گزارش فاز اول پروژه بازیابی اطلاعات

**بخش پیش پردازش**

از بخش‌های ابتدایی هر پروژه‌ی بازیابی پیش پردازش متنی متون اولیه است . در این بخش با برخی از توابع به ساده‌سازی و یکسان سازی متون می پردازیم. با توجه به اینکه دو نوع داده‌ی اولیه ( فارسی و انگلیسی) داریم از دو کتابخانه Hazm برای متون فارسی و از nltk برای متون انگلیسی استفاده شده است.

برای ساده سازی عملیات‌ها ابتدا برای بخش فارسی فایل xml داده شده را با یک تابع به صورت فایل csv تبدیل نموده که فرمت دو داده یکسان شده و استخراج داده‌های لازم از آنها راحت تر باشد.

دو تابع اصلی این بخش تابع pre\_proccess و most\_freq\_words  می‌باشد که طبق خواسته داک متن پیش پردازش شده و کلمات پر تکرار را نشان میدهد. تابع pre\_proccess رشته‌ی متنی را ورودی گرفته و داخل آن با فراخوانی تابع prepare\_text متن نرمال سازی شده را برگردانده می‌شود . عملیا‌ت‌هایی که در این تابع اصلی صورت می‌گیرند :

- عملیات tokenize:   
به این صورت که با استفاده از تابع‌های اماده‌ی کتابخانه nltk و hazm متن به صورت کلمه به کلمه توکن می‌شود.

- عملیات normalization:   
در این عملیات برای داده‌ی Ted Talk ، ابتدا تمامی حروف را lower case کرده سپس از لیست آماده stopwords استفاده شده است تا کلمات پر تکرار حذف شوند اما برخی از متنها که تنها شامل این کلمات بودند کاملا حذف می‌شدند ، لذا

لیستی از کلمات پرتکرار ایجاد کرده و ۳۰ کلمه‌ی پر تکرار را از کل متن حذف شده است برای حذف علائم نگارشی نیز از تابع آماده‌ی isalpha() کمک گرفته ‌شد.  
برای داده‌ی WIKI نیز آرایه‌ی  punctuations حذف شدند و با  تابع آماده‌ی normalize متن نرمال سازی شد.

-عملیات stemming \lemmatization:

در این عملیات از lemmatize دو کتابخانه ذکر شده استفاده شده است ، lemmatize کردن کلمات و افعال در متون انگلیسی بر اساس verb یا noun بودن صورت میگیرد و در فارسی بن فعل‌ها را باز می‌گرداند. stemming  کردن متون فارسی موجب تغییر مفهوم کلمات می‌شد.

کلاس tokenizer در hazm قابلیت جایگزین سازی برخی دیگر از علائم مانند ایموجی‌ها ، هشتگ‌، لینک و … را نیز دارد که اگر در متنی باشد با ". " جایگزین شده و در تابع prepare\_text به space تغییر یافته است.

در نهایت داده ی پیش پردازش شده به صورت یک فایل csv در محل پروژه جهت استفاده در سایر بخشها ذخیر شد.

برای شمارش تعداد کلمات پرتکرار نیز از تابع FreqDist ( کتابخانه nltk) در هر دو نوع متن استفاده شده است. این تابع تکرار token ها را شمارش می‌کند.

برای پیاده سازی مراحل پیش پردازش ذکر شده از این [لینک](https://agailloty.rbind.io/en/project/nlp_clean-text/) برای متون کمک گرفته‌ شده است.

**بخش نمایه سازی**

دراین بخش برای انجام مراحل مختلف توابع مختلفی برای اجرای این دستورات به ازای هر منبع ویکی و تد تاک در دوبخش wiki و Eng نوشته شد. توابع فراخوانی شده مشابه هم هستند ولی توابع انگلیسی با پیشوند Eng و فارسی با پیشوند wiki در فایل Indexing.py وجود دارند.

برای ساخت posting list از داده ساختار defaultdict استفاده شد و به ازای هرکلمه مستند هایی که حاوی این کلمه هستند و به ازای هر مستند ایندکس هایی که این کلمه در آن ها قرار گرفته ذخیره شد. eng\_create\_index()

پس از این مرحله این قابلیت وجود دارد که لیست ساخته شده در فایلی نوشته شده و سپس بازیابی شود که برای عملی کردن این بخش توابع مختلفی نیز نوشته شده است: eng\_load\_index() و eng\_save\_index()

همچنین می توان با وارد کردن یک کلمه لیست مربوطه را بازیابی کرد و نمایش داد: eng\_get\_index()

امکان حذف یک مستند از لیست ها و اضافه کردن مستند جدید نیز وجود دارد: eng\_delete\_index() و eng\_add\_index()

همچنین در این بخش بایگرام های هرواژه نیز ذخیره شده و امکان نوشتن آنها در فایل و سپس بازیابی نیز وجود دارد. برای این کار نیز از داده ساختار defaultdict استفاده کردیم و به ازای هر بایگرام لیستی از لغاتی که حاوی آن هستند ذخیره می گردد: eng\_create\_bigram() و eng\_save\_bigram() و eng\_load\_bigram()

همچنین می توان به ازای بایگرام خاصی لغات حاوی این ترکیب را بدست آورد: eng\_get\_bigram()

و درنهایت امکان حذف و اضافه نیز وجود دارد: eng\_add\_bigram() و eng\_delete\_bigram()

تمامی موارد فوق مثالی از توابع برای داده انگلیسی بودند به طور مشابه برای داده های ویکی نیز وجود دارد.

نکته حائز اهمیت این می باشد که تمامی موارد فوق برای title و text (بدنه) دربخش های مختلف ذخیره شده تا قابلیت بازیابی جداگانه فراهم باشد.

**بخش فشرده‌ سازی**

در حوزه بازیابی اطلاعات به‌طور معمول با داده‌های زیادی روبرو هستیم که بخشی از این داده‌ها در حافظه کامپیوتر قرار می‌گیرد که با توجه به محدودیت حافظه‌های کامپیوتر, تنها قادر هستیم بخش کوچکی از داده‌ها را در آن قرار دهیم. فشرده‌ سازی یکی از روش‌هایی است که بخش اعظمی از این معضل را حل‌ می‌کند و قادر می‌سازد بتوانیم با داده‌های بیشتری سروکار داشته باشیم بدون داشتن نگرانی بابت محدودیت حافظه یا دیسک . ازین رو در این قسمت از دو نوع روش فشرده سازی به نام‌های variable byte و gamma code استفاده‌ شده است که می‌توان نحوه‌ی پیاده سازی آنها را در فایل compress.py, به ترتیب در تابع‌های variable\_byte\_encoder و gamma\_code\_encoder یافت. هردو روش به‌جای اعداد دهدهی, با بیت‌ها سروکار دارند, بنابراین پس از اندکی تغییر در نمایه‌های ساخته‌شده در قسمت‌های قبل و تبدیل به نمایه‌ فاصله‌ای (لطفا به تابعcreate\_distance\_index توجه شود), تمامی اعداد موجود در نمایه با استفاده از کتابخانه bitarray به تعدادی از بیت‌های متناظر تبدیل می‌شوند (که این‌ تبدیل نیز در هردو تابع های encoder انجام می‌شود). درنهایت در تابع memory\_usage, می‌توان میزان بهینه‌کردن استفاده از حافظه را در دو قالب مشاهده می‌کرد:

۱. بدست‌آوردن حافظه مصرفی index ها به طور مستقیم با استفاده از متد ()getsizeof, که هردو چاپ اول این تابع مربوط به این روش ‌می‌باشد.

۲. بدست‌آوردن حافظه مصرفی مربوط به هر term با استفاده از ()getsizeof و حساب کردن مجموع آنها, که دو چاپ دوم این‌تابع مربوط به این روش می‌باشد.

اما علت استفاده از روش دوم چیست؟ در روش اول, با توجه به [این لینک](https://stackoverflow.com/questions/6579757/memory-usage-of-dictionary-in-python/37687855), گویا دیکشنری در پایتون خود یک روش فشرده‌ سازی و براساس تعداد کلید‌های موجود می‌زند (گزاره آخر براساس شواهد خود تیم ماست), از طرفی طبق توضیحات [این لینک](https://stackoverflow.com/questions/26862437/bytearray-is-not-json-serializable), کتابخانه bitarray متاسفاته serializable نیست و در ابتدا بایستی به رشته های کاراکتری تبدیل شود, در نتیجه در روش اول نتایج نا‌خواستانه‌ای را مشاهده کردیم و برای بهتر نشان‌دادن عملکرد توابع پیاده‌سازی شده, از روش دوم اسنفاده شده است.

همچنین با توجه به توضیحات داک پروژه, تنها تابع ناهمفشرده‌سازی برای روش variable\_byte در تابع variable\_byte\_decoder پیاده‌سازی شده است.

**بخش اصلاح پرسمان**‬‬

همانطور که می‌دانید, سیستم‌های بازیابی اطلاعات عملکردشان براساس پرسمان‌هایی که کاربران می‌دهند بررسی می‌شوند, از طرفی وجود خطاهایی بین این پرسمان‌ها انکارناپذیر است و سیستم‌های بازیابی بایستی توانایی تشخیص و اصلاح خطا‌ها تا حداکثر ممکن داشته باشند. ازین رو در فایل spelling\_correction.py بااستفاده از متدها jaccard\_distance و مارتیس لونشتاین (که با توجه به توضیحات داک پروژه تنها به پیاده‌سازی فاصله بین دو کلمه بسنده کرده‌ایم) به دنبال پیاده‌سازی این ویژگی بودیم. با توجه به توضیحات [این لینک](https://www.geeksforgeeks.org/spelling-correction-using-k-gram-overlap/) و [این لینک](https://python.gotrained.com/nltk-edit-distance-jaccard-distance/#Jaccard_Distance), دو نوع تابع jaccard\_distance پیاده‌سازی شده است که نام‌های آنها به ترتیب jaccard\_dist\_type1 و jaccard\_dist\_type2 می‌باشد. این روش نمایه های bigram ساخته‌شده در قسمت‌های قبلی را استفاده می‌کند.

**بخش جستجو و بازیابی**

قسمت اول)دراین بخش برای سرچ به روش ltc-lnc که در فایل retrieval.py قرار دارد ابتدا توابعی برای ساخت جدول tf برای هریک از title و text نوشته شده : create\_title\_tf\_table() و create\_text\_tf\_table()

سپس مقدار idf هر مستند محاسبه می شود: create\_title\_idf() و create\_text\_idf()

و درنهایت مقدار بردار وزن هر مستند به همراه نرمال سازی انجام می گیرد calc\_title\_weights() و calc\_text\_weights()

پس از این مراحل باید خود کوئری ارسال شده توسط کاربر بررسی شود برای این کار بردار وزن کوئری نیز محاسبه می شود و میزان شباهت آن برای مستند های مختلف به ازای title و text محاسبه می شود که با استفاده از تابع ltc\_lnc() استفاده می شود

پس از این مرحله باید جمع امتیاز های هر مستند که حاصل امتیاز های بخش title و text می باشد محاسبه شود و 10 مستند برتر به

همراه امتیاز به دست آورده به کاربر ارسال بشود

قسمت دوم ) در ابتدای این قسمت از روش proximity search برای جست‌و جوی دو کلمه بین فاصله داده شده استفاده شده‌است . کوئری به صورت دو کلمه و عدد فاصله در بین آن وارد شده و سپس کوئری پردازش شده و کلمات آن جدا میشوند.  
در این نوع سرچ از مکانی که هر کلمه در هر مستند دارد استفاده می‌کنیم ، به این صورت که متن پیمایش شده و اگر کلمه داده شده در آن موجود باشد نمایه‌ی آن را نگه می‌دارد . آرایه‌ی position\_occurance برای هر کلمه نشان میدهد در جایگاه I ام تا به حال چند بار کلمه‌ی خواسته شده تکرار شده است .

هم چنین این روش با تابع CalculateOccurencesInRange تعداد تکرارهای هر دوکلمه را با در نظر گرفتن یکی قبل از دیگری محاسبه کرده و در نهایت شماره‌ی سندهایی که پرسمان در آنها با فاصله خواسته شده موجود است را بر میگرداند.

سپس از بین این ایدیها از معیارقسمت یک برای امتیازدهی استفاده کردیم.