بسم الله الرحمن الرحيم

پروژه پیاده‌سازی سیستم بازیابی اطلاعات

استاد درس:

دکتر شهلا نعمتی

نام دانشجویان:

علی بدیعی گورتی – اشکان پورعبدالله

بهار 1403

فهرست مطالب

مقدمه

در این پروژه، ما به دنبال پیاده‌سازی یک سیستم بازیابی اطلاعات (IR) هستیم که به صورت قدم به قدم پیش می‌رویم. در این مرحله اول، داده‌های مورد نیاز از سایت مربوطه را دانلود و پیش‌پردازش می‌کنیم تا برای مراحل بعدی آماده شوند. برنامه‌ای که برای این منظور نوشته شده، قادر به دریافت متن از کاربر به صورت مستقیم یا از طریق فایل است و با استفاده از کتابخانه‌ی Hazm، متن را پردازش کرده و نتیجه را به کاربر نمایش می‌دهد.

اهداف و انگیزه ایجاد پروژه

هدف اصلی این پروژه، ایجاد یک سیستم بازیابی اطلاعات کارآمد و قابل اعتماد است که بتواند به سرعت و با دقت بالا، اطلاعات مورد نیاز کاربران را بازیابی کند. این پروژه به دنبال ارائه‌ی یک راه‌حل عملی و کاربردی برای مسأله‌ی بازیابی اطلاعات است که بتواند نیازهای گسترده‌ی کاربران را برآورده کند.

1. حل یک مسأله:

با پیاده‌سازی این سیستم بازیابی اطلاعات، به مسأله‌ی دسترسی به اطلاعات و داده‌ها در زمان مناسب و با سرعت بالا پرداخته می‌شود. این پروژه سعی دارد با استفاده از فناوری‌های مدرن و روش‌های به‌روز، به یک راه‌حل کارآمد و موثر برای بازیابی اطلاعات دست یابد.

2. ابتکار و خلاقیت:

در این پروژه، به دنبال ارائه‌ی یک سیستم نوآورانه و خلاقانه برای بازیابی اطلاعات هستیم. با بهره‌گیری از ابتکار و خلاقیت، تلاش می‌شود تا یک راه‌حل منحصر به فرد و با کیفیت برای مسأله‌ی بازیابی اطلاعات ارائه شود که به بهترین شکل ممکن، نیازهای کاربران را برآورده سازد.

3. ساخت یک کسب و کار:

این پروژه، علاوه بر حل یک مسأله فنی، فرصتی استوار برای ایجاد یک محصول تجاری محبوب است. با ارائه‌ی یک سیستم بازیابی اطلاعات کارآمد، این پروژه امکان ایجاد یک کسب و کار موفق و پربازده را فراهم می‌آورد که می‌تواند به عنوان یک منبع درآمد پایدار برای توسعه‌دهندگان خود شناخته شود.

هدف و انگیزه ایجاد پروژه معمولاً به عوامل فردی، اجتماعی، یا تجاری مرتبط است و ممکن است با گذر زمان تغییر کند یا با توسعه پروژه تغییر یابد.

فصل اول: مروری از مبانی و مفاهیم سیستم بازیابی اطلاعات و کتابخانه Hazm

1.1 مقدمه

فصل مروری از مبانی و مفاهیم سیستم بازیابی اطلاعات یک فرصت است تا به عمق در مفاهیم اساسی این حوزه نگاهی دقیق بیندازیم. این سیستم‌ها در واقع به تجزیه و تحلیل محتوای متنی برای یافتن اطلاعات مورد نیاز کاربران کمک می‌کنند. یکی از ابزارهایی که برای پردازش متن و ایجاد سیستم بازیابی اطلاعات استفاده می‌شود، زبان برنامه‌نویسی پایتون است. پایتون به عنوان یک زبان برنامه‌نویسی قدرتمند و پرکاربرد، امکانات فراوانی برای پردازش متن و ایجاد سیستم‌های بازیابی اطلاعات فراهم می‌کند.

سیستم بازیابی اطلاعات چیست؟

این سیستم‌ها به کمک الگوریتم‌ها و فنون مختلف، به کاربران امکان می‌دهند تا اطلاعات مورد نیاز خود را از مجموعه‌ی اسناد موجود بازیابی کنند. این اسناد ممکن است متنی، صوتی، تصویری یا حتی ویدیویی باشند. سیستم‌های بازیابی اطلاعات با استفاده از الگوریتم‌های مختلفی مانند TF-IDF، مدل‌های برداری اسناد (Document Embedding)، یادگیری ماشین، و شبکه‌های عصبی، سعی در یافتن اسناد مرتبط با کوئری‌های کاربر دارند.

1.2 کتابخانه Hazm

استفاده از کتابخانه‌های پایتون مانند Hazm، که برای پردازش متن به زبان فارسی طراحی شده است، بسیار مفید است. این کتابخانه ابزارها و توابعی را ارائه می‌دهد که به کاربران این امکان را می‌دهد تا متون فارسی را به آسانی پردازش کنند، بدون

اینکه نیاز به نوشتن کدهای پیچیده و زمان‌بر داشته باشند. به عنوان مثال، با استفاده از توابع توکن‌بندی، نرمالیزاسیون، و حذف کلمات توقف در کتابخانه Hazm، می‌توانیم مراحل پیش‌پردازش متن را به راحتی انجام دهیم. علاوه بر این، ابزارهای متنی مانند استخراج واژگان کلیدی، تشخیص و تحلیل موضوع، و تحلیل احساسات نیز از مفاهیم اساسی در زمینه پردازش متن هستند که می‌توانند در سیستم بازیابی اطلاعات مورد استفاده قرار گیرند. این ابزارها نیز بر اساس الگوریتم‌های مختلفی که در کتابخانه‌ی Hazm یافت می‌شوند، پیاده‌سازی می‌شوند و می‌توانند به بهبود کیفیت و دقت سیستم بازیابی اطلاعات کمک کنند.

1.1.2 نحوه‌ی نصب Hazm

برای نصب کتابخانه Hazm، ابتدا با دستور زیر در ترمینال یا کامندلاین آن را نصب کنید:

pip install hazm

سپس باید منتظر بمانید تا همه‌ی کتابخانه‌های مرتبط با آن نیز دانلود و نصب شوند. پس از پایان نصب، این کتابخانه آماده‌ی استفاده است.

