



دانشکده مهندسی کامپیوتر

بازیابی پیشرفته اطلاعات

مدرس: دکتر بیگی

شماره گروه: ۵

تهیه کنندگان: نیما جمالی - سپهر فعلی - سینا کاظمی

گزارش فاز اول پروژه

فهرست مطالب

۳	پیش پردازش اولیه
۳	پیش پردازش مستندات انگلیسی
۴	کلمات پرتکرار مستندات انگلیسی
۵	پیش پردازش مستندات فارسی
۵	کلمات پرتکرار مستندات فارسی
۷	نمایه سازی
۷	نمایه bigram
۷	نمایه positional
۸	توابع درج و حذف مستندات
۱۰	توابع ذخیره سازی و لود نمایه ها
۱۱	فشرده سازی نمایه ها
۱۱	فشرده سازی با variable byte
۱۲	فشرده سازی با gamma code
۱۳	ذخیره سازی و لود نمایه ها
۱۴	اصلاح پرسمان
۱۴	استفاده از نمایه bigram و معیار جا کارد برای یافتن کلمات مشابه
۱۵	تابع محاسبه ی فاصله ویرایش دو کلمه
۱۵	نمایش پرسمان اصلاح شده
۱۶	جستجو و بازیابی اسناد
۱۶	جستجوی ltc-lnc پرسمان
۱۷	جستجوی proximity با اندازه ی پنجره ی داده شده در ورودی

۱۹	نحوه‌ی تقسیم وظایف
۱۹	نیم‌جمالی
۱۹	سپهر فعلی
۱۹	سینا کاظمی
۲۰	مراجع

پیش پردازش اولیه

در این قسمت در کلاس IRSystem تابعی به نام prepare_text ایجاد کردیم که مستندات، زبان مد نظر و ایست‌واژه‌ها را ورودی می‌گیرد. این تابع هنگامی که کاربر در ورودی دستور prepare [lang] را فراخوانی می‌کند، صدا زده می‌شود. همچنین برای بررسی کوثری نیز از همین تابع استفاده می‌شود. در واقع اگر آرایه ایست‌واژه خالی باشد به معنای بررسی اولیه مستندات است و در صورتی که خالی نباشد، بدین معناست که یک کوثری در حال پردازش است. در هنگام ایجاد یک نمونه از کلاس، دو تابع initialize_persian و initialize_english صدا زده می‌شوند و متون موجود در فایل‌های csv و xml داده شده در یک مستند به فرمت زیر ذخیره می‌شوند:

```
document = [[title], [description]]
```

و برای هر زبان یک مجموعه مستند (collections) به عنوان ورودی تابع prepare_text پاس داده می‌شود. تابع prepare_text بسته به زبان مورد نظر، یک سری اعمال نظیر نرمال‌سازی متنی، جداسازی، حذف حروف اضافه، یافتن ایست‌واژه‌ها و بازگرداندن کلمات به ریشه را انجام می‌دهد و در نهایت لیستی از تمامی توکن‌های باقی‌مانده، تمامی مستندات ساختار یافته، ترم‌های باقی‌مانده و ایست‌واژه‌ها را باز می‌گرداند. مستندات ساختار یافته (structured documents) آرایه‌ای از مستندات به صورت زیر است.

```
document = [[array of title tokens], [array of description tokens]]
```

در ادامه به طور مجزا به بررسی اعمال انجام گرفته برای پیش‌پردازش داده‌های هر زبان می‌پردازیم.

پیش‌پردازش مستندات انگلیسی

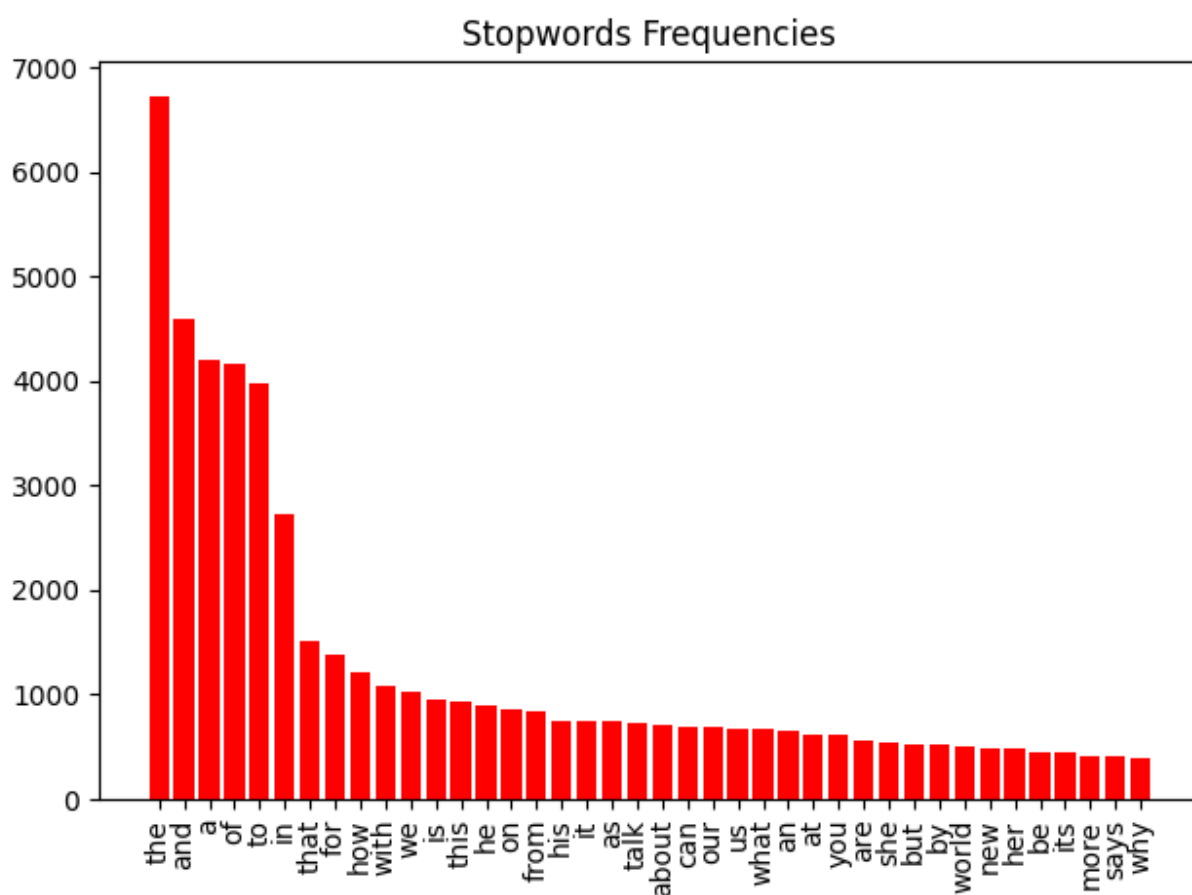
تابع prepare_english برای پیاده‌سازی این قسمت طراحی شده است. ابتدا هر قسمت از مستند را به صورت جداگانه و با استفاده از تابع word_tokenize از کتابخانه NLTK، به صورت مجموعه‌ای از توکن‌ها در می‌آوریم و تمام کلمات را نیز به حالت حروف کوچک تبدیل می‌کنیم. سپس علائمی نظیر ‘-’، ‘,’ و ‘?’ را از کلماتی که در آنها موجودند، حذف می‌کنیم. پس از آن نیز حروف اضافه را از توکن‌ها حذف می‌کنیم. پس از یافتن ایست‌واژه‌ها (اینجا)، با استفاده از SnowballStemmer کلمات را به ریشه بر می‌گردانیم و در نهایت لیستی از تمامی توکن‌ها، مستندات ساختار یافته، تمامی ترم‌های باقی‌مانده و ایست‌واژه‌های زبان انگلیسی را باز می‌گردانیم.

