate که استفاده می‌شود عملاً فرمت به صورت (index, (doc_id, score)) در می‌آید که از آنجا که لیست results مرتب شده بوده است، index همان رنک آن سند در بازیابی می‌باشد. پس از آن با توجه به شماره سند، اطلاعات لازم سند مانند عنوان و url از مجموعه اسنادی که قبلاً ذخیره کرده بودیم در هنگام خواندن دیتابیس ورودی، بازیابی می‌شود و به همراه رنک و امتیاز آن سند چاپ می‌شوند.

همچنین این تابع اطلاعات این اسناد را نیز در قالب یک دیکشنری به عنوان خروجی برمی‌گرداند.

```
def print_results(results):
    dict_result = {}
    for rank, result in enumerate(results):
        # print(f'my\n {result}')
        doc_id = result[0]
        if doc_id is None:
            continue
        print(100* '-' + '\n')
        print(f'Rank: {rank + 1}')
        print(f'Score: {result[1]}')
        print(f'ID: {doc_id}')
        print(f'{docs[f'{doc_id}"]["title"]}')
        print(f'{docs[f'{doc_id}"]["url"]}')
        dict_result[rank + 1] = {'docID': doc_id,
                               'title': docs[str(doc_id)]["title"],
                               'url' : docs[str(doc_id)]["url"]}
    return dict_result
```

3.2 گزارش چند پرسمان از موتور جستجو

1. یک پرسمان از کلمات ساده و متداول تک کلمه‌ای نتیجه جستجو پرسمان "وزیر" در تصویر زیر قابل مشاهده است.

```
r1 = query_search('وزیر', result_numbers= 3, champion_list= False)
Executed at 2024.06.28 23:25:13 in 248ms

{1 'وزیر'}
Time: 0.24480628967285156
Cosine Scores:
=====
-----
Rank: 1
Score: 0.14329647678742605
ID: 12191
آغاز نشست غیرعلنی مجلس با حضور امیرعبداللهیان
https://www.farsnews.ir/news/14000724000697
-----
Rank: 2
Score: 0.13972084497123127
ID: 10893
زارعی کوشان استاندار کردستان شد
https://www.farsnews.ir/news/14000826000505
-----
Rank: 3
Score: 0.1373662946239786
ID: 10640
جزئیات برگزاری جلسه رأی اعتماد به وزیر پیشنهادی آموزش و پرورش
https://www.farsnews
```

علت آنکه امتیازها پایین می‌باشند آن است که این کلمه میان اسناد پرتکرار است و در نتیجه idf پایینی دارد در نتیجه امتیاز محاسبه شده برای آن کم می‌باشد. متن خبر اول به همراه تکرارهای کلمه وزیر را در تصویر زیر می‌توان مشاهده نمود.



به گزارش خبرنگاری اسلامی خبرگزاری فارس، نشست غیر علنی امروز (یکشنبه ۲۵ مهر) مجلس شورای اسلامی با حضور [وزیر خارجه](#) و [وزیر امور خارجه](#) در مورد مسائل منطقه‌ای و بین‌المللی دقایقی قبیل آغاز شد.

قرار است در این نشست موضوعات منطقه‌ای و سیاست خارجی بررسی شده و گزارشی هم از سوی [وزیر خارجه](#) درباره اقدامات دیپلماتیک دستگاه دیپلماسی به نمایندگان مجلس ارائه شد.

انتهای پیام /

علت اینکه این خبر رتبه اول را دارد آن است که طول سند بسیار کوتاه است و به همین دلیل در محاسبه شباهت کسینوسی هنگامی که امتیاز را بر طول سند تقسیم می‌کنیم، این طول بسیار کوچک است و امتیاز سند بالا می‌ماند. به عبارتی دیگر در ضرب کسینوسی، مشابه با کوئری بسیاری از درایه‌های بردار این سند ۰ هستند. در صورتی که از list `champions` استفاده شود، سندی که تعداد تکرار بیشتری از کلمه را در خود دارد و البته طولانی‌تر است رنک اول را تصاحب خواهد کرد.

