بازیابی پیشرفته اطلاعات

فاز اول پروژه

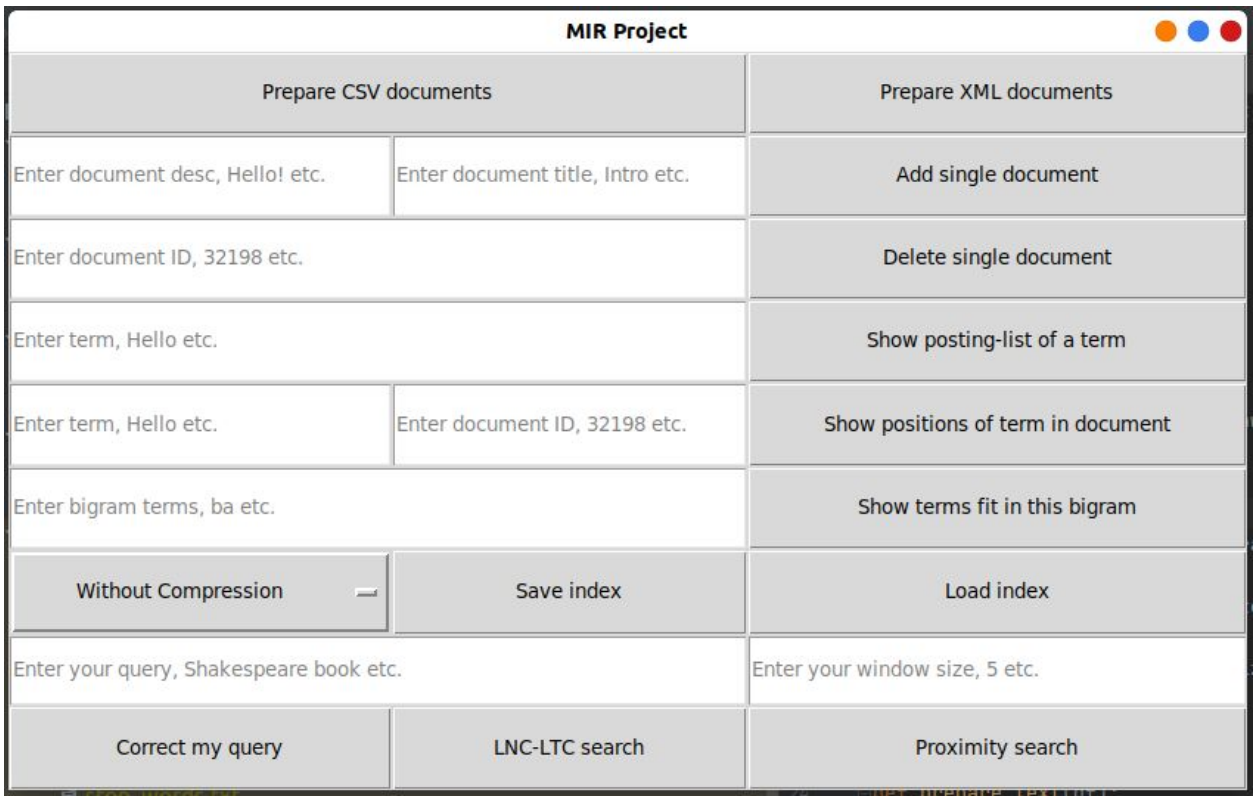
اعضای گروه:

حامد علی‌محمدزاده - ۹۶۱۰۲۰۲۹
حمیدرضا هدایتی - ۹۶۱۰۹۹۳۹
آرمین سعادت بروجنی - ۹۶۱۰۵۸۲۹

آبان ۱۳۹۹

بخش 0. کنسول

ما به طور کلی از کتابخانه tkinter پایتون برای رابط کاربری استفاده کردیم (که با sudo apt-get install python-tk باید آن را بر روی کامپیوتر خود نصب کنید) و به ازای هر بخش از پروژه، قسمتی در رابط کاربری برای آن بخش مورد نظر اختصاص دادیم، در این اسکرین‌شات شکل کلی رابط کاربری ما را می‌توانید مشاهده کنید:



برای مثال بخش prepare کردن اطلاعات در دو دکمه بالا پیاده‌سازی شده، برای تغییر پویای نمایه از add single document و delete single document و ...

بخش ۱. پیش پردازش اولیه

ما در xml فرض را بر این گذاشتیم که تعریف یک document یک text و title به ازای هر page در صفحات ویکی‌پدیا است و در csv فرض‌مان این بود که یک title و description در هر سطر تعریف یک document است، و در نهایت هر دو تعریف درون پروژه ما تبدیل به document هایی می‌شود که هر کدام title و description خاص خودشان را دارند. در این پروژه توانایی اضافه کردن csv و xml به هر زبانی دارید و پروژه خود با تشخیص زبان داده import شده آن را پیش‌پردازش می‌کند و نمایه مربوطه را می‌سازد.
دو فایل کلی واسه پیش‌پردازش به هر دو زبان وجود دارد، preprocess_eng.py و preprocess_per.py که هر دو در پکیج preprocess قرار گرفته‌اند، و در هر دو تابعی به نام prepare_text وجود دارد که ورودی آن یک دیتافریم است که فرض شده ستون‌های description و title دارند.
برای پیش‌پردازش فارسی از کتابخانه hazm و انگلیسی از کتابخانه nltk استفاده می‌کنیم.

1. **جداسازی**

از تابع word_tokenize کتابخانه nltk برای زبان انگلیسی و از تابع word_tokenize کتابخانه hazm برای زبان فارسی استفاده کردیم که raw را به لیستی از tokenها تبدیل کنیم.

2. **نرمال‌سازی متنی**

برای زبان فارسی از تابع normalizer استفاده کردیم و پس از آن جدا سازی را انجام دادیم.

3. **حذف علائم نگارشی**

برای نرمال کردن و حذف token های غیر حرفی از تابع استاندارد پایتون isalpha() استفاده کردیم و اگر کلمه‌ای isalpha بود آن را نگه داشتیم.

