گزارش پروژه پایانی بازیابی - صبا سحبان ۹۹۳۱۰۹۶

۱- ساخت شاخص مکانی

۱.۱ مجموعه داده:

```
def load_docs():
    docs = {}
    contents = []
    urls = []
    with open("IR_data_news_12k.json", 'r') as file:
        docs = json.load(file)
        for key in docs.keys():
            idx = str(key)
            docs[idx] = {'title': docs[idx]['title'],
                         'content': docs[idx]['content'],
                         'url': docs[idx]['url'],
                         }
            contents.append(docs[idx]['content'])
    return docs, contents, urls

docs, contents, urls = load_docs()
docs['1']
```

این کد فایل JSON شامل داده‌های خبری را بارگذاری می‌کند، اطلاعات هر خبر (عنوان، محتوا و لینک) را در یک دیکشنری به نام `docs` ذخیره می‌کند. همچنین محتوای تمام اخبار را در لیست `contents` قرار می‌دهد. خروجی تابع، دیکشنری `docs` و لیست محتواهاست به همراه لینک خبر.

۲.۱ پیش پردازش اسناد

1. بارگذاری مخفف‌ها (`load_abbreviations`)

```python
def load_abbreviations(file_path):
    """Loads abbreviations from the file into a dictionary."""
    abbreviations = {}
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            try:
                entry = eval(line.strip())
                abbreviations.update(entry)
            except Exception as e:
                print(f"Error parsing line: {line.strip()} - {e}")
    return abbreviations
```

- فایل `abbreviations.txt` که شامل مخفف‌ها و معانی آن‌ها به صورت خط‌به‌خط است، خوانده می‌شود.
- هر خط به یک دیکشنری تبدیل شده و به دیکشنری کلی `abbreviations` اضافه می‌شود.

2. جایگزینی مخفف‌ها (`expand_abbreviations`)

```python
def expand_abbreviations(text, abbreviations):
    """Expands abbreviations using the loaded abbreviations dictionary."""
    for short_form, expansion in abbreviations.items():
        pattern = fr"\b{re.escape(short_form)}\b"
        text = re.sub(pattern, expansion, text)
    return text
```

- با استفاده از دیکشنری `abbreviations`، تمامی مخفف‌ها در متن ورودی پیدا شده و با معانی کاملشان جایگزین می‌شوند.

3. اصلاح فاصله‌ها (`spacing_correction`)

```python
def spacing_correction(text):
    """Applies spacing corrections."""
    for pattern, repl in SPACING_PATTERNS:
        text = re.sub(pattern, repl, text)
    return text
```

- الگوهای مشخصی برای تصحیح فاصله‌ها تعریف شده است.
    - مثال: تبدیل می خواهم به می‌خواهم یا اتصال پسوندهای فارسی مثل ها، ترین به کلمه اصلی با **نیم‌فاصله**.
- متن بررسی شده و تمامی فاصله‌های اشتباه اصلاح می‌شوند.

- هدف: ایجاد متنی خواناتر و سازگارتر برای پردازش.

4. حذف علائم اضافی (`remove_punct`)

```python
def remove_punct(text):
    """Removes diacritics and extra punctuation."""
    text = re.sub(EXTRA_PUNCT_PATTERN, "", text)
    return text
```

- **علائم نگارشی** و **دیگر کاراکترهای غیرضروری** (مثل نقطه، ویرگول، نقل‌قول‌ها، ...) از متن حذف می‌شوند.
- این کار با استفاده از یک الگوی Regex انجام می‌شود که تمامی این کاراکترها را شناسایی و حذف می‌کند.

5. نرمال‌سازی متن (`Normalizer`)

- متن با استفاده از ماژول هضم نرمال‌سازی می‌شود.
- تغییرات شامل موارد زیر است:
  - تبدیل کاف و ی عربی به فارسی.
  - حذف نیم‌فاصله‌های اضافه.
  - حذف علائم تزیینی یا غیرضروری.
- هدف: یکنواخت‌سازی ساختار متن.

6. توکنیزه کردن (`word_tokenize`)

- متن نهایی به **توکن‌های مجزا** (کلمات) تقسیم می‌شود.
- این مرحله، خروجی نهایی را به شکل لیستی از کلمات آماده می‌کند.
- هدف: آماده‌سازی متن برای مراحل بعدی تحلیل (مثل محاسبه فراوانی کلمات یا ساخت شاخص معکوس).

7. تابع اصلی (`preprocess_single_text`)

```python
def preprocess_single_text(text, expand_abbr=True, do_spacing_corr=True,
                           remove_diacritics=True):

    if expand_abbr:
        text = expand_abbreviations(text, abbreviations)

    text = normalizer.normalize(text)

    if do_spacing_corr:
        text = spacing_correction(text)

    if remove_diacritics:
        text = remove_punct(text)

    tokens = word_tokenize(text)
    return tokens
```

- این تابع مراحل بالا را با توجه به ورودی‌های کاربر اجرا می‌کند:
  - **جایگزینی مخففها** (در صورت نیاز).
  - **اصلاح فاصله‌ها**.
  - **حذف علائم نگارشی**.
  - **نرمال‌سازی متن**.
  - در نهایت **توکنیزه کردن** متن.
- خروجی: لیستی از توکن‌های پردازش‌شده

توضیح نرمالایزر Hazm:

استفاده از WordTokenizer:

جداسازی کلمات با استفاده از فاصله، نیم‌فاصله، تب و خطوط جدید.

شناسایی و ترکیب کلمات با پیشوندها و پسوندها (مثل "می‌روم").

توکنایز صحیح ایمیل‌ها، آیدی‌ها و اعداد.

```
)self.email_pattern = re.compile

,"{,r"[a-zA-Z0-9._+-]+@([a-zA-Z0-9-]+\.)+[A-Za-z]{2

(
```

```
("(+[_self.id_pattern = re.compile(r"(?<![\w._])(@[\w
```

- `WordTokenizer` با قابلیت `join_verb_parts=False` برای شناسایی و جداسازی صحیح کلمات استفاده می‌شود. این ابزار:
  - فاصله‌ها و نیم‌فاصله‌ها را مدیریت می‌کند.
  - پیشوندها (مثل "می" و "نمی") را به درستی شناسایی می‌کند.

```
    self.tokenizer = WordTokenizer(join_verb_parts=False)
    self.words = self.tokenizer.words
```

ترکیب پیشوندها و پسوندها:

- در تابع `token_spacing`، کلماتی که با نیم‌فاصله یا پسوند ترکیب می‌شوند (مثل "کتاب‌ها") شناسایی و ترکیب می‌شوند

```
result = []
for t, token in enumerate(tokens):
    joined = False

    if result:
        token_pair = result[-1] + "ZWNJ" + token
        if (
            token_pair in self.verbs
            or token_pair in self.words
            and self.words[token_pair][0] > 0
        ):
            joined = True

            if (
                t < len(tokens) - 1
```

مدیریت ایمیل‌ها، آیدی‌ها و اعداد:

- با استفاده از regex، ایمیل‌ها و اعداد شناسایی و به صورت توکن مستقل در نظر گرفته می‌شوند.

فاصله‌گذاری صحیح

نکات:

- شناسایی و اصلاح فاصله‌ها برای پیشوندها (مثل "می")، پسوندها (مثل "ها") و علائم نگارشی.