1.2.۲ کلاس Normalizer

این کلاس شامل پارامترهایی برای نرمال‌سازی متن است:

نام	نوع	توضیحات	پیش‌فرض
`correct_spacing`	`bool`	اگر `True` فاصله‌گذاری‌ها را در متن، نشانه‌های سجاوندی و پیشوندها و پسوندها اصلاح می‌کند.	`True`
`remove_diacritics`	`bool`	اگر `True` باشد اعرابِ حروف را حذف می‌کند.	`True`
`remove_specials_chars`	`bool`	اگر `True` باشد برخی از کاراکترها و نشانه‌های خاص را که کاربردی در پردازش متن ندارند حذف می‌کند.	`True`
`decrease_repeated_chars`	`bool`	اگر `True` باشد تکرارهای بیش از ۲ بار را به ۲ بار کاهش می‌دهد. مثلاً «سلاممم» را به «سلامم» تبدیل می‌کند.	`True`
`persian_style`	`bool`	اگر `True` باشد اصلاحات مخصوص زبان فارسی را انجام می‌دهد؛ مثلاً جایگزین‌کردن کوتیشن با گیومه.	`True`
`persian_numbers`	`bool`	اگر `True` باشد ارقام انگلیسی را با فارسی جایگزین می‌کند.	`True`
`unicodes_replacement`	`bool`	اگر `True` باشد برخی از کاراکترهای یونیکد را با معادل نرمال‌شدهٔ آن جایگزین می‌کند.	`True`
`seperate_mi`	`bool`	اگر `True` باشد پیشوند «می» و «نمی» را در افعال جدا می‌کند.	`True`

تمامی این موارد به صورت تابع نیز موجود هستند. ما در این پروژه از آن به صورت زیر استفاده کردیم:

```
hazm.Normalizer(correct_spacing=True, remove_diacritics=True, remove_specials_chars=True,
                decrease_repeated_chars=True, persian_style=True, persian_numbers=True,
                seperate_mi=True)
```

و تابع normalize را با آن نوشتیم.

همچنین، از تابع `token_spacing(tokens)` نیز استفاده کردیم و آن را برای پروژه بازنویسی کردیم. این تابع توکن‌های ورودی را به فهرستی از توکن‌های نرمال‌سازی شده تبدیل می‌کند. در این فرایند، ممکن است برخی از توکن‌ها به یکدیگر بچسبند؛ استفاده از این قابلیت در اختیاری است.

به عنوان مثال، ['زمین', 'لرزه', 'ای'] تبدیل می‌شود به ['زمین‌لرزه‌ای'].

۱.۲.۳ کلاس InformalNormalizer

این کلاس شامل پارامترهایی برای نرمال‌سازی متن‌های محاوره‌ای است:

نام	نوع	توضیحات	پیش‌فرض
`verb_file`	`str`	فایل حاوی افعال محاوره‌ای.	`informal_verbs`
`word_file`	`str`	فایل حاوی کلمات محاوره‌ای.	`informal_words`
`seperation_flag`	`bool`	اگر `True` باشد و در بخشی از متن به فاصله نیاز بود آن فاصله درج می‌شود.	`False`
`**kargs`	`str`	پارامترهای نامدارِ اختیاری	`{}`

این موارد به صورت تابع نیز موجود هستند. ما در این پروژه از آن به صورت زیر استفاده کردیم:

```
hazm.InformalNormalizer(seperation_flag=True)
```

و تابع informal_normalize را با آن نوشتیم.

1.2.4 کلاس Stemmer

این کلاس شامل پارامترهایی برای نرمال‌سازی ریشه‌یابی کلمات است:

نام	نوع	توضیحات	پیش‌فرض
`word`	`str`	کلمه‌ای که باید ریشهٔ آن پیدا شود.	**اجباری**

ما در این پروژه از آن به صورت زیر استفاده کردیم:

```
hazm.Stemmer()
```

و تابع stem را با آن نوشتیم.

فرق بین **Lemmatizer** و **Stemmer** این است که اِستمر درکی از معنای کلمه ندارد و صرفاً براساس حذف برخی از پسوندهای ساده تلاش می‌کند ریشه‌ی کلمه را بیابد؛ بنابراین ممکن است در ریشه‌یابیِ برخی از کلمات نتایج نادرستی ارائه دهد؛ اما لماتایزر براساس لیستی از کلمات مرجع به همراه ریشه‌ی آن این کار را انجام می‌دهد و نتایج دقیق‌تری ارائه می‌دهد. البته هزینه‌ی این دقت، سرعتِ کمتر در ریشه‌یابی است.

1.2.5 کلاس Lemmatizer

این ماژول شامل کلاس‌ها و توابعی برای ریشه‌یابی کلمات است:

نام	نوع	توضیحات	پیش‌فرض
`words_file`	`str`	ریشه‌یابی کلمات از روی این فایل صورت می‌گیرد. هضم به صورت پیش‌فرض فایلی برای این منظور در نظر گرفته است؛ با این حال شما می‌توانید فایل موردنظر خود را معرفی کنید. برای آگاهی از ساختار این فایل به فایل پیش‌فرض مراجعه کنید.	`default_words`
`verbs_file`	`str`	اشکال صرفی فعل از روی این فایل ساخته می‌شود. هضم به صورت پیش‌فرض فایلی برای این منظور در نظر گرفته است؛ با این حال شما می‌توانید فایل موردنظر خود را معرفی کنید. برای آگاهی از ساختار این فایل به فایل پیش‌فرض مراجعه کنید.	`default_verbs`
`joined_verb_parts`	`bool`	اگر `True` باشد افعال چندبخشی را با کاراکتر زیرخط به هم می‌چسباند.	`True`

ما در این پروژه از آن به صورت زیر استفاده کردیم:

```
self.lemmatizer = hazm.Lemmatizer()
```

و تابع lemmatize را با آن نوشتیم.

این کلاس، زیر کلاس ها و توابع بسیاری در اختیار دارد ولی از آن ها در این پروژه صرف نظر میکنیم.

1.2.6 کلاس WordTokenizer

این کلاس شامل توابعی برای استخراج کلماتِ متن است:

نام	نوع	توضیحات	پیش‌فرض
`words_file`	`str`	مسیر فایل حاوی لیست کلمات. هضم به صورت پیش‌فرض فایلی برای این منظور در نظر گرفته است؛ با این حال شما می‌توانید فایل موردنظر خود را معرفی کنید. برای آگاهی از ساختار این فایل به فایل پیش‌فرض مراجعه کنید.	`default_words`
`verbs_file`	`str`	مسیر فایل حاوی افعال. هضم به صورت پیش‌فرض فایلی برای این منظور در نظر گرفته است؛ با این حال شما می‌توانید فایل موردنظر خود را معرفی کنید. برای آگاهی از ساختار این فایل به فایل پیش‌فرض مراجعه کنید.	`default_verbs`
`join_verb_parts`	`bool`	اگر `True` باشد افعال چندبخشی را با خط زیر به هم می‌چسباند؛ مثلاً «گفته شده است» را به صورت «گفته_شده_است» برمی‌گرداند.	`True`
`join_abbreviations`	`bool`	اگر `True` باشد مخفف‌ها را نمی‌شکند و به شکل یک توکن برمی‌گرداند.	`False`
`separate_emoji`	`bool`	اگر `True` باشد اموجی‌ها را با یک فاصله از هم جدا می‌کند.	`False`
`replace_links`	`bool`	اگر `True` باشد لینک‌ها را با کلمهٔ `LINK` جایگزین می‌کند.	`False`
`replace_ids`	`bool`	اگر `True` باشد شناسه‌ها را با کلمهٔ `ID` جایگزین می‌کند.	`False`
`replace_emails`	`bool`	اگر `True` باشد آدرس‌های ایمیل را با کلمهٔ `EMAIL` جایگزین می‌کند.	`False`
`replace_numbers`	`bool`	اگر `True` باشد اعداد اعشاری را با `NUMF` و اعداد صحیح را با `NUM` جایگزین می‌کند. در اعداد غیراعشاری، تعداد ارقام نیز جلوی `NUM` می‌آید.	`False`
`replace_hashtags`	`bool`	اگر `True` باشد علامت `#` را با `TAG` جایگزین می‌کند.	`False`