4. **بازگرداندن کلمات به ریشه**

در زبان فارسی از دو تابع stemmer.stem و lemmatizer.lemmatize استفاده کردیم که به ترتیب stemming و lemmatizing را انجام می‌دهند.
در زبان انگلیسی پس از stem کردن به وسیله snowball.stem اول کلمات را به حالت گذشته آن‌ها برده lemma.lemmatize word, pos=v و سپس به حالت کنونی می‌آوریم pos=n

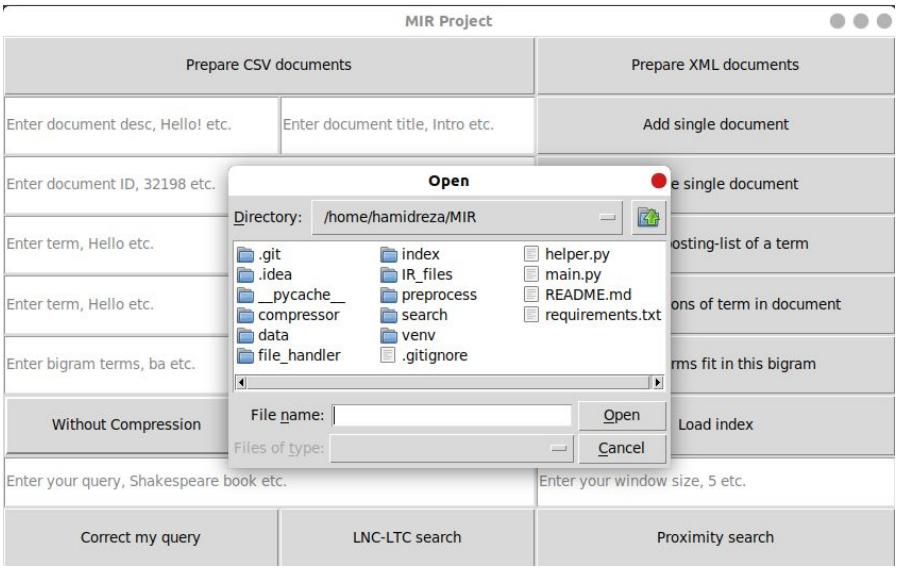
5. **یافتن و حذف نکات پرتکرار**

بعد از clean کردن یک raw و جداسازی آن‌ها وقت آن می‌رسد که کلمات پرتکرار را پیدا کرده و آن‌ها را حذف کنیم که در اینجا از تابع پیاده‌سازی شده توسط خودمان به اسم find_stop_words استفاده کردیم که کلمات را بر اساس تعداد تکرار آن‌ها در تمامی document ها مرتب می‌کند و به اندازه درصد تعیین شده (stop_words_ratio) کلمات پرتکرار را باز می‌گرداند و پس از آن از دیتافریم حذف می‌شود.

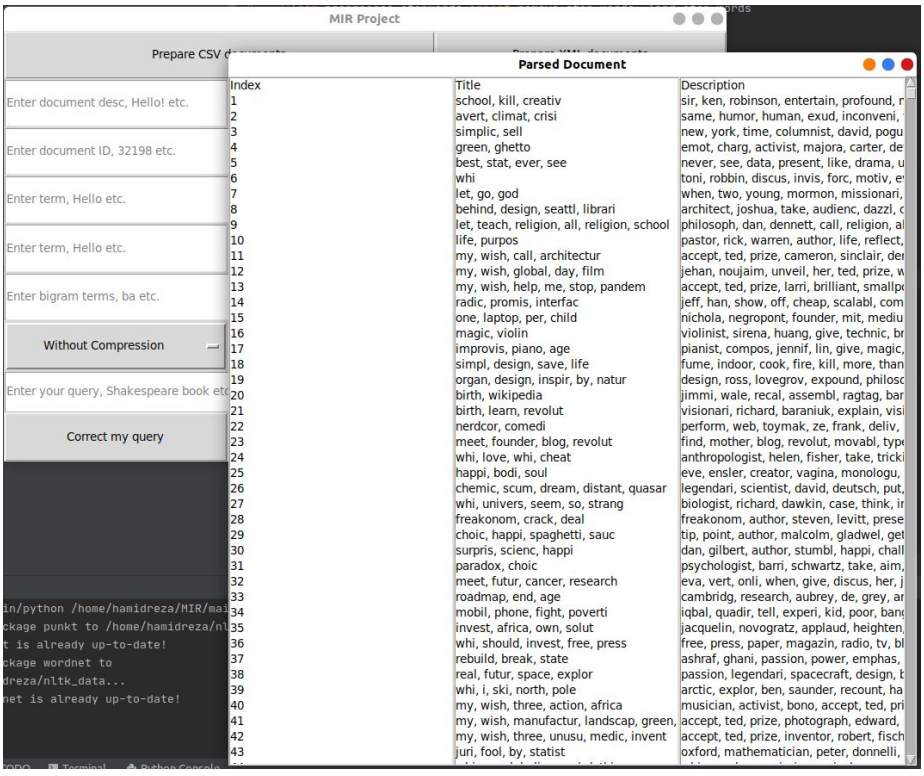
بارم‌بندی:

1. **گرفتن متن از کاربر و نمایش لغات آن بعد از پیش‌پردازش متنی**

   بعد از انجام مراحل پیش‌پردازش، جدول مربوطه پوشه‌ها را دوباره به کاربر نشان می‌دهیم که مراحل آن به شکل زیر است:
   با فشار دکمه prepare csv یا prepare xml صفحه زیر برای انتخاب داده مربوطه نشان داده می‌شود.