نتایج پرسمان در صورتی که در list `champions` جستجو را انجام دهیم:

```
r1c = query_search('وزیر', result_numbers= 3, champion_list= True)
Executed at 2024.06.28 23:31:19 in 36ms

{1 : 'وزیر'}
Time: 0.029927730560302734
Cosine Scores:
=====
-----
Rank: 1
Score: 0.05450334231579835
ID: 10268
دستور کار کمیسیون های مجلس/ دیدار نمایندگان با وزیر خارجه و بررسی تخلفات روحانی و زنگنه در اعقاد
قرارداد کرست
https://www.farsnews.ir/news/14000910000920
-----
Rank: 2
Score: 0.05054510548588114
ID: 10890
سومین وزیر پیشنهادی آموزش و پرورش در ایستگاه بهارستان/ از سرعت عمل رئیس جمهور در معرفی تا گزینه ای از
بدنه فرهنگیان
https://www.farsnews.ir/news/14000826000559
```

یک نکته‌ای که قابل مشاهده است، کاهش چشم گیر زمان کوئیر می‌باشد به گونه‌ای که نزدیک به $\frac{1}{8}$ شده است. متن خبر اول بازگردانده شده توسط لیست champions در تصاویر زیر قابل مشاهده است:

همانطور که قابل مشاهده است هم طول سند بیشتر است و هم تعداد تکرار کلمه وزیر در آن بیشتر است.

2. یک پرسمن از عبارات ساده و متداول چند کلمه‌ای
عبارة جستجو شده: "سازمان لیگ"

```
r2 = query_search('سازمان لیگ', result_numbers= 3, champion_list= False)
Executed at 2024.06.28 23:39:38 in 410ms

Time: 0.3890845775604248
Cosine Scores:
=====
-----
Rank: 1
Score: 0.19017963053603698
ID: 2
محل برگزاری نشستهای خبری سرخابی‌ها؛ مجیدی در سازمان لیگ، گل محمدی در تمرین پرسپولیس
https://www.farsnews.ir/news/14001224000971
-----
Rank: 2
Score: 0.17547625862038246
ID: 2607
دیدار حسام لیگ برتر فوتسال لغو شد
https://www.farsnews.ir/news/14001119000691/
-----
Rank: 3
Score: 0.17091165680175882
ID: 3789
دیدار هوادار و استقلال لغو شد
https://www.farsnews.ir/news/14001104000380/
```

مجدد امتیازها به این دلیل پایین می‌باشند که این کلمات بسیار پرکاربرد هستند و idf بسیار پایینی دارند.
سنند اول برگردانده شده را می‌توان در تصویر زیر مشاهده نمود. همانطور که قابل مشاهده است به نسبت طول
سنند کوتاه است و تعداد تکرار کلمات سازمان و لیگ در آن به نسبت بالاست به همین دلیل نتیجه اول است.

به گزارش خبرگزاری فارس، نشست خبری پیش از مسابقه سرمربیان دو تیم پرسپولیس و استقلال از هفته بیست و سوم لیگ برتر با مدیریت سازمان لیگ و هماهنگی پاشگاه میزان(پرسپولیس) به شرح زیر برگزار می‌شود:

چهارشنبه ۲۵ اسفند ساعت ۱۵ فرداد مجیدی سرمربی استقلال در سازمان لیگ فوتبال ایران

ساعت ۱۳:۰۰ پیش گل محمدی سرمربی پرسپولیس در ورزشگاه شهید کاظمی مسابقه دو تیم روز پنجشنبه در ورزشگاه آزادی برگزار می‌شود.