ما در این پروژه از آن به صورت زیر استفاده کردیم:

```
hazm.WordTokenizer(join_verb_parts=True, replace_links=True, replace_emails=True,
                   replace_ids=True, replace_numbers=True, replace_hashtags=True)
```

و تابع tokenize را با آن نوشتیم.

1.2.7 تابع remove_string_stopwords

این تابع بر اساس دیکشنری‌ای که برای آن مشخص می‌کنیم، تمامی کلمات توقف را از متن حذف می‌کند. به صورت پیش‌فرض، کوچک‌ترین دیکشنری برای آن تنظیم شده است که می‌توان با پارامترهای مناسب آن را تعویض کرد.

1.2.8 تابع remove_punctuations

این تابع برای حذف نشان‌های اضافی استفاده شده در متن استفاده می‌شود. دیکشنری کاراکترهای اضافی موجود در متون انگلیسی با فارسی جمع شده و آن را بسیار کامل می‌کند.

فصل دوم: انجام فاز اول پروژه

اهداف اصلی این فاز

بخش اول آماده‌سازی اولیه داده‌ها

این بخش شامل مراحلی مانند یکسان‌سازی متن، جدا کردن لغات، بازگرداندن به ریشه، حذف کلمات پرتکرار و سایر فرآیندهای پیش‌پردازش است.

بخش دوم پیاده‌سازی یک رابط کاربری

در این بخش، یک رابط کاربری برای تعامل با سیستم طراحی و پیاده‌سازی می‌شود.

بخش اول

1. در این بخش، ابتدا فایل Phases ساخته شد و سپس موارد زیر به صورت کامل در آن پیاده‌سازی شد که در فصل قبل به طور مفصل در مورد آن‌ها صحبت شده است:
2. Normalization
3. Tokenization
4. Remove Stop Words:

تابعی به نام read_stopwords نوشته شد تا بر اساس مسیر ورودی، دیکشنری را خوانده و مقدار آن را برگرداند.

5. Lemmatization
6. Stemming
7. Remove Punctuations

و ...

همچنین توابع زیر نیز پیاده‌سازی شدند تا در برنامه‌ی اصلی استفاده شوند:

1. تابع state_updater:

این تابع برای آپدیت کردن مقدار progressbar در پروژه استفاده می‌شود که همزمان درصد آن را جلو برده و مقدار رشته‌ای را روی آن مشخص می‌کند.

2. تابع preprocess_text:

این تابع برای بخش Text Process استفاده می‌شود و با ورود یک متن، آن را بر اساس checkbox هایی که انتخاب شده‌اند پروسس می‌کند. همچنین همزمان از تابع state_updater برای نشان دادن روند استفاده می‌شود.

3. تابع internal_preprocess_text:

این تابع برای بخش File Process استفاده می‌شود و با توجه به ورودی متن از یک فایل، آن را بر اساس checkbox هایی که انتخاب شده‌اند پروسس کرده و مقدار آن را بر می‌گرداند.

4. تابع preprocess_files:

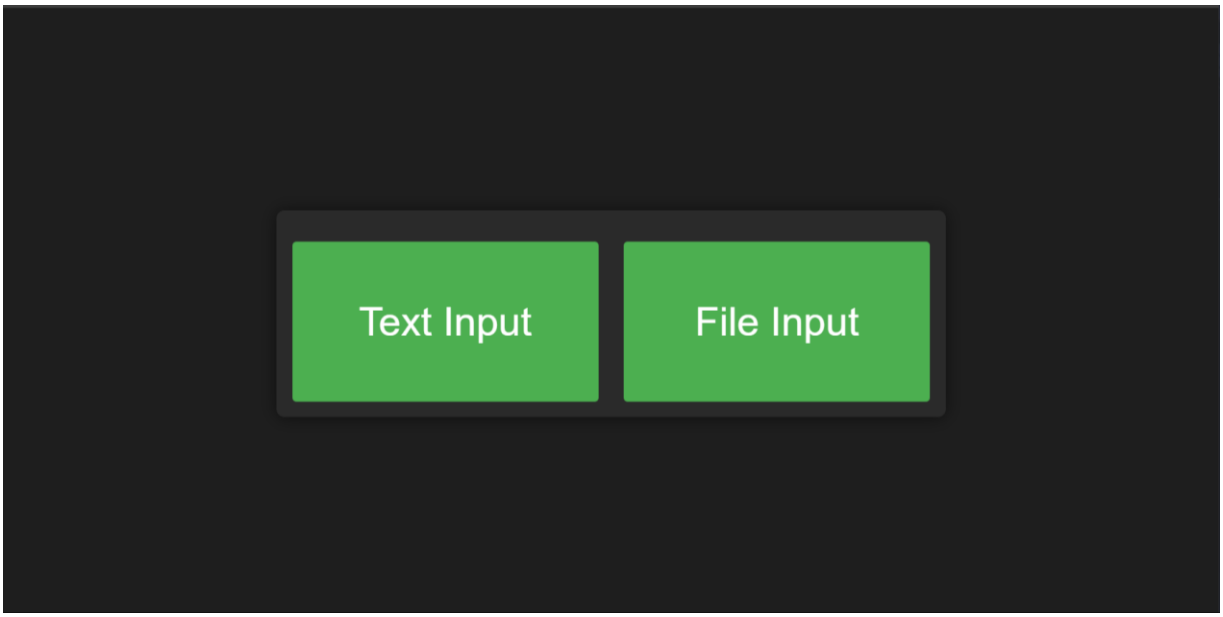
این تابع برای بخش File Process استفاده می‌شود به صورتی که نام دایرکتوری‌های مبدا و مقصد را به صورت پارامتر گرفته و آن‌ها را بررسی و اعمال را بر روی آن‌ها انجام می‌دهد. همچنین با استفاده از internal_preprocess_text و state_updater هر فایل را پروسس و مقادیر را آپدیت می‌کند.