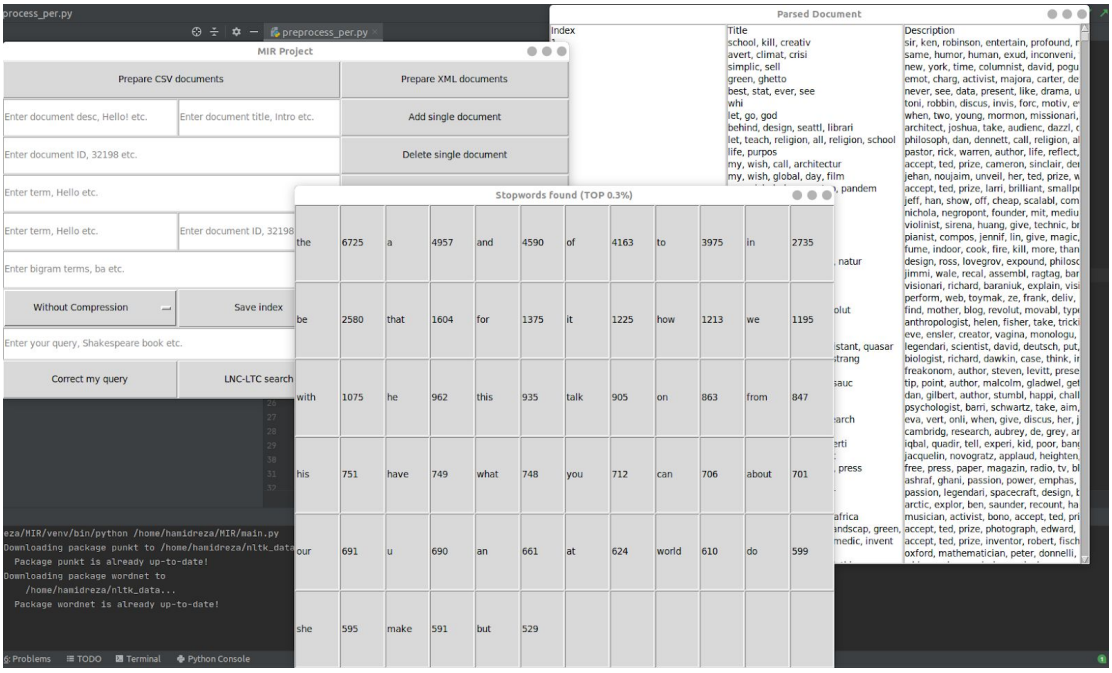
و بعد از انتخاب فایل داده، لغات بعد از پیش پردازش نشان داده می‌شود:



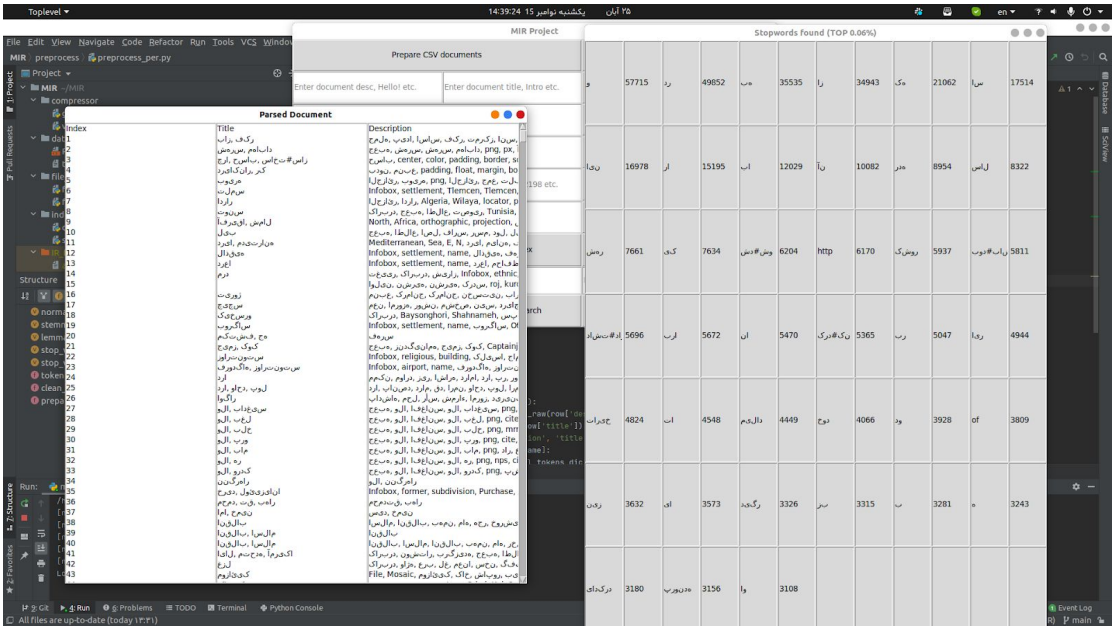
که index، آیدی داکیومنت مورد نظر و title و description اطلاعات تمیز شده مربوط به آن داکیومنت است.

2. نمایش متون پرتکرار

همزمان با نشان دادن لغات لغات پرتکرار به همراه تعداد تکرارهای آن‌ها در جدولی جداگانه نشان‌دهنده می‌شود:



به طور مشابه بعد از import کردن داده فارسی:



بخش ۲. نمایه‌سازی

بعد از پیش‌پردازش متن، پوشه‌ها را تک تک به نمایه‌مان اضافه می‌کنیم. به طور کلی state کلی برنامه بدین صورت است:

:positional
یک دیکشنری که کلید آن آیدی یک توکن و مقدار آن برابر پوستینگ لیست جایگاهی آن توکن است.
:bigram
یک دیکشنری که کلید آن یک دو حرفی است و مقدار آن ترم‌هایی که آن بای‌گرام در آن‌ها مشاهده شده.
:all_tokens
لیست تمامی توکن‌ها به صورتی که شماره ۱۰ آن برابر با ۱۰مین توکنی‌ست که مشاهده شده.
:doc_is_available
لیستی از پوشه‌های به صورتی که خانه شماره ۱۰ آن در صورتی که true باشد یعنی داکیومنت شماره ۱۰ حذف نشده است و هنوز وجود دارد.
:normalized_doc
مربع اندازه بردار هر داکیومنت که در سرچ از آن استفاده می‌شود.
:stop_words
لیست لغات پرتکرار که در پیش پردازش بدست آورده‌ایم.

به جز اطلاعات بالا، داده‌های دیگری نیز که برای افزایش سرعت و راحتی مورد نیاز است وجود دارد که بدین ترتیب‌اند:

:Token_map
دیکشنری‌ای است که معکوس آرایه قبلی می‌باشد و به این معنی است که با دادن توکن می‌توان به آیدی آن رسید.