انتهای پیام /

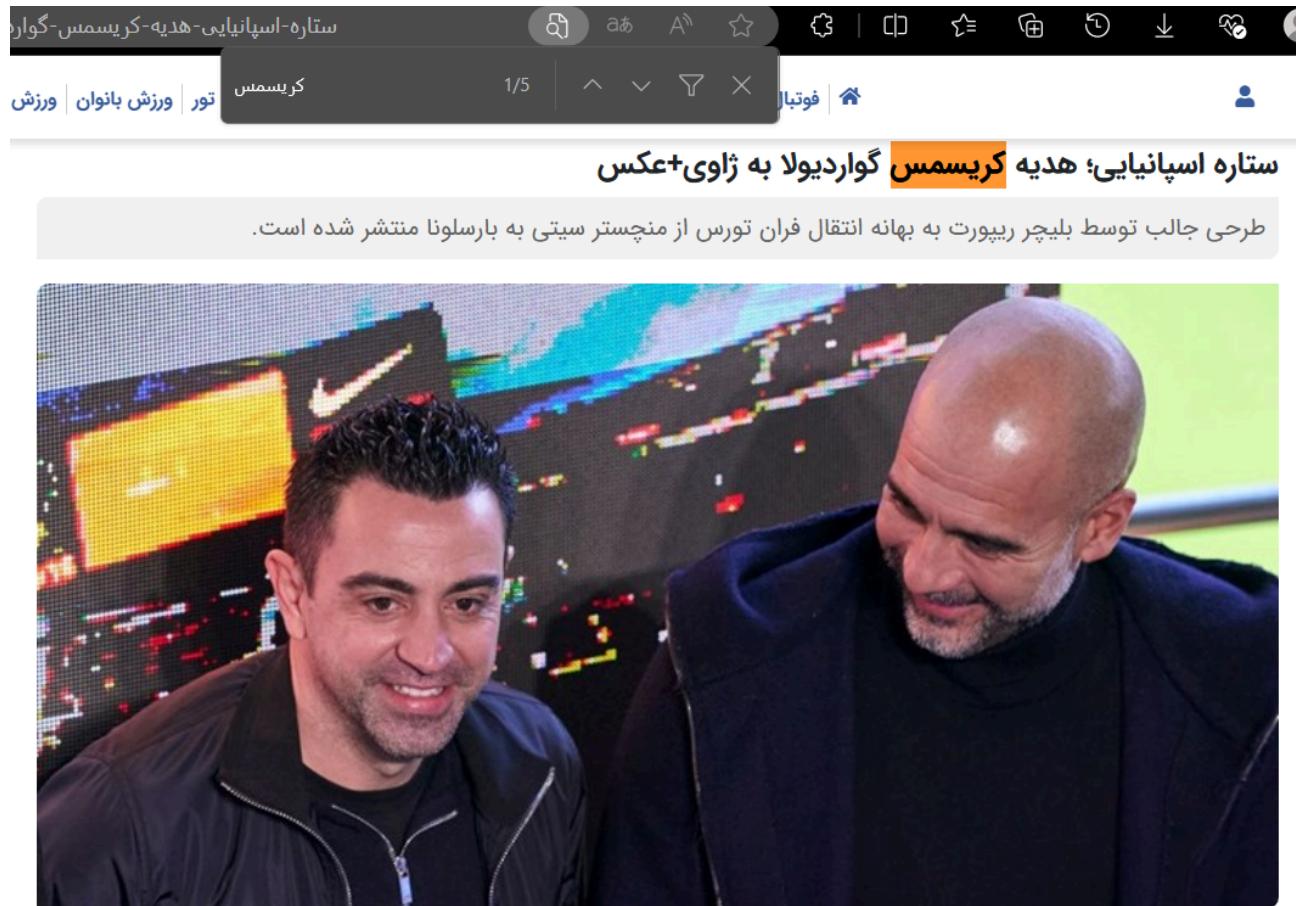
3. یک پرسمان دشوار و کم تکرار تک کلمه‌ای
عبارةت مورد جستجو: کریسمس