در hazm:

- اصلاح فاصله‌ها در تابع `correct_spacing` با الگوهای زیر انجام می‌شود:

حذف فاصله‌های اضافی:

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

اضافه کردن نیم‌فاصله به پیشوندها و پسوندها:

```
self.affix_spacing_patterns = [
    (" ی (ه[ ^])"r, " یr"\1),  # space ی fix
    (" (ن?می)( |^)"r, r"\1\2ZWNJ"),  # می, نمی put zwnj after
    # تر, تری, ترین, گر, گری, ها, های put zwnj before
    (
```

اصلاح فاصله‌گذاری علائم نگارشی:

```
punc_after + "])", r"\1"),  # remove space before + "]) ")
```

اضافه کردن فاصله بعد از اعداد:

```
(r"(\d)([آابپتثجچحخدذرزژسشصضطظعغفقکگلمنوهی])", r"\1 \2"),
```

تعویض یونیکد

نکات:

- تبدیل کاراکترهای خاص (مثل "ي" و "ك") به فرم استاندارد فارسی.
- تبدیل " " به "بسم الله الرحمن الرحیم".

در hazm:

- تبدیل یونیکدها با استفاده از maketrans و جایگزینی خاص انجام می‌شو

تبدیل "ي" و "ك" عربی به "ی" و "ک"

```
        (translations = maketrans(self.translation_src, self.translation_dst
                                            (text = text.translate(translations
```

جایگزینی کاراکترهای خاص:

```
                                                        ] = self.replacements

                                                    ("﷽", "بسم الله الرحمن الرحیم"),

                                                        ("﷼", "ریال"),

                                                    ("(صلے|صلی)", "صلی"),

                                                        ("الله", "الله"),

                                                        ("اکبر", "اکبر"),

                                                        ("محمد", "محمد"),

                                                        ("صلعم", "صلعم"),

                                                        ("رسول", "رسول"),

                                                        ("علیه", "علیه"),

                                                        ("وسلم", "وسلم"),

                                                ("لآ|لآ|لأ|لأ|لإ|لإ|لا|لا", "لا"),

                                                                        [
```

حذف کاراکترهای اضافی

حذف اعراب:

```
                                                    :if self._remove_diacritics

                                                ] = self.diacritics_patterns

remove FATHATAN, DAMMATAN, KASRATAN, FATHA, DAMMA, KASRA, SHADDA, #
                                                                        SUKUN

        ,("" ,"[u064b\u064c\u064d\u064e\u064f\u0650\u0651\u0652\]")

                                                                        [
```

حذف علائم اضافی:

```
                                                :if self._remove_specials_chars
```

```
                                        ] = self.specials_chars_patterns

      Remove almoast all arabic unicode superscript and subscript #
characters in the ranges of 00600-06FF, 08A0-08FF, FB50-FDFF, and FE70-FEFF

                                                                    )
```

تبدیل اعداد انگلیسی به فارسی

```
                                                    :if self._persian_number

                        "self.number_translation_src = "0123456789%۰۱۲۳٤٥٦۷۸۹

                          "self.number_translation_dst = "۰۱۲۳۴۵۶۷۸۹٪۰۱۲۳۴۵۶۷۸۹
```

جداسازی "می" و "نمی"

- اضافه کردن نیم‌فاصله بین پیشوند "می" و فعل.

در hazm:

- در تابع `seperate_mi`، پیشوندها با regex شناسایی و اصلاح می‌شوند:

```
                    :def seperate_mi(self: "Normalizer", text: str) -> str
```

توضیح توکنایزر در hazm:

```
  def tokenize(self: "WordTokenizer", text: str) -> List[str]:
      """توکن‌های متن را استخراج می‌کند.

      Examples:
          >>> tokenizer = WordTokenizer()
          >>> tokenizer.tokenize('این جمله (خیلی) پیچیده نیست!!!')
          ['این', 'جمله', '(', 'خیلی', ')', 'پیچیده', 'نیست', '!!!']
          >>> tokenizer = WordTokenizer(join_verb_parts=False)
```

متد `tokenize` متن ورودی را به واحدهای کوچک‌تر به نام توکن تقسیم می‌کند. این واحدها شامل کلمات، علائم نگارشی، ایموجی‌ها، اعداد و سایر عناصر متن است. عملکرد آن به شرح زیر است:

ابتدا اگر قابلیت تشخیص مخفف‌ها فعال باشد، مخفف‌ها شناسایی شده و به‌طور موقت با مقادیر یکتا جایگزین می‌شوند تا در فرآیند شکستن متن، به بخش‌های جدا تقسیم نشوند. در پایان مخفف‌ها به حالت اولیه برمی‌گردند.

ایموجی‌های موجود در متن شناسایی شده و با یک فاصله از سایر بخش‌های متن جدا می‌شوند تا به‌عنوان توکن مستقل پردازش شوند.

عناصر خاص مثل ایمیل‌ها، لینک‌ها، آیدی‌ها (مانند @username)، هشتگ‌ها، اعداد صحیح و اعشاری شناسایی شده و با مقادیر جایگزین مانند NUM، TAG، ID، LINK، EMAIL و NUMF جایگزین می‌شوند. این جایگزینی برای کاهش پیچیدگی متن انجام می‌شود.

علائم نگارشی مانند نقطه، ویرگول و علامت سوال از کلمات مجاور جدا شده و به توکن‌های مستقل تبدیل می‌شوند.

متن پس از این پردازش‌ها بر اساس فاصله‌ها شکسته شده و هر بخش به‌عنوان یک توکن جداگانه در لیست ذخیره می‌شود.

اگر قابلیت ترکیب افعال چندبخشی فعال باشد، افعالی مثل "خواهد رفت" یا "گفته شده است" با علامت _ به هم متصل می‌شوند و به‌عنوان یک توکن واحد برگردانده می‌شوند.

در نهایت، مخفف‌هایی که به‌طور موقت جایگزین شده بودند، به حالت اولیه خود بازمی‌گردند و لیستی از توکن‌های پردازش‌شده شامل کلمات، علائم نگارشی، ایموجی‌ها، و عناصر خاص آماده استفاده خواهد بود.

:Test 1

Original: من می خواهم که نمی روم به این مکان. همچنین کتاب‌هایم ترمیم شدند.

Processed Tokens: ['من', 'می\u200cخواهم', 'که', 'نمی\u200cروم', 'به', 'این', 'مکان', 'همچنین', 'کتاب\u200cهایم', 'ترمیم', 'شدند']

:Test 2

Original: کاف و یای عربی به شکل فارسی تبدیل می‌شود. بسم‌الله الرّحمن الرّحیم.

Processed Tokens: ['کاف', 'و', 'یای', 'عربی', 'به', 'شکل', 'فارسی', 'تبدیل', 'می\u200cشود', 'بسم\u200cالله', 'الرحمن', 'الرحیم']

:Test 3

Original: إِنَّ اللَّهَ غَفورٌ رَحیمٌ.

Processed Tokens: ['إن', 'الله', 'غفور', 'رحیم']

:Test 4

Original: سلام! این یک جمله است... یا شاید جمله‌ای دیگر؟!

Processed Tokens: ['سلام', 'این', 'یک', 'جمله', 'است', 'یا', 'شاید', 'جمله\u200cای', 'دیگر']

:Test 5

Original: شماره تماس 1234567890 به شماره ۱۲۳۴۵۶۷۸۹۰ تبدیل شود.

Processed Tokens: ['شماره', 'تماس', '۱۲۳۴۵۶۷۸۹۰', 'به', 'شماره', '۱۲۳۴۵۶۷۸۹۰', 'تبدیل', 'شود']

:Test 6

Original: در سال‌های گذشته کتاب‌های زیادی خواندم. سال‌ها می‌گذرد و کتاب‌ها تکرار می‌شوند.

Processed Tokens: ['در', 'سال\u200cهای', 'گذشته', 'کتاب\u200cهای', 'زیادی', 'خواندم',
'سال\u200c\u200cها', 'می\u200cگذرد', 'و', 'کتاب\u200c\u200cها', 'تکرار', 'می\u200cشوند']

:Test 7

Original: بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ و .