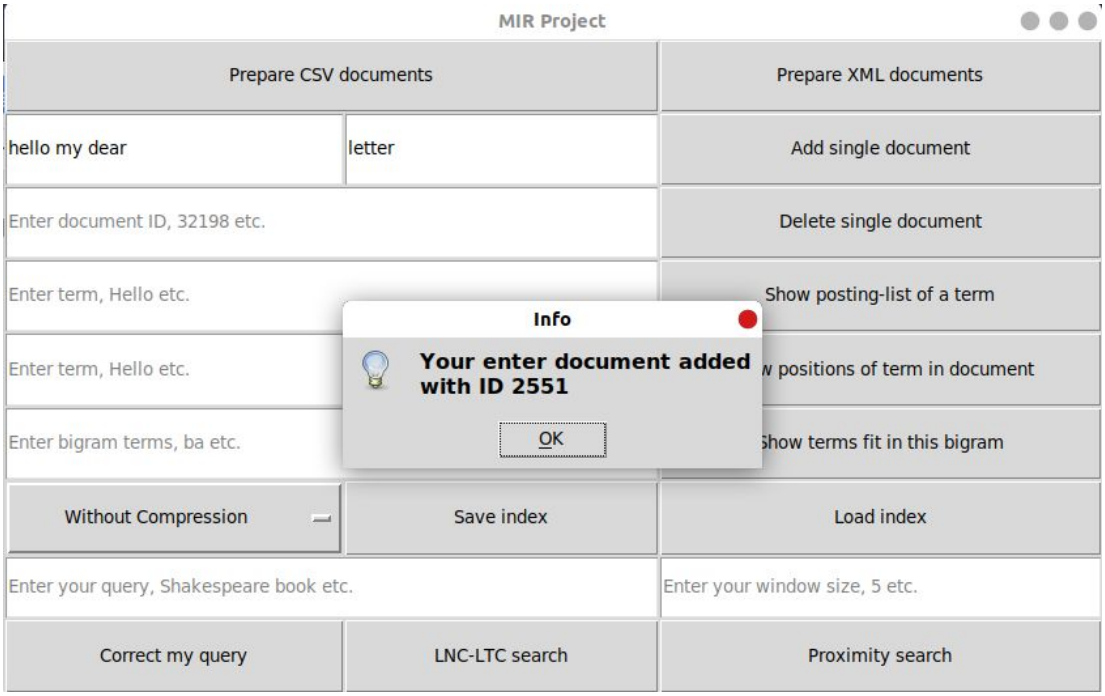
از آنجا که positional و bigram هر دو دیکشنری هستند عملا از دادساختار hashmap برای پیادسازی index هایمان استفاده کرده‌ایم، بعد از اضافه شدن هر داکیومنت به نمایه ما پس از تبدیل کردن token به token_id برای اضافه کردن یک position جدید به نمایه جایگاهی‌مان نخست یک باینری سرچ اجرا کرده و به document مربوطه می‌رسیم و پس از آن برای insert کردن position جدید در لیست مربوطه یک باینری‌سرچ دیگر را اجرا می‌کنیم و پس از یافتن نقطه مناسب آن position را insert می‌کنیم، تلاش شده که اضافه کردن یک document در بهترین اردر زمانی انجام شود که عملا پیچیدگی زمانی آن از order زیر است:
O(number_of_tokens_in_document*log(df(t))*log(cf(t)))
و از آن جا که ما عملا اضافه کردن یک فایل داده را به اضافه کردن یک مجموعه داده تبدیل می‌کنیم کد ما قابلیت چند بار اضافه کردن فایل‌ها و همچنین هر مقدار اضافه کردن تنهای اطلاعات را دارد.

بارم‌بندی:

1. **نمایه‌سازی از روی پوشه‌های در اختیار قرار داده شده**

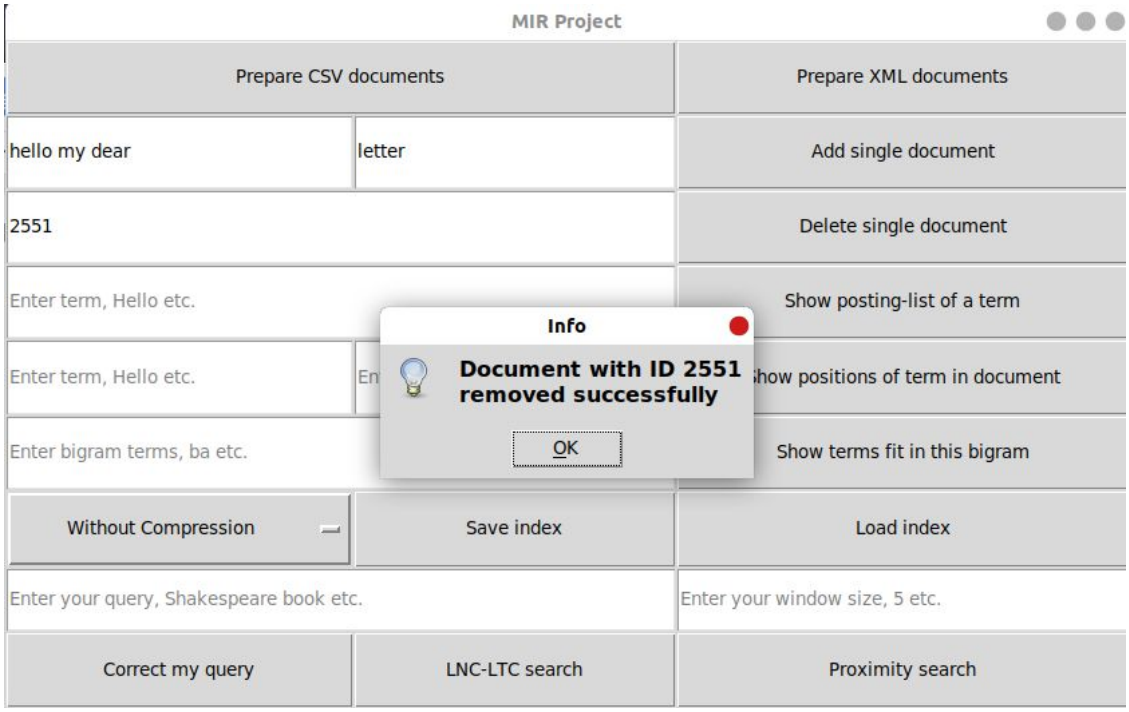
پس از انتخاب فایل برای اضافه کردن به نمایه یا اضافه کردن یک داکیومنت تنها به نمایه، ما استیت برنامه (متغیرهای گفته شده در بالا) را تغییر می‌دهیم، همانطور که گفته شد برای هر دو نمایه positional و bigram یک باینری سرچ اجرا می‌کنیم که روند اضافه کردن بسیار سریع‌تر شود و پس از هر بار ورودی کاربر نمایه ما آپدیت می‌شود.
برای مثال با اضافه کردن یک داکیومنت تنها به نمایه داریم:

بعد از فشردن دکمه:



همچنین برای حذف:

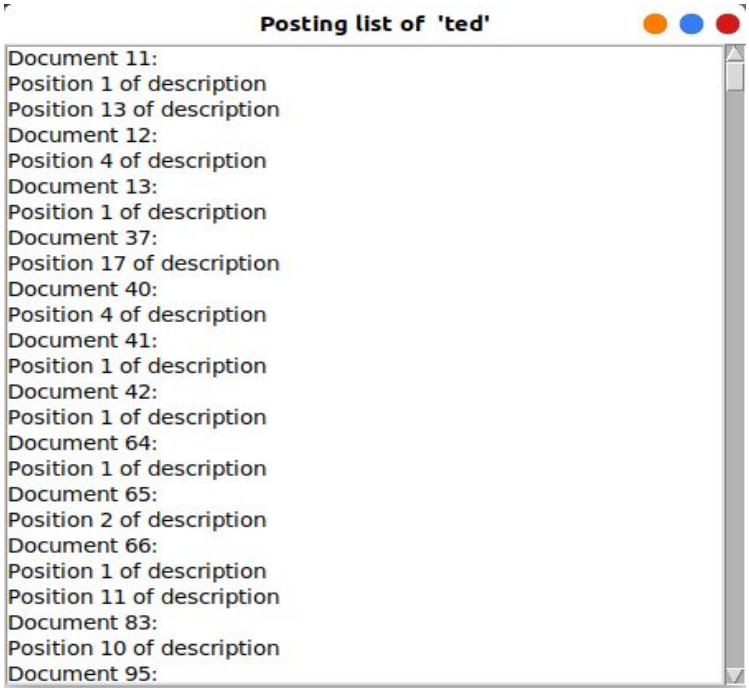
بعد از فشردن دکمه:



2. نمایش posting list کلمه ورودی توسط کاربر

با استفاده از نمایه positional این درخواست را پاسخ می‌دهیم:

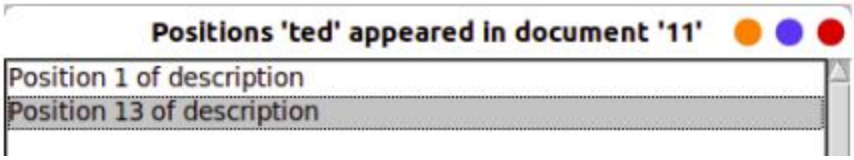
و بعد از فشردن دکمه:



3. **نمایش جایگاه کلمه وارد شده توسط کاربر در هر سند**

به طور مشابه می‌توانیم یک توکن و یک داکیومنت را وارد کنیم و مکان‌های مختلف آن‌ها را مشاهده کنیم:

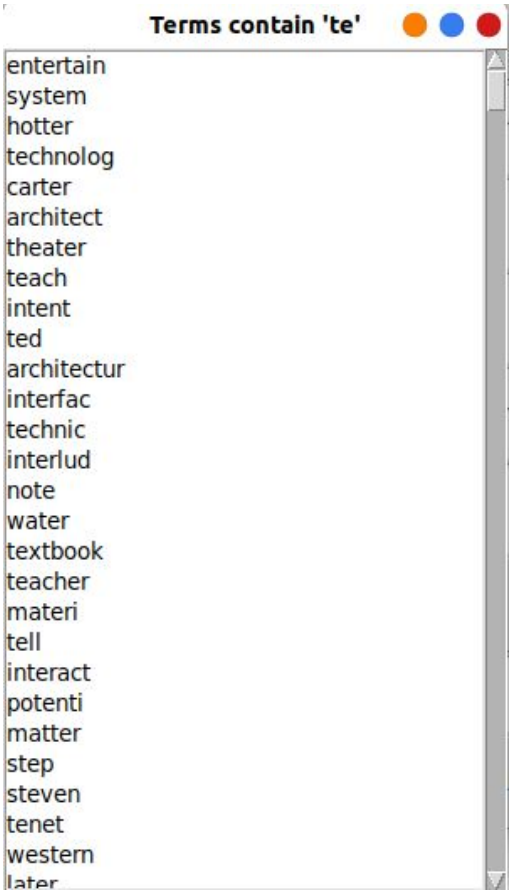
و بعد از فشردن دکمه:



4. **مشاهده تمام کلماتی که دارای یک bigram خاص درون خود هستند**

با استفاده از نمایه bigram که یک دیکشنری بر اساس bigramهاست این درخواست‌ها را پاسخ می‌دهیم: (دقت کنید که برای محاسبه bigramها به شروع و پایان کلمه‌ها $ اضافه می‌کنیم)

و بعد از فشردن دکمه:



بخش ۳. فشرده‌سازی نمایه‌ها

روش‌های فشرده‌سازی gamma-code و variable-byte مطابق با اسلایدهای درس پیاده‌سازی شدند. در نهایت ۴ class طراحی شد که به شکل زیر است:

GammaCodeCompressor
GammaCodeDecompressor
VariableByteCompressor
VariableByteDecompressor

که مسئول فشرده‌سازی و استخراج به حالت عادی هستند.
همانطور که در دستور کار پروژه ذکر شده بود،تنها posting list ها فشرده شدند و روی دیکشنری فشرده‌سازی صورت نگرفته است.
توابع مربوط به فشرده‌سازی به صورت جنریک پیاده‌سازی شده‌اند به این صورت که هم positional و هم bigram به هر دو صورت می‌توانند فشرده شوند.
در ضمن در فشرده‌سازی positional، هم شناسه داکیومنت‌ها (در واقع اختلافشان) و هم ایندکس تعیین جایگاه (آن هم اختلافشان) فشرده شده است.
علاوه بر آن مطابق اسلایدهای درس، عدد صفر نمایش gamma-code ندارد که در این پیاده‌سازی این قضیه نیز مورد بررسی قرار گرفت و عدد صفر نیز ذخیره می‌شود.
همچنین برای اینکه اختلاف اعداد (چه شناسه داکیومنت‌ها چه جایگاه‌ها) ذخیره می‌شود، posting-list همواره به صورت sorted ذخیره و نگه‌داری می‌شود تا از این بابت مشکلی نباشد و عملکرد سامانه نیز بهینه باشد.
در ایندکس جایگاهی، posting list به صورت زیر است:

Posting = [[doc_number1, [10,20,30]], [doc_number2, [5, 10]], ...]