نتایج جستجو را در تصویر زیر می‌توان مشاهده نمود. همانطور که دیده می‌شود امتیازها به نسبت بسیار بالا هستن دچرا که کریسمس یک کلمه کم تکرار است idf بسیار پایینی دارد.

```
r3 = query_search('کریسمس', result_numbers= 3, champion_list= False)
Executed at 2024.06.28 23:46:24 in 130ms

{1: 'کریسمس'}
Time: 0.015781641006469727
Cosine Scores:
=====
-----
Rank: 1
Score: 0.8669566044973201
ID: 5933
ستاره اسپانیایی؛ هدیه کریسمس گواردیولا به ژاوای+عکس
ستاره - اسپانیایی - هدیه - کریسمس - گواردیولا - به - ژاوای - عکس
https://www.farsnews.ir/news/14001007000739
-----
Rank: 2
Score: 0.7741970328238605
ID: 6117
کیروش «دیکتاتور» لقب گرفت/اختلاف مرد پرتغالی با مصری‌ها به خاطر کریسمس+عکس
کیروش-دیکتاتور-لقب-گرفت-اختلاف-مرد-پرتغالی-با-مصری‌ها-به-خاطر-کریسمس
https://www.farsnews.ir/news/14001005000165
-----
Rank: 3
Score: 0.7332855143304998
ID: 6120
```

متن خبر اول را می‌توان در تصویر زیر مشاهده نمود. این کلمه بسیار کم تکرار است و در اسناد کمی به کار رفته است و حتی در این سند که به عنوان رتبه اول بازگردانده شده است، تنها یک بار تکرار شده. علت آنکه این خبر امتیاز بیشتری از خبر با رتبه دوم دارد آن است که طول آن کمتر و کوتاه‌تر است در نتیجه به نسبت وزن و امتیاز کلمه کریسمس در این سند بالاست.



به گزارش خبرگزاری فارس، فران تورس ستاره اسپانیایی باشگاه منچستر سیتی با قراردادی تا سال ۲۰۲۷ به باشگاه بارسلونا پیوست.

انتقال این بازیکن ۲۱ ساله اسپانیایی مورد توجه بلیچر ریپورت قرار گرفته و طرحی جالب در این باره را منتشر کرده است.

در این طرح، تورس به هدیه کریسمس پپ گواردیولا به ژاوه تشبیه شده است.

4. یک پرسمان دشوار و کم تکرار چند کلمه‌ای

عبارت: جیانی اینفانتینو

هر دو کلمه بسیار خاص و نادر هستند و df بالایی دارند در نتیجه امتیاز اسناد بازگردانده شده بالاست.

(دقیق شود عت آنکه امتیاز اسناد از 1 بیشتر است این است که مقادیر شباهت کسینوسی بر طول کوئری

تقسیم نشده‌اند زیرا این مقدار در محاسبه امتیاز برای اسناد مختلف یکسان است)

```
r4 = query_search('جیانی اینفانتینو', result_numbers= 3, champion_list= False)  
Executed at 2024.06.28 23:53:53 in 141ms
```

{'جیانی': 1, 'اینفانتینو': 1}

Time: 0.02007269859313965

Cosine Scores:

```
=====
```

Rank: 1

Score: 1.5289023053255142

ID: 6731

حضور مهدوی کیا در بازی نمادین ستارگان در ورزشگاه جام جهانی

<https://www.farsnews>

.ir/news/14000926000283/حضور-مهدوی-کیا-در-بازی-نمادین-ستارگان-در-ورزشگاه-جام-جهانی

Rank: 2

Score: 1.2640737943569136

ID: 4033

احتمال لغو سفر اینفانتینو به تهران / وضعیت دستیاران اسپانیایی شمسایی بررسی می شود

<https://www.farsnews>

.ir/news/14001102000084/احتمال-لغو-سفر-اینفانتینو-به-تهران-وضعیت-دستیاران-اسپانیایی-شمسایی

Rank: 3

در این دیدار نمادین که از سوی **فیفا** برگزار می شود دو تیم منتخب ستارگان عرب ها با منتخب ستارگان جهان رقابت خواهند کرد.



البته نام گذاری تیم ها به صورت نمادین است و لزوما بازیگنان تیم ها عرب زبان نیستند. در همین رابطه مهدی مهدوی کیا ستاره سابق تیم ملی کشورمان و مربی امیدهای ایران در ترکیب تیم منتخب اعراب قرار گرفت.

چیانی اینفانتینو رئیس فیفا هم در تیم منتخب اعراب هم بازی مهدوی کیا است. این بازی نمادین در ورزشگاه الثمامه یکی از میزبان های جام جهانی ۲۰۲۲ برگزار می شود.

انتهای پیام /