بخش دوم

در این بخش، ما برای ایجاد یک رابط کاربری کارآمد و کاربرپسند از فریم‌ورک Django استفاده کردیم. پس از نصب نیازمندی‌های پروژه که در فایل requirements.txt نوشته شده‌اند، می‌توانید با دستور زیر آن را اجرا کنید:

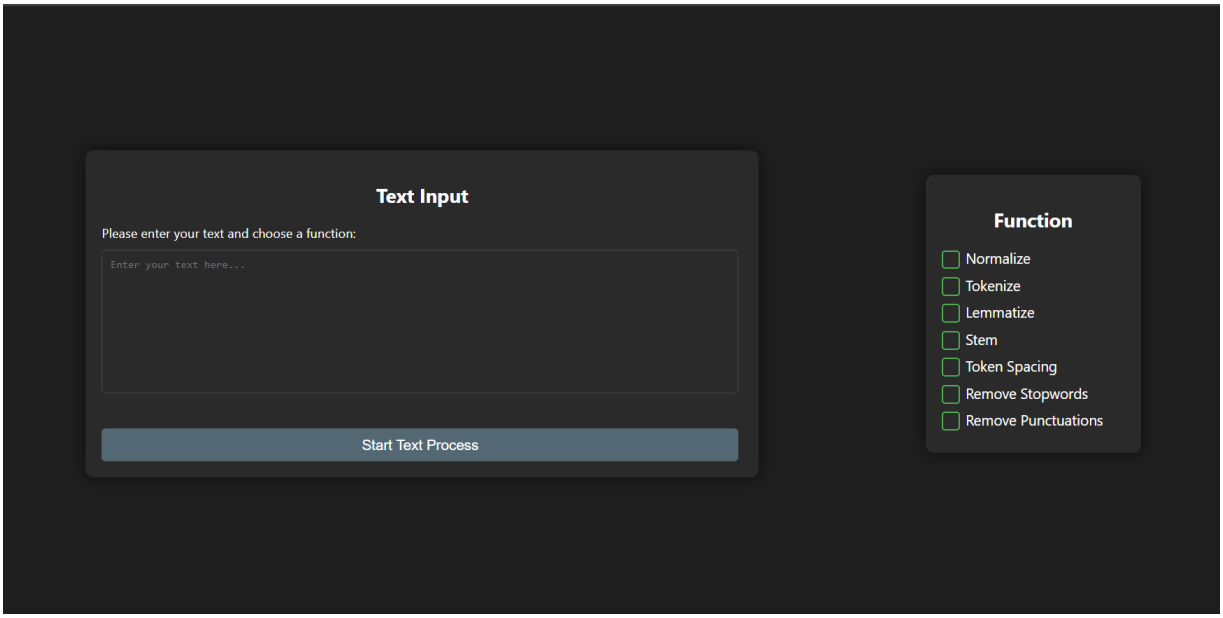
python manage.py runserver

سپس می‌توانید به آدرس 127.0.0.1:8000 مراجعه کرده و صفحه‌ی زیر را مشاهده کنید:

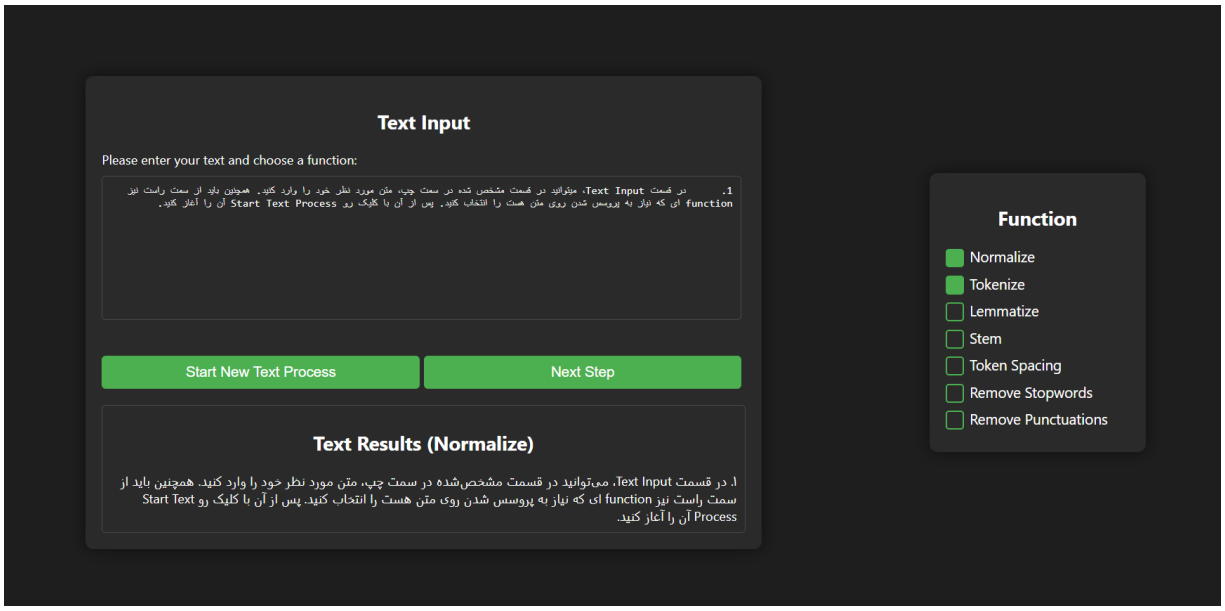


سپس با انتخاب هر کدام، می‌توانید به صفحه‌ی مربوطه رفته و از ویژگی‌ها و عملکردهای مربوطه استفاده کنید.

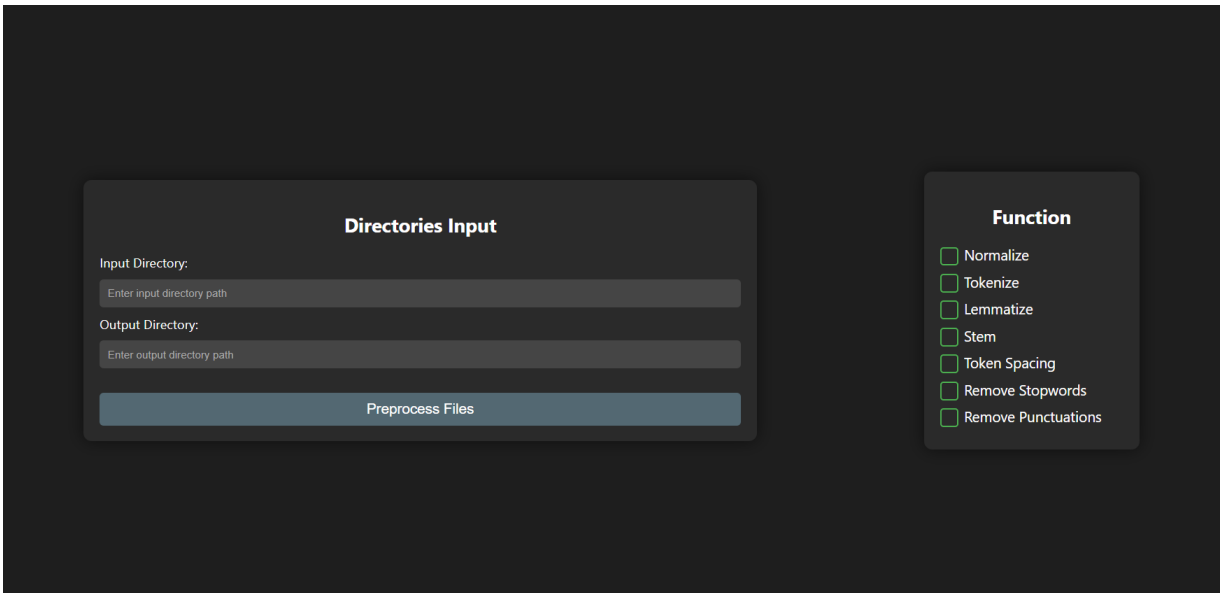
1. در بخش Text Input، می‌توانید در جایگاه مشخص شده در سمت چپ، متن مورد نظر خود را وارد کنید. همچنین، از سمت راست نیز می‌بایست تابعی که نیاز به پردازش روی متن دارید را انتخاب کنید. پس از آن، با کلیک بر روی Start Text Process، پردازش متن را آغاز کنید.



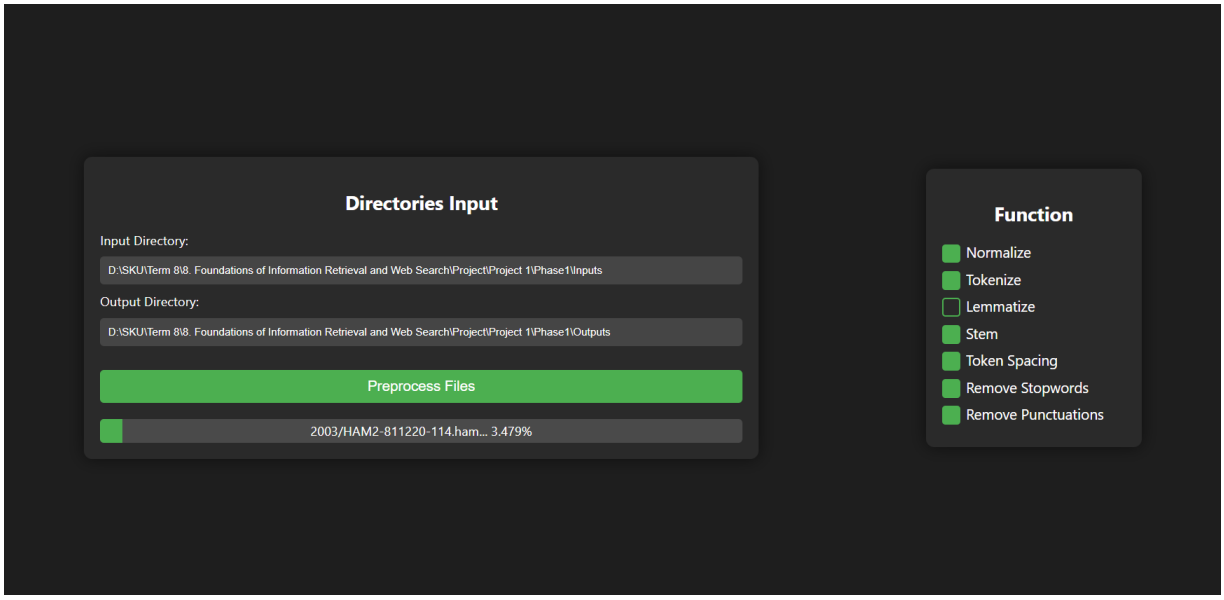
پس از تکمیل مراحل مورد نیاز، می‌توانید تصویر زیر را مشاهده کنید. در این تصویر، می‌توانید بین مراحل مختلف جابجا شده و خروجی هر بخش را به صورت جداگانه مشاهده کنید. این مراحل به ترتیب اعمال می‌شوند و ترتیب آن‌ها بر اساس ترتیب نوشته شده در ستون سمت راست است. به عنوان مثال، اگر گزینه‌های Normalize و Tokenize را انتخاب کنید، ابتدا Normalize اعمال شده و خروجی آن ذخیره می‌شود. سپس، tokenize روی خروجی قسمت قبل اعمال می‌شود و در آخر، خروجی آن ذخیره و به کاربر نمایش داده می‌شود.



2. در بخش File Input، می‌توانید در قسمت سمت چپ، آدرس دایرکتوری‌های ورودی و خروجی را مشخص کرده و از سمت راست نیز گزینه‌های مورد نیاز خود را انتخاب کنید. سپس بر روی گزینه Preprocess Files کلیک کرده و منتظر اجرای آن توابع بر روی همه‌ی فایل‌های ورودی باشید.



همچنین، میزان پیشرفت اعمال این توابع در قسمت progress bar مشخص می‌شود:



فاز دوم را به صورت کلاسی نوشته ایم که وظیفه مدیریت شاخص‌های متنی (indexing) را بر عهده دارد. این شاخص‌ها شامل شاخص‌های غیرمکانی (non-positional index)، شاخص‌های مکانی (positional index) و شاخص‌های جایگشتی (wildcard index) می‌باشند. علاوه بر این، کلاس قابلیت فشرده‌سازی شاخص‌ها و به‌روزرسانی وضعیت پردازش را نیز دارد. در این گزارش به بررسی جزئیات عملکرد هر بخش از کد این کلاس پرداخته می‌شود.

۱. متغیرها و سازنده کلاس

در ابتدای کلاس، متغیرهای مختلفی تعریف شده‌اند:

```
doc_id = 0
```

این متغیر به عنوان شمارنده اسناد استفاده می‌شود.

```
def __init__(self):
    self.file_name = dict()
    self.non_positional_index = defaultdict(set)
    self.positional_index = defaultdict(lambda: defaultdict(list))
    self.wildcard_index = defaultdict(set)
    self.state = [0, "Not Started Yet!", '']
```

سازنده کلاس `__init__`، چهار متغیر را مقداردهی می‌کند:

- `file_name`: یک دیکشنری برای نگهداری نام فایل‌ها.
- `non_positional_index`: یک دیکشنری از مجموعه‌ها برای شاخص‌های غیرمکانی.

- `positional_index`: یک دیکشنری تو در تو برای شاخص‌های مکانی.

- `wildcard_index`: یک دیکشنری از مجموعه‌ها برای شاخص‌های جایگشتی.

- `state`: یک لیست برای نگهداری وضعیت پردازش.

۲. روش‌های به‌روزرسانی و مدیریت اسناد

۲.۱. روش `next_doc_id`

```
@staticmethod
def next_doc_id:()
  Phase2.doc_id += 1
  return Phase2.doc_id
```

این روش یک شناسه جدید برای سند تولید می‌کند. هر بار که این روش فراخوانی می‌شود، شمارنده `doc_id` یک واحد افزایش می‌یابد و مقدار جدید برگردانده می‌شود.

۲.۲. روش `state_updater`

```
def state_updater(self, done : int, process_length : int, section : str, directory : str = '') :
  self.state = [round((done / process_length) * 100, 3), section, directory]
```

این روش وضعیت پردازش را به‌روزرسانی می‌کند. پارامترهای ورودی شامل تعداد آیتم‌های پردازش شده (`done`)، تعداد کل آیتم‌ها (`process_length`)، بخش فعلی پردازش (`section`) و دایرکتوری مربوطه (`directory`) می‌باشند. وضعیت جدید به صورت درصدی از پیشرفت پردازش محاسبه و در متغیر `state` ذخیره می‌شود.

۲.۳. روش `add_document_single`

```
def add_document_single(self, doc_id, text) :
```

```
    words = text.split()
    done = 0
    process_length = len(words)
    for pos, word in enumerate(words) :
        self.non_positional_index[word].add(doc_id)
        self.positional_index[word][doc_id].append(pos)
        self._add_to_wildcard_index(word, doc_id)
        self.state_updater(done, process_length, "Adding…")
        done += 1
    self.state_updater(done, process_length, "Done!")
```

این روش یک سند جدید به شاخص‌ها اضافه می‌کند. ابتدا متن سند به کلمات تقسیم می‌شود و سپس برای هر کلمه:

- آن کلمه به شاخص غیرمکانی اضافه می‌شود.
- موقعیت کلمه در سند به شاخص مکانی اضافه می‌شود.
- کلمه به شاخص جایگشتی اضافه می‌شود.

در هر مرحله، وضعیت پردازش به‌روزرسانی می‌شود.

۲.۴. روش `remove_document_single`

```
def remove_document_single(self, doc_id, text) :
    words = text.split()
    done = 0
    process_length = len(words)
    for pos, word in enumerate(words) :
```

```
        self._remove_from_index(word, doc_id, pos)
        self.state_updater(done, process_length, "Removing…")
        done += 1
    self.state_updater(done, process_length, "Done!")
```

این روش یک سند را از شاخص‌ها حذف می‌کند. ابتدا متن سند به کلمات تقسیم می‌شود و سپس برای هر کلمه، با استفاده از روش `_remove_from_index`، کلمه از شاخص‌های مختلف حذف می‌شود. در هر مرحله، وضعیت پردازش به‌روزرسانی می‌شود.

۲.۵. روش `add_document`

```
def add_document(self, doc_id, text, **kwargs):
    words = text.split()
    for pos, word in enumerate(words):
        if kwargs['non-positional']:
            self.non_positional_index[word].add(doc_id)
        if kwargs['positional']:
            self.positional_index[word][doc_id].append(pos)
        if kwargs['wildcard']:
            self._add_to_wildcard_index(word, doc_id)
```

این روش نیز برای اضافه کردن یک سند به شاخص‌ها استفاده می‌شود، اما این‌بار با استفاده از پارامترهای اختیاری (`kwargs`) که تعیین می‌کند آیا کلمه به شاخص غیرمکانی، مکانی و یا جایگشتی اضافه شود یا نه.

۲.۶. روش‌های داخلی برای مدیریت شاخص‌ها

- `remove_from_index_`: حذف یک کلمه از شاخص‌های مختلف.
- `add_to_wildcard_index_`: اضافه کردن یک کلمه به شاخص جایگشتی.
- `remove_from_wildcard_index_`: حذف یک کلمه از شاخص جایگشتی.

۳. روش‌های ذخیره و بارگذاری شاخص‌ها

۳.۱. روش `save_index`

```
def save_index(self):
  with open("index.file", "wb") as file:
    pickle.dump})
"       file_names": {str(k): v for k, v in self.file_name.items()},
"       non_positional_index": {str(k): list(v) for k, v in self.non_positional_index.items()},
"       positional_index": {str(k): {str(dk): dv for dk, dv in v.items()} for k, v in
                self.positional_index.items,{()
"       wildcard_index": {str(k): list(v) for k, v in self.wildcard_index.items()}
,{    file(
  try:
    with open('index.json', 'w', encoding='utf8') as file:
      json.dump)
}
"           file_names": {str(k): v for k, v in self.file_name.items()},
"           non_positional_index": {str(k): list(v) for k, v in self.non_positional_index.items()},
```

```
"            positional_index": {str(k): {str(dk): dv for dk, dv in v.items()} for k, v in
                    self.positional_index.items,{()
"            wildcard_index": {str(k): list(v) for k, v in self.wildcard_index.items()}
,{           file, ensure_ascii=False(
  except IOError as e:
    print(f'Error saving index to 'index.json': {e}")
```

این روش شاخص‌ها را در دو فایل ذخیره می‌کند: `index.file` با استفاده از کتابخانه `pickle` و `index.json` با استفاده از کتابخانه `json`. در هر دو حالت، شاخص‌ها به فرمت‌های قابل ذخیره تبدیل می‌شوند.

۳.۲. روش `load_index`

```
def load_index(self):
  try:
    with open("index.file", "rb") as file:
      data = pickle.load(file)
      self.file_name = {str(k): v for k, v in data["file_names"].items()}
      self.non_positional_index = defaultdict(set,
}                                 str(k): set(v) for k, v in
                                 data["non_positional_index"].items({()
      self.positional_index = defaultdict(lambda: defaultdict(list), {str(k): defaultdict(list, v) for k, v in
                                        data["positional_index"].items({()
      self.wildcard_index = defaultdict(set, {str(k): set(v) for k, v in data["wildcard_index"].items()})
  except IOError as e:
    print(f'Error loading index from 'index.json': {e}")
```

این روش شاخص‌ها را از فایل `index.file` بارگذاری می‌کند. در صورت بروز خطا، پیغام خطای مناسبی نمایش داده می‌شود.