کلمات پرتکرار مستندات انگلیسی

در این قسمت از تابع `process_stop_words` استفاده کردیم که سائز مورد نظر برای ایست‌واژه‌ها و آرایه‌ای از تمامی توکن‌ها ورودی می‌گیرد و با استفاده از توابع `Counter`، توکن‌ها را بر اساس تکرار آنها و به صورت نزولی مرتب می‌کند. در نهایت هم ترم‌های مجموعه مستندات و ایست‌واژه‌ها را باز می‌گرداند. در بخش انگلیسی با امتحان کردن مقادیر مختلف به عنوان سائز مجموعه‌ی ایست‌واژه، به عدد ۴۰ رسیدیم که تعداد توکن‌ها را از ۱۴۴۸۲۸ به ۹۳۶۹۴ عدد کاهش می‌داد. این آرایه به ترتیب فراوانی متشکل از کلمات زیر بود:

['the', 'and', 'a', 'of', 'to', 'in', 'that', 'for', 'how', 'with', 'we', 'is', 'this', 'he', 'on', 'from', 'his', 'it', 'as', 'talk', 'about', 'can', 'our', 'us', 'what', 'an', 'at', 'you', 'are', 'she', 'but', 'by', 'world', 'new', 'her', 'be', 'its', 'more', 'says', 'why']

نمودار فراوانی این ایست‌واژه‌ها نیز در تصویر زیر قابل مشاهده‌است.



تصویر ۱: نمودار فراوانی ایست‌واژه‌های مستندات انگلیسی

پیش پردازش مستندات فارسی

تابع `prepare_persian` برای پیاده سازی این قسمت طراحی شده است، ابتدا یک لیست از حروف اضافه و علائم نگارشی و عملگرها و اعداد را انتخاب می کنیم و هر قسمت از مستند را به طور جداگانه پردازش می کنیم، بدین صورت که در هر بخش به جای اعضای لیست یک " " (کاراکتر `space`) قرار می دهیم. همچنین هر رشته ای از کلمات را که بین دو آکلاد ({ }) وجود دارد را نیز حذف می کنیم. با استفاده از یک رجکس نیز تمامی اعداد از مستندات حذف می شوند.

سپس با استفاده از کتابخانه هضم (`hazm`) کلمات را در هر بخش `normalize` و `tokenize` می کنیم ، پس از آن در هر کلمه ای که نیم فاصله در آن وجود داشت، نیم فاصله (و هر کاراکتر اضافی دیگر) را حذف می کنیم و دو قسمت کلمه را به هم می چسبانیم:

می شود ← میشود

در نهایت با استفاده از `stemmer` که در کتابخانه هضم موجود بود کلماتی که انتهای آنها " ان - ات - ترین - تر - یی - ها - ا و ... " بودند را کوتاه می کنیم:

کتابهایشان -> کتاب

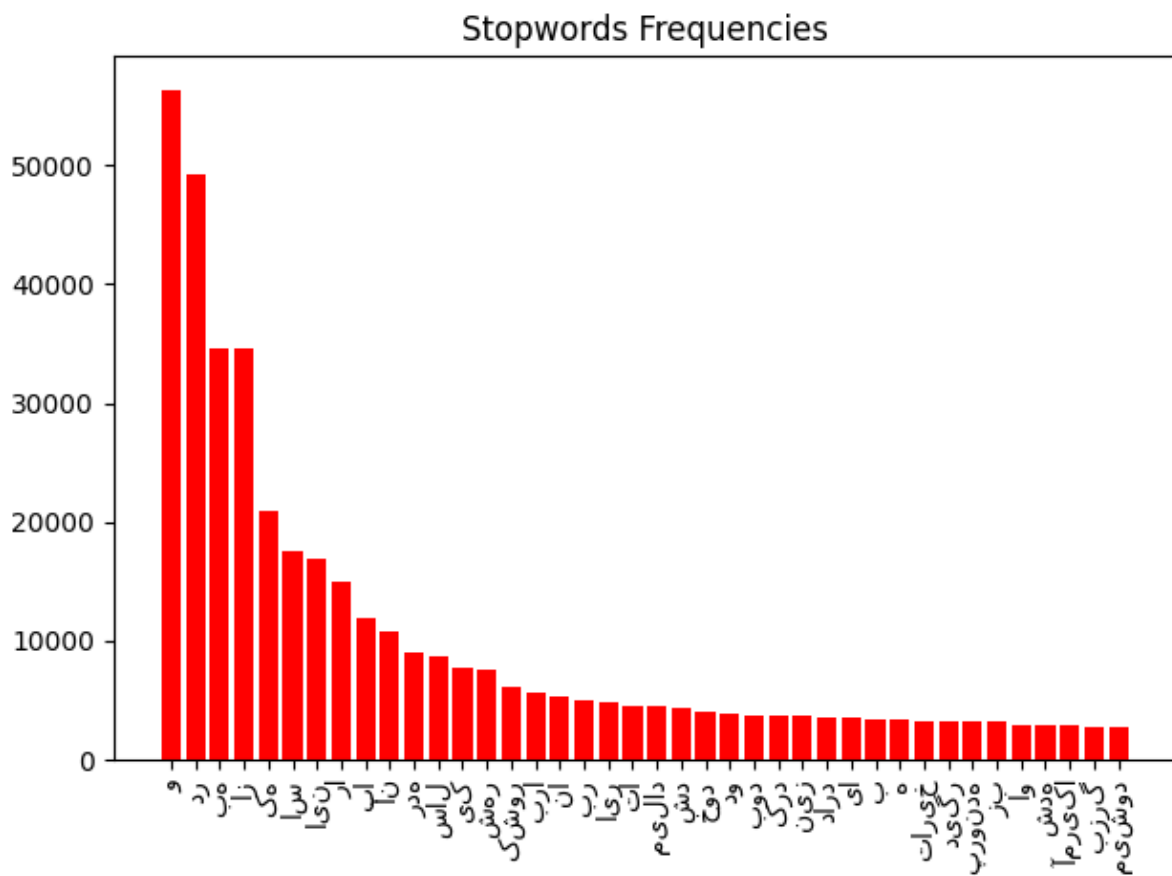
در نهایت هم پس از پیدا کردن ایست واژه ها، خروجی شامل تمامی توکن ها، مستندات ساختار یافته، لیستی از ترم ها و ایست واژه ها باز گردانده می شود.

کلمات پرتکرار مستندات فارسی

برای این بخش نیز از تابعی که برای پیدا کردن ایست واژه ها در متون انگلیسی استفاده شد، بهره گرفتیم و با امتحان کردن مقادیر مختلف، به عدد ۴۰ برای اندازه ایست واژه ها رسیدیم. این عمل باعث شد تا اندازه ای مجموعه توکن ها از ۱۳۱۲۰۸۳ به ۹۱۱۴۷۷ کاهش پیدا کند. این آرایه به ترتیب فراوانی از کلمات زیر تشکیل شده بود:

['او', 'در', 'به', 'از', 'که', 'اس', 'این', 'را', 'با', 'آن', 'رده', 'سال', 'یک', 'شهر', 'کشور', 'بر', 'تا', 'بر', 'ایر', 'تا', 'میلاد', 'شد', 'خود', 'دو', 'بود', 'کرد', 'نیز', 'دارد', 'یا', 'ب', 'ه', 'تاریخ', 'دیگر', 'پرونده', 'زب', 'او', 'شده', 'آمریکا', 'بزرگ', 'میشود']

نمودار فراوانی این واژه‌ها نیز در تصویر زیر قابل رویت است.



تصویر ۲: نمودار فراوانی ایست‌واژه‌های مستندات انگلیسی

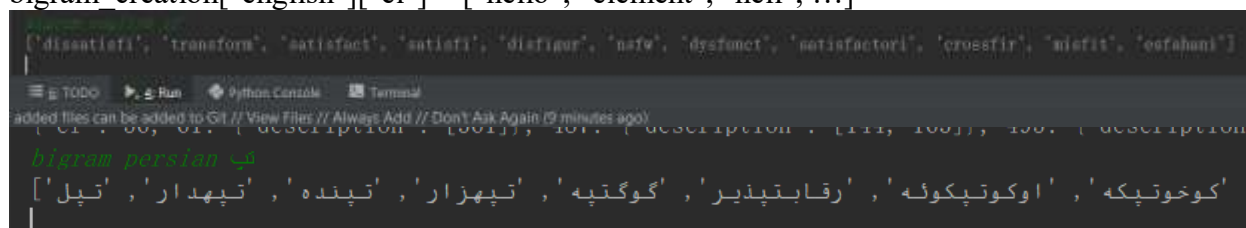
نمایه سازی

در این بخش نمایه های bigram و positional به کمک ساختمان داده ی دیکشنری طراحی می شوند و سپس توابعی برای ذخیره سازی و بارگذاری این نمایه پیاده سازی می شوند که در ادامه به شرح بیشتر آنها می پردازیم.

نمایه bigram

با وارد کردن دستور `create bigram [lang]` تابع `call_create_bigram` فراخوانی می شود که آن نیز تابع `bigram` را صدا می زند که برای استخراج این نوع نمایه پیاده سازی شده است. بدین صورت که با دریافت مجموعه مستندات پیش پردازش شده که با عنوان `structured_documents` ذخیره شده اند و برای هر دو زبان انگلیسی و فارسی در دسترس اند، `sub_term` هایی به طول ۲ به ازای هر ترم ایجاد می کند و این `sub_term` ها به نوعی به ترم مربوط به خود اشاره می کنند. نتیجه این تابع، در واقع دیکشنری `bigram_creation` است که نحوه پیکربندی و یک مثال از آن در ادامه ذکر شده است:

```
bigram_creation[lang][sub_term] = [terms which contain sub_term]
bigram_creation["english"]["el"] = ["hello", "element", "hell", ...]
```



تصویر ۳: نمونه ای از نمایه bigram در مستندات انگلیسی و فارسی

نمایه positional

با وارد کردن دستور `create positional [lang]` تابع `call_create_positional` فراخوانی می شود که آن نیز تابع `positional` را صدا می زند که برای استخراج این نوع نمایه پیاده سازی شده است. بدین صورت که با دریافت مجموعه ی `structured_documents` که در بخش bigram نیز توضیح داده شد، ترم های موجود را بررسی می کند و تغییرات مورد نیاز را در متغیر `positional_index_creation` که یک دیکشنری است و برای ذخیره سازی این نوع نمایه در نظر گرفته شده لحاظ می کند.

به ازای هر ترم، مجموعه مستنداتی که ترم مذکور در آنها آمده است، ذخیره شده و به ازای هریک از این مستندات نیز، ابتدا این ویژگی ذخیره شده که ترم مذکور، در این مستند، در ستون `title` آمده یا `description` وبعد، چندمین

term از این ستون است و در چه مکانی آمده است. همچنین collection frequency ترم مورد نظر نیز در این نمایه ذخیره می شود.

نحوه پیکربندی در ادامه آمده است:

```
positional_index_creation[lang][term][docId][column] = [places where term occurs]
```

```
positional english soccer
{'cf': 3, 2263: {'title': [3], 'description': [10]}, 2316: {'description': [27]}}
```

6: TODO 4: Run Python Console Terminal

Positional index Saved Successfully with Gamma Method

```
positional persian گراف
{'cf': 56, 61: {'description': [361]}, 487: {'description': [144, 163]}, 495: {'description': [144, 163]}}
```

kinam persian گرامر

تصویر ۴: نمونه‌ای از نمایه positional در مستندات انگلیسی و فارسی

توابع درج و حذف مستندات

توابع این قسمت شامل توابع `insert`، `csv_insert`، `xml_insert` و `delete` هستند. در مجموعه مستندات فرض کردیم شناسه‌ی هر مستند برابر با شماره‌ی آن در مجموعه مستندات است و هر شماره‌ی دیگری که درج می‌شود به این اعداد اضافه می‌شود. به این منظور برای هر زبان عددی به نام `docs_size` در نظر گرفته شده‌است. تابع `insert` مجموعه مستنداتی را ورودی می‌گیرد و با توجه به زبان ورودی، نمایه‌های `bigram` و `positional` مربوطه را نیز ورودی می‌گیرد و پس از اعمال پیش‌پردازش، تمامی توکن‌های این مستندات را به توکن‌های کلی اضافه می‌کند، مستندات را به مجموعه‌ی قبلی اضافه می‌کند و ترم‌های جدید را به دیکشنری اضافه می‌کند و سپس نمایه‌های `positional` و `bigram` را با فراخوانی توابع مربوطه به‌روز رسانی می‌کند. کاربر می‌تواند با دستور `insert [lang] [docs_number] [parts]` به تعداد `docs_number` داکيومنت را به صورت دستی در مجموعه مستندات درج کند.

توابع `csv_insert` و `xml_insert` نیز زبان ورودی و `path` مورد نظر را دریافت می کنند و در صورت صحت `path`، پردازش های مربوط به `csv` یا `xml` را انجام می دهند. در نهایت هم تابع `insert` بر روی مستندات فایل ورودی اعمال می شود.

تابع delete اما یک شماره‌ی مستند را ورودی می‌گیرد و با توجه به زبان ورودی، اگر مستند مورد نظر در مجموعه موجود باشد، نمایه‌های bigram و positional را نیز به‌روز رسانی می‌کند. در واقع برای مجموعه مستندات بردار deleted_documents از True یا False داریم که هنگام پاک شدن یک مستند، اندیس مربوط به آن در این بردار True می‌شود. به‌روز رسانی نمایه‌های مذکور نیز به این صورت است که به ازای هر توکن در مستند پاک شده، در نمایه positional مربوط به این ترم دیکشنری، cf یک واحد کم می‌شود و شناسه‌ی مستند هم از posting list آن

کلمه پاک می شود. اگر cf برابر صفر شود، این ترم از نمایه positional حذف می گردد و در نمایه bigram نیز این ترم از posting list همه ی دوحرفی هایی که این ترم در آنها موجود بوده، پاک می شود. دستوری که کاربر در کنسول وارد می کند به فرمت زیر است:

delete [lang] [doc_id]

در ادامه مثال هایی از عملکرد توابع insert و delete مشاهده می کنیم. این مثال ها مربوط به دو اجرای مجزا هستند.

```
corppoo: korppoo> insert
{"cf": 148, "id": 1, "description": [5]}, 54: {"description": [15]}, 186: {"title": [2], "description": [9]}, 188: {"title": [1], "description": [11]}, 187: {"descrip
corppoo: korppoo> insert
Here (nima) doesn't match any term in english documents.
corppoo: korppoo> insert
{"cf": 142, "id": 58, {"title": [3], "description": [5]}, 54: {"description": [15]}, 186: {"title": [2], "description": [9]}, 188: {"title": [1], "description": [11]}, 187: {"descrip
corppoo: korppoo> insert
{"cf": 1, "id": 2558: {"description": [1]}}
corppoo: korppoo> insert
2530: {"title": [1], "description": [16, 46]}, 2542: {"description": [19]}, 2549: {"title": [72], "description": [5, 18, 29, 43, 9455]}
corppoo: korppoo> insert
2550: {"title": [3], "description": [16, 46]}, 2542: {"description": [19]}, 2549: {"title": [7], "description": [5, 18, 29, 43, 16]}, 2558: {"title": [1], "description": [3]}
```

تصویر ۵: مثالی از عملکرد تابع insert

در مثال بالا یک مستند با موضوع city و بدنه ی Nima loves this city! درج شده است. قبل از اعمال این درج ترم citi که stem شده ی کلمه ی city است، ۱۴۰ بار در مستندات استفاده شده و ترمی به نام nima نیز وجود نداشته است. اما پس از درج این مستند city ۱۴۲ بار در مجموعه مستندات حضور دارد و ترم nima نیز به نمایه ها اضافه شده است.

```
corppoo: korppoo> insert
{"cf": 1, "id": 2549: {"description": [11]}}
corppoo: korppoo> insert
{"cf": 148, "id": 58, {"title": [1], "description": [5]}, 54: {"description": [15]}, 186: {"title": [2], "description": [9]}, 188: {"title": [2], "description": [11]}, 187: {"descrip
corppoo: korppoo> insert
Korppoo is not a keyword!
corppoo: korppoo> insert
["happi", "happier", "applaud", "approach", "support", "disappear", "appli", "happen", "opportun", "collapsingcu", "hippo", "appear", "philip", "appreci", "appli", "upper", "E
corppoo: korppoo> insert
term (korppoo) doesn't match any term in english documents.
corppoo: korppoo> insert
{"cf": 136, "id": 1, {"title": [1], "description": [4]}, 54: {"description": [15]}, 186: {"title": [2], "description": [9]}, 188: {"title": [2], "description": [11]}, 187: {"descrip
corppoo: korppoo> insert
["happi", "happier", "applaud", "approach", "support", "disappear", "appli", "happen", "opportun", "collapsingcu", "hippo", "appear", "philip", "appreci", "appli", "upper", "E
```

تصویر ۶: مثالی از عملکرد تابع delete

در مثال بالا نیز مستند آخر یعنی ۲۵۵۰ پاک شده است. ترم korppoo تنها در این مستند موجود بوده که بعد از پاک شدن چون دیگر در مستندی وجود ندارد، از نمایه های bigram و positional پاک شده است. همچنین ترم citi ۶ بار در این مستند به کار رفته که با حذف آن، cf این ترم ۶ واحد کاهش پیدا کرده است.

توابع ذخیره‌سازی و لود نمایه‌ها

این بخش برای ذخیره‌سازی و بارگذاری نمایه‌های ساخته شده طراحی شده است. هنگامی که دستورات صفحه‌ی بعد در کنسول نوشته شود باعث می‌شود ذخیره‌سازی یا بارگذاری اطلاعات از/به سامانه‌ی بازیابی اطلاعات صورت پذیرد. همچنین با استفاده از توابع این قسمت می‌توان ایست‌واژه‌ها و مستندات ساختار یافته را نیز ذخیره و بارگذاری کرد تا هر بار نیاز به استفاده از دستور prepare نباشد:

```
save [ positional - bigram - stop_words - structured_documents ] [lang]
load [ positional - bigram - stop_words - structured_documents ] [lang]
```

در حالت ذخیره‌سازی پس از این که زبان و نوع نمایه یا ساختار دیگری که می‌خواهیم ذخیره کنیم بررسی شد، تابع `call_save_index` فراخوانی می‌شود. در این تابع با توجه به زبان و نوع چیزی که می‌خواهیم ذخیره کنیم، یک فایل با پیشوند `type_of_indexing + lang` با حالت نوشتن باز می‌کنیم - برای `serialize` کردن و نوشتن درون یک فایل از کتابخانه‌ی `pickle` استفاده می‌کنیم - و دیکشنری مورد نظر را در فایل باز شده `write` می‌کنیم. سپس فایل را می‌بندیم تا در فضای حافظه صرفه‌جویی کنیم.

در آخر هم عبارتی مبنی بر موفقیت آمیز بودن ذخیره‌سازی در کنسول چاپ می‌کنیم. برای بارگذاری نمایه‌ها در سامانه‌ی بازیابی اطلاعات هم مانند ذخیره‌سازی نوع نمایه یا ساختار دیگری که می‌خواهیم بارگذاری کنیم و زبان را مشخص می‌کنیم، سپس با فراخوانی تابع `call_load_index` بارگذاری نمایه‌ها در دیکشنری مورد نظر صورت می‌گیرد.

فشرده‌سازی نمایه‌ها

این بخش برای فشرده‌سازی نمایه‌های مکانی (positional) در نظر گرفته شده‌است. هدف از این بخش کاهش حجم نمایه‌هاست و با این عمل سرباری محاسباتی را برای فشرده‌سازی و خارج کردن از حالت فشرده پذیرفته‌ایم. به این منظور دو روش variable byte و gamma code را طراحی و پیاده‌سازی می‌کنیم و در نهایت پس از فشرده‌سازی نمایه‌ی مکانی به هر دو روش، حجم فایل ذخیره شده را با ذخیره‌سازی عادی مقایسه می‌کنیم. در ادامه این قسمت‌ها را بیشتر توضیح می‌دهیم.

فشرده‌سازی با variable byte

با استفاده از دستور `compress variable_byte [lang]` تابع `call_compress_variable_byte` فراخوانی می‌شود و آن نیز، تابع `positional_index_to_variable_byte` را صدا می‌زند. همان‌طور که از نام این تابع پیداست، ما برای فشرده‌سازی نمایه‌ها با variable byte، از `positional_index` بهره می‌بریم و گپ جایگاه‌هایی که برای هر ترم درون مستندات ذخیره کرده‌ایم را به شکل variable byte ذخیره می‌کنیم.

دیکشنری `vb_positional_index` برای ذخیره‌سازی این مقادیر در نظر گرفته شده‌است:

```
vb_positional_index[lang][term][docId] = [gaps in variable byte form]
```

به ازای هر ترم در مستند و ستونی مشخص، فاصله دو جایگاه متوالی را بدست می‌آوریم و برای تبدیل آن به فرم variable byte، آن را به تابع `create_variable_byte` پاس می‌دهیم. این تابع، فاصله دو جایگاه و ستونی که در آن رخ داده‌اند را به عنوان ورودی می‌گیرد و مقدار مناسب به فرم variable byte را خروجی می‌دهد. اما ساختار این خروجی چیست؟

خروجی، حداقل یک بایت است. به ازای هر بایت، بیت سمت چپ مشخص می‌کند که آخرین بایت است (۱) یا خیر (۰) و بیت سمت راست نیز ستون رخداد را مشخص می‌کند که description است (۱) یا title (۰). هنگام بازیابی `positional index` از variable byte نیز از تابع `variable_byte_to_positional_index` که به این منظور پیاده‌سازی شده و در آن برای دیکود کردن variable byte ها، از تابع `decode_variable_byte` استفاده می‌گردد، بهره می‌گیریم.

فشرده‌سازی با gamma code

با استفاده از دستور زیر تابع `call_compress_gamma_code` فراخوانی می‌شود.

```
compress gamma_code [lang]
```

عملیات فشرده‌سازی بدین شکل انجام می‌شود که ابتدا گپ‌های مکان کلمات در هر مستند را معین می‌کنیم و کد گاما را برای هر یک از اعداد به‌دست می‌آوریم علاوه بر آن در انتهای کد گاما یک بیت برای معین کردن zone کلمه (عنوان مستند یا شرح مستند) در نظر گرفته می‌شود.

مثال:

```
positional english → 434 (doc id) = {title: [4, 10], description: [23, 54, 87]}
```

```
after compression → 434 (doc id) = {title: ["110000110100"], description: ["111100111111101111111110000011"]}
```

چون در ذخیره‌سازی با پایتون بهترین data type قابل استفاده byte بود مجبور شدیم رشته‌های بالا را در آرایه‌ای از بایت‌ها ذخیره‌سازی کنیم و هنگام خواندن از این آرایه به عدد تبدیل کنیم و سپس آن را به رشته تبدیل کرده، با پردازش رشته نمایه‌ی اصلی را بسازیم.

بیت آخر را برای شناسایی zone در نظر گرفتیم که اگر ۰ باشد یعنی کلمه در title بوده‌است و اگر ۱ باشد بدین معنی است که کلمه در بخش description ظاهر شده‌است.

برای خارج کردن از حالت فشرده باید کد گامای مربوطه را شناسایی کنیم و zone را به درستی تشخیص دهیم. برای این کار از تابع `gamma_code_to_positional_index` استفاده کردیم که خود از تابع `decode_gamma_code` استفاده می‌کند که یک گاما کد ورودی می‌گیرد و عدد مربوط به آن و zone آن عدد را باز می‌گرداند. سپس این گپ را با آخرین مقدار position قبلی جمع می‌کند و نمایه‌ی positional بازسازی می‌شود.

با استفاده از دستور زیر در کنسول نمایه‌های مکانی را از حالت فشرده خارج کرده و در دیکشنری متناظر در زبان مقصد قرار می‌دهیم:

```
decompress gamma_code [lang]
```

نکته: یک بیت با مقدار ۱ هم اول هر zone (در صورت وجود) اضافه گردیده‌است که اگر کلمه‌ای در جایگاه نخست آن zone بود، بیت‌های ۰ اول `gamma_code` را از دست ندهیم (با توجه به استفاده از بایت).

ذخیره‌سازی و لود نمایه‌ها

برای ذخیره‌سازی هر دو نوع variable byte و gamma code از کتابخانه‌ی pickle بهره بردیم.







برای حالت variable byte:

- پس از وارد کردن دستور compress variable_byte [lang] و انجام فرآیند فشرده‌سازی، نتیجه کار در فایل‌ی ذخیره می‌گردد. به صورت خودکار اگر زبان انگلیسی بود، در فایل variable_byte_english و اگر زبان فارسی بود، در فایل variable_byte_persian.

- پس از وارد کردن کوئری decompress variable_byte [lang]، با توجه به زبان موردنظر، فایل مربوطه انتخاب و برای خروج از حالت فشرده به تابع variable_byte_to_positional_index تحویل داده می‌شود. برای حالت gamma code:

- پس از وارد کردن دستور compress gamma_code [lang] و انجام فرآیند فشرده‌سازی، نتیجه‌ی کار در فایل‌ی ذخیره می‌گردد. به صورت خودکار، اگر زبان انگلیسی بود، در فایل gamma_code_english و اگر زبان فارسی بود، در فایل gamma_code_persian.

- پس از وارد کردن کوئری decompress gamma_code<lang>، با توجه به زبان موردنظر، فایل مربوطه انتخاب و برای خروج از حالت فشرده به تابع gamma_code_to_positional_index تحویل داده می‌شود. در ادامه حجم نمایه‌های ایجاد شده را مقایسه می‌کنیم. حجم فایل‌های ایجاد شده در [جدول ۱](#) قابل مشاهده است.

 gamma_code_english	11/13/2020 8:32 PM	File	833 KB
 positional_english_indexing	11/13/2020 8:31 PM	File	1,308 KB
 variable_byte_english	11/13/2020 8:32 PM	File	1,060 KB
 gamma_code_persian	11/13/2020 8:36 PM	File	5,362 KB
 positional_persian_indexing	11/13/2020 8:35 PM	File	8,034 KB
 variable_byte_persian	11/13/2020 8:36 PM	File	7,730 KB

تصویر ۷: فایل‌های ذخیره شده قبل و بعد از فشرده‌سازی

جدول ۱: حجم نمایه‌ی positional قبل و بعد از فشرده‌سازی

فارسی	انگلیسی	
۸۰۳۴	۱۳۰۸	حجم قبل از فشرده‌سازی (KB)
۷۷۳۰	۱۰۶۰	حجم با فشرده‌سازی variable byte (KB)
۵۳۶۲	۸۳۳	حجم با فشرده‌سازی gamma code (KB)

اصلاح پرسمان

در این بخش به اصلاح پرسمان ورودی کاربر می پردازیم. پس از پیش پردازش پرسمان، با استفاده از نمایه bigram و معیار جاکارد، لیستی از کلمات پیشنهادی را برای هر کلمه‌ی پرسمان ورودی پیدا می کنیم و سپس فاصله‌ی ویرایش کلمه‌ی پرسمان با هر یک از این کلمات را باز می گردانیم و در نهایت کلمه‌ای که این فاصله ویرایش را کمینه می کند به عنوان کلمه‌ی پیشنهادی استفاده می کنیم. در نهایت نیز پرسمان اصلاح شده را به کاربر نشان می دهیم. در ادامه به شرح بیشتر توابع این بخش می پردازیم.

استفاده از نمایه bigram و معیار جاکارد برای یافتن کلمات مشابه

توابع این قسمت شامل jaccard_similarity و correction_list هستند.

تابع jaccard_similarity یک کلمه‌ی پرسمان و یک ترم دیکشنری دریافت می کند و مجموعه‌ی دوحرفی‌های پرسمان را محاسبه می کند (از \$ صرف نظر می کنیم). در واقع معیار جاکارد به صورت حاصل تقسیم اندازه‌ی اشتراک مجموعه‌ی دوحرفی‌های پرسمان و ترم، بر اندازه‌ی اجتماع آنها محاسبه می شود. از این رو در posting list هر کدام از دوحرفی‌های پرسمان در نمایه bigram، اگر ترم ورودی را پیدا کند، یک واحد به اشتراک دو مجموعه اضافه می شود و در نهایت با استفاده از فرمول زیر معیار جاکارد باز گردانده می شود:

$$\text{jaccard}(A, B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

سپس نوبت به باز گرداندن لیست کلمات پیشنهادی می رسد. تابع correction_list یک کلمه و یک threshold را علاوه بر زبان ورودی می گیرد. سپس آن کلمه‌ی پرسمان را به تعدادی دوحرفی می شکند و به ازای هر ترم در posting list، شباهت جاکارد بین کلمه‌ی پرسمان و ترم انتخاب شده را محاسبه می کند و اگر میزان این شباهت از threshold ورودی بیشتر بود، این ترم را به مجموعه کلمات پیشنهادی اضافه می کند و در نهایت آرایه‌ای از این کلمات باز می گرداند.

کاربر با استفاده از دو دستور زیر می تواند دو تابع معرفی شده در این بخش را بررسی کند:

```
jaccard [lang] [query] [dictionary term]
```

```
correction_list [lang] [query]
```

در فراخوانی دستور دوم، ابتدا عدد ۰,۴ به عنوان threshold داده می شود اما اگر آرایه‌ی خروجی تابع تهی باشد، ۰,۱ از آن کم می شود و این عمل تا زمانی که لیست پیشنهادی تهی نباشد، ادامه پیدا می کند.

در ادامه مثالی از کار با دستورات jaccard و correction_list مشاهده می‌شود.

```
jaccard english life life
1.0
jaccard english nima lima
0.5
correction_list nima
not a valid command!
correction_list english nima
['mani', 'anim', 'unimagin', 'minimart', 'imag', 'imam', 'maim', 'lima']
```

تصویر ۸: مثالی از فراخوان دستورات jaccard و correction_list

تابع محاسبه‌ی فاصله ویرایش دو کلمه

تابع edit_distance برای این قسمت طراحی شده‌است. این تابع با استفاده از برنامه‌نویسی پویا طراحی شده‌است و در هر مرحله $dp[i][j]$ به صورت زیر محاسبه می‌شود:

$$dp[i][j] = \begin{cases} dp[i-1][j-1] & \text{if query}[i-1] = \text{term}[j-1] \\ 1 + \min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]) & \text{otherwise} \end{cases}$$

و در نهایت $dp[\text{len}(\text{query})][\text{len}(\text{term})]$ به عنوان خروجی بازگردانده می‌شود.

در ادامه مثال‌هایی از عملکرد تابع edit_distance دیده می‌شود.

```
edit_distance english create cerate
2
edit_distance english snow oslo
3
edit_distance persian میباید میباید
1
```

تصویر ۹: مثالی از محاسبه‌ی فاصله‌ی ویرایش

نمایش پرسمان اصلاح‌شده

در این قسمت از توابع دو قسمت قبلی استفاده می‌کنیم. در صورتی که پس از پیش‌پردازش پرسمان، کلمه‌ای موجود باشد که در دیکشنری موجود نباشد، با استفاده از قسمت اول لیستی از واژه‌های پیشنهادی را باز می‌گردانیم، دقیقاً مانند عملیاتی که هنگام اجرای دستور correction_list انجام می‌شود. سپس از تابع edit_distance استفاده می‌کنیم و کلمه‌ای را که کمترین فاصله‌ی ویرایش را با کلمه‌ی پرسمان دارد، جایگزین می‌کنیم. مثالی از عملکرد این قسمت در بخش جستجوی [lrc-lrc](#) پرسمان قابل مشاهده‌است.

جستجو و بازیابی اسناد

در این بخش کاربر می‌تواند پرسمان خود را وارد کند و پس از اصلاح پرسمان (در صورت نیاز)، در بخش اول می‌تواند مستندات مرتبط را بر اساس امتیاز ltc-lnc مشاهده کند و در بخش دوم نیز بعد از اصلاح پرسمان، بررسی می‌کند که در کدام مستندات تمامی این کلمات در پنجره‌ی داده شده وجود دارند و سپس آنها را به ترتیب امتیاز به کاربر نشان می‌دهد. در ادامه به شرح توابع این بخش می‌پردازیم.

جستجوی ltc-lnc پرسمان

این بخش با فراخوانی دستور [lang] query فعال می‌شود. پس از ورود کوئری توسط کاربر، تابع اصلاح کوئری صدا زده می‌شود تا در صورت لزوم اصلاح شود. سپس تابع process_usual_query فراخوانی می‌شود. این تابع کوئری را به روش lnc بررسی می‌کند و طول پرسمان و تکرار هر ترم در پرسمان را به دست می‌آورد و سپس به ازای هر شناسه‌ی مستند، تابع tf_idf صدا زده می‌شود. این تابع ابتدا مقدار عبارت $(1 + \log tf) \times idf$ را برای هر ترم مستند محاسبه می‌کند و سپس با فراخوانی تابع doc_length، طول بردار مستند را محاسبه می‌کند و در نهایت ضرب داخلی بردار مستند و بردار پرسمان را خروجی می‌دهد. سپس این امتیازات مرتب می‌شوند و در نهایت حداکثر ۱۰ مستندی که امتیاز ltc-lnc مخالف صفر دارند، خروجی داده می‌شود.