که نشان می‌دهد هر ترم در چه داکیومنت‌هایی و در چه جایگاهی از آن‌ها آمده‌اند.
برای فشرده‌سازی این نوع posting، ابتدا شماره داکیومنت، سپس تعداد دفعات تکرار ترم در آن داکیومنت (که همان طول آرایه جایگاهی است) فشرده می‌شود. پس از آن جایگاه‌ها فشرده می‌شوند. دلیل فشرده‌سازی طول آرایه این است که در زمان decode معلوم شود چه زمان به سراغ داکیومنت بعدی باید رفت.
در ایندکس بایگرم، posting list به صورت زیر است:

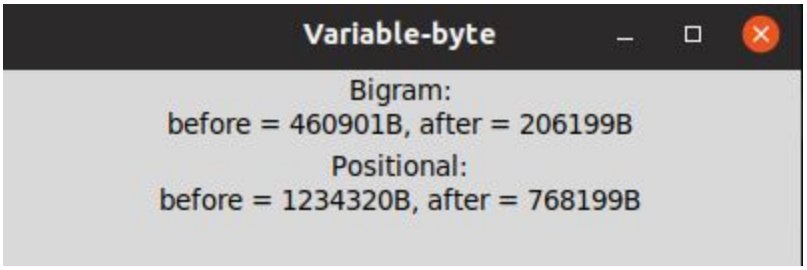
Posting = [doc_number1, doc_number2, …]

که صرفا نشان می‌دهد هر ترم در چه داکیومنت‌هایی ظاهر شده است. برای فشرده‌سازی اختلاف این اعداد که سورت شده هستند ذخیره می‌شود.
برای ذخیره‌سازی اطلاعات از json استفاده شد. اطلاعاتی مانند stop_words یا all_token که هر عدد را به یک term مپ کرده است بدون فشرده‌سازی در فایل جدا ذخیره شدند.
برای ذخیره‌سازی positional و bigram ابتدا مطابق با compress_type مشخص شده فشرده‌سازی صورت گرفت و نتیجه در فایل جداگانه برای هر کدام ذخیره شد.
همچنین پس از ذخیره، حجم فایل ذخیره شده محاسبه شده است.
برای خواندن از فایل نیز از json استفاده شد. پس از خواندن اطلاعات ذخیره شده، بعضی اطلاعات مانند token_map تولید می‌شود. Token_map یک داده ساختار است که هر term را به یک عدد map می‌کند و از روی all_token که ذخیره شده بود به دست می‌آید. سامانه به هر دوی این داده ساختارها نیاز دارد اما چون از روی هم به دست می‌آیند، تنها یکی از آن‌ها ذخیره می‌شود و دیگری به دست می‌آید تا سربار خواندن و نوشتن از فایل بهینه شود.
در زمان خواندن همچنین positional و bigram به توابع decompressor داده شده و از حالت فشرده خارج می‌شوند. و در نهایت ایندکس عادی به سامانه تحویل داده می‌شود.

بارم‌بندی:

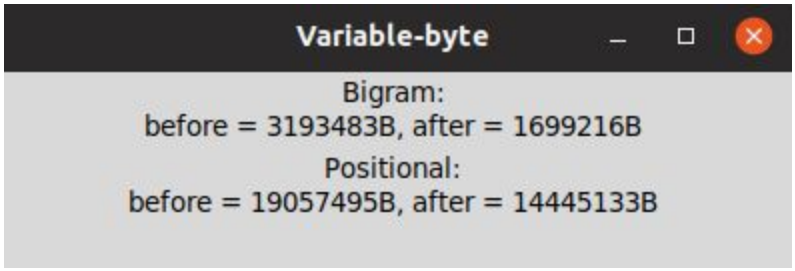
1. **پیاده‌سازی و نمایش میزان حافظه اشغال شده قبل و بعد از اعمال variable-byte**

فشرده‌سازی روی داده‌های csv:



مشاهده می‌شود که bigram از حدود 460KB به حدود 206KB کاهش یافته که بیش از ۵۰ درصد فشرده شده است.
مشاهده می‌شود که positional از حدود 1.2MB به حدود 0.7MB کاهش یافته که حدود ۴۰ درصد فشرده شده است.

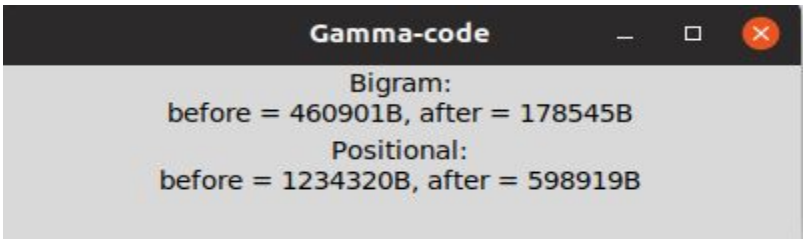
فشرده‌سازی روی داده‌های xml:



مشاهده می‌شود که bigram از حدود 3.2MB به حدود 1.7MB کاهش یافته که حدود ۵۰ درصد فشرده شده است.
مشاهده می‌شود که positional از حدود 19MB به حدود 14.5MB کاهش یافته که حدود ۲۵ درصد فشرده شده است.

2. **پیاده‌سازی و نمایش میزان حافظه اشغال شده قبل و بعد از اعمال gamma-code**

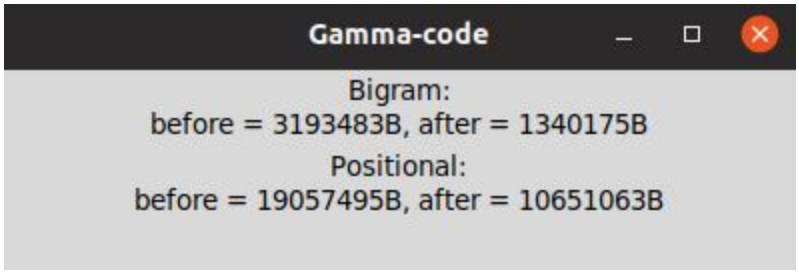
فشرده‌سازی روی داده‌های csv:



مشاهده می‌شود که bigram از حدود 460KB به حدود 178KB کاهش یافته که بیش از ۶۰ درصد فشرده شده است.

مشاهده می‌شود که positional از حدود 1.2MB به حدود 0.6MB کاهش یافته که حدود ۵۰ درصد فشرده شده است.