۴. روش‌های فشرده‌سازی و اندازه‌گیری حافظه

۴.۱. روش `get_memory_size`

```
def get_memory_size(self, obj) :
  if isinstance(obj, (str, bytes, bytearray)) :
    return sys.getsizeof(obj)
  elif isinstance(obj, dict) :
    return sys.getsizeof(obj) + sum(self.get_memory_size(k) + self.get_memory_size(v) for k, v in
obj.items())
  elif isinstance(obj, (list, set, tuple)) :
    return sys.getsizeof(obj) + sum(self.get_memory_size(i) for i in obj)
  return sys.getsizeof(obj)
```

این روش به صورت بازگشتی اندازه حافظه اشیاء را در بایت محاسبه می‌کند. ابتدا نوع شیء بررسی می‌شود و بسته به نوع آن (رشته، دیکشنری، لیست، مجموعه، یا تاپل)، اندازه حافظه محاسبه و برگردانده می‌شود. برای دیکشنری‌ها، لیست‌ها، مجموعه‌ها و تاپل‌ها، این روش به صورت بازگشتی اندازه حافظه کل محتوا را نیز محاسبه می‌کند.

۴.۲. روش `variable_byte_encode`

```
def variable_byte_encode(self, numbers, length) :
  bytes_stream[] =
  for number in numbers :
    byte_segments[] =
    while True :
```

```
        byte_segments.insert(0, number % 128)  # Get 7 least significant bits
        if number < 128:
            break
        number //= 128
    byte_segments[-1] += 128  # Set the continuation bit
    bytes_stream.extend(byte_segments)
return bytes_stream
```

این روش لیستی از اعداد را با استفاده از روش کدگذاری متغیر بایتی (Variable Byte Encoding) فشرده می‌کند. هر عدد به صورت قطعه‌های بایتی با 7 بیت کم‌اهمیت‌ترین بیت‌ها و یک بیت ادامه‌دهنده کدگذاری می‌شود. این بیت ادامه‌دهنده مشخص می‌کند که آیا بایت بعدی نیز به عدد تعلق دارد یا خیر.

۴.۳. روش `gamma_encode`

```
def gamma_encode(self, number):
    if number == 0:
        return '0'
    binary = bin(number)[2:]
    offset = binary[1:] # Remove the leading 1
    length = len(offset)
    unary = '1' * length + '0'
    return unary + offset
```

این روش یک عدد را با استفاده از روش کدگذاری گاما (Gamma Encoding) فشرده می‌کند. این کدگذاری شامل یک بخش یونی (unary) که طول باینری عدد را نشان می‌دهد و یک بخش آفست (offset) که باینری عدد بدون بیت پیش‌رو است، می‌باشد.

۴.۴. روش `gamma_encode_list`

```
def gamma_encode_list(self, numbers) :
  return '' .join(self.gamma_encode(number) for number in numbers)
```

این روش لیستی از اعداد را با استفاده از روش کدگذاری گاما فشرده می‌کند. هر عدد به صورت جداگانه کدگذاری شده و سپس نتایج به هم پیوسته می‌شوند.

۴.۵. روش `compress_index`

```
def compress_index(self, method='variable_byte') :
  done = 0
  process_length = 3

  if method == 'variable_byte :'
    self.state_updater(done, process_length, "Compressing Non-Positional Index…")
    non_positional_index} =
      k : self.variable_byte_encode(list(v), len(self.non_positional_index)) for k, v
      in self.non_positional_index.items()
{
    done += 1
    self.state_updater(done, process_length, "Compressing Positional Index…")
    positional_index} =
      k : {dk : self.variable_byte_encode(dv, len(self.positional_index)) for dk, dv in v.items()} for k, v
      in self.positional_index.items()
```

```
{
    done += 1
    self.state_updater(done, process_length, "Compressing Wildcard Index…")
    wildcard_index} =
      k : self.variable_byte_encode(list(v), len(self.wildcard_index)) for k, v in
      self.wildcard_index.items()
{
    done += 1
    self.state_updater(done, process_length, "Done!")
    return}
"       non_positional_index" : non_positional_index,
"       positional_index" : positional_index,
"       wildcard_index" : wildcard_index
{
  elif method == 'gamma :'
    return}
"       non_positional_index" : {k : self.gamma_encode_list(list(v)) for k, v in
                    self.non_positional_index.items,{()
"       positional_index" : {k : {dk : self.gamma_encode_list(dv) for dk, dv in v.items()} for k, v in
                  self.positional_index.items,{()
"       wildcard_index" : {k : self.gamma_encode_list(list(v)) for k, v in self.wildcard_index.items()}
}
```

این روش شاخص‌ها را با استفاده از روش‌های کدگذاری متغیر بایتی یا گاما فشرده می‌کند. ابتدا وضعیت پردازش به‌روزرسانی شده و سپس هر شاخص به صورت جداگانه فشرده می‌شود. در نهایت، وضعیت پردازش به حالت "انجام شده" به‌روزرسانی می‌شود و شاخص‌های فشرده شده برگردانده می‌شوند.

نتیجه‌گیری برای فاز دوم را میتوان به این صورت گفت که :

کلاس `Phase2` یک ابزار قدرتمند برای مدیریت شاخص‌های متنی است. این کلاس با استفاده از روش‌های مختلف، از جمله شاخص‌های غیرمکانی، مکانی و جایگشتی، به بهبود کارایی جستجوها کمک می‌کند. علاوه بر این، روش‌های فشرده‌سازی مختلفی ارائه می‌دهد که می‌توانند به کاهش حجم داده‌های ذخیره شده کمک کنند. با به‌روزرسانی وضعیت پردازش، کاربران می‌توانند به‌طور مداوم پیشرفت فرآیند شاخص‌گذاری را مشاهده کنند. این ویژگی‌ها موجب می‌شوند که کلاس `Phase2` برای کاربردهای مختلفی در زمینه پردازش متون و جستجو مفید باشد.

فصل چهارم: پیاده سازی فاز سوم پروژه

در این فاز به بررسی پیاده‌سازی یک سیستم بازیابی اطلاعات پرداخته می‌شود. این سیستم از روش‌های مختلفی مانند جستجوی رتبه‌بندی‌شده و جستجوی دقیق عبارات استفاده می‌کند تا به کاربران در یافتن اسناد مرتبط با پرسش‌هایشان کمک کند. همچنین، روش‌های ارزیابی مختلفی برای اندازه‌گیری دقت و کارایی این سیستم ارائه شده‌اند. این گزارش شامل توضیحات مفصل درباره‌ی هر بخش از کد و عملکرد آن است.

بخش اول: محاسبه TF-IDF

توضیحات کلی:

TF-IDF (Term Frequency-Inverse Document Frequency) یک روش متداول برای ارزیابی اهمیت کلمات در یک سند نسبت به یک مجموعه اسناد است. این روش شامل دو بخش اصلی است:

1. **TF (Term Frequency)**: تعداد دفعات تکرار یک کلمه در یک سند.
2. **IDF (Inverse Document Frequency)**: میزان اهمیت یک کلمه در مجموعه اسناد.

پیاده‌سازی:

کد مربوط به محاسبه TF-IDF به صورت زیر پیاده‌سازی شده است:

```
def compute_tf(term, document) :
  return document.count(term) / len(document)

def compute_idf(term, documents) :
  num_docs_containing_term = sum(1 for doc in documents if term in doc)
  if num_docs_containing_term > 0 :
    return math.log(len(documents) / num_docs_containing_term)
  else :
```

```
    return 0


def compute_tf_idf(term, document, documents) :
    return compute_tf(term, document) * compute_idf(term, documents)
```

بخش دوم: کلاس 'RankedRetrieval'

توضیحات کلی:

این کلاس مسئولیت جستجوی رتبه‌بندی‌شده اسناد بر اساس پرسش کاربران را دارد. در این روش، هر سند بر اساس محتوای آن و پرسش کاربر امتیازدهی می‌شود و سپس اسناد بر اساس این امتیازات رتبه‌بندی می‌شوند.

پیاده‌سازی

این کلاس به صورت زیر پیاده‌سازی شده است:

```python
class RankedRetrieval:
    matching_terms = []

    def __init__(self, phase3_instance):
        self.phase3 = phase3_instance

    1 usage
    def rank_documents(self, query):
        RankedRetrieval.matching_terms = []
        query_terms = query.split()
        doc_scores = defaultdict(float)
```

```
    for term in query_terms:
        if '*' in term:  # Handle wildcard term
            matching_terms = self.expand_wildcard(term)
            RankedRetrieval.matching_terms.extend(matching_terms)
            for m_term in matching_terms:
                self.update_doc_scores( is_wildcard: True, m_term, doc_scores)
        else:
            self.update_doc_scores( is_wildcard: False, term, doc_scores)

    ranked_docs = sorted(doc_scores.items(), key=lambda item: item[1], reverse=True)
    return ranked_docs

1 usage
def expand_wildcard(self, term):
    wildcards = self.phase3.wildcard_index.keys()

    pre, post = term.split("*")
    terms = []
    if pre and post:
        for text in wildcards:
            if text.startswith(pre) and text.endswith(post):
                terms.append(text)

    elif pre:
        for text in wildcards:
            if text.startswith(pre):
                terms.append(text)

    else:
        for text in wildcards:
            if text.endswith(post):
                terms.append(text)

    return terms
    # return self.phase3.wildcard_index[term.replace("*", "")]
```

```
2 usages
def update_doc_scores(self, is_wildcard, term, doc_scores):
    search_from = self.phase3.wildcard_index if is_wildcard else self.phase3.non_positional_index

    for doc_id in search_from[term]:
        # print(doc_id, self.phase3.positional_index[term][str(doc_id)])
        # doc_scores[str(doc_id)] += compute_tf_idf(term, self.phase3.positional_index[term][str(doc_id)],

        try:
            doc_scores[str(doc_id)] += compute_tf_idf(term, self.get_doc_content(str(doc_id)),
                                                      self.phase3.positional_index)
        except ZeroDivisionError:
            pass

1 usage
def get_doc_content(self, doc_id):
    doc_content = []
    for term, postings in self.phase3.positional_index.items():
        if str(doc_id) in postings.keys() or int(doc_id) in postings.keys():
            doc_content.extend([term] * len(postings[str(doc_id)]))
    return doc_content
```

بخش سوم: کلاس `PhraseSearch`

توضیحات کلی:

این کلاس مسئولیت جستجوی دقیق عبارات را دارد. در این روش، پرسش کاربر به صورت یک یا چند عبارت مشخص بررسی می‌شود و اسناد دقیقی که این عبارات را شامل می‌شوند بازیابی می‌شوند.

پیاده‌سازی:

این کلاس به صورت زیر پیاده‌سازی شده است:

```python
class PhraseSearch:
    def __init__(self, phase3_instance):
        self.phase3 = phase3_instance

    1 usage
    def match_phrases(self, query):
        phrases = self.extract_phrases(query)
        matching_docs = self.find_matching_docs(phrases)
        ranked_docs = self.rank_phrase_documents(query, matching_docs)
        return ranked_docs

    1 usage
    def extract_phrases(self, query):
        import re
        phrases = re.findall( pattern: r'"([^"]*)"', query)
        return phrases

    1 usage
    def find_matching_docs(self, phrases):
        doc_sets = [set(self.phase3.non_positional_index[term]) for phrase in phrases for term in phrase.split()]
        if not doc_sets:
            return set()
        matching_docs = set.intersection(*doc_sets)
        return matching_docs

    1 usage
    def rank_phrase_documents(self, query, matching_docs):
        query_terms = query.replace('"', '').split()
        doc_scores = defaultdict(float)
```

```
    for doc_id in matching_docs:
        doc_content = self.get_doc_content(str(doc_id))
        for term in query_terms:
            doc_scores[str(doc_id)] += compute_tf_idf(term, str(doc_content), self.phase3.positional_index)

    ranked_docs = sorted(doc_scores.items(), key=lambda item: item[1], reverse=True)
    return ranked_docs

1 usage
def get_doc_content(self, doc_id):
    doc_content = []
    for term, postings in self.phase3.positional_index.items():
        if str(doc_id) in postings.keys() or int(doc_id) in postings.keys():
            doc_content.extend([term] * len(postings[str(doc_id)]))
    return doc_content
```

بخش چهارم: کلاس `Phase3`

توضیحات کلی

این کلاس اصلی سیستم بازیابی اطلاعات است که شامل شاخص‌های غیرمکانی، مکانی و جایگشتی می‌شود. همچنین، این کلاس شامل روش‌هایی برای به‌روزرسانی و ذخیره‌سازی شاخص‌ها است.

پیاده‌سازی این این کلاس را میتوانید در کد مشاهده کنید.

به عنوان نتیجه‌گیری این فصل میتوان گفت که :

کد پیاده‌سازی شده در این پروژه یک سیستم بازیابی اطلاعات کارآمد را فراهم می‌کند که از روش‌های مختلف جستجو مانند جستجوی رتبه‌بندی‌شده و جستجوی دقیق عبارات بهره می‌برد. همچنین، این سیستم شامل مکانیزم‌هایی برای مدیریت شاخص‌ها و به‌روزرسانی آن‌ها است که می‌تواند به راحتی با مجموعه داده‌های بزرگ مقیاس‌پذیر باشد. بهینه‌سازی‌های مختلفی مانند استفاده از TF-IDF برای امتیازدهی اسناد و استفاده از شاخص‌های جایگشتی برای جستجوی عبارات نیز به دقت و کارایی این سیستم کمک می‌کند

منابع

1. https://www.roshan-ai.ir/hazm/

2. https://github.com/roshan-research/hazm

3. https://github.com/ziaa/Persian-stopwords-collection

4. https://github.com/kharazi/persian-stopwords

5. https://www.djangoproject.com/

6. https://developer.mozilla.org/en-US/docs/Web/JavaScript