در ادامه نمونه‌ای از فراخوانی دستور مرتبط با این بخش دیده می‌شود. شایان ذکر است که در صورتی که خود tf به جای $1 + \log tf$ در نظر گرفته می‌شد، نتایج بهتری به دست می‌آمد.

```
Enter Your Query: ancient disease
suggested correction for the query: ancient disease
document 1218: [[ 'trach', 'ancient', 'disease', 'use', 'plagu', ['imagin', 'smile', 'learn', 'disease', 'stud', 'histori', 'human', 'disease', 'ancient', 'humind', 'present', '
ltc-lnc score: 0.2834818871427994
document 367: [[ 'nose', 'photo', 'asthma', 'drugexist', 'th', ['ancient', 'disease', 'take', 'dead', 'fore', 'pass', 'hachway', 'stare', 'power', 'photograph', 'adrb', 'swell
ltc-lnc score: 0.1981999178946846
document 766: [[ 'disce', 'ancient', 'climate', 'ocean', 'ice', 'rop', 'dome', 'hunt', 'data', 'climate', 'year', 'egg', 'find', 'clap', 'inside', 'ancient', 'seab', 'coral', '
ltc-lnc score: 0.1525621197942225
document 1858: [[ 'ancient', 'ancient', 'captiv', ['ancient', 'monument', 'give', 'slav', 'autonish', 'peak', 'civil', 'they', 'order', 'threat', 'pollut', 'war', 'neglect', 'h
ltc-lnc score: 0.3558361816543887
document 346: [[ 'ancient', 'engine', 'make', 'harvest', ['window', 'wit', 'musam', 'ajshra', 'tala', 'amaz', 'feat', 'begin', 'bull', 'centuri', 'egg', 'pengi', 'india', 'g
ltc-lnc score: 0.1487841117481993
document 58: [[ 'killer', 'american', 'diet', 'that', 'sawp', 'planet', ['forget', 'latest', 'disease', 'news', 'cardiovascular', 'disease', 'hill', 'pengi', 'than', 'everyth', '
ltc-lnc score: 0.1382504369129442
document 2137: [[ 'is', 'disce', 'secret', 'ancient', 'text', ['gregori', 'heyarthy', 'factual', 'lab', 'work', 'way', 'read', 'ancient', 'manuscript', 'map', 'use', 'spectral'
ltc-lnc score: 0.1372881516599236
document 2888: [[ 'scoe', 'well', 'core', 'disease', 'cell', 'not', 'poll', ['survive', 'media', 'treatment', 'boll', 'cow', 'lit', 'word', 'have', 'disease', 'take', 'bill', 'h
ltc-lnc score: 0.13615779151982878
document 829: [[ 'end', 'shape', 'kill', ['work', 'day', 'kill', 'side', 'london', 'million', 'west', 'sare', 'where', 'die', 'all', 'find', 'time', 'architect', 'caroten', '
ltc-lnc score: 0.1353885731744978
document 942: [[ 'heart', 'photo', 'polar', 'ice', ['photograph', 'canis', 'season', 'shoot', 'iceberg', 'show', 'complex', 'beast', 'these', 'massiv', 'ancient', 'think', 'h
ltc-lnc score: 0.131428898811944
```

تصویر ۱۰: نمونه‌ای از پرسمان انگلیسی، همان‌طور که دیده می‌شود، ابتدا پرسمان اصلاح شده و سپس مستندات مرتبط و شواهد آن چاپ شده‌اند.

فراخوانی این بخش با دستور [lang] proximity query امکان پذیر است و در ادامه مثالی از این بخش دیده می شود.

```

Please Enter Size Of Window:
Enter Your Query: also score
no spell correction needed
document 1796: [{"dir", "orchestra", "futur"}, {"ge", "sang", "make", "domest", "music", "lung", "all", "code", "kling", "big", "stanford", "laptop", "orchestra", "great", "ins
ltr-lms score: 8.2879915489427586
document 1843: [{"act", "made", "store"}, {"artist", "nathali", "mishuch", "have", "weather", "date", "massiv", "store", "turn", "into", "complex", "amulet", "emodi", "form",
ltr-lms score: 8.17110310164231192
document 181: [{"cape", "breton", "fidd", "real", "time"}, {"violinist", "natali", "become", "ted", "music", "director", "thoma", "dolbi", "play", "dolbi", "origin", "song", "t
ltr-lms score: 8.1799216426313554
document 184: [{"by", "tita", "ash", "other"}, {"alt", "grad", "student", "held", "verit", "dome", "siftabl", "bookies", "computer", "tita", "stank", "shuffl", "your", "hand
ltr-lms score: 8.181429687613895
document 179: [{"fidd", "real", "time"}, {"natali", "become", "music", "partner", "daniel", "lashi", "play", "sever", "time", "cape", "breton", "tradit", "sprint", "soul", "s
ltr-lms score: 8.1593270575878453

"Blip", "stanford", "laptop", "orchestra", "great", "inkromant", "out", "unexpected", "like", "alone", "mullian", "play", "music", "that", "both", "beauti", "express"]}

"turn", "into", "complex", "amulet", "emodi", "form", "nature", "time", "these", "amulet", "then", "become", "music", "score", "string", "musical", "play"]}

", "thoma", "dolbi", "play", "dolbi", "origin", "song", "blue", "river", "ether", "dust", "littl", "danc"]}

sa", "computer", "tita", "stank", "shuffl", "your", "hand", "these", "futurtoy", "de", "math", "play", "music", "their", "friend", "tan", "east", "thing", "handson", "learn"]}

"tine", "cape", "breton", "tradit", "sprint", "soul", "style", "folk", "fidd", "inspire", "collabor", "all", "have", "flag", "everyb", "danc", "along"]}

```

تصویر ۱۲: نمونه ای از پرمسان همسایگی

نحوه‌ی تقسیم وظایف

وظایف اختصاص یافته به هر فرد به شرح زیر بود:

نیما جمالی

- ۱- پیش پردازش متون انگلیسی
- ۲- پیاده‌سازی توابع درج و حذف مستند
- ۳- پیاده‌سازی تابع محاسبه‌ی معیار جاکارد
- ۴- محاسبه‌ی فاصله‌ی ویرایش و تصحیح پرسمان کاربر
- ۵- پیاده‌سازی تابع محاسبه‌ی tf-idf بر اساس امتیاز ltc-lnc

سپهر فعلی

- ۱- طراحی و پیاده‌سازی نمایه bigram
- ۲- طراحی و پیاده‌سازی نمایه positional
- ۳- فشرده‌سازی به روش variable byte
- ۴- استخراج نمایه مکانی از حالت فشرده‌ی variable byte
- ۵- ذخیره‌ی فایل‌های فشرده شده و مقایسه‌ی حجم آنها

سینا کاظمی

- ۱- پیش پردازش متون فارسی
- ۲- پیاده‌سازی ذخیره‌سازی و بارگذاری نمایه‌های bigram و positional
- ۳- فشرده‌سازی به روش gamma code
- ۴- استخراج نمایه مکانی از حالت فشرده‌ی gamma code
- ۵- پیاده‌سازی جستجوی proximity

مراجع

- [1] https://agailloty.rbind.io/en/project/nlp_clean-text/
- [2] <https://www.sobhe.ir/hazm/>
- [3] <https://nlp.stanford.edu/IR-book/html/htmledition/context-sensitive-spelling-correction-1.html>
- [4] <https://docs.python.org/3/library/stdtypes.html>