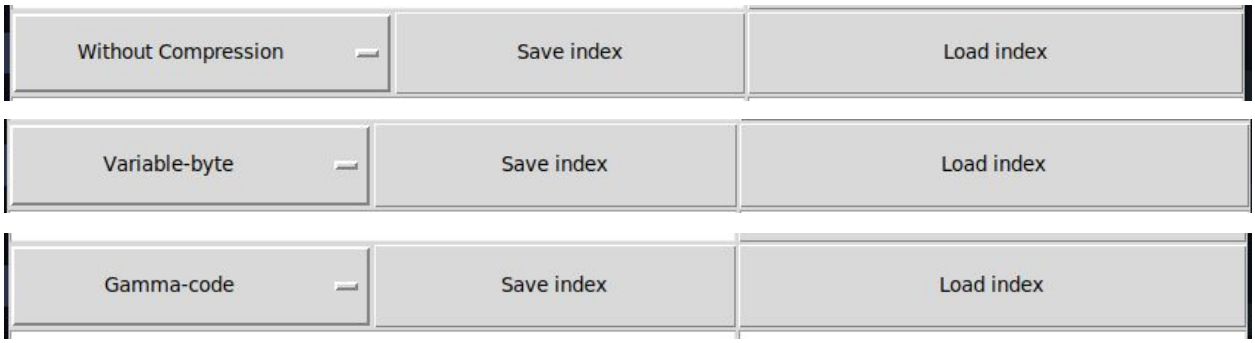
فشرده‌سازی روی داده‌های xml:



مشاهده می‌شود که bigram از حدود 3.2MB به حدود 1.3MB کاهش یافته که حدود ۶۰ درصد فشرده شده است.
مشاهده می‌شود که positional از حدود 19MB به حدود 10.5MB کاهش یافته که حدود ۴۵ درصد فشرده شده است.

3. ذخیره‌سازی نمایه‌ها در فایل و بارگذاری از آن

پس از انتخاب شیوه فشرده‌سازی، برای ذخیره کافی است دکمه save و برای بارگذاری دکمه load زده شود.
پس از save حجم فایل فشرده شده و فایل اولیه نمایش داده می‌شود.
پس از load داده‌ها در سامانه بارگذاری شده وسامانه آماده کار است.



همانطور که مشاهده می‌شود، ۳ حالت برای فشرده‌سازی وجود دارد:

Without Compression
Variable-byte
Gamma-code

دقت شود که حالت فشرده‌سازی یک فایل در هنگام save و load باید یکسان باشد.

بخش ۴. اصلاح پرسمان

با استفاده از نمایه bigram می‌خواهیم که ۱۰ (top_jaccard_items_candidate) کاندید نخست فاصله jaccard با term وارد شده را پیدا کنیم، برای انجام اینکار، نخست bigramهای term وارد شده برای اصلاح را بررسی می‌کنیم، به ازای هر کدام از آن‌ها کلماتی که آن bigram دارند را بررسی می‌کنیم و تعداد bigramهای مشترک آن کلمه را با کلمه وارد شده محاسبه می‌کنیم و در نهایت jaccard distance را بدست می‌آوریم (که برابر با یک منهای اشتراک مجموعه bigramها تقسیم بر اجتماع مجموعه هر دو است). پس از آن آن‌هایی که کمترین فاصله را دارند مرتب می‌کنیم (تمامی منطق گفته شده در تابع correct در فایل spell_checker زده شده) و آن‌ها را برای محاسبه edit distance به تابع edit_distance ورودی می‌دهیم.

برای محاسبه فاصله ویرایش، الگوریتم dynamic programming نوشته شده که بدین ترتیب تعریف شده:

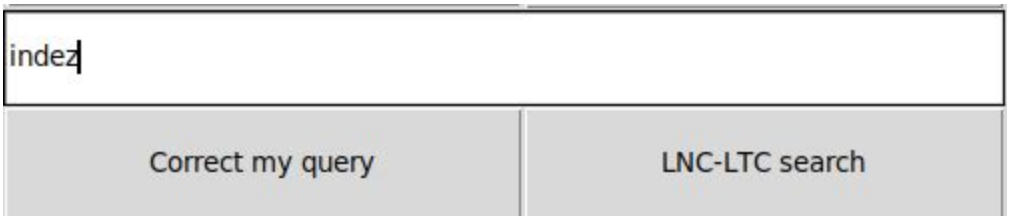
dis[i][j] = max(i,j) if i=0 or j=0 else min(dis[i-1][j],dis[i][j-1],dis[i-1][j-1] + 1 if a[i] == b[j])

که به این معنی است که اگر بخواهیم تا پریفیکس i استرینگ اول را به پریفیکس j استرینگ دوم تبدیل کنیم چقدر هزینه خواهد داشت.

بارم‌بندی:

1. **نمایش پرسمان اصلاح شده**

بعد وارد کردن کوئری در فیلد خواهیم داشت:



و بعد از زدن دکمه correct my query داریم:

و در خود فیلد هم کلمه وارد شده تغییر می‌کند که برای search کردن کارمان ساده‌تر شود.

2. **محاسبه فاصله جاکارد دو کلمه**

همانطور که در بالا می‌بینیم فاصله جاکارد کلمه تمیز شده و کلمه اصلاح شده نشان داده شده است.

3. **محاسبه فاصله ویرایش دو کلمه بدون استفاده از کتابخانه‌های آماده**

همانطور که در بالا می‌بینیم فاصله ویرایشی کلمه تمیز شده و کلمه اصلاح شده نشان داده شده است.

بخش ۵. جستجو و بازیابی اسناد

برای محاسبه وزن‌دهی lnc-ltc روی تمامی داکیومنت‌هایی که termهای query را دارند حرکت می‌کنیم (در تابع search در فایل LNC_LTC در پکیج ، وزن یک term در کوئری برابر می‌شود با:

```
df = len(index.positional[token_id])
weight_term_query = (1 + log(count)) * log(N/df) / sqrt(normalize_query)
```

و به ازای هر document برای محاسبه وزن term در document داریم:

```
weight_term_doc = (1+log(tf_document) * 1 / sqrt(index.normalize_doc[document])
```

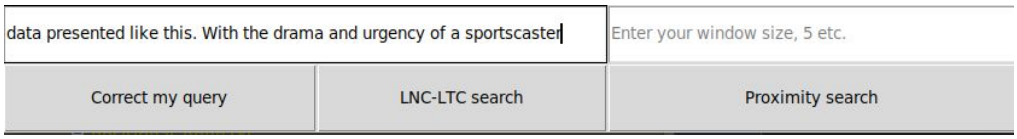
سپس با ضرب داخلی این دو عدد در یکدیگر و مرتب کردن داکیومنت‌ها نتایج سرچ را به کاربر نشان می‌دهیم.

