حسن اردشیر ۶۱۰۳۹۸۲۰۲

در برنامه مذکور یک کلاس IRSystem تعبیه شده است که در آن document ها تنظیم می شوند و سپس inverted index ها برای کوئری های Boolean و positional index ها برای کوئری های proximity ساخته می شوند. برای تجزیه document نیز، در ابتدا preprocess روی آنها انجام می شود. Preprocess حالت های مختلفی دارد و از لایبری nltk استفاده شده است. در ابتدا ورودی را توکنایز می کند و سپس punctuation ها را از آنها حذف می کند و آنها را تبدیل به حروف کوچک می کند. حال بر حسب پارامتر های تنظیم شده دو کار زیر را انجام می دهد:

۱- اگر rm_stop فعال باشد، حروف stop words حذف می شوند مانند to یا is.
۲- اگر do_stem فعال شده باشد، stemming بر روی آنها اتفاق می افتد تا کلمات در یک کلاسه و از یک ریشه به احتمال خوبی با یکدیگر شناخته شوند.

در بخش build_inverted_index، بر روی document ها حرکت می کنیم و ابتدا بر روی آن document، پیش پردازش گفته شده را انجام می دهیم و سپس به ازای هر term آن، شماره این document را ذخیره می کنیم (posting list)

حال برای هر Boolean query ابتدا بررسی می کنیم که چه نوع سوالی از ما پرسیده شده است:

۱- برای مدل and، ابتدا دو ترم اول و سوم را پیش پردازش می کنیم تا مانند ترم های document ها شود، سپس با با الگوریتم گفته شده در کلاس intersect sorted، اشتراک دو مجموعه inverted index متناظر با آنها را گرفته و خروجی می دهیم. لازم به ذکر است که برای افزایش سرچ، اولویت را با ارایه ای قرار می دهیم که اندازه اش کم تر است (فرکانس کم تر) چرا که این ارائه محدودیت بیشتری در محاسبه اشتراک خواهد داشت.
۲- برای مدل or، مانند and اقدام می کنیم اما به جای اشتراک، اجتماع آنها را می گیریم که چون طبق الگوریتم استفاده شده در ساخت inverted index، می دانیم این دو مجموعه مرتب هستند پس از الگوریتم merge استفاده می کنیم.
۳- در حالت not، ترم دوم را پیش پردازش می کنیم و چون inverted index آن sort است، پس به سادگی document هایی که این term را ندارند را محاسبه می کنیم.

نکته حائز اهمیت این است که پارامتر rm_stop و do_stem را چه بگذاریم؟! اگر rm_stop را فعال کنیم در این صورت اگر یکی از term های مورد سوال یک stop words باشد، در این صورت کوئری ارور می گیرد

چرا که پیش پردازش آن خالی می شود. مانند "example or is". پس بهتر است rm_stop فعال نباشد مگر اینکه تضمین شود که ترم های سوال stop words نیستند که با این تضمین طبیعتاً سرعت سرچ بیشتر می شود چون در ساخت inverted index، تعداد ترم کمتری وجود دارد و اعداد به طور کل کوچک تر هستند.

do_stem نیز بهتر است فعال باشد که ترم های از یک ریشه یکی تشخیص داده شوند تا تعداد ترم ها هم کمتر شود و سرعت سرچ بیشتر شود البته که دقت مسئله کاهش می یابد و بستگی به دقت مورد نیاز دارد.

برای سوالات proximity، باید positional index بسازیم چرا که موقعیت ترم ها اهمیت پیدا می کند پس برای هر document پیش پردازش انجام می دهیم و به ازای هر ترم آن، آن document و موقعیت مکانی در آن document را ذخیره می کنیم.

حال به ازای هر کوئری proximity، ابتدا ترم های اول و سوم را برداشت کرده و پیش پردازش می کنیم. از ترم دوم فاصله ماکسیمم قابل قبول را استخراج می کنیم. حال اشتراک document های ترم های اول و دوم را محاسبه می کنیم. (چون positional index هم مرتب است پس از همان الگوریتم intersect sorted استفاده می کنیم.)

حال به ازای هر document مشترک، الگوریتم positional intersect را برای موقعیت های رخداد این دو ترم در این document پیاده می کنیم. اگر موقعیت رخداد این دو کلمه شرط فاصله را رعایت کرده باشد، شماره این document را خروجی می دهیم. در الگوریتم positional intersect، بر روی موقعیت های ترم اول حرکت می کنیم. و به ازای هر موقعیت بررسی می کنیم که موقعیت رخداد ترم دوم نزدیک به آن داریم یا خیر. بدین منظور طبیعتا موقعیت رخداد ترم دوم که قبلا بررسی شده است، در قبل از این موقعیت قرار دارد پس اگر فاصله آن بیش تر از kprox (فاصله خواسته شده) موقعیت ترم دوم بعدی را در نظر می گیریم ولی اگر باز هم این فاصله خواسته شده نشود، موقعیت را دوباره بعدی قرار می دهیم. اگر موقعیت ترم دوم از موقعیت فعلی بیشتر شد یا فاصله ان کم تر مساوی با kprox شد، تمام موقعیت دوم هایی که با این موقعیت (ترم اول) شرط مذکور را دارد را ذخیره می کنیم. لازم به ذکر است که این الگوریتم به طور کل بهتر از صرفا دو حلقه تو در تو بر روی تمام موقعیت های رخداد این دو ترم عمل می کند و از مرتب بودن این دو موقعیت استفاده کردیم.

حال به طور مشابه نکته حائز اهمیت این است که پارامتر rm_stop و do_stem را چه بگذاریم؟! طبیعتا بهتر است که هیچ کلمه ای در document حذف نشود چرا که فاصله ها را تغییر می دهد و در نتیجه ممکن است نتیجه برای کوئری خاص اشتباه شود. پس بهتر است rm_stop فعال نباشد اما فعال بودن do_stem ایراد

چندانی ندارد. برای مثال با فعال بودن rm_stop، نتیجه برای کوئری "example near/1 test" اشتباها خروجی دارد چرا که "to" حذف می شود و فاصله دو ترم test و example در document سوم یکی کم تر حساب می شود.

حال تابع prep_correct_col را بررسی می کنیم. هدف این تابع استخراج داده هایی می باشد که در docs قرار داده شده است. برای هر doc، پیش پردازش انجام می شود و term های آنها در یک مجموعه all_cterms ذخیره می شود تا در spell checking از آن استفاده کنیم. مجموعه all_kterms نیز ساخته می شود که key آنا برابر عددی یکتا برای هر term متفاوت می باشد.

پس در تابع correct_spelling، query دریافت می شود و پیش پردازش می شود. طبق تعریف، در این پیش پردازش ها نیازی به فعال بودن do_stem و rm_stop نمی باشد. حال به وسیله روش labenshtein distance، به ازای هر term در کوئری، نزدیک ترین term در docs در نظر گرفته می شود این روش دقت خوبی دارد و طبق بررسی های انجام شده با اینکه تعداد term های دیتاست کم می باشد، نتیجه قابل قبولی دارد.

حال برای کوئری های wildcard، از روش kgram استفاده می کنیم. بدین منظور، ابتدا kgram index بر اساس پارامتر k (دیفالت ۲) ساخته می شود. بدین منظور دیکشنری kgram_index در نظر گرفته می شود که کلید آن برابر k gram حرفی است و مقدار متناظر هر gram برابر شماره term هایی است که این gram را دارا می باشد. در حقیقت به ازای هر term که در all_kterms قبلا در نظر گرفته شده است، gram های آن محاسبه می شود و kgram_index آن پر می شود. لازم به ذکر است که چون شماره term ها صعودی است، پس kgram_index برای هر gram نیز صعودی (مرتب) خواهد بود.

حال به ازای هر wildcard کوئری، ابتدا آن را lower می کنیم و $ را به ابتدا و انتهای آن اضافه می کنیم تا مشخص کنیم هر کوئری ابتدا و انتهای آن چیست و بر اساس آن، gram های این کوئری محاسبه می شود. لازم به ذکر است که زیر رشته ای که * دارد نباید به عنوان gram در نظر گرفته شود. حال به ازای هر gram، kgram_index آن در نظر گرفته می شود و اشتراک آنها را به عنوان جواب در نظر می گیریم. در حقیقت این جواب برابر شماره term هایی است که این gram ها را دارند (نه لزوما به ترتیب درست) پس با تبدیل این شماره به term متناظر آنها، جواب مسئله به دست می آید. اما همانطور که گفتیم چون در این الگوریتم تا الان به ترتیب درست gram و حتی تعداد gram در term جواب دقت نشده است، لزوما همه جواب های به دست

آمده درست نیستند و باید false positive ها را پاک کنیم. برای مثال nobody و nobobody به عنوان جواب *nobod در نظر گرفته می شود ! (ایراد T)

در نتیجه در بخش post filtering، به ازای هر جواب بررسی می کنیم که آیا می تواند جوابی برای کوئری باشد یا خیر! بدین منظور اولاً در سوال فرض شده است که هر کوئری حداکثر یک ستاره دارد. پس ابتدا بررسی می کنیم که ایا جواب پیدا شده تا اولین ستاره کوئری یکی است یا خیر! (ممکن است ستاره ای نباشد) چون تا ستاره باید یکی باشد! حال بررسی می کنیم که ایا از انتها تا ستاره اخر (دوم یا اول یا هیچ کدام) یکی می باشند یا خیر! سپس روشن است که اگر دو ستاره در کوئری باشد، زیر رشته بین دو ستاره در کوئری باید در زیر بررسی نشده جواب باشد. در نتیجه روشن است که تمام رشته جواب بررسی شده است و می توانیم به طور دقیق بیان کنیم که ایا می تواند جوابی درست قلمداد شود یا خیر!

همچنین بررسی کردیم که ایا برای دیگر k ها نیز به درستی کار می کند یا خیر و دیدیم که به ازای k=3 حتی بدون post filter، دیگر ایراد T را نداریم.