Processed Tokens: ['بسم\u200cالله', 'الرحمن', 'الرحیم', 'و', 'بسم', 'الله', 'الرحمن', 'الرحیم']

:Test 8

Original: آیا !؟ و ... را باید حذف کنیم؟

Processed Tokens: ['آیا', 'و', 'را', 'باید', 'حذف', 'کنیم']

:Test 9

Original: نمی‌شود "می‌خواهم" را به دو توکن تبدیل کرد.

Processed Tokens: ['نمی\u200cشود', 'می\u200cخواهم', 'را', 'به', 'دو', 'توکن', 'تبدیل', 'کرد']

Test 10:

Original: ه.ش.

Processed Tokens: ['هجری', 'شمسی']

تابع `stem_tokens`

این تابع برای ریشه‌یابی (stemming) لیستی از توکن‌ها استفاده می‌شود. از طریق کتابخانه `Stemmer`، هر توکن به ریشه‌ی اصلی خود تبدیل می‌شود تا کلمات مشابه از نظر معنایی مثل "کتاب‌ها" و "کتاب" به یک شکل واحد پردازش شوند.

```
def stem_tokens(tokens):
    """Applies stemming to a list of tokens."""
    return [stemmer.stem(t) for t in tokens]
```

تابع `compute_top_k_frequent`

این تابع پرتکرارترین توکن‌ها را محاسبه می‌کند:

1. تعداد تکرار هر توکن را با استفاده از `Counter` محاسبه می‌کند.
2. توکن‌ها را بر اساس تعداد تکرار مرتب کرده و `k` توکن پرتکرار را به همراه تعداد تکرارشان بازمی‌گرداند.

```
def compute_top_k_frequent(tokens, k):
    """Computes the top-K most frequent tokens along with their frequencies."""
    token_counts = Counter(tokens)
    sorted_tokens = sorted(token_counts.items(), key=lambda x: x[1], reverse=True)
    return dict(sorted_tokens[:k])
```

تابع `simple_preprocess`

این تابع یک سند را به‌صورت زیر پیش‌پردازش می‌کند:

1. **نرمال‌سازی و توکن‌سازی:** متن ورودی با استفاده از تابع `preprocess_single_text` نرمال و به توکن‌ها شکسته می‌شود.
2. **ریشه‌یابی:** با استفاده از تابع `stem_tokens`، تمامی توکن‌ها ریشه‌یابی می‌شوند.
3. **حذف توکن‌های پرتکرار:** اگر لیستی از توکن‌های پرتکرار (`top_k_tokens`) فراهم شود، این توکن‌ها از لیست نهایی حذف می‌شوند.

```python
def simple_preprocess(content, top_k_tokens=None):
    """
    Simplified preprocessing pipeline for a single document:
      - Normalize and tokenize the content.
      - Stem tokens.
      - Optionally remove tokens present in `top_k_tokens`.
    """
    # Step 1: Normalize and tokenize the content
    tokens = preprocess_single_text(content)

    # Step 2: Stem tokens
    tokens = stem_tokens(tokens)

    # Step 3: Remove top-K frequent tokens if provided
    if top_k_tokens:
        tokens = [token for token in tokens if token not in top_k_tokens]

    return tokens
```

تابع `preprocess_all_docs`

این تابع همه اسناد را پیش‌پردازش می‌کند و مراحل زیر را انجام می‌دهد:

1. **پردازش تک‌تک اسناد:**
   - برای هر سند، متن نرمال شده، توکن‌سازی شده و توکن‌ها ریشه‌یابی می‌شوند.
   - لیست توکن‌های پردازش‌شده برای هر سند ذخیره می‌شود و همه توکن‌ها در لیستی ترکیبی جمع‌آوری می‌شوند.
2. **محاسبه پرتکرارترین توکن‌ها:**
   - با استفاده از `compute_top_k_frequent`، k توکن پرتکرار محاسبه می‌شود.
   - فقط خود توکن‌ها (بدون تعداد تکرار) برای حذف استفاده می‌شوند.
3. **حذف توکن‌های پرتکرار از هر سند:**
   - توکن‌های پرتکرار از لیست توکن‌های هر سند حذف می‌شوند.
   - محتوای هر سند با لیست فیلتر شده به‌روزرسانی می‌شود.

```python
def preprocess_all_docs(docs, top_k=50):
    """
    Preprocesses all documents while returning the same structure as required:
      - Normalizes, tokenizes, and stems content for each document.
      - Computes top-K frequent tokens across all documents.
      - Removes top-K frequent tokens from each document's content.
    """
    combined_tokens = []
    all_tokens = {}

    for doc_id, doc_data in docs.items():
        content = doc_data['content']
        tokens = simple_preprocess(content)
        all_tokens[doc_id] = tokens
        combined_tokens.extend(tokens)

    top_k_tokens_with_counts = compute_top_k_frequent(combined_tokens, top_k)
    top_k_tokens = set(top_k_tokens_with_counts.keys())  # Extract tokens only

    for doc_id in all_tokens:
        filtered_tokens = [token for token in all_tokens[doc_id] if token not in top_k_tokens]
        docs[doc_id]['content'] = filtered_tokens

    return docs, top_k_tokens_with_counts
```

خروجی نهایی

این تابع:

1. اسناد پیش‌پردازش‌شده را برمی‌گرداند که در آن‌ها توکن‌های پرتکرار حذف شده‌اند.
2. لیستی از پرتکرارترین توکن‌ها به همراه تعداد تکرارشان را ارائه می‌دهد.

```python
docs, contents, _ = load_docs()

pre_processed_docs, top_k_tokens_with_counts = preprocess_all_docs(docs, top_k=20)

print("\nTop-K Frequent Tokens:")
for token, count in top_k_tokens_with_counts.items():
    print(f"Token: {token}, Count: {count}")

print("\nProcessed Document Tokens:")
for doc_id, doc_data in list(pre_processed_docs.items())[:5]:
    print(f"Doc {doc_id} final tokens:", doc_data['content'])
```

Executed at 2025.01.11 02:31:50 in 1m 34s 625ms

```
Top-K Frequent Tokens:
Count: 234908 ,و :Token
Count: 165135 ,در :Token
Count: 136045 ,به :Token
Count: 92977 ,از :Token
Count: 83094 ,این :Token
Count: 75480 ,که :Token
Count: 69233 ,با :Token
Count: 68677 ,را :Token
Count: 48513 ,اس :Token
Count: 31029 ,ا ا :Token
```

```
Processed Document Tokens:
['گزار', 'خبرگزار', 'فارس', 'کنفدراسیون', 'فوتبال', 'آسیا', 'AFC', 'نامه', 'رسم', 'فدراسیون', 'فوتبال', 'ایر', 'باشگاه', 'گیت',
'پسند', 'ز', 'قرعه\u200cکش', 'چ', 'باشگاه', 'فوتسال', 'آسیا', 'رسما', 'اعل', 'اساس', '۲۵', 'فروردین\u200cماه', '۱۴۰۱', 'مراس', 'قرعه\u200cکش',
'باشگاه', 'فوتسال', 'آسیا', 'مالز', 'برگزار', 'می\u200cشود', 'باشگاه', 'گیت', 'پسند', 'بعنو', 'قهر', 'فوتسال', 'ایر', 'سال', '۱۴۰۰', 'مسابق',
'راه', 'پیدا', 'کرده\_اس', 'پ', 'گیت', 'پسند', 'تجربه', '۳', 'دوره', 'حضور', 'چ', 'باشگاه', 'فوتسال', 'آسیا', 'داشته', 'هر', 'سه', 'دوره', 'فینال',
['/', 'پی', 'انت', 'آورده\_اس', 'بس', 'دوم', 'ق', 'دو', 'قهرمان', 'عنو', 'کرده', 'پیدا', 'راه', 'مسابق',
'گزار', 'خبرگزار', 'فارس', 'سید', 'حمید', 'سجاد', 'حاشیه', 'مراس', 'گرامیدا', 'روز', 'جو', 'جمع', 'خبرنگار', 'رابطه', 'عرضه',
```

```python
pre_processed_docs['1']
```

Executed at 2025.01.11 02:31:50 in 4ms

```
'content': ['گزار',
 'خبرگزار',
 'فارس',
 'سید',
 'حمید',
 'سجاد',
 'حاشیه',
 'مراس',
 'گرامیدا',
 'روز',
```

حضور تماشاگران در لیگ برتر فوتبال تابع نظر فدراسیون و سازمان لیگ است',

توضیح تابع stemming hazm:

۱. متد `__init__`

در این متد، یک لیست از پسوندهای رایج فارسی مانند "ات"، "ان"، "ترین"، "ها"، "یی"، "م" و غیره در متغیر `self.ends` ذخیره می‌شود. این پسوندها برای حذف از انتهای کلمات در متد `stem` استفاده می‌شوند.

۲. متد `stem`

این متد برای یافتن ریشهٔ کلمه طراحی شده است.
ابتدا بررسی می‌کند که آیا کلمه به یکی از پسوندهای موجود در لیست `self.ends` ختم می‌شود یا خیر.
اگر پسوندی یافت شود، آن را از انتهای کلمه حذف می‌کند.
سپس، اگر کلمه به "ۀ" ختم شود، آن را به "ه" تغییر می‌دهد. در نهایت، ریشهٔ کلمه بازگردانده می‌شود.

```python
class Stemmer(StemmerI):
    """این کلاس شامل توابعی برای ریشه‌یابی کلمات است."""

    def __init__(self: "Stemmer") -> None:
        self.ends = [
            "ات",
            "ان",
            "ترین",
            "تر",
            "م",
            "ت",
            "ش",
            "یی",
            "ی",
            "ها",
            "",
            "ا",
            "ZWNJ",
        ]

    def stem(self: "Stemmer", word: str) -> str:
        """ریشهٔ کلمه را پیدا می‌کند.
```

۳.۱ ساخت شاخص مکانی:

```python
def _add_new_token_posting(token_dict, token, doc_id, position):
    """
    Initialize a new posting entry for a token that has not yet been seen.
    """
    token_dict[token] = {
        'frequency': 1,
        'docs': {
            doc_id: {
                'positions': [position],
                'number_of_token': 1
            }
        }
    }
```

این تابع برای اضافه کردن یک **توکن جدید** به دیکشنری پستینگ‌ها استفاده می‌شود.

- اگر توکن جدید باشد:
  - تعداد تکرار آن (`frequency`) برابر 1 تنظیم می‌شود.
  - در دیکشنری `docs`، شناسهٔ سند (`doc_id`) ذخیره می‌شود.
  - موقعیت توکن در متن (لیستی از موقعیت‌ها) به همراه تعداد تکرار در همان سند نگهداری می‌شود.

```python
def _update_existing_token_posting(token_dict, token, doc_id, position):
    """
    Update an existing token entry with a new position in an existing or new document.
    """
    token_dict[token]['frequency'] += 1

    if doc_id in token_dict[token]['docs']:
        token_dict[token]['docs'][doc_id]['positions'].append(position)
        token_dict[token]['docs'][doc_id]['number_of_token'] += 1
    else:
        token_dict[token]['docs'][doc_id] = {
            'positions': [position],
            'number_of_token': 1
        }
```

. تابع `_update_existing_token_posting`

این تابع برای به‌روزرسانی **توکنی که قبلاً وجود دارد** استفاده می‌شود.

- تعداد تکرار توکن (`frequency`) در کل دیکشنری افزایش می‌یابد.
- اگر توکن در همان سند (`doc_id`) باشد:
  - موقعیت جدید آن به لیست موقعیت‌ها اضافه می‌شود.

- تعداد تکرار توکن در همان سند به‌روزرسانی می‌شود.
- اگر توکن در سند جدید باشد:
  - ورودی جدیدی برای آن سند اضافه می‌شود که شامل موقعیت و تعداد تکرار است.

```python
def _build_postings_dict(Docs):
    """
    Build a postings dictionary from the input Docs, collecting token frequency
    and positions per document.
    """
    token_dict = {}
    for doc_id, doc_content in Docs.items():
        for position, token in enumerate(doc_content['content']):
            if token in token_dict:
                _update_existing_token_posting(token_dict, token, doc_id, position)
            else:
                _add_new_token_posting(token_dict, token, doc_id, position)
    return token_dict
```

3. تابع `_build_postings_dict`

این تابع یک **دیکشنری پستینگ** برای تمامی اسناد می‌سازد.

- اسناد (Docs) به صورت `doc_id` و محتوای آن بررسی می‌شوند.
- تمامی توکن‌های هر سند پیمایش شده و:
  - اگر توکن جدید باشد، به کمک تابع `_add_new_token_posting` اضافه می‌شود.
  - اگر توکن قبلاً وجود داشته باشد، به کمک تابع `_update_existing_token_posting` به‌روزرسانی می‌شود.

خروجی:
یک دیکشنری که شامل توکن‌ها، تعداد تکرار آن‌ها و اطلاعات مربوط به هر سند است.

```python
def _calculate_tf_idf(token_dict, total_docs):
    """
    Calculate TF-IDF for each token in each document.
    """
    for term, term_data in token_dict.items():
        term_docs = term_data['docs']
        n_t = len(term_docs)  # Number of documents containing this term

        for doc_id, doc_info in term_docs.items():
            tf = doc_info['number_of_token']
            # TF-IDF = log10(N / n_t) * (1 + log10(tf))
            tf_idf_value = (np.log10(total_docs / n_t)) * (1 + np.log10(tf))
            doc_info['tf_idf'] = tf_idf_value
```

این تابع **مقدار TF-IDF** را برای هر توکن در هر سند محاسبه می‌کند:

- برای هر توکن:
  - تعداد اسنادی که شامل این توکن هستند (n_t) محاسبه می‌شود.
  - سپس برای هر سند (doc_id) شامل توکن:
    - مقدار TF (تعداد تکرار توکن در سند) محاسبه می‌شود.
    - مقدار TF-IDF با فرمول زیر محاسبه شده و ذخیره می‌شود:

```
# TF-IDF = log10(N / n_t) * (1 + log10(tf))
```

تابع build_champions_list_and_docs_vectors_

این تابع دو وظیفه دارد:

1. **ساخت لیست قهرمانان (Champions List):**
   - توکن‌ها براساس تعداد تکرارشان در هر سند مرتب می‌شوند.
   - اگر تعداد اسناد بیش از مقدار champ_len باشد، **بالاترین champ_len سند** در لیست قهرمانان ذخیره می‌شود.
   - این لیست شامل اطلاعات number_of_token و TF-IDF برای هر سند است.

```python
def _build_champions_list_and_docs_vectors(token_dict, champ_len):
    """..."""
    docs_vectors = {}

    for term, term_data in token_dict.items():
        term_docs = term_data['docs']

        # Sort documents by number_of_token (descending) for the champion list
        sorted_term_docs = sorted(
            term_docs,
            key=lambda d: term_docs[d]['number_of_token'],
            reverse=True
        )

        # Build champion list (take top `champ_len` if needed)
        champions_list = {}
        for doc_id in sorted_term_docs:
            champions_list[doc_id] = {
                'number_of_token': term_docs[doc_id]['number_of_token'],
                'tf_idf': term_docs[doc_id]['tf_idf']
            }

        if champ_len < len(term_docs):
            champions_list = dict(list(champions_list.items())[:champ_len])

        token_dict[term]['champions_list'] = champions_list
```

2. **ساخت `docs_vectors`:**
   - بردار توکن‌های هر سند شامل اطلاعات TF-IDF و TF در قالب دیکشنری ذخیره می‌شود.

```python
    # Populate docs_vectors
    for doc_id, doc_info in term_docs.items():
        if doc_id not in docs_vectors:
            docs_vectors[doc_id] = {}
        docs_vectors[doc_id][term] = {
            'tf_idf': doc_info['tf_idf'],
            'tf': doc_info['number_of_token']
        }

return docs_vectors
```

تابع اصلی `Postings_List`

این تابع تمامی مراحل بالا را مدیریت می‌کند:

1. **ایجاد لیست پستینگ:**
   - ابتدا دیکشنری پستینگ با استفاده از تابع `_build_postings_dict` ساخته می‌شود.
2. **محاسبه TF-IDF:**
   - برای تمامی توکن‌ها و اسناد مقدار TF-IDF محاسبه می‌شود.
3. **ساخت لیست قهرمانان و بردارهای اسناد:**
   - با استفاده از تابع `_build_champions_list_and_docs_vectors`، لیست قهرمانان و بردارهای سند ایجاد می‌شود.

خروجی:

- دیکشنری پستینگ با اطلاعات کامل هر توکن.
- دیکشنری `docs_vectors` که بردارهای توکن هر سند را شامل می‌شود.

```python
def Postings_List(Docs, champ_len):
    """
    Orchestrates the creation of the postings list (token_dict) and docs_vectors.
    Steps:
      1) Build an initial postings dictionary with token frequencies/positions.
      2) Compute TF-IDF for each token in each document.
      3) Build the champions lists and docs_vectors.
    """
    # Step 1: Build the core postings dictionary
    token_dict = _build_postings_dict(Docs)

    # Step 2: Calculate TF-IDF
    total_docs = len(Docs)
    _calculate_tf_idf(token_dict, total_docs)

    # Step 3: Create champions lists and docs_vectors
    docs_vectors = _build_champions_list_and_docs_vectors(token_dict, champ_len)

    return token_dict, docs_vectors
```

```
dictionary, docs_vectors = Postings_List(pre_processed_docs, 20)
```
Executed at 2025.01.11 02:32:03 in 12s 559ms

```
docs_vectors['8332']
```
Executed at 2025.01.11 02:32:03 in 2ms

```
  ,{tf_idf': 3.003141219509179, 'tf': 2'} :'برف'
  ,{tf_idf': 0.6759912344460222, 'tf': 1'} :'پارلمان'
  ,{tf_idf': 1.9860604755388058, 'tf': 1'} :'دام'
  ,{tf_idf': 2.5549521036141134, 'tf': 1'} :'شر'
  ,{tf_idf': 3.38746101632035, 'tf': 1'} :'غلامرضاU200c\'
  ,{tf_idf': 1.8216131976468322, 'tf': 1'} :'شهرساز'
  ,{tf_idf': 1.928068528561119, 'tf': 1'} :'انتخابیه'
  ,{tf_idf': 2.3788608445584325, 'tf': 1'} :'دامدار'
  ,{tf_idf': 2.8559820992780947, 'tf': 1'} :'گلوگاه'
  ,{tf_idf': 3.132188511217044, 'tf': 1'} :'علوفه'
  ,{tf_idf': 6.036154116557285, 'tf': 3'} :'بهشهر'
  {{tf_idf': 4.086431020656368, 'tf': 1'} :'نکاء'
```

```
dictionary['فیلم']['champions_list']
```
Executed at 2025.01.11 02:32:03 in 4ms

```
'821': {'number_of_token': 25, 'tf_idf': 0.005047725200337222},
'2388': {'number_of_token': 24, 'tf_idf': 0.0050104056913280555},
'7744': {'number_of_token': 24, 'tf_idf': 0.0050104056913280555},
'525': {'number_of_token': 19, 'tf_idf': 0.00479683475620343},
'8680': {'number_of_token': 16, 'tf_idf': 0.004639729076116687},
'10183': {'number_of_token': 16, 'tf_idf': 0.004639729076116687},
'10025': {'number_of_token': 15, 'tf_idf': 0.00458072789429455},
'2831': {'number_of_token': 14, 'tf_idf': 0.00451765454084172},
'8897': {'number_of_token': 14, 'tf_idf': 0.00451765454084172},
'9687': {'number_of_token': 14, 'tf_idf': 0.00451765454084172},
'1684': {'number_of_token': 13, 'tf_idf': 0.004449904957144393},
'2098': {'number_of_token': 13, 'tf_idf': 0.004449904957144393}}
```

```
dictionary['کمپینسیون']
```
Executed at 2025.01.11 03:50:50 in 70ms

```
  {'frequency': 6298,
   'docs': {'13': {'positions': [79, 91],
     'number_of_token': 2,
     'tf_idf': 1.084825715800724},
    '89': {'positions': [301],
     'number_of_token': 1,
     'tf_idf': 0.8338206800889957},
    '100': {'positions': [149],
     'number_of_token': 1,
     'tf_idf': 0.8338206800889957},
    '106': {'positions': [34],
     'number_of_token': 1
```

```
'champions_list': {'11882': {'number_of_token': 55,
  'tf_idf': 2.2849710814446},
 '7506': {'number_of_token': 48, 'tf_idf': 2.2356743920311724},
 '9970': {'number_of_token': 46, 'tf_idf': 2.2202625461250296},
 '7401': {'number_of_token': 42, 'tf_idf': 2.187319507362553},
 '10042': {'number_of_token': 35, 'tf_idf': 2.121296546932828},
 '11449': {'number_of_token': 29, 'tf_idf': 2.053198373257889},
 '9500': {'number_of_token': 28, 'tf_idf': 2.040490973979017},
 '11143': {'number_of_token': 28, 'tf_idf': 2.040490973979017},
 '11163': {'number_of_token': 26, 'tf_idf': 2.0136547194018264},
```

ذخیره دیکشنری:

این کد یک دیکشنری را به دو قسمت مساوی تقسیم می‌کند. ابتدا تعداد کل کلیدهای دیکشنری محاسبه می‌شود و سپس کلیدها به دو بخش تقسیم می‌شوند: نیمه اول شامل اولین نصف کلیدها و نیمه دوم شامل بقیه کلیدها است. هر بخش به صورت جداگانه در فایل‌های `first_half.json` و `second_half.json` ذخیره می‌شود. داده‌ها در این فایل‌ها با فرمت JSON و به صورت خوانا ذخیره می‌شوند.

```
half_length = len(dictionary) // 2
first_half = {key: dictionary[key] for key in list(dictionary.keys())[:half_length]}
second_half = {key: dictionary[key] for key in list(dictionary.keys())[half_length:]}

with open('first_half.json', "w", encoding="utf-8") as first_file:
    json.dump(first_half, first_file, indent=4)

with open('second_half.json', "w", encoding="utf-8") as second_file:
    json.dump(second_half, second_file, indent=4)
```

۲. پاسخ‌دهی به پرسمان در فضای برداری

تابع `vector_length`

این تابع طول بردار وزن‌های TF-IDF یک سند را محاسبه می‌کند. این طول در محاسبه شباهت کوساین استفاده می‌شود.

- **ورودی**: دیکشنری‌ای که شامل وزن‌های TF-IDF کلمات یک سند است.
- **محاسبات**:
  - مربع تمام وزن‌های TF-IDF محاسبه و جمع زده می‌شود.
  - جذر جمع این مقادیر به عنوان طول بردار برگردانده می‌شود.
- **خروجی**: طول بردار.

```python
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
```

تابع `get_query_tokens`

عبارت جستجو را نرمال‌سازی و توکنایز می‌کند.

```python
def get_query_tokens(query):
    """
    Preprocesses the query into tokens.
    Returns a list of tokens.
    """
    return simple_preprocess(query)
```

تابع `get_query_tokens_count`

تعداد دفعات تکرار هر کلمه در عبارت جستجو را محاسبه می‌کند.

```python
def get_query_tokens_count(query_tokens):
    """
    Counts how many times each token appears in the query.
    Returns a dictionary of token -> frequency.
    """
    return dict(collections.Counter(query_tokens))
```

وزن‌دهی به کلمات عبارت جستجو

تابع `compute_query_weight`

این تابع وزن TF-IDF هر کلمه از عبارت جستجو را محاسبه می‌کند.

- **ورودی‌ها:**
  - `term`: کلمه‌ای که باید وزن آن محاسبه شود.
  - `query_tokens_count`: تعداد دفعات تکرار کلمه در عبارت جستجو.
  - `dictionary`: دیکشنری حاوی اطلاعات مربوط به کلمات و اسناد.
  - `champion_list`: مشخص می‌کند که آیا از لیست منتخب (champion list) استفاده شود یا کل اسناد.
  - `total_number_of_docs`: تعداد کل اسناد.
- **منطق:**
  - ابتدا بررسی می‌کند که آیا کلمه در دیکشنری وجود دارد یا نه. اگر کلمه موجود نباشد، مقدار 0 و یک دیکشنری خالی بازگردانده می‌شود.
  - وزن TF-IDF برای کلمه محاسبه می‌شود.
  - لیست اسناد مرتبط (لیست کامل یا منتخب) بازیابی می‌شود.

- **خروجی**: وزن TF-IDF کلمه و لیست اسناد حاوی آن

```
def compute_query_weight(term, query_tokens_count, dictionary, champion_list, total_number_of_docs):
    """
    Computes the TF-IDF weight of a query term.
    Returns w_tq (the query-term weight) and the term_docs to iterate over.
    """
    if term not in dictionary:
        return 0, {}

    # Retrieve the appropriate document list (champion list or full list)
    term_docs = (dictionary[term]['champions_list']
                 if champion_list
                 else dictionary[term]['docs'])

    w_tq = calculate_tf_idf(query_tokens_count[term],
                            total_number_of_docs,
                            len(term_docs))
    return w_tq, term_docs
```

به‌روزرسانی امتیازات اسناد

محاسبه طول بردار سند

تابع `vector_length`

طول بردار وزن‌های TF-IDF یک سند را محاسبه می‌کند که برای نرمال‌سازی شباهت کسینوسی استفاده می‌شود.

- **منطق**:
  - مربع تمام مقادیر TF-IDF جمع زده می‌شود.
  - جذر مجموع این مقادیر طول بردار را می‌دهد.
- **خروجی**: طول بردار.

```
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
```

پیش‌پردازش عبارت جستجو

تابع `get_query_tokens`

عبارت جستجو را نرمال‌سازی و توکنایز می‌کند و کلمات مجزا را استخراج می‌کند.

- **خروجی**: لیستی از کلمات جستجو.

تابع `get_query_tokens_count`

تعداد تکرار هر کلمه در عبارت جستجو را محاسبه می‌کند.

- **خروجی**: دیکشنری‌ای شامل هر کلمه و تعداد تکرار آن.

```
def get_query_tokens(query):
    """
    Preprocesses the query into tokens.
    Returns a list of tokens.
    """
    return simple_preprocess(query)
```

وزن‌دهی به کلمات جستجو

تابع `compute_query_weight`

وزن TF-IDF هر کلمه از عبارت جستجو را محاسبه می‌کند و لیست اسناد حاوی آن را برمی‌گرداند.

- **منطق**:
  - اگر کلمه در دیکشنری وجود نداشته باشد، مقدار 0 و یک دیکشنری خالی بازگردانده می‌شود.
  - لیست اسناد مرتبط با کلمه از دیکشنری (لیست کامل یا لیست منتخب) دریافت می‌شود.
  - وزن TF-IDF کلمه محاسبه می‌شود.
- **خروجی**: وزن TF-IDF کلمه و لیست اسناد مرتبط.

```
def compute_query_weight(term, query_tokens_count, dictionary, champion_list, total_number_of_docs):
    """
    Computes the TF-IDF weight of a query term.
    Returns w_tq (the query-term weight) and the term_docs to iterate over.
    """
    if term not in dictionary:
        return 0, {}

    # Retrieve the appropriate document list (champion list or full list)
    term_docs = (dictionary[term]['champions_list']
                 if champion_list
                 else dictionary[term]['docs'])

    w_tq = calculate_tf_idf(query_tokens_count[term],
                            total_number_of_docs,
                            len(term_docs))
    return w_tq, term_docs
```

به‌روزرسانی امتیازات اسناد

تابع `update_doc_scores`

امتیازات کسینوسی را برای اسنادی که کلمه مورد نظر را دارند، به‌روزرسانی می‌کند.

- **منطق:**
  - وزن TF-IDF کلمه در سند (`w_td`) در وزن کلمه در عبارت جستجو (`w_tq`) ضرب می‌شود.
  - اگر سند قبلاً امتیاز داشته باشد، به آن اضافه می‌شود؛ در غیر این صورت، مقدار اولیه تنظیم می‌شود.
- **خروجی:** به‌روزرسانی امتیازات در دیکشنری `cosine_scores`.

```
def update_doc_scores(term_docs, w_tq, cosine_scores):
    """
    Updates the cosine scores for each document that contains the term.
    """
    for doc in term_docs:
        w_td = term_docs[doc]['tf_idf']
        doc_id = int(doc)

        # Update cosines similarity
        if doc_id in cosine_scores:
            cosine_scores[doc_id] += w_td * w_tq
        else:
            cosine_scores[doc_id] = w_td * w_tq
```

محاسبه نهایی امتیازات کسینوسی

تابع `finalize_cosine_scores`

امتیاز نهایی شباهت کسینوسی را برای هر سند محاسبه می‌کند.

- **منطق:**
  - امتیاز هر سند بر طول بردار آن تقسیم می‌شود تا نرمال‌سازی انجام شود.
- **خروجی:** امتیازات نهایی نرمال‌شده.

```
def finalize_cosine_scores(cosine_scores):
    """
    Divides each document's cosine score by its vector length.
    """
    for doc_number in cosine_scores:
        cosine_scores[doc_number] /= vector_length(docs_vectors[str(doc_number)])
```

مرتب‌سازی امتیازات

تابع `sort_scores`

امتیازات اسناد را به ترتیب نزولی مرتب می‌کند.

- **خروجی**: لیستی از جفت‌های `(doc_id, score)` مرتب‌شده بر اساس امتیاز.

```python
def sort_scores(scores_dict):
    """
    Sorts scores in descending order by value.
    Returns a list of (doc_id, score) tuples.
    """
    return sorted(scores_dict.items(), key=lambda x: x[1], reverse=True)
```

تابع `query_scoring`

این تابع تمام مراحل بالا را ترکیب می‌کند و امتیازات نهایی را برای عبارت جستجو محاسبه می‌کند.

- **مراحل**:
  1. عبارت جستجو پیش‌پردازش می‌شود و تعداد تکرار کلمات آن محاسبه می‌شود.
  2. برای هر کلمه در عبارت جستجو:
     - وزن TF-IDF محاسبه می‌شود.
     - امتیازات کسینوسی برای اسناد مرتبط به‌روزرسانی می‌شود.
  3. امتیازات کسینوسی نهایی نرمال‌سازی می‌شوند.
  4. اسناد بر اساس امتیاز کسینوسی مرتب می‌شوند.
  5. تاپ k سند برتر برگردانده می‌شود.
- **خروجی**: لیستی از k سند برتر بر اساس شباهت کسینوسی

```python
def query_scoring(query, total_number_of_docs, dictionary, k, champion_list=False):
    # Initialize scores
    cosine_scores = {}

    # Preprocess query
    query_tokens = get_query_tokens(query)
    query_tokens_count = get_query_tokens_count(query_tokens)
    query_terms_num = sum(query_tokens_count.values())

    print(query_tokens_count)

    # Compute and update scores for each term in the query
    for term in query_tokens_count:
        w_tq, term_docs = compute_query_weight(term,
                                               query_tokens_count,
                                               dictionary,
                                               champion_list,
                                               total_number_of_docs)
        if w_tq != 0:
            update_doc_scores(term_docs, w_tq, cosine_scores)

    # Finalize scores
    finalize_cosine_scores(cosine_scores)

    # Sort and retrieve top k results
    sorted_doc_cosine = sort_scores(cosine_scores)

    return sorted_doc_cosine[:k]
```

```
r1, r2 = query_search('کریسمس', result_numbers = 5, champion_list = True)
Executed at 2025.01.11 03:51:12 in 5ms

{1 :'کریسمس'}
=== Cosine Scores ===
---------- Results ----------
Rank: 1 | ID: 5933
ستاره اسپانیایی؛ هدیه کریسمس گواردیولا به ژاوی+عکس : Title
ستاره-اسپانیایی-هدیه-کریسمس-گواردیولا-به-ژاوی-عکس/URL   : https://www.farsnews.ir/news/14001007000739
--------------------------------------------------
Rank: 2 | ID: 6117
کی‌روش «دیکتاتور» لقب گرفت/اختلاف مرد پرتغالی با مصری‌ها به خاطر کریسمس+عکس : Title
کی‌روش-دیکتاتور-لقب-گرفت-اختلاف-مرد-پرتغالی-با-مصری‌ها-به-خاطر-کریسمس/URL   : https://www.farsnews.ir/news/140
--------------------------------------------------
Rank: 3 | ID: 6120
```

الف) ساده:کلمه اخبار

{'اخبار': 1}

=== Cosine Scores ===

---------- Results ----------

Rank: 1 | ID: 4082

Title : انتصاب های جدید در باشگاه پرسپولیس رد شد

URL : https://www.farsnews.ir/news/14001101000140/انتصاب-های-جدید-در-باشگاه-پرسپولیس-رد-شد

Rank: 2 | ID: 3781

Title : گفت‌وگوی فارس با مسئول فدراسیون عراق درباره جزئیات کرونایی‌ها قبل از بازی با ایران

URL : https://www.farsnews.ir/news/14001104000419/گفت‌وگوی-فارس-با-مسئول-فدراسیون-عراق-درباره-جزئیات-کرونایی‌ها-قبل-از

Rank: 3 | ID: 5749

Title : پیش بینی مفسر مشهور فوتبال روسیه از آینده آزمون+عکس

URL : https://www.farsnews.ir/news/14001010000074/پیش-بینی-مفسر-مشهور-فوتبال-روسیه-از-آینده-آزمون-عکس

Rank: 4 | ID: 10436

Title : مقتدایی: درباره خلل یا آسیب در روند نگهداری برخی از اسناد سازمان اوقاف شفاف سازی شود

URL : https://www.farsnews.ir/news/14000909000220/مقتدایی-درباره-خلل-یا-آسیب-در-روند-نگهداری-برخی-از-اسناد-سازمان-اوقاف

Rank: 5 | ID: 5330

Title : از تکذیب توافق آزمون با لیون تا خوشحالی روس‌ها+عکس

URL : https://www.farsnews.ir/news/14001016000067/از-تکذیب-توافق-آزمون-با-لیون-تا-خوشحالی-روس‌ها-عکس

گفت‌وگوی فارس با مسئول فدراسیون عراق درباره جزئیات کرونایی‌ها قبل از بازی با ایران

مدیر روابط عمومی فدراسیون فوتبال عراق توضیحاتی را درباره جزئیات ابتلای بازیکنان و اعضای کاروان این تیم در تهران ارائه کرد.

به گزارش خبرگزاری فارس، امروز رسانه های عراق اخباری متفاوت از کرونایی های تیم ملی این کشور منتشر کردند. برخی از رسانه از 6 کرونایی، برخی دیگر از 5 کرونایی و اخباری از ابتلای 3 بازیکن مطرح شد. به همین خاطر خبرنگار خبرگزاری فارس برای تایید رسمی این اخبار به سراغ محمد عماد مدیر روابط عمومی فدراسیون فوتبال عراق رفت از واقعیت این اخبار مطلع شود. مدیر روابط عمومی فدراسیون فوتبال عراق در این زمینه به خبرنگار فارس، گفت: بله اخبار مطرح شده در زمینه ابتلای چند بازیکن و عضو تیم ملی صحت دارد. ما روز گذشته در تهران تست کرونا دادیم و امروز پاسخ تست ها آمد. وی درباره تعداد دقیق کرونایی‌های تیم ملی عراق در بدو ورود به تهران، گفت: 5 تست کرونا اعضای کاروان در تهران مثبت شد. در میان این کرونایی‌ها تست دو بازیکن به نام های یاسر قاسم و مناف یونس مثبت شد. همچنین تست کرونای احمد جاسم مربی دروازه بان های عراق، غیث مهنا مدیر اداری و یاسین خبیر پزشک تیم مثبت شده است. مدیر روابط عمومی فدراسیون عراق

در این ها خبر تعداد تکرارهای کلمه اخبار زیاد است چون از لیست قهرمان استفاده شده که در آن داکیومنت ها با بیشترین tf قرار دارندو اگر همین سرچ بدون لیست قهرمان انجام میشد احتمالا داکیومنت ها کوتاه‌تر و با تعداد تکرار کمتر کلمه اخبار بود.
زیاد بودن تکرار کلمه اخبار نشانه از درست بودن کوئری است.

ب) ساده و مرکب: باشگاه والیبال

{'باشگاه': 1, 'فوتبال': 1}

=== Cosine Scores ===

---------- Results ----------

Rank: 1 | ID: 4238

Title : منافی: امیدواریم درویش بتواند از پس مشکلات پرسپولیس بر بیاید/ هیات مدیره جای بگو و بخند نیست

URL : https://www.farsnews.ir/news/14001028000893/منافی-امیدواریم-درویش-بتواند-از-پس-مشکلات-پرسپولیس-بر-بیاید-هیات

Rank: 2 | ID: 2642

Title : شهرداری ارومیه به دنبال دور زدن فدراسیون جهانی والیبال/ نه پول هست، نه وکیل

URL : https://www.farsnews.ir/news/14001118000468/شهرداری-ارومیه-به-دنبال-دور-زدن-فدراسیون-جهانی-والیبال-نه-پول-هست-نه

Rank: 3 | ID: 5403

Title : اولیایی: امیدواریم AFC قول وزیر ورزش را بپذیرد/ واگذاری سهام استقلال و پرسپولیس به مردم سراب است

URL :
https://www.farsnews.ir/news/14001014001185/اولیایی-امیدواریم-AFC-قول-وزیر-ورزش-را-بپذیرد-واگذاری-سهام-استقلال-و

Rank: 4 | ID: 2391
Title : عطابخش: به جز ۲-۳ بازیکن سایر نفرات‌مان می‌توانند در لیگ امید بازی کنند/ پیکان در بودجه مضیقه دارد
URL :
https://www.farsnews.ir/news/14001122000769/عطابخش-به-جز-۲-۳-بازیکن-سایر-نفرات‌مان-می‌توانند-در-لیگ-امید-بازی

Rank: 5 | ID: 534
Title : پولادگر: خیلی از باشگاه‌های ما فقط تابلوی خصوصی دارند/ ساختمان‌ها به استقلال و پرسپولیس هبه شده
URL :
https://www.farsnews.ir/news/14001216001105/پولادگر--خیلی-از-باشگاه‌های-ما-فقط-تابلوی-خصوصی-دارند-ساختمان‌ها-به


ورزشی
@Sports
تکلیف میزبان مسابقات والیبال جام باشگاه‌های آسیا مشخص شد
با انصراف باشگاه شهرداری ارومیه از میزبانی رقابت‌های والیبال قهرمانی باشگاه‌های مردان آسیا، این مسابقات در سالن فدراسیون در تهران برگزار خواهد شد.
به گزارش خبرگزاری فارس، رقابت‌های والیبال قهرمانی باشگاه های مردان آسیا از 25 اردیبهشت تا یکم خردادماه سال 1401 با حضور 11 تیم از 10 کشور قاره کهن برگزار خواهد شد. پیش از این قرار بود این رقابت‌ها به میزبانی شهرداری ارومیه برگزار شود که با توجه به محرومیت‌های مالی این باشگاه ازسوی فدراسیون جهانی والیبال و با وجود دادن چندین مهلت و فرصت فدراسیون والیبال به شهرداری ارومیه، این باشگاه نتوانست رضایت شاکیان خود از FIVB را جلب نماید. مدیرکل ورزش و جوانان استان آذربایجان غربی و مسئولین باشگاه شهرداری ارومیه با وجود تلاش‌های و مذاکرات فراوان اعلام کردند که نمی‌توانند این مشکلات را برطرف کنند و از حضور و میزبانی رقابت‌های والیبال قهرمانی باشگاه‌های مردان آسیا انصراف دادند. این در حالی بود که مسئولین فدراسیون والیبال در نظر داشتند آذرباتری را به عنوان میزبان این رقابت‌ها اعلام کنند که مسئولین این تیم هم اعلام کردند نمی‌توانند میزبانی این رقابت‌ها را بر عهده گیرند. بر این اساس و با توجه به ترافیک رویدادهای بین‌المللی والیبال ایران در سال آینده، فدراسیون والیبال تصمیم گرفت که رقابت‌های والیبال قهرمانی باشگاه‌های مردان آسیا 2022 را در سالن والیبال مجموعه ورزشی آزادی تهران برگزار کند و تهران را به عنوان شهر میزبان این رقابت ها به کنفدراسیون والیبال آسیا معرفی کرد.
بدین ترتیب علاوه بر قهرمان لیگ برتر والیبال سال 1400 ایران، یک تیم دیگر از ایران به‌عنوان میزبان این رقابت‌ها در مسابقات باشگاه‌های مردان آسیا حضور خواهد یافت که فدراسیون والیبال در روزهای آینده پس از پایان لیگ برتر ۱۴۰۰ در این خصوص تصمیم‌گیری خواهد کرد انتهای پیام/.

این سند با اینکه دو کلمه کوئری را دارد ولی چون هر دو به نسبت ساده و رایج هستند و در ترکیب با کلمات دیگر هستند. برای همین در سند می توان دید که این دو کلمه در کنار هم نیامده اند و ممکن است اسناد غیرمرتبط برگردانده شود.

پ) دشوار تک کلمه ای:کریسمس

{'کریسمس': 1}

=== Cosine Scores ===

---------- Results ----------

Rank: 1 | ID: 5933

Title : ستاره اسپانیایی؛ هدیه کریسمس گواردیولا به ژاوی+عکس

URL :

https://www.farsnews.ir/news/14001007000739/ستاره-اسپانیایی-هدیه-کریسمس-گواردیولا-به-ژاوی-عکس

Rank: 2 | ID: 6117

Title : کی‌روش «دیکتاتور» لقب گرفت/اختلاف مرد پرتغالی با مصری‌ها به خاطر کریسمس+عکس

URL : https://www.farsnews.ir/news/14001005000165/کی‌روش-دیکتاتور-لقب-گرفت-اختلاف-مرد-پرتغالی-با-مصری‌ها-به-خاطر-کریسمس

Rank: 3 | ID: 6120

Title : کشتار در ورزشگاه فوتبال در آستانه سال جدید

URL : https://www.farsnews.ir/news/14001005000143/کشتار-در-ورزشگاه-فوتبال-در-آستانه-سال-جدید

Rank: 4 | ID: 5926

Title : مهاجم خارجی مس رفسنجان،4 کودک را به محل تحصیل برگرداند +عکس

URL : https://www.farsnews.ir/news/14001007000809/مهاجم-خارجی-مس-رفسنجان4-کودک-را-به-محل-تحصیل-برگرداند-عکس

Rank: 5 | ID: 5483

Title : مسی چه زمانی به پاریس برمی گردد؟

URL : https://www.farsnews.ir/news/14001014000228/مسی-چه-زمانی-به-پاریس-برمی-گردد

تعداد تکرار با توجه به دشوار بودن کلمه پایین امده است ولی با توجه به اینکه کریسمس کلمه ای خاص است و به شکل کلی تکرار نمیشود در برخی مواقع مانند خبر بالا ارتباط بیشتری نسبت به حالت های قبل دارد به سند زیرا رایج نیست و محدوده جست و جو کمتر است.

ج)دشوار چند کلمه‌ای: کمیسیون کریسمس

{'کمیسیون': 1, 'کریسمس': 1}

=== Cosine Scores ===

---------- Results ----------

Rank: 1 | ID: 5933

Title : ستاره اسپانیایی؛ هدیه کریسمس گواردیولا به ژاوی+عکس

URL : https://www.farsnews.ir/news/14001007000739/ستاره-اسپانیایی-هدیه-کریسمس-گواردیولا-به-ژاوی-عکس

Rank: 2 | ID: 6117

Title : کی‌روش «دیکتاتور» لقب گرفت/اختلاف مرد پرتغالی با مصری‌ها به خاطر کریسمس+عکس

URL : https://www.farsnews.ir/news/14001005000165/کی‌روش-دیکتاتور-لقب-گرفت-اختلاف-مرد-پرتغالی-با-مصری‌ها-به-خاطر-کریسمس

Rank: 3 | ID: 6120

Title : کشتار در ورزشگاه فوتبال در آستانه سال جدید

URL : https://www.farsnews.ir/news/14001005000143/کشتار-در-ورزشگاه-فوتبال-در-آستانه-سال-جدید

Rank: 4 | ID: 5926

Title : مهاجم خارجی مس رفسنجان،4 کودک را به محل تحصیل برگرداند +عکس

URL : https://www.farsnews.ir/news/14001007000809/مهاجم-خارجی-مس-رفسنجان4-کودک-را-به-محل-تحصیل-برگرداند-عکس

Rank: 5 | ID: 5483

Title : مسی چه زمانی به پاریس برمی گردد؟

URL : https://www.farsnews.ir/news/14001014000228/مسی-چه-زمانی-به-پاریس-برمی-گردد

کریسمس چون کلمه خاصی هست امتیاز آن بالاتر رفته و به عنوان مثال در داکیومنت اول میبینیم که هیچ نشانی ای از کمیسیون وجود ندارد چون کلمه عام تری هستش. بنابراین در این حالت امکان دارد ارتباط با

کوئری پایین تر بیاید هنگامی که کوئری خاص است.