آرایه normalize_doc آرایه‌ای است که موقع اضافه کردن یک داکیومنت به نمایه محاسبه می‌شود و ذخیره می‌شود و در جستجوی‌مان از آن استفاده می‌کنیم.

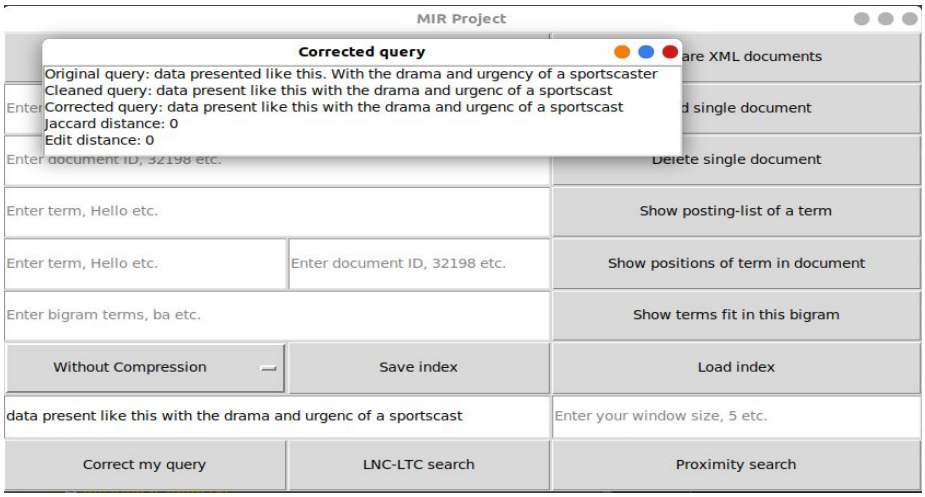
برای جستجوی proximity نیز با توجه به سایز پنجره وارد شده، بر روی تمامی داکیومنت‌هایی که termهای query را شامل می‌شوند حرکت می‌کنیم و چک میکنیم که آیا در description یا title آن document تمامی queryها در یک بازه به اندازه پنجره وجود دارند یا نه و لیستی از documentهای مناسب را پیدا می‌کنیم (در تابع search در proximity در پکیج search) و سپس تمامی آن‌ها را به جستجوی lnc_ltc پاس می‌دهیم و نتایج را به کاربر نشان می‌دهیم.

بارم‌بندی:

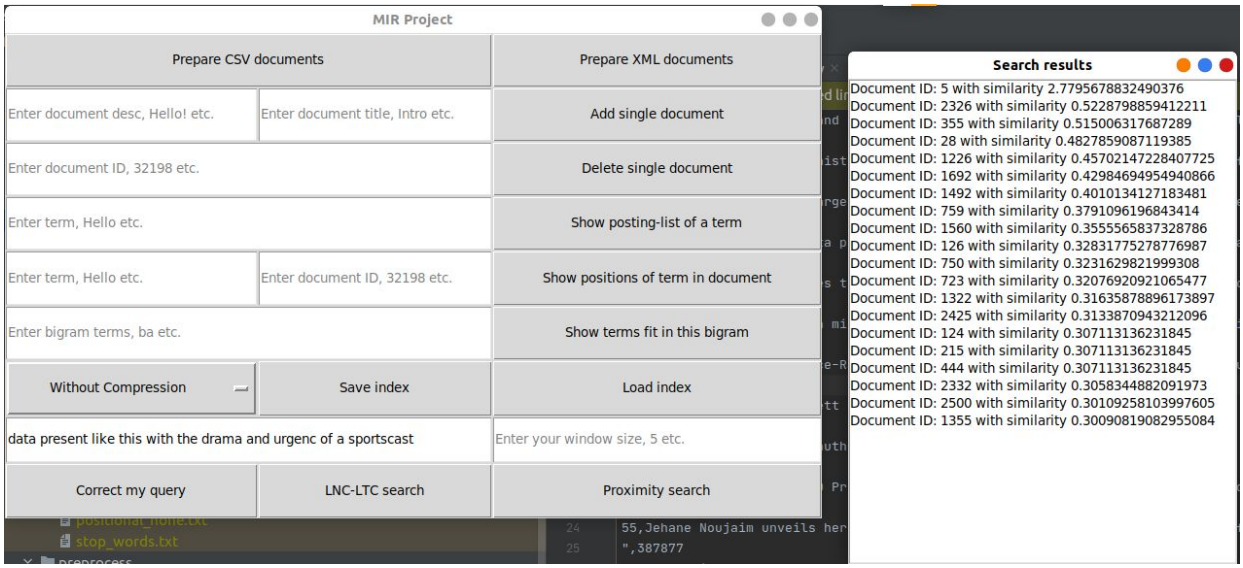
1. **نمایش لیست اسناد مرتبط به ترتیب شباهت در جستجو ترتیب‌دار در فضای برداری tf-idf به روش lnc-ltc**
   با وارد کردن کوئری (و قبل از آن correct کردن آن) می‌توانیم بوسیله ویژگی lnc-ltc پیادسازی شده نتایج سرچمان را مشاهده کنیم:



و بعد از اصلاح کردن پرسمان داریم:

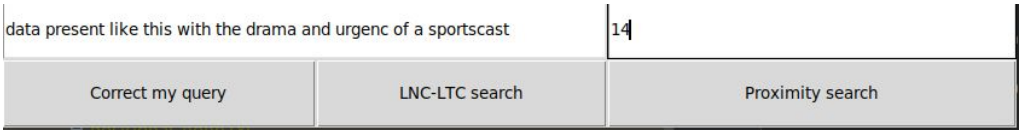


و بعد از زدن دکمه جستجوی lnc-ltc:



2. نمایش لیست اسناد مطابق با پرسمان و اندازه پنجره ورودی در جستجو proximity

با محدود کردن همان پرسمان در یک پنجره داریم:



که نتایج آن بدین شکل است: