وزارت علوم، تحقیقات، و فناوری

پژوهشگاه علوم و فناوری اطلاعات ایران

(ایرانداک)

طرح پژوهشی

بررسی موتور جستجوی گنج به منظور بهبود بازیابی اطلاعات در زبان فارسی

مجری

مرضیه زرین بال ماسوله

شهریور ۱۳۹۸


بنام خداوند جان و خرد

بررسی موتور جستجوی گنج به منظور بهبود بازیابی اطلاعات در زبان فارسی

مدیر طرح پژوهشی: مرضیه زرین بال ماسوله، پژوهشگاه علوم و فناوری اطلاعات ایران

نشانی: تهران، خیابان انقلاب، چهارراه فلسطین، شمارهٔ ۱۰۹۰،

صندوق پستی:۱۳۱۵۷۷۳۳۱۴، **تلفن:** ۰۲۱۶۶۴۹۴۹۸۰، **دورنگار:** ۰۲۱۶۶۴۹۴۹۸۰

وب‌گاه: www.irandoc.ac.ir

رایانامه: zarinbal@irandoc.ac.ir

به نام خدا

آغاز نیم‌قرن دوم خدمات ارزشمند ایرانداک به علم، فناوری، و نوآوری کرامی باد

۱۳۹۸ – ۱۳۴۷

گواهی

انجام طرح پژوهشی:

- **عنوان:** بررسی موتور جستجوی گنج به منظور بهبود بازیابی اطلاعات در زبان فارسی
- **مجری:** دکتر مرضیه زرین‌بال ماسوله
- **همکار(ان):** -
- **شمارهٔ قرارداد:** ۳۳/۶۷۵۰
- **تاریخ قرارداد:** ۱۳۹۷/۶/۱۲
- **تاریخ شروع:** ۱۳۹۷/۶/۱۲
- **تاریخ پایان:** ۱۳۹۸/۶/۶
- **ناظر:** دکتر آزاده محبی

در پژوهشگاه علوم و فناوری اطلاعات ایران گواهی می‌شود. گزارش پایانی این طرح، بر پایهٔ داوری در نشست ۳۴۴ (تاریخ ۱۳۹۸/۷/۲) شورای پژوهشی پژوهشگاه به تصویب رسیده است. لازم می‌دانم مراتب قدردانی خود را به مجری و ناظر این طرح تقدیم دارم.

با آرزوی بهروزی
پرویز شهریاری
معاون پژوهش و آموزش

چکیده

بازیابی اطلاعات به معنای یافتن محتوایی با طبیعت غیرساخت‌یافته (معمولاً متن) از مجموعه‌هایی به‌منظور برآورده ساختن نیاز(های) اطلاعاتی کاربران است. با بهره‌گیری از روش‌های بازیابی اطلاعات موجود در ادبیات پلتفرم‌های متن‌باز متعددی طراحی شده‌اند. کتابخانه آپاچی لوسن و موتور جستجوی آپاچی سولر از این نوع پلتفرم‌ها هستند. سامانه گنج نیز به منظور جستجو در اسناد و مدارک ذخیره شده در پایگاه داده خود، از این کتابخانه و موتور جستجو به عنوان هسته اصلی فرآیند بازیابی اطلاعات استفاده می‌کند. علی‌رغم کاربردهای گسترده، سولر دارای کاستی‌هایی بوده، تاکنون نیز هیچ‌گونه پژوهشی در زمینه شناسایی و واکاوی این کتابخانه و موتور جستجو در پژوهشگاه انجام نشده و تنها عملکرد رابط کاربری آن و مشکلات موجود بررسی شده است. لذا علی‌رغم ضرورت بررسی کارایی رابط کاربری گنج، تازمانی که هسته اصلی آن (خصوصا با تمرکز بر زبان فارسی) بررسی و اصلاح نشود نمی‌توان انتظار داشت عملکرد سامانه مطلوب باشد. لذا ضروری است تا با بررسی محدودیت‌ها و مشکلات موجود در روش‌های مورد استفاده در موتور جستجوی آپاچی سولر (از منظر بازیابی اطلاعات) دیدگاهی کلان نسبت به آن در پژوهشگاه ایجاد شده تا از این طریق بتوان پیشنهادهایی به منظور ارتقاء عملکرد هسته اصلی فرآیند بازیابی اطلاعات سامانه گنج ارائه داد. براین اساس، طرح پژوهشی با عنوان «**بررسی موتور جستجوی گنج به منظور بهبود بازیابی اطلاعات در زبان فارسی**» پیشنهاد شد. در این طرح دانش فنی درمورد این موتور جستجو و روش‌های مورد استفاده آن کسب شده و راهکارهایی به منظور ارتقاء نتایج بازیابی در زبان فارسی و در سامانه گنج ارائه شد. راهکارهای پیشنهاد شده در قالب دو دسته راهکارهای نیازمند پژوهش و راهکارهای فنی و عملیاتی دسته‌بندی شدند.

کلید واژه‌ها: بازیابی اطلاعات؛ آپاچی لوسن؛ آپاچی سولر؛ سامانه گنج

فهرست مطالب

عنوان **شماره صفحه**

چکیده ..... ب

فهرست مطالب ..... ب

فهرست جدول‌ها ..... ج

فهرست شکل‌ها ..... خ

فصل اول: بیان مسئله ۲

۱-۱ بیان مسئله ۲

۱-۲ هدف‌ها ۴

۱-۳ پرسش‌ها ۴

۱-۴ روش پژوهش ۵

فصل دوم: کتابخانه لوسن و موتور جستجوی سولر ۷

۲-۱ مقدمه ۷

۲-۲ بازیابی اطلاعات ۸

۲-۳ کتابخانه لوسن ۱۰

۲-۴ موتور جستجوی سولر ۱۷

فصل سوم: مباحث پایه‌ای در موتور جستجوی سولر ۲۱

۳-۱ مقدمه ۲۱

۳-۲ مفاهیم کلیدی در سولر ۲۳

۳-۲-۱- تنظیمات در سولر ۲۳

۳-۲-۲- سند ۲۹

۳-۲-۳- تطابق فازی ۳۰

۳-۲-۴- ارتباط ۳۲

۵-۲-۳- معیارهای سنجش بازیابی اطلاعات ۳۵
۳-۳ زیرسیستم مدیریت اسناد ۳۶
۱-۳-۳- تنظیمات ورود و بروزرسانی اسناد ۳۷
۳-۴ زیرسیستم تحلیل متن ۳۹
۱-۴-۳- ریشه‌یابی ۴۲
۲-۴-۳- زبان‌های پیش‌فرض در سولر ۴۶
۳-۵ زیرسیستم پردازش پرس‌وجو ۴۸
۱-۵-۳- مولفه Query ۵۱
۲-۵-۳- مولفه Facet ۵۸
۳-۵-۳- مولفه More Like This ۵۸
۴-۵-۳- مولفه Highlight ۵۹
۵-۵-۳- مولفه Last-Component ۶۲
۶-۵-۳- مولفه Response Writer ۶۵
۳-۶ جمع‌بندی ۶۶

فصل چهارم: مباحث پیشرفته در موتور جستجوی سولر ۷۰
۴-۱ مقدمه ۷۰
۴-۲ استفاده از داده‌های منابع بیرونی ۷۰
۴-۳ عملیات جستجوی پیچیده ۷۲
۴-۴ ارتقاء ارتباط سند با عبارت پرس‌وجو ۷۳
۴-۵ شخصی‌سازی جستجو ۷۶
۴-۶ جمع‌بندی ۸۱

فصل پنجم: موتور جستجوی گنج ۸۳
۵-۱ مقدمه ۸۳
۵-۲ سند Solrconfig سامانه گنج ۸۳
۵-۳ سند Schema سامانه گنج ۸۳
۵-۴ جمع‌بندی ۸۴

فصل ششم: راهکارها و روش‌های بهبود ۸۶
۶-۱ مقدمه ۸۶
۶-۲ راهکارهای بهبود بازیابی اطلاعات در سامانه گنج ۸۶

۱-۲-۶- راهکارهای نیازمند پژوهش ۸۹
۲-۲-۶- راهکارهای فنی و عملیاتی ۹۱

جمع‌بندی و نتیجه‌گیری ۹٤

ضمائم ۹۹
۱-۷ توابع ۹۹
۱-۱-۷- توابع ریاضی ۹۹
۲-۱-۷- توابع فاصله ۱۰۰
۳-۱-۷- توابع منطق بولی ۱۰۱
۲-۷ عوامل محاسبه ارتباط ۱۰۱
۳-۷ کلاس‌های محاسبه ارتباط ۱۰۲
۴-۷ خصوصیات Update Handler ۱۰۳
۵-۷ پارامترهای مولفه Highlight ۱۰۳
۶-۷ پارامترهای مولفه Spellcheck ۱۰۴

منابع ۱۰۵

ABSATRACT 109

فهرست جدول‌ها

جدول ۲-۱- فرمت‌های ورودی قابلیت پردازش در لوسن (McCandless, Hatcher, and Gospodnetic 2010) ۱۱

جدول ۳-۱- جستجوی Wildcard: عبارت پرس‌وجوی کاربر و عبارتی که سولر جستجو می‌کند..... ۳۰

جدول ۳-۲- جستجوی نزدیکی: عبارت پرس‌وجوی کاربر و توضیحات آن ۳۱

جدول ۳-۳- جستجوی بازه‌ای: عبارت پرس‌وجوی کاربر و عبارتی که سولر برای جستجو استفاده می‌کند..... ۳۲

جدول ۳-۴- درنظرگیری وزن برای فیلدهای مختلف ۳۴

جدول ۳-۵- متن اولیه مثال شکل ۳-۷، تغییرات اعمال شده و ابزارهای سولر برای تغییرات ۴۰

جدول ۳-۶- عملکرد ریشه‌یاب‌های KStemFilter،PorterStemFilter و EnglishMinimalStemFilter ۴۳

جدول ۳-۷- پارامتر نوع نتایج خروجی ۶۵

جدول ۴-۱- توابع محاسبه میزان ارتباط سند و عبارت پرس‌وجو ۷۲

جدول ۷-۱- توابع ریاضی در سولر ۹۹

جدول ۷-۲- توابع فاصله در سولر ۱۰۰

جدول ۷-۳- توابع منطق بولی در سولر ۱۰۱

جدول ۷-۴- عوامل محاسبه ارتباط در سولر ۱۰۱

جدول ۷-۵- کلاسهای محاسبه ارتباط در سولر ۱۰۲

جدول ۷-۶- خصوصیات Update Handler ۱۰۳

جدول ۷-۷- برخی از پارامترهای مولفه highlight ۱۰۳
جدول ۷-۸- برخی از پارامترهای مولفه Spellcheck ۱۰۴

فهرست شکل‌ها

شکل ۲-۱- نحوه ارتباط بین آپاچی لوسن و آپاچی سولر (Grainger and Potter 2014) ۱۰

شکل ۲-۲- کلاس‌های مورد استفاده در لوسن برای نمایه‌سازی متن (McCandless, Hatcher, and Gospodnetic 2010) ۱۱

شکل ۲-۳- منوی مرور کلی در Luke (McCandless, Hatcher, and Gospodnetic 2010) ۱۳

شکل ۲-۴- منوی اسناد در Luke (McCandless, Hatcher, and Gospodnetic 2010) ... ۱۴

شکل ۲-۵- منوی جستجو در Luke (McCandless, Hatcher, and Gospodnetic 2010) ۱۵

شکل ۲-۶- توضیحات مربوط به نحوه محاسبه وزن در Luke (McCandless, Hatcher, and Gospodnetic 2010) ۱۵

شکل ۲-۷- منوی موارد افزوده در Luke (McCandless, Hatcher, and Gospodnetic 2010) ۱۶

شکل ۳-۱- اجزای سولر ۲۲

شکل ۳-۲- نمونه‌ای از سند Solrconfig ۲۴

شکل ۳-۳- نتایج جستجوی عبارت «text analysis» برای سطح ریزدانگی هر کتاب و هر فصل هر کتاب ۲۷

شکل ۳-۴- نمودار کلاس انواع فیلدها در Schema ۲۹

شکل ۳-۵- شباهت کسینوسی بین بردارهای واژگان ۳۳

شکل ۳-۶- زیرسیستم مدیریت اسناد: فرآیند نمایه‌سازی و ذخیره‌سازی اسناد ۳۷
شکل ۳-۷- مثال تحلیل متن ۴۰
شکل ۳-۸- مثالی از StandardTokenizer ۴۱
شکل ۳-۹- اعمال دو فیلتر StopFilterFactory و LowerCaseFilterFactory در مثال شکل ۳-۸ ..... ۴۱
شکل ۳-۱۰- نمونه‌هایی از نتایج متفاوت بکارگیری روش‌های ریشه‌یابی و بُن‌واژه‌سازی ۴۵
شکل ۳-۱۱- درنظرگیری فیلدهای جداگانه در تحلیل‌های چند زبانی ۴۶
شکل ۳-۱۲- دسته‌بندی اسناد براساس زبان در تحلیل‌های چند زبانی ۴۷
شکل ۳-۱۳- تشخیص خودکار زبان سند در تحلیل‌های چند زبانی ۴۷
شکل ۳-۱۴- نمای کلی مدیریت درخواست جستجو ۴۸
شکل ۳-۱۵- کلاس‌های مدیریت درخواست در سولر ۴۹
شکل ۳-۱۶- نمای درختی کلاس‌های SearchHandler ۴۹
شکل ۳-۱۷- فرآیند کلی SearchHandler ۵۰
شکل ۳-۱۸- نمونه‌ای از کاربرد مولفه‌های بخش ۳ شکل ۳-۱۷ ..... ۵۱
شکل ۳-۱۹- گام‌های پردازش q و fq ۵۳
شکل ۳-۲۰- نمایش نمادین عبارت پرس‌و جو در eDisMax در حالت تفاوت وزن فیلدها ۵۵
شکل ۳-۲۱- نمونه‌ای از نتایج جستجو پس از فعال کردن مولفه hithighlight ۵۹
شکل ۳-۲۲- فرآیند نمایش snippet و پررنگ کردن بخشهایی از سند ۶۰
شکل ۳-۲۳- فرآیند پردازش اسناد و پرس‌وجو در سولر ۶۷
شکل ۳-۲۴- فرآیند پردازش اسناد و پرسوجو در سولر در قالب مثال ۶۸
شکل ۴-۱- سیستم پیشنهاد دهنده آمازون برای کالاهای مشابه ۸۰
شکل ۴-۲- سیستم پیشنهاد دهنده آمازون برای کالاهای مکمل ۸۰
شکل ۴-۳- جدول ارتباط بین کاربران و اسناد ۸۱

فصل اول

بیان مسئله

۱-۱ بیان مسئله

بازیابی اطلاعات شامل سازماندهی، ذخیره سازی، بازنمایی (نمایش) و دسترسی به اقلام اطلاعاتی است. به عبارت دیگر به فناوری و دانش پیچیده جستجو و استخراج اطلاعات، داده‌ها و فراداده‌ها در انواع گوناگون منابع اطلاعاتی مانند بانک اسناد، مجموعه‌های تصاویر، وب، و...، و در نهایت نمایش آنها برای کاربرِ آغاز کننده عملیات جستجو، بازیابی اطلاعات گفته می شود. سیستم‌های بازیابی اطلاعات جهت خودکارسازی فرآیند بازیابی بکار می‌روند. مراکز اطلاعاتی بسیاری همچون دانشگاه‌ها و کتابخانه‌های عمومی از سیستم‌های بازیابی اطلاعات بمنظور فراهم آوردن امکان دستیابی به کتب، روزنامه‌ها، و اسناد مختلف استفاده می‌کنند. سامانه گنج نیز به منظور جستجو در اسناد[۱] ذخیره شده در پایگاه داده خود، از کتابخانه آپاچی لوسن[۲] و موتور جستجوی آپاچی سولر[۳] استفاده می‌کند. این کتابخانه، کتابخانه‌ای متن‌باز[۴] برای راه‌اندازی موتورهای جستجو برپایه متن بوده و با زبان جاوا نوشته شده‌است. سولر نیز موتور جستجوی این کتابخانه بوده و مانند لوسن با زبان جاوا پیاده‌سازی شده‌است. به بیانی دیگر هسته اصلی سامانه گنج سولر و لوسن است

---

[۱] Document

[۲] Apache Lucene

[۳] Apache Solr

[۴] Open source

که هردو متن‌باز بوده و با زبان جاوا پیاده‌سازی شده‌اند. برای تعامل با این هسته نیز رابط‌های کاربری متعددی توسعه داده شده‌اند، رابط کاربر سامانه گنج[۱] پژوهشگاه علوم و فناوری اطلاعات ایران (ایرانداک) نمونه‌ای از این موارد است.

لذا عملکرد سامانه گنج را می‌توان از دو منظر بررسی کرد؛ (۱) عملکرد رابط کاربری طراحی شده و تعامل آن با کاربران و (۲) عملکرد هسته اصلی سامانه. در مورد عملکرد رابط کاربری مطالعات بسیاری در پژوهشگاه صورت پذیرفته که از این میان می‌توان به (ارشادی و همکاران، ۱۳۹۴)، (فتاحی، ۱۳۹۶) و (زرین‌بال، ۱۳۹۷) اشاره نمود. اما درمورد عملکرد هسته سامانه تاکنون نیز هیچ‌گونه پژوهشی در زمینه شناسایی و واکاوی این کتابخانه و موتور جستجو در پژوهشگاه انجام نشده و تنها به استفاده از تنظیمات پیش‌فرض اکتفا شده است. این هسته علی‌رغم کاربردهای وسیع و گسترده، کاستی‌هایی نیز دارد؛ به عنوان نمونه، در سولر از رویکردهای بهینه‌سازی موتور جستجو استفاده نشده است و ارتباط بین اسناد در بازیابی‌ها درنظر گرفته نمی شود؛ در صورتی که داده‌ای بین چند مدرک و یا فیلد اطلاعاتی مشترک باشد، این داده باید برای هر مدرک تکرار شود؛ در سولر نمی‌توان به سادگی اطلاعات مربوط به یک مدرک را بروزرسانی نمود و باید از ابتدا فرآیند نمایه سازی را انجام داد و سولر برای جستجوهایی بهینه است که عبارات پرس‌وجو از تعداد کمی از کلمات تشکیل شده باشد. علاوه‌براین، روش‌های مورد استفاده روش‌های پایه‌ای بازیابی اطلاعات هستند درحالی‌که در طی سال‌های گذشته توسعه‌های بسیاری برای آنها ارائه شده است. با این وجود، متن باز بودن موتور جستجوی سولر امکان تغییر ماژول‌ها و روش‌های بکاررفته در آن را برای طراحان و پیاده‌سازان فراهم آورده است.

بدیهی است که اکتفا به استفاده از تنظیمات پیش‌فرض سولر در سامانه گنج و عدم توسعه آن، با توجه به نیازهای کاربران سامانه گنج و خصوصا زبان فارسی، سبب خواهد شد تا مشکلاتی در عملکرد رابط کاربری این سامانه ایجاد شده و در تعامل آن با کاربران مشکلاتی ایجاد شود، به برخی از این مشکلات در (ارشادی و همکاران، ۱۳۹۴) اشاره شده است. همچنین به منظور بهبود و ارتقاء عملکرد این سامانه ضروری است تا هسته اصلی بازیابی اطلاعات آن بهبود یابد. بنابراین بایستی کتابخانه آپاچی لوسن و موتور جستجوی آپاچی سولر و روش‌های مورد استفاده در برخی از مراحل و اجزای مختلف آن به صورت دقیق بررسی شده، مشکلات فعلی این روش‌ها در حوزه بازیابی اطلاعات و خصوصا در زبان فارسی استخراج شده و راهکارهای بهبود پیشنهاد گردد.

---

[۱] https://ganj-beta.irandoc.ac.ir/

براین اساس، انجام طرح پژوهشی حاضر با هدف بررسی هسته سامانه گنج و با عنوان «بررسی موتور جستجوی گنج به منظور بهبود بازیابی اطلاعات در زبان فارسی» پیشنهاد می‌شود. انتقال دانش فنی این کتابخانه و موتور جستجو و معرفی روش‌های مورد استفاده در هریک از ماژول‌های آن به پژوهشگاه و ارائه پیشنهادهایی به منظور بکارگیری روش‌های جایگزین معرفی شده در ادبیات در برخی از ماژول‌هابا تمرکز بر زبان فارسی از خروجی‌های این طرح خواهد بود.

بنابراین، بررسی رابط کاربری گنج و عملکرد آن در حیطه این طرح نیست. همچنین سطح شناسایی این کتابخانه و موتور جستجو محدود به نمودار مفهومی[1] سطح صفر و یک بوده و فعالیت هایی چون بررسی دقیق و تفسیر کدهای موجود، ارائه الگوریتم جدید، کدنویسی و پیاده‌سازی آن در حیطه این طرح قرار ندارند.

۱-۲ هدف‌ها

براساس موارد مطرح شده، اهداف اصلی طرح پژوهشی مذکور بشرح زیر است:

- □ کسب و انتقال دانش فنی کتابخانه آپاچی لوسن و موتور جستجوی آپاچی سولر برای ارتقاء بازیابی اطلاعات در گنج

جهت دستیابی به اهداف فوق اهداف فرعی طرح پژوهشی به شرح زیر تعیین می‌گردند:

- □ بررسی سطح صفر و یک ماژول‌ها و روش‌های بکاررفته در کتابخانه آپاچی لوسن و موتور جستجوی آپاچی سولر؛
- □ شناسایی و بررسی مشکلات و محدودیت‌های روش‌های بکاررفته درکتابخانه لوسن و موتور جستجوی سولر (با تمرکز بر زبان فارسی) برای بازیابی اطلاعات؛
- □ ارائه روش‌های پیشنهادی و یا جایگزین به منظور بهبود عملکرد بازیابی اطلاعات در زبان فارسی در موتور جستجوی سولر.

۱-۳ پرسش‌ها

این طرح به دنبال یافتن پاسخ به پرسش‌های زیر است:

---

[1] Context diagram

- □ ساختار کتابخانه آپاچی لوسن و موتور جستجوی آپاچی سولر چگونه است و از چه ماژول‌هایی تشکیل شده است؟
- □ چه روش‌ها و یا الگوریتم‌هایی در این ماژول‌ها بکار رفته‌اند؟
- □ این روش‌ها از چه مشکلات و یا محدودیت‌هایی رنج می‌برند؟
- □ به منظور بهبود نتایج بازیابی شده در زبان فارسی از چه روش‌ها و یا الگوریتم‌های دیگری می‌توان استفاده کرد؟

۱-۴ روش پژوهش

پژوهش حاضر در قالب دو مرحله اصلی زیر صورت پذیرفته است:

مرحله اول: شناسایی و بررسی کتابخانه آپاچی لوسن و موتور جستجوی آپاچی سولر

- □ شناسایی ماژول‌های آپاچی لوسن و آپاچی سولر
- □ بررسی روش‌های بکاررفته در هریک از ماژول‌ها

مرحله دوم: ارائه راهکارها و روش‌های بهبود

- □ شناسایی مشکلات فعلی سامانه گنج از منظر عملکرد هسته آن (ماژول‌های آپاچی لوسن و آپاچی سولر) خصوصا در زبان فارسی
- □ ارائه پیشنهادهایی به منظور بهبود عملکرد بازیابی اطلاعات در زبان فارسی (شامل پیشنهادهایی برای جایگزینی روش‌های موجود با روش‌های بهبود یافته معرفی شده در ادبیات با تمرکز بر زبان فارسی. این پیشنهادها شامل تفسیر کدهای موجود، ارائه الگوریتم جدید، کدنویسی و پیاده‌سازی نخواهند بود.)

فصل دوم

کتابخانه لوسن

و موتور جستجوی سولر

۲-۱ مقدمه

اطلاع‌یابی یکی از بنیادی‌ترین نیازهای بشـــر برای توســـعه و پیشـــرفت بوده و وجود منابع بزرگ اطلاعاتی و در راس آنها وب، اهمیت بازیابی اطلاعات را در ســطح گســترده افزایش داده اســت. آنچه امروزه اهمیت بسیار دارد طراحی و بکارگیری روش‌هایی است که به کاربران کمک می‌کنند اطلاعات مورد نیاز خود را در انبوهی از اطلاعات غیرساخت‌یافته بیابد. مفهوم بازیابی اطلاعات[۱] در چنین بافتی تعریف می‌شود (Manning, Ragahvan, and Schutze 2009).

در سال‌های اخیر سامانه‌های بسیاری با هدف بازیابی اطلاعات طراحی شده‌اند و بسیاری از آنان از پلتفرم‌های متن‌باز برای این منظور بهره می‌برند. یکی از پرکاربردتربن این پلتفرم‌ها، پلتفرم طراحی شـده توسـط آپاچی، کتابخانه لوسـن و موتور جسـتجوی سـولر، اسـت. سـامانه گنج پژوهشـگاه ایرانداک نیز به منظور مدیریت و جسـتجو در اسـناد ذخیره شـده در پایگاه داده خود از این پلتفرم استفاده می‌کند.

این فصل با هدف بررسی بیشتر موتور جستجوی سولر نگاشته شده است هرچند نیم نگاهی نیز به کتابخانه لوسن دارد.

---

[۱] Information retrieval

۲-۲ بازیابی اطلاعات

جمع‌آوری مدارک و مستندات موجود در پایگاه داده و نمایه‌سازی[۱] آنها دو گامی است که عموماً پیش از شـــروع فرآیند بازیابی اطلاعات انجام می‌شـــود. پس از ورود نیاز اطلاعاتی کاربر در قالب عبارت پرس‌وجو[۲]، عملیات متنی بر عبارت پرس‌وجوی اعمال شـــده و فرآیند بازیابی اســـناد آغاز می‌شود. در انتها و پس از رتبه‌بندی نیز نتایج بازیابی شده به کاربر نمایش داده می‌شود. مقاله بوش با عنوان «As we may Think» (Bush 1945) را می‌توان نقطه آغازین فعالیت‌های بازیابی اطلاعات دانست و پس از آن روش‌ها و مدل‌های متعددی ارائه شده است که از این میان می‌توان به موارد زیر اشاره نمود:

۱۹۶۰- مارون و کوهن: بازیابی اطلاعات احتمالی (Maron and Kuhns 1960)

۱۹۷۱- روچیو: بازخورد در بازیابی اطلاعات (Rocchio 1971)

۱۹۷۱ و ۱۹۷۵- ســـالتون و همکاران: مدل برداری در بازیابی اطلاعات (Salton 1971) و (Salton, Wong, and Yang 1975)

۱۹۷۶- رابرتسون و اسپارک-جونز: وزن‌دهی به عبارت پرس‌وجو (Robertson and Sparck-Jones 1976)

۱۹۸۰- بوک‌اشتین: مدل بولین فازی در بازیابی اطلاعات (Bookstein 1980)

۱۹۸۶- ون ریجزبرگر: منطق در بازیابی اطلاعات (Van Rijsbergen 1986)

۱۹۸۸- دومیاس و همکاران: تحلیل معنایی پنهان در بازیابی اطلاعات (Dumais et al. 1988)

۱۹۸۹- کوک: شبکه‌های عصبی در بازیابی اطلاعات (Kwok 1989)

۱۹۹۵- رابرتسون و همکاران: BM25 (Robertson et al. 1995) و (Robertson, Walker, and Hancock-Beaulieu 1995)

۱۹۹۵- چن: یادگیری ماشین در بازیابی اطلاعات (Chen 1995)

۱۹۹۶- ریبریو و مانتز: شبکه باور در بازیابی اطلاعات (Ribeiro and Muntz 1996)

۱۹۹۸- پونت و کرافت: مدل زبانی در بازیابی اطلاعات (Ponte and Croft 1998)

۱۹۹۸- اورد و دیکما: بازیابی اطلاعات بین زبانی (Oard and Diekema 1998)

۲۰۰۰- هیمســـترا: ترکیب فراوانی کلمه-فراوانی معکوس ســـند[۳] و مدل زبانی (Hiemstra 2000)

---

[۱] Indexing

[۲] Query

[۳] Term frequency-inverse document frequency (TF-IDF)

۲۰۰۱- براون و جونز: بازیابی اطلاعات بافت آگاه (Brown and Jones 2001)

۲۰۰۲- آماتیو ون ریجزبرگر: بکارگیری معیار Divergence from Randomness (DFR) در بازیابی اطلاعات (Amati and Van Rijsbergen 2002)

۲۰۰۳- بروزا و سانگ و زاراگوزا و همکاران: بکارگیری احتمال در مدل‌های زبانی (Bruza and Song 2003) و (Zaragoza, Hiemstra, and Tipping 2003)

۲۰۰۴- متزلر و کرافت: شبکه‌های استنباطی در بازیابی اطلاعات (Metzler and Croft 2004)

۲۰۰۵- فنگ و ژای: رویکردهای معنایی در بازیابی اطلاعات (Fang and Zhai 2005)

۲۰۰۸- ژای: مدل‌های زبانی احتمالی در بازیابی اطلاعات (Zhai 2008)

۲۰۱۱- الای و دمیستر: مدل‌های احتمالی در بازیابی اطلاعات (Aly and Demeester 2011)

۲۰۱۲- گراوسمن و فریدر: روش‌های هیورستیک در بازیابی اطلاعات (Grossman and Frieder 2012)

۲۰۱۳- هافمن و همکاران: رتبه‌بندی نتایج بر اساس یادگیری برخط (Hofmann, Whiteson, and de Rijke 2013)

۲۰۱۴- شن و همکاران: تحلیل معنایی پنهان و یادگیری عمیق در بازیابی اطلاعات (Shen et al. 2014)

۲۰۱۵- لیو و همکاران: دسته‌بندی معنایی با کمک یادگیری عمیق در بازیابی اطلاعات (Liu et al. 2015)

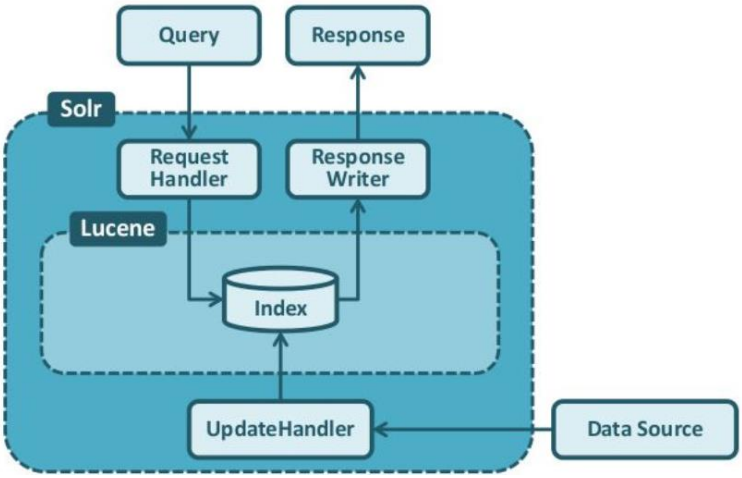
۲۰۱۶- آی و همکاران: مدل برداری پاراگراف در بازیابی اطلاعات (Ai et al. 2016)

۲۰۱۷- کلوچ و همکاران: رتبه‌دهی بصری به نتایج بازیابی اطلاعات (Klouche et al. 2017)

با بهره‌گیری از روش‌های ارائه شده، پلتفرم‌های بازیابی اطلاعات متعددی طراحی شده است و نکته اساسی در یک پلتفرم مناسب آن است که این پلتفرم برای جستجو باید بتواند نتایج مرتبط را در زمان کوتاهی بازیابی کند؛ اشتباهات تایپی کاربر را در عبارت پرس‌وجو اصلاح نماید؛ عبارت پرس‌وجوی مشابه را به صورت خودکار پیشنهاد دهد؛ کلمات جستجوی هم‌معنی و مشابه را شناسایی کرده و در جستجو از آن استفاده کند؛ قادر باشد عبارت‌های پرس‌وجوی حاوی کلمات پرکاربرد را مدیریت کند؛ و کاربر بتواند با کمک ابزارهای طراحی شده در رابط کاربر نتایج مورد نیاز خود را در سریع‌ترین زمان مشاهده کند (Grainger and Potter 2014).

برخی از پلتفرم‌های طراحی شـــده متن‌باز هســـتند مانند آپاچی ســـولر[۱]، Lemur[۲]، MeTA[۳] و Terrier[۴] و برخی نیز به صورت اختصاصی توسعه داده شده‌اند مانند Google، Bing و Amazon.

از آنجا که ســامانه گنج از پلتفرم آپاچی، لوســن و ســولر، به منظور مدیریت و جســتجو در اســناد ذخیره شده در پایگاه داده خود استفاده می‌کند، تمرکز این طرح بر این پلتفرم خواهد بود. کتابخانه آپاچی لو سن و موتور جستجوی آپاچی سولر با زبان جاوا نو شته شده‌اند اما در عین حال قابلیت پیاده‌ســازی با زبان‌هایی چونDelphi, Perl, C#, C++, Python, Ruby و PHP را دارند. نحوه ارتباط کتابخانه لوسن و موتور جستجوی سولر در شکل ۲-۱ نمایش داده شده است. لوسن وظیفه نمایه سازی داده‌های ورودی را برعهده دا شته و وظایف سولر نیز تحلیل عبارت پرس‌وجو، اجرای جستجو و نمایش اطلاعات بازیابی شده به کاربر است (Grainger and Potter 2014).



شکل ۲-۱- نحوه ارتباط بین آپاچی لوسن و آپاچی سولر (Grainger and Potter 2014)

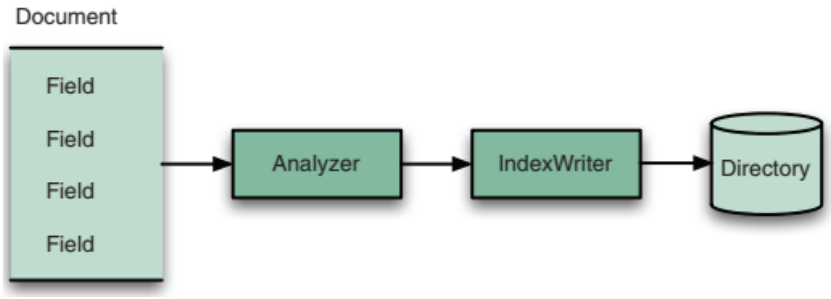
۲-۳ کتابخانه لوسن

لوســن کتابخانه بازیابی اطلاعات مقیاس‌پذیر[۵] با کارایی بالا بوده و امکان اضــافه کردن جســتجو را برای طراحان سامانه‌ها فراهم می‌کند. به بیانی دیگر، لوسن هسته پردازشی برای نمایه‌سازی داده‌های

[۱] http://lucene.apache.org/solr/
[۲] http://www.lemurproject.org/
[۳] https://meta-toolkit.org/
[۴] http://terrier.org/
[۵] Scalable

ورودی در قالب متن و تا سطح ریزدانگی کلمه بوده و برای بکارگیری به عنوان موتور جستجو نیازمند ابزارهای دیگری است (McCandless, Hatcher, and Gospodnetic 2010).

کلاس‌های مورد استفاده در لوسن برای نمایه سازی متن به صورت شماتیک در شکل ۲-۲ نشان داده شده‌اند. در ابتدا هر سند (متشکل از یک یا چند فیلد) توسط کلاس `Analyzer` تحلیل شده و به واژه[۱] شکسته می‌شود و سپس نمایه‌های[۲] استخراج شده توسط `IndexWriter` در مکانی (`Directory`) ذخیره می‌شوند (McCandless, Hatcher, and Gospodnetic 2010).



شکل ۲-۲- کلاس‌های مورد استفاده در لوسن برای نمایه‌سازی متن (McCandless, Hatcher, and Gospodnetic 2010)

لوسن قابلیت پردازش ورودی‌ها با فرمت‌های متعددی را دارد. این موارد به همراه کتابخانه مورد استفاده در جدول ۲-۱ گزارش شده‌اند (McCandless, Hatcher, and Gospodnetic 2010).

جدول ۲-۱- فرمت‌های ورودی قابلیت پردازش در لوسن (McCandless, Hatcher, and Gospodnetic 2010)

فرمت وردی	کتابخانه مورد استفاده
Microsoft's OLE2 Compound Document Format (Excel, Word, PowerPoint, Visio, Outlook)	Apache POI
Adobe Portable Document Format (PDF)	PDFBox
Rich Text Format (RTF)	Java Swing API (`RTFEditorKit`)
Plain text character set detection	ICU4J library
HTML	CyberNeko library
XML	Java's `javax.xml` classes
ZIP Archives	Java's built-in zip classes, Apache Commons Compress
TAR Archives	Apache Ant, Apache Commons Compress
BZIP2 compression	
AR Archives	Apache Commons Compress
CPIO Archives	

[۱] Token

[۲] Index

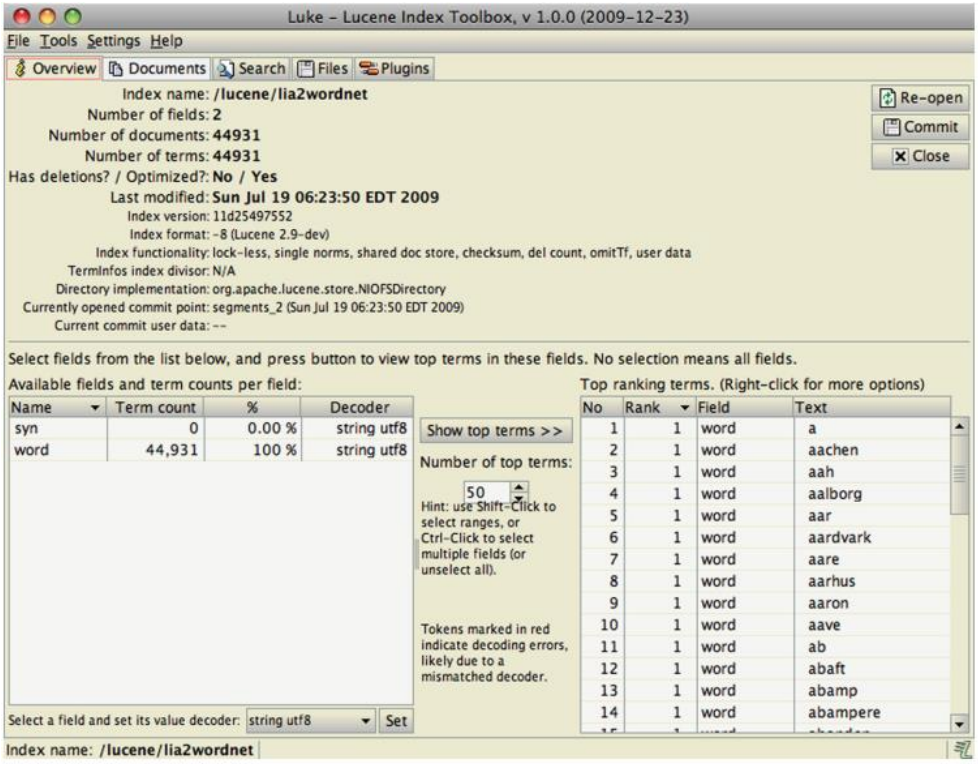
Java's built-in support (`GZIPInputStream`) , Apache Commons Compress	GZIP compression
ava's `javax.imageio` classes	Image formats (metadata only)
ASM library (JCR-1522)	Java class files
Java's built-in zip classes and ASM library, Apache Commons Compress	Java JAR files
Implemented directly	MP3 audio (ID3v1 tags)
Parses XML directly	OpenDocument
Parses metadata from FLV files directly	Adobe Flash
Java's built-in support (`javax.audio.midi.*`)	MIDI files (embedded text, eg song lyrics)
	WAVE Audio (sampling metadata)

پس از ورود داده به لوسن متن باید تحلیل گردد که این امر توسط کلاس `Analyzer` انجام می‌شود. به جهت اهمیت، به این کلاس به صورت مفصل در بخش **Error! Reference source not found.**، زیرسیستم مدیریت اسناد، پرداخته می‌شود.

یکی از ابزارهای مهم لوسن برای مدیریت نمایه‌ها، مرور اسناد و واژگان بر اساس نمایه‌ها، گزارش آمارهای مربوط به نمایه‌ها و ساختاردهی مجدد به اسناد، ابزار Luke[۱] است که برای اجرای آن بهره‌مندی از نسخه ۱,۵ به بعد Java Runtime Environment (JRE) ضروری است (McCandless, Hatcher, and Gospodnetic 2010).

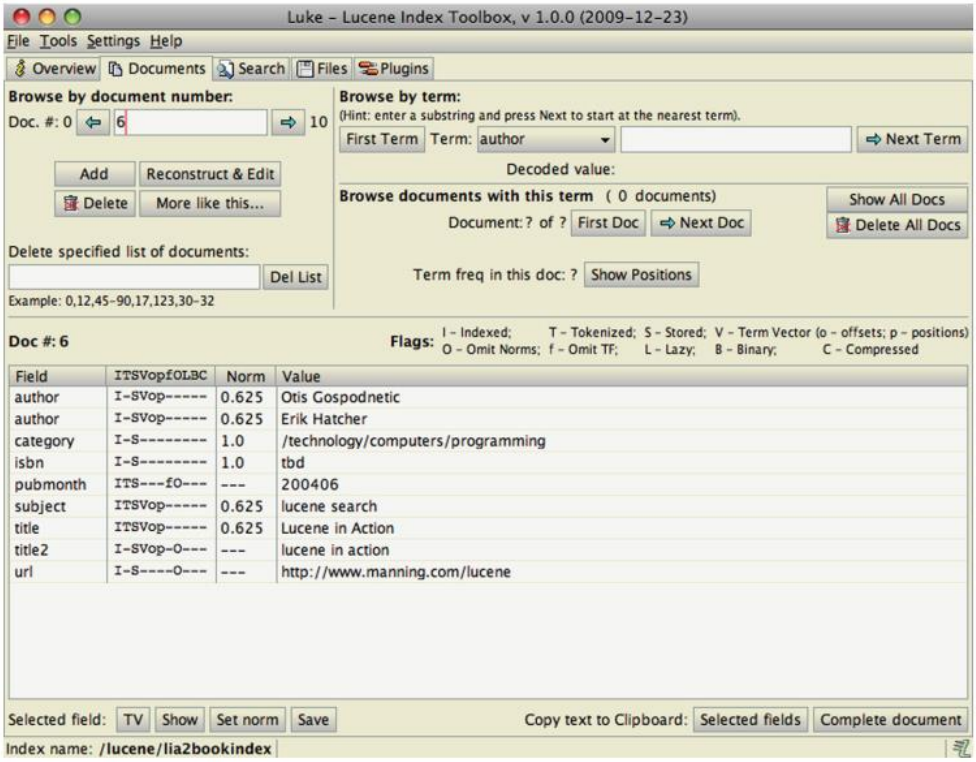
این رابط از پنج منو تشکیل شده است: نمای کلی (Overview)، اسناد (Documents)، جستجو (Search)، فایل‌ها (Files) و موارد افزوده (Plugins). در منوی نمای کلی، کاربر یا مدیرسیستم قادر است تصویر کلان نمایه‌های لوسن را مشاهده کند. این نما شامل تعداد فیلدها، تعداد اسناد و تعداد واژگان است (شکل ۲-۳). همچنین واژگان پرکاربرد در یک یا چند فیلد انتخابی در پنل Top Ranking Terms نشان داده شده‌است و با انتخاب هر واژه، اسناد حاوی آن نمایش داده می‌شود (McCandless, Hatcher, and Gospodnetic 2010).

[۱] http://code.google.com/p/luke



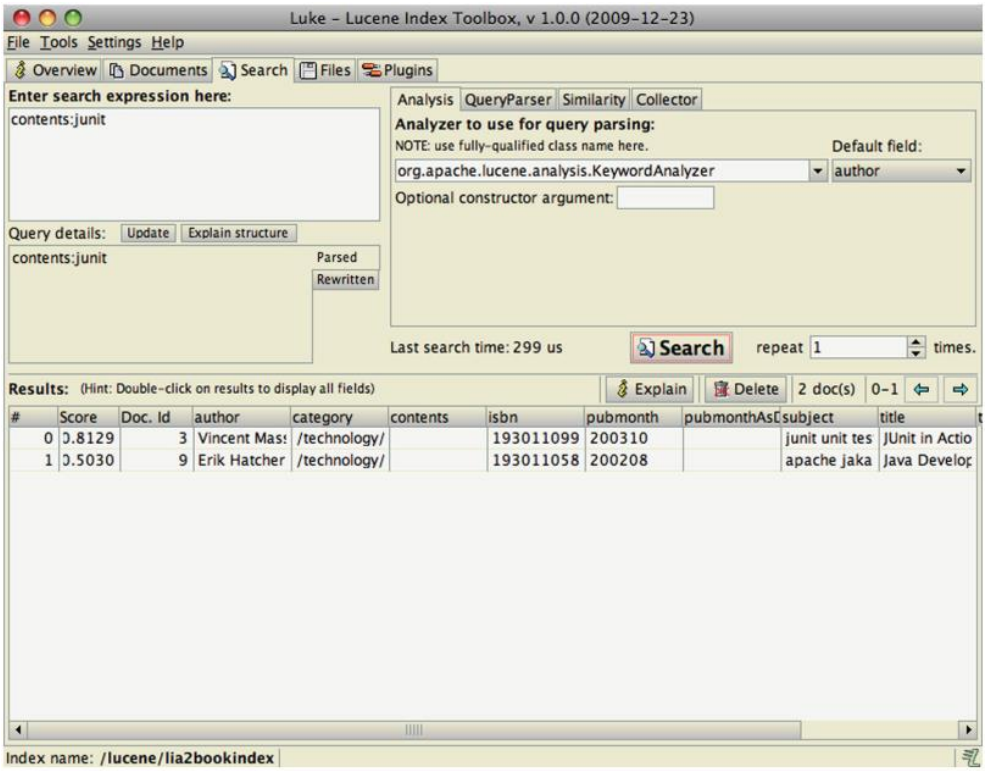
شکل ۲-۳- منوی مرور کلی در Luke (McCandless, Hatcher, and Gospodnetic 2010)

در منوی اسناد کاربر قادر است اسناد را با کمک دو روش مشاهده و مرور نماید: (۱) با انتخاب شماره سند (Browse by document number)، که در این حالت فیلدهای مربوط به سند به همراه مشخصات و اطلاعات هر فیلد را در بخش پایین صفحه نمایش داده می شود (شکل ۲-۴) و یا (۲) با انتخاب واژه (Browse by term)، که در این حالت یا با انتخاب پرکاربردترین واژه در فیلد منتخب (در شکل ۲-۴ Frist Term در فیلد author فعال است) و یا با تایپ واژه مورد نظر در محل مربوطه می‌توان اسناد حاوی چنین واژگانی را مشاهده نمود (McCandless, Hatcher, and Gospodnetic 2010).

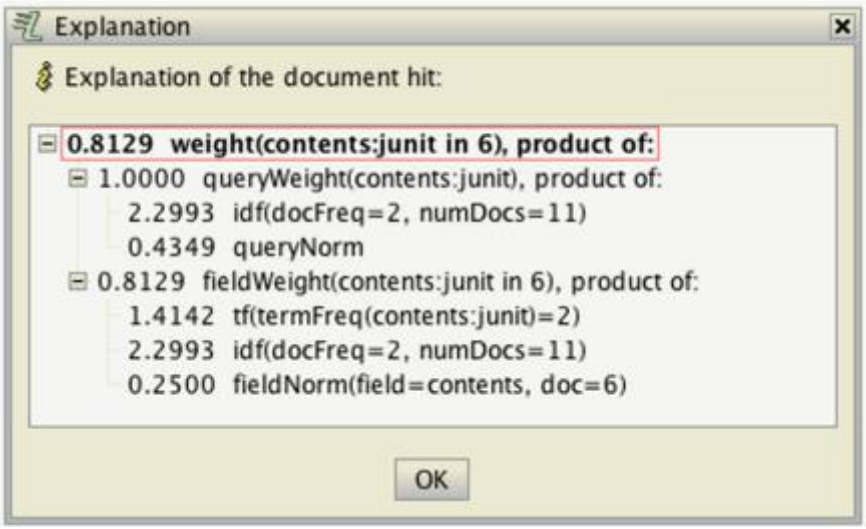


شکل ۲-٤- منوی اسناد در Luke (McCandless, Hatcher, and Gospodnetic 2010)

در منوی جستجو، کاربر می‌توان عبارت پرس‌وجوی خود را وارد کرده و از منوهای Analysis و QueryParse روش تجزیه را انتخاب کرده و جستجو را انجام دهد. با انتخاب سند بازیابی شده کاربر مجدداً به منوی اسناد منتقل شده و اطلاعات جزئی‌تر سند به وی نمایش داده می‌شود (شکل ۲-۵). همچنین با انتخاب گزینه «Explane» در بخش میانی این صفحه نحوه محاسبه وزن اسناد بازیابی شده برای کاربر نمایش داده می‌شود (شکل ۲-۶) (McCandless, Hatcher, and Gospodnetic 2010).

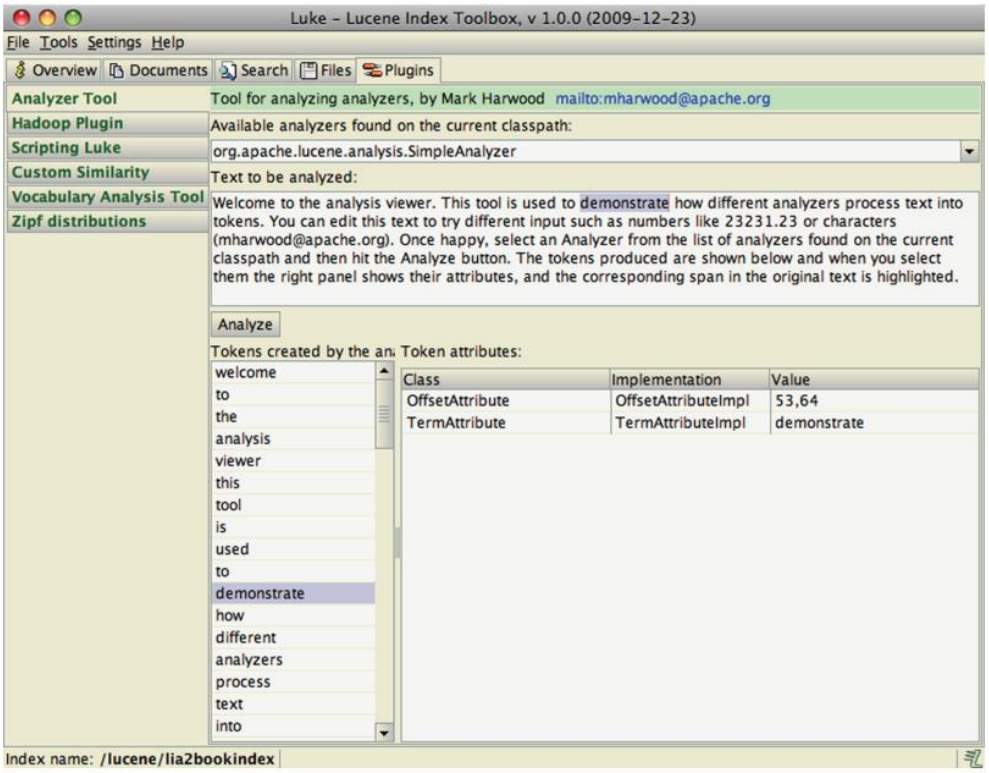


شکل ۲-۵- منوی جستجو در Luke (McCandless, Hatcher, and Gospodnetic 2010)



شکل ۲-٦- توضیحات مربوط به نحوه محاسبه وزن در Luke (McCandless, Hatcher, and Gospodnetic 2010)

در منوی موارد افزوده، تو ضیحات مربوط به ابزارهای تو سعه داده شده برای luke قرار داده شده است و کاربر می‌تواند به صورت آزمایشی آنها را برای متن محدود بکار گیرد ()‌. همچنین می‌توان ابزار های دیگری را نیز توســعه داد و به Luke افزود(برای مطالعه بیشــتر در مورد این ابزار به (McCandless, Hatcher, and Gospodnetic 2010) مراجعه شود).



شکل ۲-۷- منوی موارد افزوده در Luke (McCandless, Hatcher, and Gospodnetic 2010)

یکی دیگر از موارد مهم در تنظیمات لوسن، تعیین فضای مورد نیاز برای ذخیره‌سازی نمایه‌های ایجاد شده است. این فضا برای یک سند به عوامل مختلفی بستگی دارد؛ در بهترین حالات حدوداً یک سوم متن اصلی و در بدترین حالت (درنظرگیری بردار واژگان، اسناد حذف شده و مرورگرهای باز برای بررسی نمایه‌ها) حدوداً ده برابر متن اصلی است (به عنوان مثال برای نمایه‌سازی اسناد ویکی‌پدیا در نهایت ۱۴,۲ گیگابایت فضا نیاز است اما در طی پردازش فضای مورد استفاده لوسن به ۳۲,۴ گیگابایت هم می‌رسد). از معادله زیر برای تخمین فضای مورد نیاز استفاده می‌شود (McCandless, Hatcher, and Gospodnetic 2010):

$$\frac{1}{3}\ \text{indexed} + \text{stored} + 2\ \text{term vectors} \qquad (2\text{-}1)$$

به عنوان مثال اگر نیاز باشد علاوه‌بر نمایه و ذخیره شدن یک فیلد برای تمامی اسناد پایگاه داده، بردار واژگان آن نیز محاسبه شود، فضای مورد انتظار برابر خواهد بود با $10/3$ فضای مورد نیاز برای ذخیره‌سازی تمامی اسناد. این فضا را می‌توان با غیرفعال کردن برخی از ویژگی‌ها نظیر نگهداری TF) فیلدهای غیرضروری، اطلاعات مکانی و offsetها در هنگام نمایه‌سازی بردارهای

واژگان و نمایه‌سازی و ذخیره‌سازی فیلدهای کمتر برای هر سند مدیریت نمود (McCandless, Hatcher, and Gospodnetic 2010).

۲-۴ موتور جستجوی سولر

موتور جستجوی سولر وظیفه جستجو براساس عبارت پرس‌وجوی کاربر و نمایش نتایج بازیابی شده را برعهده دارد. قابلیت‌های متعددی برای سولر معرفی شده‌است که می‌توان آنها را در قالب موارد زیر دسته‌بندی نمود (Grainger and Potter 2014):

- تجربه کاربر: برای فراهم آوردن تجربه مناسب سولر از سرویس REST[۱] برای استانداردهای XML، JSON و HTTP استفاده می‌کند تا کاربران قادر باشند از طریق زبان‌ها و وب‌سرویس‌های مختلف با آن ارتباط برقرار کنند. همچنین وجود قابلیت‌هایی چون مرتب‌سازی و صفحه‌بندی[۲] نتایج، جستجوی چندوجهی[۳]، پیشنهاد خودکار[۴]، املای کلمات[۵]، پررنگ کردن کلمات پرس‌وجو[۶] و جستجوی جغرافیایی از دیگر امکاناتی است که با پیاده‌سازی آنها سولر تلاش دارد تا تجربه مناسبی را برای کاربر فراهم آورد.
- مدلسازی داده: برای مدلسازی اسناد موجود در پایگاه داده امکاناتی در سولر طراحی شده است: تخصیص ویژگی‌هایی خاص به همه اسناد موجود در پایگاه داده، روش‌های پردازش نظیر عبارت‌های منطقی AND، OR و NOT، جستجوی Wildcard، جستجوی فازی و ...، خوشه‌بندی اسناد، پشتیبانی از فرمت‌های PDF و Word، انتقال داده‌ها از پایگاه‌های داده رابطه‌ای و پشتیبانی چندزبانه.

---

[۱] Representational State Transfer (REST)

[۲] Sorting and pagination

[۳] Facet Searching

[۴] Autosuggest

[۵] Spell checking

[۶] Hit highlighting

همچنین سولر به گونه‌ای طراحی شده است که قادر است حجم عظیمی از داده‌های متنی را با سرعت به صورت نزدیک به زمان واقعی[۱] پردازش کند. به بیانی دیگر، اسناد با فاصله زمانی اندکی از اضافه شدن به پایگاه داده قابلیت جستجو را دارند. با این حال (Grainger and Potter 2014):

- در سولر از رویکردهای بهینه‌سازی موتور جستجو استفاده نشده است؛
- به علت استفاده از طراحی NoSQL[۲] و ساختار غیر رابطه سولر، ارتباطی بین اسناد وجود ندارد و درصورتی که داده‌ای بین چند سند و یا چند فیلد اطلاعاتی مشترک باشد، این داده‌ها باید برای هر سند تکرار شوند؛
- سولر زمانی عملکرد خوبی دارد که داده‌ها غیر ساخت‌یافته باشند و در صورتی که داده‌ها ساخت‌یافته باشند بهتر است از پایگاه‌های داده رابطه‌ای استفاده شود؛
- در سولر می‌توان به راحتی سندی را اضافه و یا کم کرد اما نمی‌توان به سادگی اطلاعات مربوط به یک سند را بروز نمود و باید از ابتدا فرآیند نمایه‌سازی انجام شود؛
- سولر زمانی عملکرد بهینه دارد که عبارت پرس‌وجو از واژگان کمی تشکیل شده باشد.
- نسخه‌های فعلی سولر قابلیت مقیاس‌پذیری الاستیک را ندارد (درصورت افزایش بار محاسباتی سرورهای جدید به صورت کاملاً خودکار فعال نمی‌شوند). طراحی سولرCloud با کمک آپاچیZookeeper گامی در راستای بهبود این شرایط است.
- سولر به صورت پیش‌فرض از روش‌های پایه‌ای در بازیابی اطلاعات نظیر روش شاخص معکوس، بولین، مدل فضای برداری و TF-IDF استفاده می‌کند. درحالی‌که در طی سال‌های گذشته توسعه‌ها و اصلاحات بسیاری برای آنها ارائه شده است.

علی‌رغم موارد ذکر شده، متن باز بودن لوسن وسولر امکان تغییر ماژول‌ها و روش‌های بکاررفته در آن را برای طراحان و پیاده‌سازان موتورهای جستجو فراهم آورده و این امر را می‌توان یکی از دلایل عمده شهرت و کاربردهای متعدد آنها دانست. با این وجود و تاکنون هیچ‌گونه پژوهشی در زمینه شناسایی و واکاوی این کتابخانه و موتور جستجو در پژوهشگاه انجام نشده است. به جهت تمرکز این طرح بر موتور جستجوی سولر، فصل سوم به آن تخصیص یافته است.

---

[۱] Near real-time (NRT)

[۲] Not Only SQL (NoSQL)

فصل سوم

مباحث پایه‌ای در موتور جستجوی سولر

۳-۱ مقدمه

سولر موتور جستجویی متن‌باز و مقیاس‌پذیر برمبنای تکنولوژی NoSQL است که برای جستجو در داده‌های متنی حجیم طراحی شده و نتایج بازیابی شده را براساس میزان ارتباط[۱] با عبارت پرس‌وجو نمایش می‌دهد. داده‌ها در این موتور جستجو ساختاری تخت[۲] داشته (داده‌ها به یکدیگر وابستگی نداشته و سلسله مراتبی نیستند) و نرخ بروزرسانی کمی دارند. همچنین از آنجا که سولر به صورت NoSQL طراحی شده است ضرورتی ندارد تمامی داده‌ها دارای ساختاری یکسان باشند. به بیانی دیگر، سولر برای مدیریت داده‌هایی بهینه شده است که مبتنی بر متن (Text-centric) بوده و ساختاری تخت (Document-Oriented) و منعطف (Flexible-schema) داشته و نرخ بروزرسانی در داده‌ها کمتر از نرخ خواندن آن‌ها (Read-dominant) باشد (Grainger and Potter ۲۰۱۴)[۳].

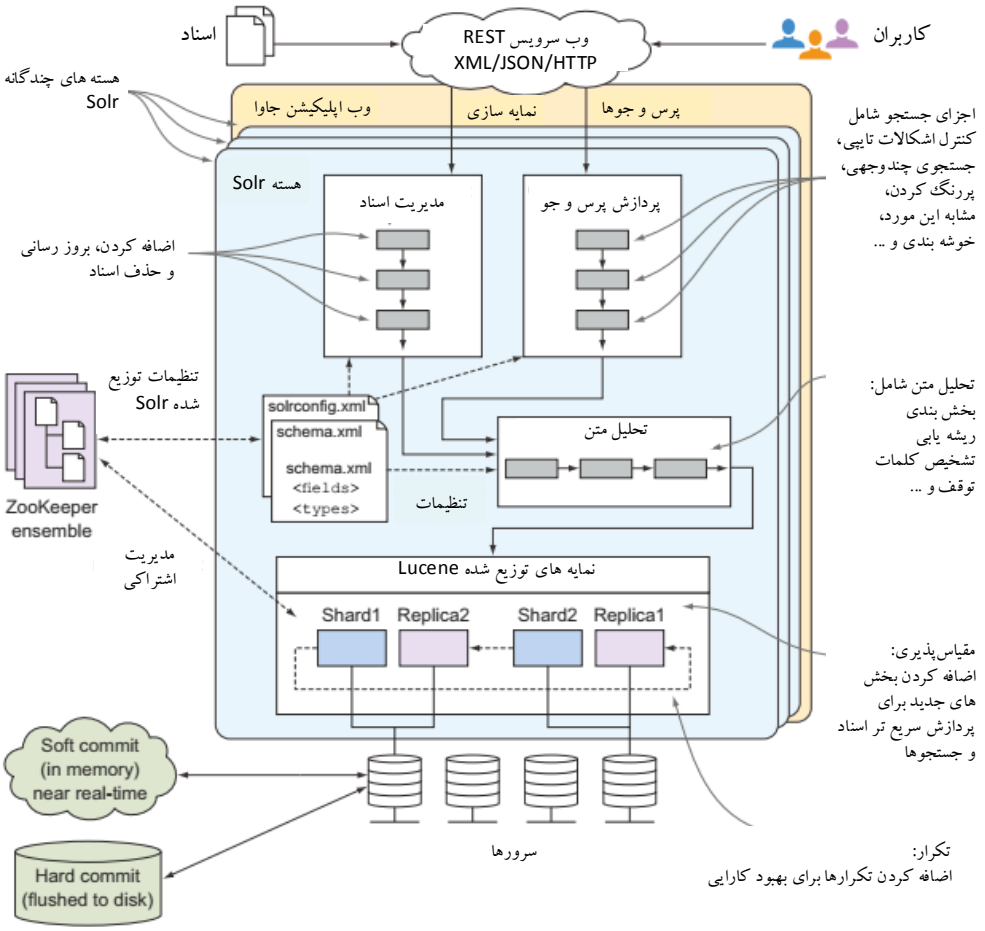
---

[۱] Relevancy

[۲] Flat

[۳] تمامی مطالب ارائه شده در این فصل از کتاب (Grainger and Potter ۲۰۱۴) استخراج شده‌اند که منبعی اصلی برای درک عملکرد سولر و ماژول‌های اصلی آن است (http://lucene.apache.org/solr/resources.html). لذا به جهت اختصار در ادامه بدان ارجاع داده نخواهد شد.

همچنین، سولر پلتفرمی تحت جاوا است که در هر موتور جستجوی مبتنی بر سرولت جاوا[۱] نظیر Jetty، Tomcat، full J2EE، JBoss و یا Oracle AS کار می‌کند. اجزای اصلی سولر در شکل ۳-۱ نمایش داده شده است.



شکل ۳-۱- اجزای سولر

کاربران سولر باید قادر باشند از طریق نرم‌افزارها و زبان‌های مختلف با آن ارتباط برقرار کنند، بدین منظور وب‌سرویس REST برای استانداردهای XML، JSON و HTTP طراحی شده است. همچنین سولر می‌تواند هسته‌های پردازشی متعددی داشته باشد که امکان پردازش موازی داده‌ها و پشتیبانی از به اشتراک‌گذاری منابع[۲] را برای رایانش ابری فراهم می‌آورد.

---

[۱] Java servlet

[۲] Multitenancy

زیر سیستم‌های (۱) مدیریت اسناد، (۲) تحلیل متن و (۳) پردازش پرس‌وجو سه زیر سیستم سولر بوده و هریک به صورت فرآیندهایی ماژولار طراحی شده‌اند. طراحی ماژولار زیرسیستم‌ها، آنها را قادر می‌سازد تا به‌صورت موازی فعالیت کرده و قابلیت توسعه و یا شخصی‌سازی را داشته باشند.

قابلیت مقیاس‌پذیری سولر توسط دو ویژگی تعداد پرس‌وجوهای پردازش شده در ثانیه و تعداد اسناد نمایه شده مشخص می شود. تعداد پرس‌وجوهای پردازش شده با قابلیت پرداز شی سرورها مرتبط است و هرچه تعداد سرورها و قابلیت‌های پرداز شی آنها بیشتر باشد، تعداد پرس‌وجوهای پردازش شده در ثانیه افزایش خواهد یافت. همچنین اگر حجم اسناد و یا تعداد آنها زیاد باشد، کارایی بازیابی اطلاعات کاهش خواهد یافت. بدین منظور ماتریس اصلی نمایه‌ها به بخش‌های کوچک‌تری بنام Shard تقسیم و شکسته شده و هر بخش به صورت موازی و جداگانه پردازش می‌شوند. در انتها نیز نتایج این بخش‌ها ترکیب شده و نتیجه نهایی برای کاربر نمایش داده می‌شود.

پیش از پرداخت عمیق‌تر به این زیر سیستم‌ها ضروری است تا برخی از مفاهیم کلیدی بیان شده و درباره آنها توضیحاتی ارائه شود.

۲-۳ مفاهیم کلیدی در سولر

۱-۲-۳- تنظیمات در سولر

انجام تنظیمات در سولر در دو سند اصلی Solrconfig و Schema انجام می شود. تنظیمات اصلی سامانه‌ای که بر سولر ساخته شده است در Solrconfig انجام شده و Schema تعریف کننده ساختار نمایه‌سازی و نوع-فیلد[۱] و فیلدهای مورد استفاده است. به خصوصیات این دو سند در ادامه پرداخته می‌شود.

SOLRCONFIG

در زمان اجرا، سولر فایل اصلی تنظیمات را با نام SolrConfig در مسیر اصلی نصب خود فراخوانی می‌کند. چهار بخش اصلی این سند شامل تنظیمات مربوط به محل ذخیره‌سازی نمایه‌ها (Data directory location)، تنظیمات حافظه cache (Cache parameters)، مدیریت درخواست‌ها

[۱] Field type

(Request handlers) و مولفه‌های جستجو (Search components) است. نمونه‌ای از این سند در شکل ۳-۲ آورده شده است.

شکل ۳-۲- نمونه‌ای از سند Solrconfig

تو ضیحات مختصری از چهار بخش فوق به شرح زیر ا ست (این بخش‌ها به صورت مفصل‌تر در زیرسیستم‌های مرتبط بررسی خواهند شد):

- محل ذخیره سازی نمایه‌ها: نمایه سازی یکی از بخش‌های ا صلی سولر و لو سن ا ست و با کمک این بخش محل ذخیره‌سازی نمایه‌های ساخته شده توسط «زیرسیستم تحلیل متن» و مسیر کتابخانه‌های مورد استفاده در «زیر سیستم مدیریت اسناد» و «زیر سیستم تحلیل متن» تعیین می‌شود.

- □ مدیریت درخواست‌ها: در این بخش درخواست‌های دریافتی از کلاینت در طی فرآیند مدیریت و پردازش جستجو تحلیل شده و نتایج را برای نمایش در کلاینت تنظیم می شود. جزئیات مربوط به این بخش در «زیرسیستم پردازش پرس‌وجو» بیان خواهد شد.

- □ تنظیمات حافظه cache: یکی از راهکارهای سولر برای بهبود کارایی پردازش پرس‌وجوها استفاده از Cache است. برای استفاده بهینه از این حافظه و مدیریت آن، توجه به موضوعات زیر ضروری است:
  - اندازه Cache: تعیین حداکثر تعداد موارد ذخیره شده در Cache سیستم موجب خواهد شد تا تعادلی بین حافظه اصلی و نهان ایجاد شده و سولر بتواند فعالیت اصلی خود را با کارایی بالاتری انجام دهد. زمانی که اندازه Cache به حد آستانه می‌رسد باید برخی از اسناد موجود پاک شود تا اسناد جدید بتوانند در آن ذخیره شوند. برای پاک کردن دو سیاست وجود دارد: سیاست Least Recently Used (LRU) (روش پیش‌فرض) که اسنادی قدیمی را پاک می کند (قدمت یک سند بر اساس آخرین زمانی که مورد استفاده قرار گرفته مشخص می‌شود)؛ و سیاست Least Frequently Used (LFU) که اسناد با کمترین استفاده (غیر جذاب‌ترین اسناد) را از Cache پاک می کند. از آنجا که این سیاست نیازمند نگهداری تعداد دفعات استفاده از اسناد است، حافظه و حجم اطلاعات بیشتری را به خود تخصیص می‌دهد.
  - سود حاصل از نگهداری و تعیین تعداد اسناد پاک شده[۱]: محاسبه سود حاصل شده از نگهداری و تعیین تعداد اسناد پاک شده سبب خواهد شد تا اندازه بهینه‌ای برای Cache تعیین گردد.
  - ناکارایی اسناد در Cache: تعیین مصادیق ناکارایی اسناد موجود در Cache ارتباط مستقیمی با سیاست پاک کردن این حافظه و تعریف اسناد ناکارامد در بافت سیستم دارد.
  - آماده سازی خودکار Cache جدید: همانطور که عنوان شد، سولر جستجوگری را به‌صورت فعال و جستجوگرهای دیگری را درحال آماده‌باش دارد که به محض بسته شدن جستجوگر فعلی، جستجوگر آماده شده بعدی بارگذاری

[۱] Hit ratio and eviction

خواهد شد. برخی از اسناد جستجوگر فعلی می‌توانند در Cache جستجوگر آماده شده بعدی ذخیره شوند. به این فرآیند آماده سازی خودکار Cache جدید گفته می شود. حداکثر تعداد اسناد جستجوگر فعلی برای ذخیره سازی و یا تعیین درصد اشغال Cache از پارامترهای مهم در مدیریت آن است.

- مولفه‌های جستجو: پرس‌وجوها در سولر توسط مولفه‌ای به نام جستجوگر[۱] پردازش می‌شود. در هر لحظه از زمان تنها یک جستجوگر فعال وجود داشته و تمامی درخواست‌ها توسط این جستجوگر پردازش می شوند. جستجوگر دستر سی read-only به نمایه‌های لوسن داشته و اگر در زمان فعال بودن جستجوگر سند جدیدی اضافه شود، اطلاعات آن در جستجو دخالت داده نخواهد شد. برای این منظور جستجوگر باید بروزرسانی شود (جستجوگر فعلی بسته شده و جستجوگر جدیدی فعال شود). دلیل این امر آن است که اگر در لحظه اضافه شدن سند جدید، جستجوگر بروزر سانی شود، صفحه نتایج کاربر تغییر کرده و این اتفاق خوبی برای کاربر نیست (تجربه نامناسب کاربر). همچنین اگر به ازای اضافه شدن هر سند جدید، جستجوگر جدیدی از ابتدا بازخوانی و اجرا شود، بازیابی و نمایش نتایج کند خواهد شد (کاهش سرعت).

برای رفع این مشکل، سولر در پشت صحنه یک یا چند جستجوگر را درحال آماده شدن (warming) نگه داشته و پارامترهای جستجو در آن را براساس مقادیر پرکاربرد در سیستم بازیابی اطلاعات تنظیم می‌کند. جستجوگر فعلی، آنقدر فعال می‌ماند تا جستجوگر پشت صحنه کاملاً آماده شود و به محض آماده شدن، جستجوگر فعلی بسته شده و جستجوگر جدید جایگزین خواهد شد. تعداد این جستجوگرها توسط پارامتر maxWarmingSearchers با مقدار پیش‌فرض ۲ در Schema تنظیم می‌شود.

همچنین اگر درخواست جستجویی به سیستم داده شود و هیچ جستجوگر فعالی در سیستم نباشد شود با کمک دستور useColdSearcher می‌توان به سولر دستور داد که جستجو را آغاز کند چه جستجوگر به طور کامل آماده باشد چه نه (مقدار True) و یا آنکه صبر کند تا جستجوگر به طور کامل آماده شود (مقدار False).

---

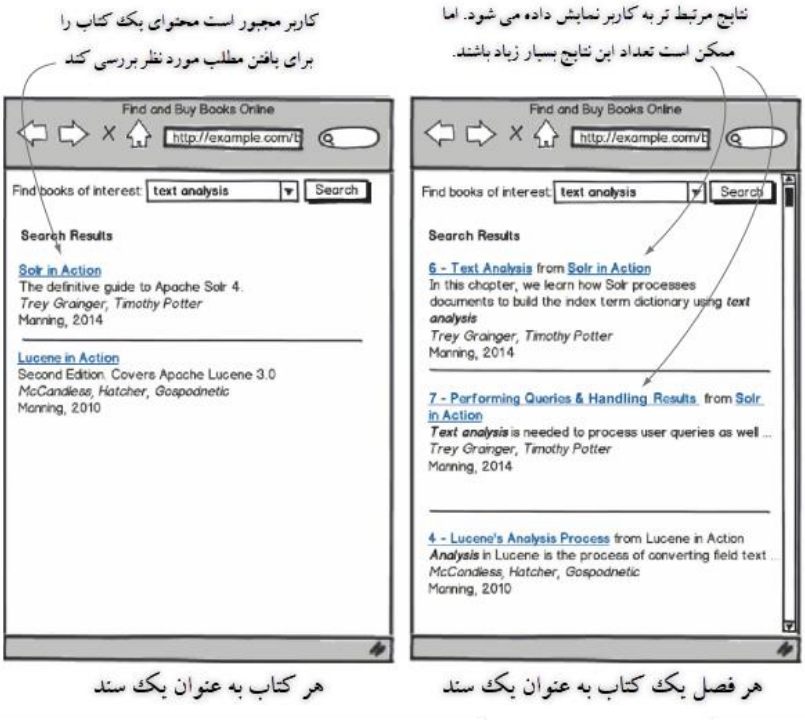
[۱] Searcher

SCHEMA

همانگونه که ذکر شد، ساختار نمایه‌سازی و نوع-فیلد و فیلدهای مورد استفاده در سندی به نام Schema انجام و ذخیره می‌شود. برای تنظیم Schema پاسخ به سوالات زیر می‌تواند کمک کننده باشد:

- معنا و مفهوم سند در سیستم طراحی شده چیست؟

پاسخ به این سوال سطح ریزدانگی بررسی و پردازش اسناد را تعیین می‌کند. به عنوان مثال، سند می‌تواند به معنای یک کتاب کامل باشد و یا هرفصل از یک کتاب سندی جداگانه محسوب شود. درحالت نخست، کاربر باید خود به دنبال فصل/مطلب مورد نظر در کل یک کتاب (سند بازیابی شده) باشد اما در حالت دوم می‌توان فصل مرتبط با عبارت جستجوی کاربر را برای وی نمایش داد (که البته در مواردی سبب سردرگمی کاربر نیز خواهد شد چرا که ممکن است نتایج بسیار زیادی صرفاً از یک کتاب برای او نمایش داده شود). شکل ۳-۳ نمونه‌ای از جستجوی عبارت «text analysis» را برای سطح ریزدانگی کتاب و فصل‌های هر کتاب به صورت جداگانه نمایش می‌دهد.



شکل ۳-۳- نتایج جستجوی عبارت «text analysis» برای سطح ریزدانگی هر کتاب و هر فصل هر کتاب

- مشخصه کلیدی[۱] هر سند چیست؟

به عنوان نمونه برای یک کتاب مشخصه کلیدی آن می‌تواند ISSN کتاب باشد. دلیل اهمیت تعیین این مشخصه آن است که اگر دو سند دارای مشخصه یکسان باشند، سولر آخرین سند اضافه شده را نگه‌داشته و بقیه اسناد را حذف خواهد کرد. همچنین اگر معماری سولر به گونه‌ای باشد که در آن از سرورهای متعدد برای پردازش استفاده شود، از مشخصه کلیدی هر سند برای تبادل اطلاعات و ادغام نتایج پردازش‌ها استفاده می‌شود.

- فیلدهایی که نمایه می‌شوند کدامند؟

سولر جستجو را برروی فیلدهای نمایه شده[۲] انجام می‌دهد. بنابراین تعیین دقیق این فیلدها بر اندازه شاخص معکوس، دقت، بازخوانی و سرعت جستجو اثر بسیار زیادی خواهد داشت.

- چه فیلدهایی باید برای کاربر نمایش داده شوند؟

فیلدهایی که قابلیت نمایش برای کاربر را دارند در فیلدهای ذخیره‌شده[۳] تعیین می‌شوند. تعیین دقیق این فیلدها نیز تاثیر بسیاری بر سرعت و کفایت نمایش نتایج برای کابر خواهد داشت.

یکی از اقدامات مهم دیگر در تنظیم Schema تعیین فیلدها ست. برخی از انواع این فیلدها در ادامه توضیح داده می‌شوند.

- فیلدهای ضروری (Required fields): تعیین کننده ضروری بودن آن فیلد است و مقدار این پارامتر برای فیلد مشخصه کلیدی باید `true` باشد.
- فیلدهای چند مقداره (Multivalued fields): یک فیلد می‌تواند شامل یک یا چند مقدار باشد و سولر هر یک از این مقادیر را به صورت جداگانه ذخیره می‌کند.
- فیلدهای دینامیک (Dynamic fields): در مواردی که تعدادی از فیلدها مشخصات یکسانی دارند، می‌توان به‌جای تکرار هر مشخصه برای تک تک فیلدها، یک مشخصه کلان برای تمامی فیلدهای همسان تعریف کرد. به عبارت بهتر، اگر:

```
<dynamicField name="*_s" type="string" indexed="true" stored="true" />
```

---

[۱] Unique key

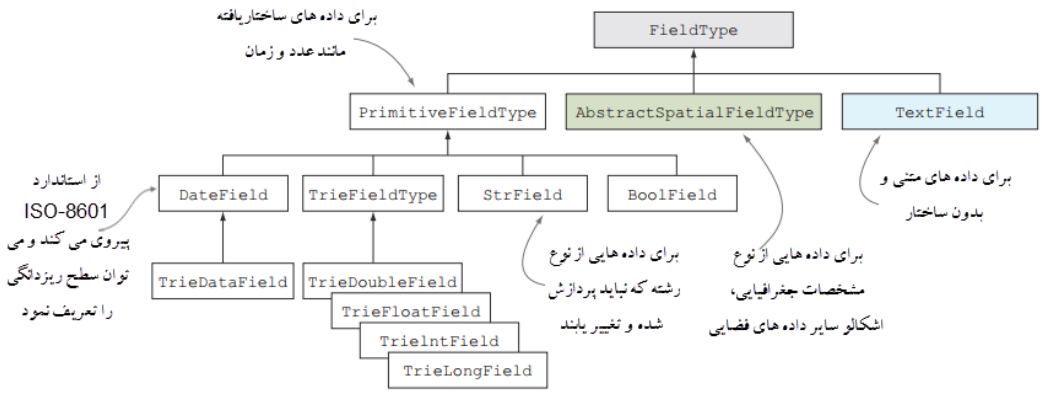
[۲] Indexed fields

[۳] Stored fields

بدین معناست که هر فیلدی که در انتهای نام آن "_s" وجود داشته باشد از نوع رشته بوده، نمایه شده و در صفحه نتایج برای کاربر نمایش داده می‌شود.

- فیلدهای کپی (Copy fields): در اغلب پایگاه‌های داده، اسناد در قالب فیلدهای مختلف ذخیره می‌شوند (فیلد نام نویسنده، فیلد عنوان، فیلد کلمات کلیدی و ...). درمواردی که کاربر بخواهد عبارت پرس‌وجوی خود را جستجو کند بدون آنکه فیلد جستجو را تعیین کرده باشد، سیستم بازیابی اطلاعات باید نتایج را مستقل از آنکه این عبارت در کجای سند قرار دارد نمایش دهد. اما اتفاقی که رخ می‌دهد آن است که موتور جستجو صرفاً در فیلدی که به عنوان فیلد پیش‌فرض برای جستجو طراحی شده است به دنبال نتایج مرتبط خواهد بود. برای رفع این مشکل دو راهکار وجود دارد: (۱) استفاده از فیلدهایی از نوع کپی که درواقع فیلدی سایه‌ای بوده و به صورت غیر سمی تمامی فیلدهای یک سند در آن کپی شده‌اند و (۲) استفاده از تجزیه‌گر[۱]های غیر از تجزیه‌گر پیش‌فرض لوسن (این موارد در بخش ۱-۵-۳-مولفه Query بررسی خواهند شد).

نمودار کلاس انواع فیلدها در Schema در شکل ۳-۴ نمایش داده شده است.



شکل ۳-٤- نمودار کلاس انواع فیلدها در Schema

۲-۲-۳- سند

سولر موتور جستجوی اسناد است و هر سند می‌تواند دارای یک یا چند فیلد اطلاعاتی از نوع رشته[۲]، بولی[۳]، زمان، مکان و غیره باشد. خصوصیات هریک از این فیلدها در سندی به‌نام Schema

---

[۱] Parser

[۲] String

[۳] Boolean

تعریف می‌شود و با کمک این سند سولر قادر است اسناد ورودی را مدیریت و پردازش نماید. ذکر مجدد این نکته نیز ضروری است که از آنجا که سولر به صورت NoSQL طراحی شده است ضرورتی ندارد همه فیلدها برای همه اسناد وجود داشته و یا دارای مقدار باشند.

همچنین درصورتی که سندی دارای فیلدی باشد که با نامی دیگر در Schema تعریف شده، سولر این قابلیت را دارد که به صورت خودکار نوع فیلد را حدس زده و آن را با ملاحظاتی به Schema اضافه کند.

۳-۲-۳- تطابق فازی

تطابق فازی[۱] به معنای توانایی یافتن عبارت‌ها و یا واژگان ناهمسان با عبارت پرس‌وجوی کاربر است. سولر این توانایی را با کمک چندین روش عملیاتی ساخته است:

- جستجوی Wildcard: زمانی که کاربر املای درست کلمه پرس‌وجو را نمی‌داند و یا می‌خواهد تمام حالت های ممکن از کلمه پرس‌وجو را در پایگاه داده جستجو کند. مثال‌هایی از حالت‌های مختلف این جستجو در جدول ۳-۱ نمایش داده شده‌اند.

جدول ۳-۱- جستجوی Wildcard**: عبارت پرس‌وجوی کاربر و عبارتی که سولر جستجو می‌کند**

عبارت پرس‌وجوی کاربر	عبارتی که سولر جستجو می‌کند
offi*	office, officer, official, …
off*r	offer, officer, officiator, …
off?r	offer
“softwar* Eng?neering”	“softwar* Eng?neering”

وجود علامت * در کلمه پرس‌وجو به معنای یافتن و جستجوی همه کاراکترهای ممکن بامعنا و وجود علامت ؟ به معنای یافتن و جستجوی تنها یک کاراکتر با معنا است (سولر برای شناسایی کلمات بامعنا از نمایه ساخته شده برای پایگاه داده استفاده می‌کند). همچنین علامت ”X“ به معنای جستجوی دقیق عبارت X بوده و اعمال روش جستجوی Wildcard برای عبارت X ممکن نیست.

جستجوی اصلاح فاصله[۲]: در جستجوی Wildcard حداکثر کاراکترهایی که باید یافته و جستجو شوند مشخص نمی شود. این امر می‌تواند باعث کاهش سرعت بازیابی اطلاعات

---

۱ Fuzzy matching

۲ Edit-distance search

شود. لذا در مواردی لازم است تعداد کاراکترها محدود شوند. در این حالت می‌توان از جستجوی اصلاح فاصله استفاده کرد. مقدار این فاصله در سولر بر اساس فاصله دامرا-لونا شتاین[۱] که تعمیم یافته فاصله لونا شتاین است (Damerau ۱۹۶۴) محاسبه می شود. البته توابع فاصله دیگری نیز تعریف شده‌اند که ازجمله آنها می‌توان به موارد جدول ۷-۲ ضمائم اشاره کرد.

برای فراخوانی این جستجو از دستور x~ استفاده می شود: 1~به معنای جستجو با یک فاصله اصلاح، 2~جستجو با دو فاصله اصلاح (حالت پیش‌فرض) و N~ به معنای جستجو با N فاصله اصلاح است. به عبارت دیگر، اگر عبارت پرس‌وجوی کاربر به صورت administrator~ باشد، سولر از عبارت‌هایی مانند adminstrator، administrater، administrator نیز علاوه بر عبارت پرس‌وجوی کاربر استفاده می‌کند.

□ جستجوی نزدیکی[۲]: حالت جستجوی اصلاح فاصله برای یافتن واژگان مشابه با واژه پرس‌وجوی کاربر بکار می‌رود، اما اگر کاربر بخواهد عبارات مشابه را بیابد (ترکیبی از واژگان)، جستجوی نزدیکی در سولر طراحی شده است. جدول ۳-۲ مثال‌هایی از این جستجو را نشان می‌دهد.

جدول ۳-۲- جستجوی نزدیکی: عبارت پرس‌وجوی کاربر و توضیحات آن

عبارت پرس‌وجوی کاربر	توضیح
"chief officer"~0	جستجوی دقیق (حداکثر فاصله دو واژه برابر ۰ است)
"chief officer"~1	chief و officer باید حداکثر فاصله‌ای برابر یک داشته باشند. مانند: "chief executive officer", "chief financial officer", …
"chief officer"~2	chief و officer باید حداکثر فاصله‌ای برابر دو داشته باشند. مانند: "chief business development officer", "officer chief", …
"chief officer"~N	chief و officer باید حداکثر فاصله‌ای برابر N داشته باشند.

□ جستجوی بازه‌ای[۳]: جستجو در بازه‌های عددی و یا حرفی نیز در سولر میسر است. نمونه‌هایی از این جستجو در جدول ۳-۳ گزارش شده است.

---

[۱] Damerau Levenshtein

[۲] Proximity search

[۳] Range search

جدول ۳-۳- جستجوی بازه‌ای: عبارت پرس‌وجوی کاربر و عبارتی که سولر برای جستجو استفاده می‌کند

عبارت پرس‌وجوی کاربر	عبارتی که سولر جستجو می‌کند
`Yearsold: [18 To 21]`	Yearsold=18, 19, 20, 21
`Yearsold: [18 To 21}`	Yearsold=18, 19, 20
`Yearsold: {18 To 21}`	Yearsold= 19, 20
`Title: [boat To boulder]`	Title= boat, boil, book, boulder, …
`Price: [12.99 To 14.99]`	Price= 12.99, 13.00009, 14.01, …

۳-۲-۴- ارتباط

یافتن اسناد مرتبط با عبارت پرس‌وجوی کاربر یکی از اصلی‌ترین فعالیت‌ها در موتورهای جستجو است و تعیین میزان این ارتباط[۱] یکی دیگر از فعالیت‌ها مهم در این رابطه است. در سولر میزان ارتباط با کمک کلاس `similarity` که در Schema تعریف شده است، تعیین می شود و سولر به صورت پیش‌فرض از تابع `DefaultSimilarity` که متلعق به لوسن است استفاده می‌کند.

در این تابع، ابتدا با استفاده از منطق بولی اسناد مرتبط با عبارت پرس‌وجوی کاربر شناسایی شده و سپس از مدل فضای برداری[۲] و شباهت کسینوسی[۳] بین بردار سند و عبارت پرس‌وجو برای امتیازدهی[۴] (RS) و یافتن میزان ارتباط استفاده می شود. شکل **۳-۵** نمایی از شباهت کسینو سی را نمایش می‌دهد. در این شکل بردار عبارت پرس‌و جو v(q) به بردار سند ۱ (v(d1)) نزدیک‌تر از بردار سند ۲ (v(q2)) است (این نزدیکی براساس زاویه بین دو بردار اندازه‌گیری می‌شود؛ هرچه زاویه کوچکتر باشد دو بردار به هم نزدیک‌تر بوده و یا به هم شباهت بیشتری دارند).

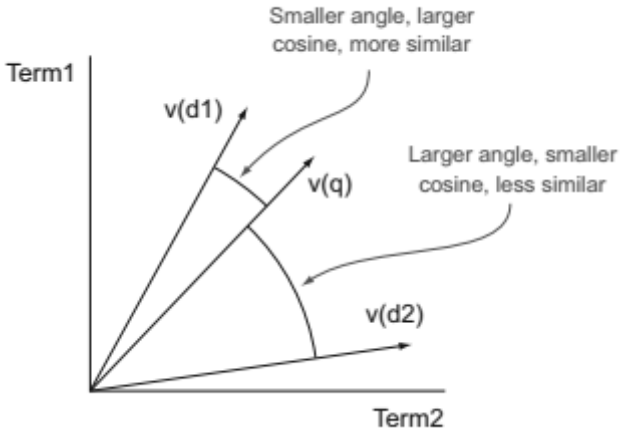
---

۱ Relevancy

۲ Vector space model (VSM)

۳ cosine

۴ Relevancy score



شکل ۳-۵- شباهت کسینوسی بین بردارهای واژگان

همچنین در این تابع برای محاسبه میزان ارتباط بین هر سند و عبارت پرس‌وجو از رابطه (3–1) استفاده می‌شود:

$$Score(q,d) = \sum_{t \text{ in } d} \left(tf\ (t \text{ in } d) \times idf\ (t)^2 \times t.getboost() \times norm(t,d) \right) \times coord(q,d) \times queryNorm(q) \quad (3\text{–}1)$$

در این رابطه t نشان‌دهنده واژه، q نشان‌دهنده عبارت پرس‌وجو و d نشان‌دهنده سند است. $tf\ (t \text{ in } d)$ بیانگر مجذور میزان تکرار واژه t در سند d و $idf\ (t)$ معیاری برای تعیین تکرار واژه t در کل اسناد بوده و از طریق رابطه (3–2) محاسبه می‌شود.

$$idf\ (t) = 1 + \log\left(\frac{\text{numDocs}}{\text{docFreq+1}} \right) \quad (3\text{–}2)$$

در رابطه (3–1)، $t.getboost()$ نشانگر وزنی[۱] است که برای فیلدهای مختلف (نظیر عنوان سند، چکیده، نام نویسنده و ...) در هنگام جستجو درنظر گرفته می‌شود. به تعبیری دیگر، اگر بدانیم فیلد خاصی در بافت جستجو اهمیت بیشتری نسبت به بقیه فیلدها دارند می‌توان در زمان پردازش و بازیابی اطلاعات وزن بیشتری را برای آن درنظر گرفت. به عنوان نمونه:

[۱] Boost

جدول ۳-٤- درنظرگیری وزن برای فیلدهای مختلف

اهمیت	عبارت پرس‌وجو	عبارتی که سولر جستجو می‌کند
عنوان مهم‌تر از توضیحات	Title: solr in action Description: solr in action	`title:(solr in action)^2 & description:(solr in action)`
جریمه توضیحات	Title: solr in action Description: solr in action	`title:(solr in action) & description:(solr in action)^0.2`
هر واژه دارای اهمیت خاص		`title:(solr^2 in^0.1 action^1.5)^3`

همچنین در رابطه (3–1)، $norm(t,d)$ وظیفه نرمال‌سازی فیلد[1] را برعهده داشته و براساس اهمیت هر سند $d.getboost()$، اهمیت هر فیلد سند $f.getboost()$ و طول سند $lenghtNorm()$ (رابطه (3–3)) محاسبه می‌شود. اگر فیلدهای یک سند دارای وزن‌های مختلف باشند، $f.getboost()$ از حاصلضرب این وزن‌ها محاسبه می‌شود.

$$norm(t,d) = d.getboost() \times lenghtNorm() \times f.getboost() \qquad (3\text{–}3)$$

عامل $coord(q,d)$ در رابطه (3–1) نشاندهنده میزان ارتباط هر سند با عبارت پرس‌وجو است؛ هرچه سند دارای تعداد تکرارهای بیشتری از عبارت پرس‌وجو باشد، میزان ارتباط بیشتری با عبارت پرس‌وجو خواهد داشت . این عامل با کمک رابطه (3–4) محاسبه می‌شود.

$$coord(q,d) = \frac{\text{num Terms in Document from Query}}{\text{num Terms in Query}} \qquad (3\text{–}4)$$

عامل $queryNorm(q)$ در در رابطه (3–1) نیز این قابلیت را به سولر می‌دهد تا امتیازهای عبارت‌های پرس‌وجو قابل مقایسه باشند. رابطه (3–5) برای محاسبه این عامل استفاده می‌شود.

$$queryNorm(q) = \frac{1}{\left(q.getboost()^2 \times \sum_{t\ in\ q} \left(idf\ (t) \times t.getboost() \right)^2 \right)^{1/2}} \qquad (3\text{–}5)$$

از عامل‌های دیگری نیز می‌توان برای محاسبه ارتباط بین عبارت پرس‌وجوی کاربر و اسناد استفاده کرد. این عوامل در جدول ۴-۷ در بخش ضمائم ارائه شده‌اند. همچنین کلاس‌های محاسبه ارتباط

[1] Field normalization

دیگری نیز در سولر تعریف شده‌اند که می‌توان از آنها به جای DefaultSimilarity استفاده کرد. به این موارد در جدول ۷-۵ بخش ضمائم اشاره شده است.

۵-۲-۳- معیارهای سنجش بازیابی اطلاعات

معیارهای دقت[۱] و بازخوانی[۲] دو معیار مورد استفاده برای سنجش کارایی بازیابی اطلاعات در سولر هستند. دقت نتایج بازیابی شده توسط معیار دقت سنجیده می شود. این معیار بدنبال پاسخ دادن به پرسش «آیا اسناد بازیابی شده همان‌هایی هستند که کاربر به دنبال آن بوده؟» است. به بیانی دیگر، precision معیاری برای سنجش میزان ارتباط نتایج با عبارت پرس‌وجو است و توجهی به میزان دربرگیری کل اسناد مرتبط در پایگاه داده ندارد (not the comprehensiveness of the results). رابطه (3–6) نحوه محاسبه این معیار را نشان می‌دهد.

$$precision = \frac{\#\text{Correct Matches}}{\#\text{Total Results Returned}} = \frac{\text{TP}}{\text{TP+FP}} \qquad (3\text{–}6)$$

معیار بازخوانی جامعیت نتایج بازیابی شده را سنجیده و بدنبال پاسخ دادن به پرسش «چه تعداد سند درست بازیابی شده؟» است. رابطه (3–7) نحوه محاسبه این معیار را نشان می‌دهد.

$$recall = \frac{\#\text{Correct Matches}}{\#\text{Correct Matches} + \#\text{Missed Matches}} = \frac{\text{TP}}{\text{TP+FN}} \qquad (3\text{–}7)$$

هدف در سیستم‌های بازیابی اطلاعات بیشینه کردن دقت و بازخوانی است اما تمرکز اصلی در سولر بر بهبود بازخوانی است و سپس به دقت توجه می‌شود. به عبارت بهتر، ابتدا تمامی نتایج درست بازیابی شده (بازخوانی) و سپس نتایج به گونه‌ای مرتب می شوند که در صفحات نخست مرتبط‌ترین نتایج با عبارت پرس‌وجوی کاربر (دقت) نمایش داده شوند.

اما این دو معیار نمی‌توانند به صورت کامل کارایی سیستم‌های بازیابی اطلاعات را بسنجند؛ به عنوان نمونه کاربر ممکن است بخواهد نتایج نزدیک به محل سکونت خود را بیابید و یا پراکندگی نتایج بازیابی شده و نمایش تمامی حالت‌های ممکن در صفحه نخست مطلوب او باشد. به عبارت دیگر، به عنوان مثال اگر کاربر کلمه hamburger را جستجو کند آیا برای او مطلوب است صفحه نخست نمایش نتایج دربرگیرنده تمامی شعب Burger King باشد و یا اینکه تمامی رستوران‌های

---

[۱] Precision

[۲] Recall

دارای hamburger برای او در صفحه نخست نمایش داده شود. بنابراین محاسبه کارایی سیستم با کمک سنجش میزان ارتباط آن با کلیدواژگان [۱] کاربر تنها یکی از معیارها است و درک مفهوم ارتباط در بافت مورد بررسی و نظر کاربر نیز در این رابطه اهمیت بسیار دارد.

برخی مفاهیم کلیدی دیگر نیز در سولر وجود دارد که در ادامه صرفاً به آنها اشاره می‌شود:

- استفاده از شاخص معکوس[۲] برای جستجو (در این ماتریس هر واژه به یک یا چند سند نگاشته شده و به صورت صعودی به ترتیب الفبایی مرتب شده است.)
- پشتیبانی از منطق بولی در پردازش عبارت‌های پرس‌وجو:
  - AND یا + درصورتی که کاربر بخواهد واژه‌ای حتماً در نتایج جستجو باشد؛
  - OR درصورتی که ضرورتی در وجود واژه در نتایج جستجو نباشد؛
  - NOT یا - درصورتی که کاربر بخواهد واژه‌ای حتماً در نتایج جستجو نباشد؛
  - "..." در صورتی که کاربر به دنبال جستجوی عبارتی خاص باشد.
- Deformalized بودن اسناد: از آنجا که سولر دارای طراحی NoSQL است، خصوصیات اسناد به صورت یکسان و واحد تعریف نمی‌شود. این امر قابلیت مقیاس‌پذیربودن سولر را به دنبال دارد.
- جستجوی توزیع‌شده[۳]: از آنجا که سولر دارای طراحی توزیع شده است، عبارت پرس‌وجو به هسته‌های پردازشی مختلفی که هریک حاوی تعدادی از اسناد پایگاه داده هستند فرستاده شده، نتایج بازیابی و ادغام شده و براساس میزان ارتباط نهایی برای کاربر نمایش داده می‌شوند.

همانگونه که ذکر شد، سولر برای به انجام رساندن جستجو از سه زیرسیستم (۱) مدیریت اسناد، (۲) تحلیل متن و (۳) پردازش پرس‌وجو استفاده می‌کند. این زیرسیستم‌ها در ادامه بررسی خواهند شد.

۳-۳ زیرسیستم مدیریت اسناد

مدیریت اسناد در سولر را می‌توان به صورت فرآیند نمایه‌سازی و ذخیره‌سازی اسناد با گام‌های زیر تعریف کرد (شکل ۳-۶):

---

[۱] Keyword relevancy

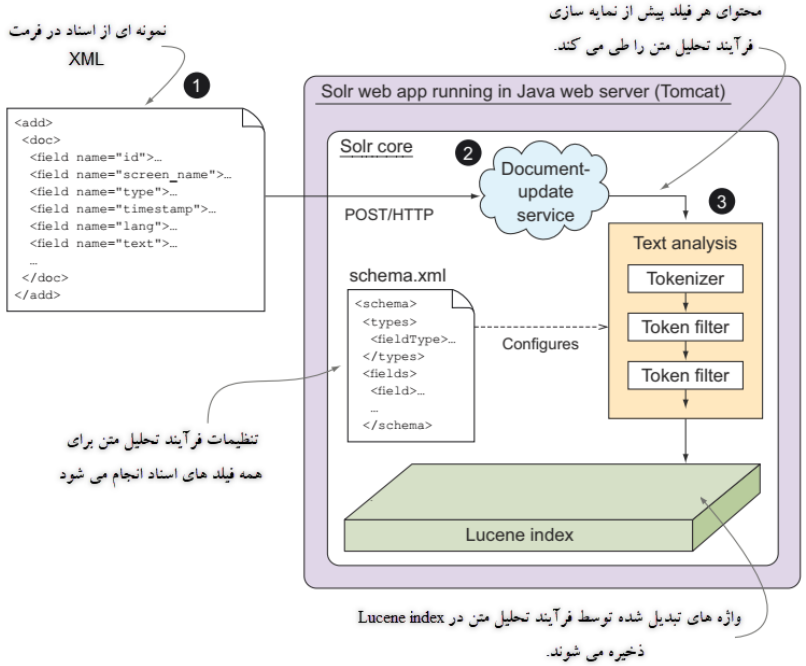
[۲] Inverted index

[۳] Distributed searching

گام ۱: تبدیل سند از فرمت اولیه به فرمت قابل پردازش در سولر مانند XML، CSV و یا JSON (❶)؛

گام ۲: اضافه کردن سند تبدیل شده به سولر از طریق واسط‌هایی نظیر HTTP POST (❷)؛

گام ۳: اعمال تغییرات و تبدیل‌های مورد نیاز در سند در طی نمایه‌سازی (❸).



شکل ۳-۶- زیرسیستم مدیریت اسناد: فرآیند نمایه‌سازی و ذخیره‌سازی اسناد

برای اجرایی سازی گام‌های فوق نیاز است تا تنظیماتی در سولر انجام شود. به این تنظیمات در دو بخش تنظیمات Schema و تنظیمات ورود و بروزرسانی اسناد پرداخته می‌شود. فرآیند تحلیل متن نیز در بخش ۴-۳، تحلیل متن، بررسی می‌شود.

۱-۳-۳- تنظیمات ورود و بروزرسانی اسناد

همانگونه که در شکل ۳-۶ نشان داده شده است، در ابتدا باید فرمت اسناد ورودی به فرمت‌های قابل پردازش در سولر مانند XML، CSV و یا JSON تبدیل شده و سپس از طریق واسط‌هایی به سولر اضافه شوند. علاوه‌بر افزودن، این واسط‌ها وظیفه بروزرسانی اسناد موجود را نیز برعهده دارند. یکی از پرکاربردترین این واسط‌ها، واسط HTTP POST است که از طریق اپلیکیشن post.jar و کلاینت سولر J وظیفه اضافه کردن و بروزرسانی اسناد را برعهده دارد. سه نمونه از سایر واسط‌های پرکاربرد عبارتند از:

- Data Import Handler (DHI): امکان اضافه کردن داده ها را از منابع بیرونی مانند وب‌سایت‌ها و یا پایگاه‌های داده رابطه‌ای پشتیبانی کننده از اتصال مبتنی بر جاوا[۱] (مانند Oracle، Postgres، MySQL) فراهم می‌آورد.
- ExtractingRequestHandler (Solr Cell): امکان اضافه کردن داده های متنی با فرمت‌هایی چون PDF، Ms Office و OpenOffice را فراهم می‌آورد. در پشت صحنه سولرCell از آپاچیTika project برای قطعه‌بندی[۲] اسناد و استخراج متن و فراداده سند استفاده می‌کند.
- Nutch: خزشگر وب مبتنی بر جاوا است و برای قابل جستجو کردن لینک‌های وب در حجم وسیع بکار می‌رود.

واسط‌های ذکر شده درخواست اضافه‌کردن و یا بروزرسانی را به سولر داده و سولر توسط Update Handler این وظیفه را مدیریت می‌کند. خصوصیات این handler در جدول ۷-۶ بخش ضمائم آمده است و به جهت اهمیت، خصوصیت سپردن[۳] اسناد برای نمایه‌سازی در ادامه بررسی بیشتری می‌شود.

به طور معمول، زمانی که یک سند به سولر اضافه می‌شود، تا وقتی نمایه نشده باشد در نتایج جستجو نشان داده نمی‌شود. در سولر4.0 دو نوع سپردن وجود دارد: سپردن معمولی[۴] (سخت) و سپردن نرم[۵].

در سپردن سخت، سولر فرآیند نمایه‌سازی را به یکباره و برای تمامی اسناد جدید انجام داده و سپس اسناد نمایه شده را در محلی به صورت کامل در مکان ذخیره‌سازی طولانی‌مدت (durable storage) ذخیره می‌کند.

سپردن نرم فرآیندی متفاوت دارد؛ پشتیبانی سولر از جستجوی نزدیک به زمان واقعی ایجاب می‌کند تا به محض اضافه شدن یک سند، آن سند قابل جستجو باشد. به عبارت دیگر، اسناد باید در نزدیک به زمان واقعی نمایه شوند. لذا در فرآیند سپردن نرم از بخش‌های زمان‌بر نمایه‌سازی

---

[۱] Java database connectivity (JDBC)

[۲] Fragment

[۳] Commit

[۴] Normal (hard) commit

[۵] Soft commit

چشم‌پوشی شده و اسناد در durable storage ذخیره نمی‌شوند، بلکه در بازه‌های زمانی مشخص شده (مثلاً انتهای هر هفته) عملیات ذخیره‌سازی انجام می‌شود.

برای تعیین زمان آغاز فرآیند سپردن می‌توان از معیارهای زیر استفاده کرد: (۱) نمایه‌سازی هر سند در زمان اضافه شدن، (۲) نمایه‌سازی اسناد زمانی که تعداد اسناد نمایه نشده به حد مشخصی برسند و یا (۳) نمایه‌سازی اسناد در بازه‌های زمانی مشخص شده. همچنین میزان بازدید کاربران از سامانه (تعداد جستجوها) و یا میزان اضافه شدن سند جدید دو عامل کلیدی در تعیین نوع سپردن هستند.

پس از نمایه‌سازی، سولر هر دسته از اسناد را در یک بخش[۱] ذخیره می‌کند و این بخش‌ها ثابت می‌شوند، لذا، دسته جدید اسناد نمایه شده در بخش جدید دیگری ذخیره خواهند شد. در هنگام جستجو، هر دسته به صورت مجزا پردازش شده و نتیجه نهایی از ترکیب نتایج بخش‌ها بدست خواهد آمد. لذا تعداد دسته‌ها و حجم اسناد هر بخش عامل مهمی در تعیین سرعت بازیابی و نمایش اطلاعات است. همچنین اگر سندی حذف شود، صرفاً از بخش مربوطه حذف شده ولی همچنان در durable storage وجود دارد تا زمانی که بخش‌ها برای ذخیره‌سازی نهایی ادغام شده و durable storage بروزرسانی شود.

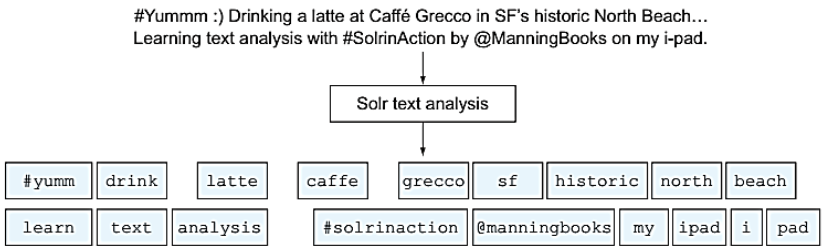
زیرسیستم تحلیل متن دومین زیرسیستم مورد بررسی در پلفترم سولر است. بخش بعد به بررسی این زیرسیستم اختصاص دارد.

۳-۴ زیرسیستم تحلیل متن

تحلیل اسناد موجود در پایگاه داده، اعمال تغییرات و تبدیل‌های مورد نیاز در سند در طی نمایه‌سازی و تحلیل عبارت‌های پرس‌وجو از جمله اهداف اجرای زیرسیستم تحلیل متن است. برخی از ابزارهای مورد استفاده سولر در این زیرسیستم در قالب یک مثال (شکل ۳-۷ و

جدول ۳-۵) نشان داده شده است.

[۱] Segment



شکل ۳-۷- مثال تحلیل متن

جدول ۳-۵- متن اولیه مثال شکل ۳-۷، تغییرات اعمال شده و ابزارهای سولر برای تغییرات

ابزارهای تحلیل سولر	تغییرات اعمال شده	متن اولیه
LowerCaseFilterFactory	تبدیل به حروف کوچک	تمامی واژه‌ها
StopFilterFactory	حذف ایست واژه[۱]	a, at, in, with, by, on
KStemFilterFactory	drink, learn (ریشه‌یابی[۲])	Drinking, learning
WordDelimiterFilterFactory	sf, san francisco	SF’s
ASCIIFoldingFilterFactory	caffe	Caffé
WordDelimiterFilterFactory	ipad, i pad	i-Pad
PatternReplaceCharFilterFactory	#yumm	#Yummm
WhitespaceTokenizer WordDelimiterFilterFactory	#solrinaction, @manningbooks	#SolrInAction, @ManningBooks

فرآیند تحلیل متن در سولر شامل دو مرحله است: (۱) واژه–واژه کردن[۳] (تجزیه) و (۲) فیلتر کردن واژه‌های تجزیه شده[۴]:

- مرحله ۱: واژه–واژه کردن

در این مرحله متن به مجموعه‌ای از واژه‌ها تجزیه می‌شود. StandardTokenizer یکی از معمولی‌ترین و پرکاربردترین ابزارها است که در طی آن واژه‌ها بر اساس فاصله و نشانه‌ها جدا شده و آدرس‌های اینترنتی، رایانامه‌ها و سرنام‌ها مدیریت می‌شوند. نمونه‌ای از این تغییرات به همراه نحوه فراخوانی آن در شکل ۳-۸ نمایش داده شده است.

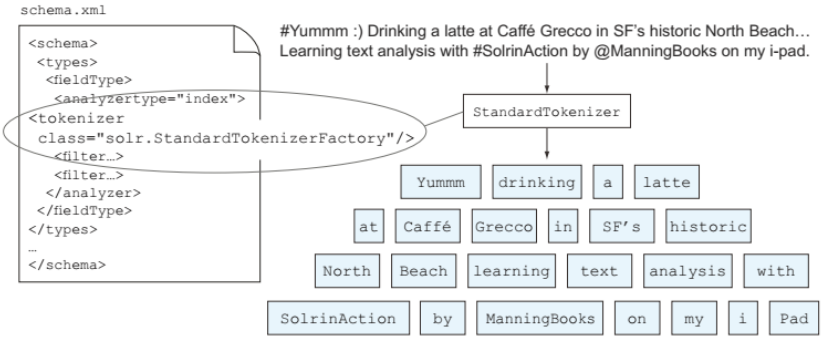
---

۱ Stop words

۲ Stemming

۳ Tokenization

۴ Token filtering

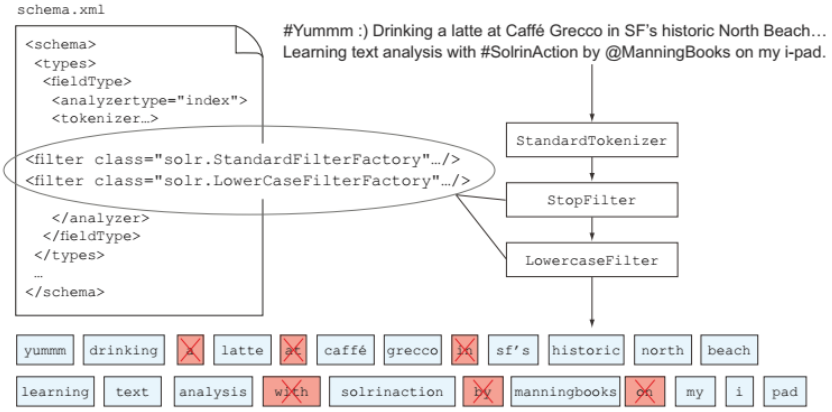


شکل ۳-۸- مثالی از StandardTokenizer

□ مرحله ۲: فیلتر کردن واژه‌های تجزیه شده

پس از جداسازی واژه‌ها در متن گام بعدی فیلتر کردن واژه‌های تجزیه شده (حذف ایست‌واژه‌ها، تبدیل حروف بزرگ به کوچک و ...) است. برای این منظور فیلترهای مختلفی در سولر طراحی شده است، نظیر:

- StopFilterFactory برای حذف ایست‌واژه‌ها بکار می‌رود. لیست این کلمات باید در فایل txt جداگانه تعریف شده و در Schema فراخوانی شود.
- LowerCaseFilterFactory برای تبدیل حروف بزرگ به کوچک بکار می‌رود. نتیجه اعمال این دو فیلتر در مثال شکل ۳-۸ به همراه نحوه فراخوانی در شکل ۳-۹ نمایش داده شده است.



شکل ۳-۹- اعمال دو فیلتر StopFilterFactory **و** LowerCaseFilterFactory **در مثال شکل ۳-۸**

- WordDelimiterFilterFactory برای حذف نشانه‌ها (مانند "; : , .") و مدیریت کاراکترهای خاص (مانند "! @ #") بکار می‌رود (شکل ۳-۷).

- ASCIIFoldingFilterFactory برای تبدیل حروف به کدهای ASCII بکار می‌رود و بهتر است بعد از LowerCaseFilterFactory اعمال شود. از این فیلتر برای تبدیل حروف لاتین استفاده می‌شود.
- KStemFilterFactory برای ریشه‌یابی متون انگلیسی بکار می‌رود. سایر فیلترهای ریشه‌یابی در بخش بعد تشریح خواهند شد.
- PatternReplaceCharFilterFactory برای جایگزین کردن الگوهای نوشتاری بکار می‌رود (به عنوان مثال جایگزین کردن #Yamm با #Yam در شکل ۳-۷). بهتر است این فیلتر قبل از شروع فرآیند واژه–واژه کردن اعمال شود و درواقع نوعی پیش‌پردازش محسوب می‌شود.
- HTMLStripCharFilterFactory برای حذف کردن آدرس‌های اینترنتی از متن بکار می‌رود و بهتر است قبل از شروع فرآیند واژه–واژه کردن اعمال شود.
- MappingCharFilterFactory برای جایگزین کردن کاراکترهای متن با کاراکترهای از قبل تعیین شده بکار می‌رود.
- SynonymFilterFactory برای اضافه و یا جایگزین‌کردن کلمات مترادف بکار می‌رود و بهتر است در آخرین حلقه زنجیره پردازش متن و صرفاً در هنگام پردازش عبارت پرس‌وجو استفاده شود. فایل txt کلمات مترادف باید از قبل تعریف شده و در Schema فراخوانی شود. برای تعریف این فایل می‌توان از پایگاه داده WordNet نیز استفاده نمود.

یکی از اصلی‌ترین اقدامات در طی فرآیند تحلیل متن یافتن ریشه کلمات یا ریشه‌یابی است. به روش‌های مورد استفاده در سولر برای این منظور در ادامه پرداخته می‌شود.

۱-۴-۳- ریشه‌یابی

جزئی از واژه پس از حذف وندهای آن ریشه نامیده شده و ریشه‌یابی را می‌توان یکی از راهکارهای بهبود کارایی سیستم‌های بازیابی اطلاعات دانست. درواقع ریشه‌یابی باعث بهبود شاخص بازخوانی می‌شود و اما می‌تواند بر شاخص دقت اثر منفی داشته باشد.

فیلترهای ریشه‌یابی متعددی در سولر طراحی شده است که از آن میان می‌توان به ریشه‌یاب Krovets با دستور KStemFilterFactory، ریشه‌یاب Porter با دستور

PorterStemFilterFactory و ریشه‌یاب EnglishMinimalStemFilter اشاره کرد. مقایسه‌ای از عملکرد این ریشه‌یاب‌ها در جدول ۳-۶ آمده است.

جدول ۳-٦- عملکرد ریشه‌یاب‌های PorterStemFilter، KStemFilter **و** EnglishMinimalStemFilter

واژه اصلی	PorterStemFilter	KStemFilter	EnglishMinimalStemFilter
country	countri	country	country
countries	countri	country	country
dogs	dog	dogs	dog
runs	run	runs	run
running	run	running	running
association	associ	association	association
associates	associ	associate	associate
listing	list	list	listing
investing	invest	invest	investing
investments	invest	investment	investment
investors	investor	invest	investor
organizations	organ	organization	organization
organize	organ	organize	organize
organic	organ	organic	organic
generous	gener	generous	generous
generals	gener	general	general
generalizations	gener	generalization	generalization

ریشه‌یاب Porter نسخه دیگری نیز دارد که با دستور SnowballPorterFilter فراخوانی می شود. تفاوت این دو نسخه در سرعت و دقت آنها ست؛ نسخه اصلی سریع‌تر بوده و برای زبان انگلیسی پیشنهاد می‌شود اما نسخه دوم (Snowball) دقیق‌تر عمل می‌کند. اگر پارامتر زبان در نسخه دوم به صورت Lovins تنظیم شود، این دو نسخه مشابه هم عمل می‌کنند[۱] و اگر پارامتر زبان به صورت English تنظیم شود، نسخه دوم بر اساس الگوریتم بهبود یافته Porter عمل خواهد کرد[۲].

گاهی در ریشه‌یابی حالات خاصی رخ می‌دهد، به عنوان نمونه:

- □ گاهی اسامی خاص و یا واژگان تخصصی وجود دارند که روش‌های ریشه‌یابی نباید تغییراتی در آنها اعمال کند. بدین منظور لیستی از اسامی محافظت شده ایجاد می‌شود. برای این منظور در زنجیره پردازش متن و قبل از فراخواندن ریشه‌یاب، دستور

[۱] http://snowball.tartarus.org/algorithms/lovins/stemmer.html

[۲] http://snowball.tartarus.org/algorithms/english/stemmer.html

KeywordMarkerFilterFactory اجرا شـده و آدرس فایل حاوی اسـامی محافظت شده در قالب txt به سولر داده شود.

- گاهی نیاز است ریشه‌یاب، برخی از ریشه‌های بدست آمده را به واژگانی خاص نگاشت کند، این امر توسط دستور StemmerOverrideFilterFactory محقق می شود. این دسـتور مانند دسـتور KeywordMarkerFilterFactory نیازمند لغت‌نامه‌ای در قالب فایل txt است که نگاشت‌ها در آن تعریف شده باشد.

نمونه‌ای از نحوه بکارگیری این دو دستور در Schema به صورت زیر است:

```
<fieldType name="text_en_kstem_with_overrides"
class="solr.TextField"
positionIncrementGap="100">
<analyzer>
<tokenizer class="solr.StandardTokenizerFactory"/>
<filter class="solr.StopFilterFactory"
ignoreCase="true"
words="lang/stopwords_en.txt"/>
<filter class="solr.LowerCaseFilterFactory"/>
<filter class="solr.EnglishPossessiveFilterFactory"/>
<filter class="solr.KeywordMarkerFilterFactory"
protected="protwords.txt"/>
<filter class="solr.StemmerOverrideFilterFactory"
dictionary="stemdict_en.txt" />
<filter class="solr.KStemFilterFactory"/>
</analyzer>
</fieldType>
```

protwords.txt

```
listing
manning
```

stemdict_en.txt

```
dogs    dog
runs    run
investors       investor
```

با استفاده از این دو دستور در زنجیره تحلیل متن می‌توان از بسیار از اشتباهات بوجود آمده در اثر اجرای ریشـه یابی جلوگیری نمود. با این حال مشـکلاتی نیز وجود دارند و اسـتفاده ناکارا از روش‌های ریشه‌یابی می‌تواند تاثیرات نامطلوبی بر معیارهای دقت و بازخوانی داشته باشد. به عنوان مثال:

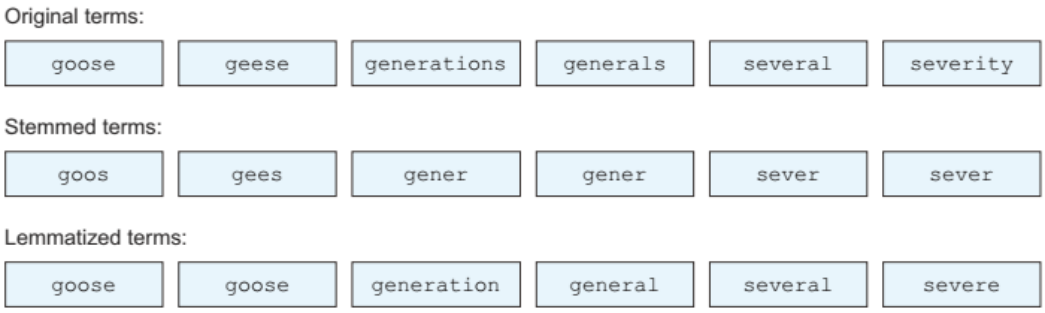
- اگر تمام حالت‌های یک واژه به یک ریشه ختم نشود (مانند نتایج روش Porter برای واژه‌های dry و dries که برای واژه اول ریشه dry و برای واژه دوم ریشه dri حاصل می‌شود) نتایج بازیابی کامل نبوده و مقدار بازخوانی کاهش می‌یابد.

مشکلی با نام overmatching**، مانند نتایج روش** Porter **برای واژه‌های** generous، generals **و** generalizations **در**

- جدول **۳-۵**، سبب می شود تا نتایج خوبی بازیابی نشده و دقت نتایج بازیابی شده کاهش یابد.

برای برطرف نمودن این کاستی‌ها دو راهکار کلی پیشنهاد شده است:

- استفاده از ریشه‌یاب‌های کمتر تهاجمی مانند `EnglishMinimalStemFilter` بجای روش‌های تهاجمی‌تر مانند `PorterStemFilter` (روش‌های تهاجمی بازخوانی بالاتری داشته و روش‌های کمتر تهاجمی دقت بالاتری دارند).
- استفاده از روش‌های بُن‌واژه‌سازی[۱]. هدف در هر دو دسته روش‌های ریشه‌یابی و بُن‌واژه‌سازی دستیابی به ریشه واژه‌هاست با این تفاوت که روش‌های ریشه‌یابی، روش‌هایی مبتنی برالگوریتم بوده و روش‌های بُن‌واژه‌سازی روش‌هایی مبتنی بر لغت‌نامه هستند (هرچند یافتن لغت‌نامه جامع دشوار و در برخی موارد غیرممکن است). شکل ۳-۱۰ نمونه‌هایی از نتایج متفاوت بکارگیری این دو دسته از روش‌ها را نمایش می‌دهد.



شکل ۳-۱۰- نمونه‌هایی از نتایج متفاوت بکارگیری روش‌های ریشه‌یابی و بُن‌واژه‌سازی

تعادل بین دقت و بازخوانی می‌تواند به عنوان معیاری برای تصمیم‌گیری در مورد بکارگیری روش‌های ریشه یابی و یا بُن واژه سازی باشد. عدم استفاده از ریشه یابی سبب می‌شود تا دقت نتایج افزایش یابد و استفاده از آن باعث بهبود بازخوانی و کاهش دقت خواهد شد. اگر هر دو این

[۱] Lemmatization

معیارها در سیستم بازیابی مهم باشند بهتر است از روش‌های بُن واژه سازی و یا روش‌های پیشرفته‌تر نظیر پردازش زبان طبیعی استفاده کرد.

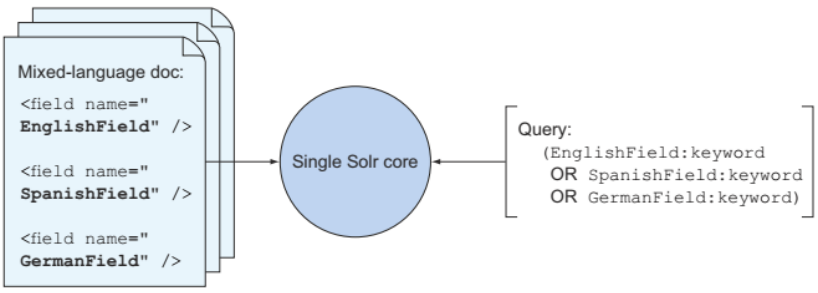
یکی دیگر از موارد موثر در انتخاب و درنتیجه نتایج روش‌های بازیابی اطلاعات زبان اسناد مورد بررسی است و توانایی تحلیل زبان‌های مختلفی در سولر طراحی شده است.

۲-۴-۳- زبان‌های پیش‌فرض در سولر

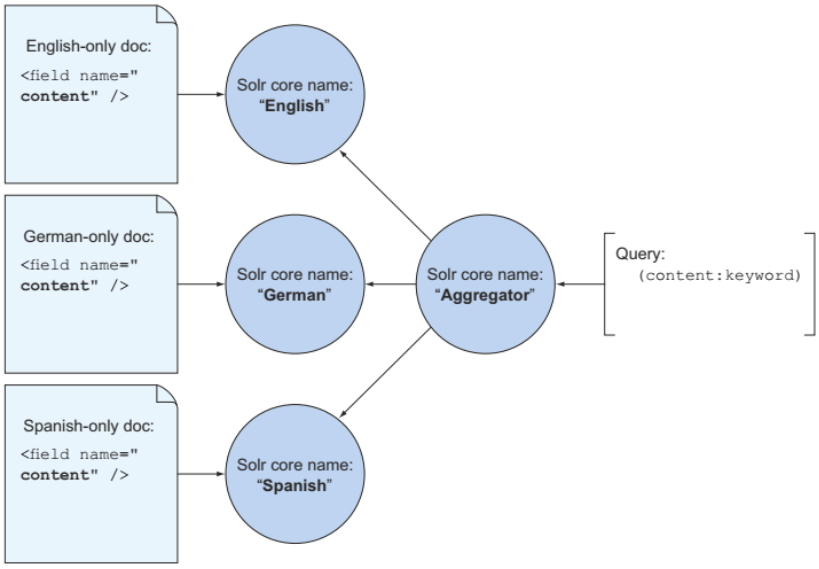
سولر به صورت پیش‌فرض توانایی تحلیل ا سناد با زبان‌های مختلفی را دارد نظیر انگلیسی، عربی، لاتین، هندی، ژاپنی، چینی و ... . د ستورات زنجیره تحلیل متن به زبان فار سی در ادامه آمده ا ست که این دستورات باید در Schema در بخش تحلیل متن و تحلیل عبارت پرس‌وجو فراخوانی شوند (برای توضیح بیشتر به بخش ۳-۵، سند Schema سامانه گنج مراجعه شود).

```
PersianCharFilterFactory
StandardTokenizerFactory
LowerCaseFilterFactory
ArabicNormalizationFilterFactory
PersianNormalizationFilterFactory
StopFilterFactory                [words="lang/stopwords_fa.txt"]
```

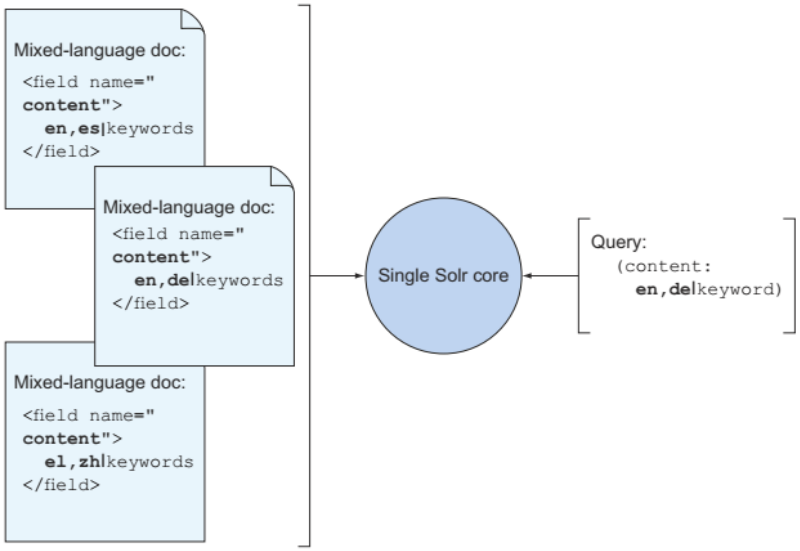
همچنین ممکن است کاربر عبارت پرس‌وجوی خود را در زبان های مختلف بیان کند. برای پردازش چنین عبارت‌هایی راهکارهایی وجود دارد: (۱) درنظرگیری فیلدهای جداگانه براساس هر زبان برای هر سند (شکل ۳-۱۱)، (۲) دسته‌بندی اسناد براساس زبان و نمایه‌سازی جداگانه آنها (شکل ۳-۱۲) و (۳) تشخیص خودکار زبان سند (شکل ۳-۱۳).



شکل ۳-۱۱- درنظرگیری فیلدهای جداگانه در تحلیل‌های چند زبانی



شکل ۳-۱۲- دسته‌بندی اسناد براساس زبان در تحلیل‌های چند زبانی



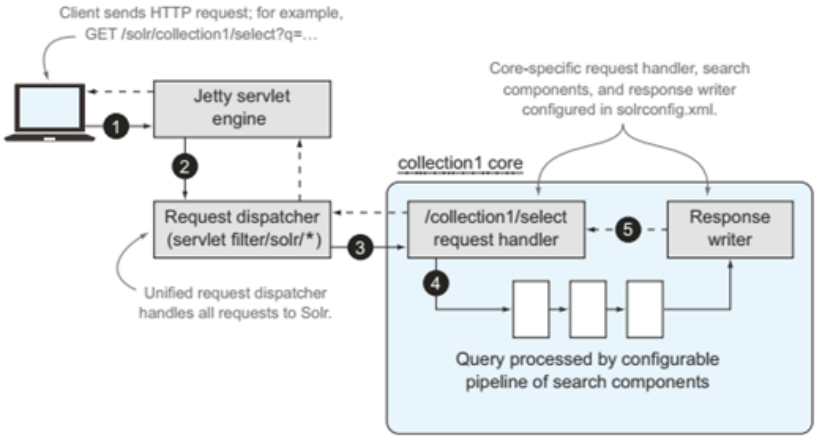
شکل ۳-۱۳- تشخیص خودکار زبان سند در تحلیل‌های چند زبانی

درمورد راهکار ۱ و ۳ نیازی به درنظرگیری هسته‌های پردازش موازی و جداگانه برای سولر نیست اما معماری سیستم بر اساس راهکار ۲ پیچیده بوده و در بسیاری از موارد باید هسته‌های پردازشی برای هر زبان درنظر گرفته شود. همچنین پیاده‌سازی راهکار ۳ نیازمند استفاده از روش‌های پیچیده در تحلیل متن است.

سومین زیرسیستم طراحی شده در سولر، زیرسیستم پردازش پرس‌وجو است که وظیفه مدیریت درخواست‌های کاربر را از سیستم جستجو برعهده دارد.

۳-۵ زیرسیستم پردازش پرس‌وجو

نمای کلی زیرسیستم پردازش پرس‌وجو در قالب مدیریت درخواست[۱] در سولر در شکل ۳-۱۴ شده است. در این شکل درخواستی از سمت کلاینت به صورت HTTP به موتور سرولت Jetty فرستاده می‌شود (❶). پس از قبول این درخواست، Jetty آن را به توزیع کننده درخواست سولر[۲] می‌فرستد (❷). در گام بعد توزیع کننده درخواست هسته collection1 را شناسایی کرده و مدیریت درخواست دستور `select`/ را از Schema بازخوانی می‌کند (❸). این handler درخواست را بر اساس فرآیند ارائه شده در شکل ۳-۱۷ پردازش کرده (❹) و نتایج را برای نمایش در کلاینت تنظیم می‌کند (❺).



شکل ۳-۱۴- نمای کلی مدیریت درخواست جستجو

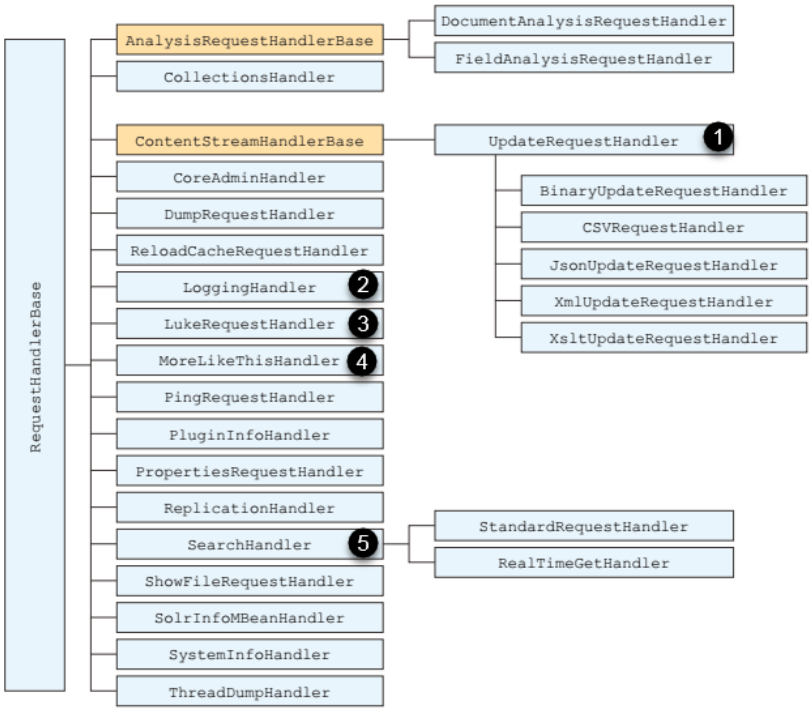
بخش مدیریت درخواست از کلاس‌های متعددی تشکیل شده‌اند که ساختار درختی آنها در شکل ۳-۱۵ نمایش داده‌شده و خصوصیات تمامی این کلاس‌ها باید در ابتدا و در Schema تعیین گردند. به عنوان نمونه، بروزرسانی نمایه‌ها از طریق کلاس `UpdateRequestHandler` (❶) و از طریق handler ۵ زیرمجموعه انجام می‌شود. همچنین ذخیره‌سازی لاگ از طریق `LoggingHandler` (❷)، گزارش فراداده مربوط به اسناد از طریق `LukeRequestHandler` (❸) و امکان مشاهده اسناد مشابه با سند انتخابی کاربر[۳] از طریق `MoreLikeThisHandler` (❹) ممکن می‌شود.

---

۱ Request handler

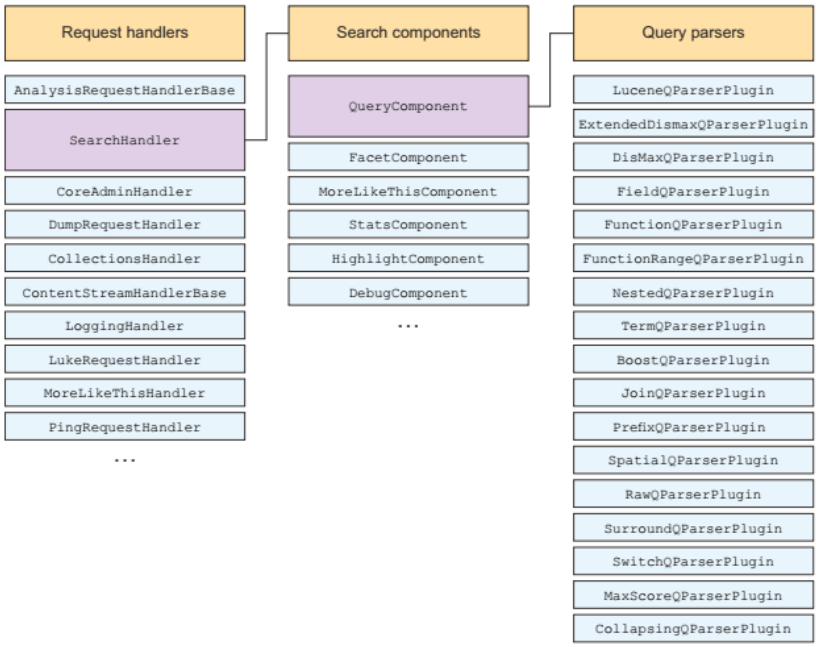
۲ Solr request dispatcher

۳ More like this

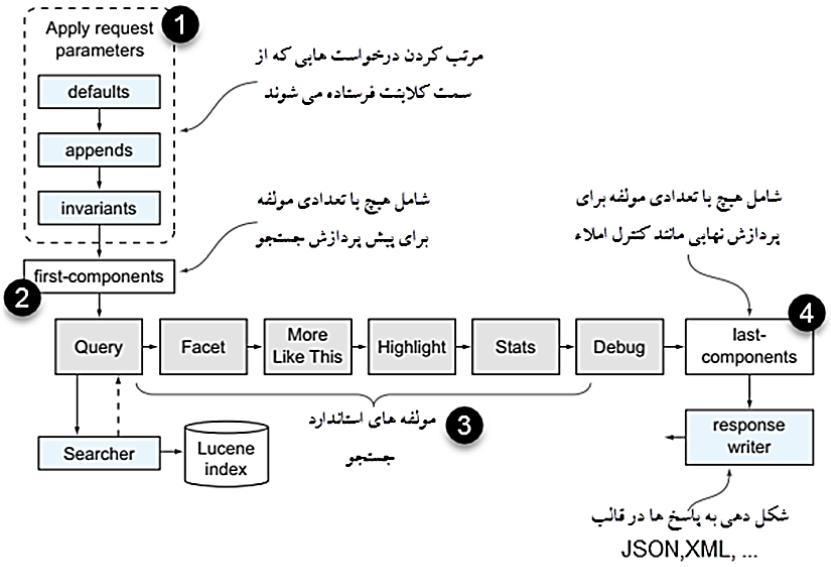


شکل ۳-۱۵- کلاس‌های مدیریت درخواست در سولر

SearchHandler (❺) وظیفه مدیریت جســتجوی کاربر را برعهده دارد و به جهت اهمیت این موضـوع در این پژوهش در ادامه بر آن تمرکز خواهد شـد. نمای درختی کلاس‌های این handler در شکل ۳-۱۶ و فرآیند کلی آن در شکل ۳-۱۷ نمایش داده شده است.



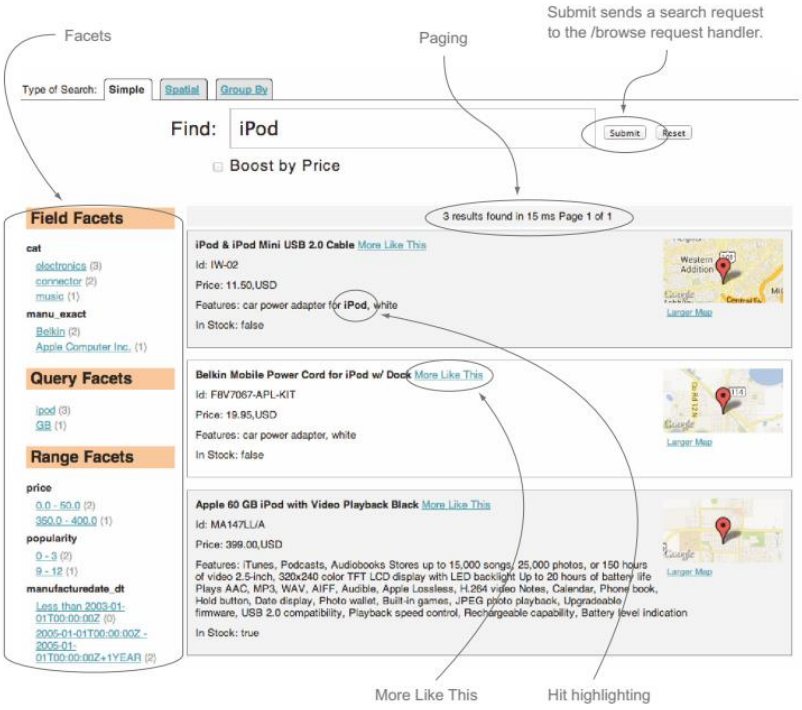
شکل ۳-۱۶- نمای درختی کلاس‌های SearchHandler



شکل ۳-۱۷- فرآیند کلی SearchHandler

بخش ❶ در شکل ۳-۱۷ وظیفه تنظیم پارامترهای اولیه و مرتب کردن درخواست‌هایی که از سوی کلاینت فرستاده می شود را برعهده دارد. اعمال تنظیمات و مقادیر پیش‌فرض، در صورتی که کابر تغییراتی را اعمال نکرده باشـــد، در default و ثابت کردن برخی از پارامترها در invariants انجام شـده و در بخش append پارامترهای دیگری به موارد مورد نظر کاربر افزوده می‌شـود. بخش ❷، First-components، وظیفه پیش‌پردازش را برعهده داشـته و به عنوان بخشـی اختیاری در زنجیره جستجو طراحی شده است. بخش ❸ شامل مولفه‌های جستجو بوده و حداقل یکی از این موارد بایستی در فرآیند جستجو فعال شده باشد (مولفه Query به صورت پیش‌فرض فعال است). بخش ❹، Last-component، مولفه‌ای اختیاری در زنجیره بوده و برای فعالیت های پس از پردازش طراحی شده است.

نمونه‌ایی از کاربرد مولفه‌های بخش ❸ در شـکل ۳-۱۸ نمایش داده شـده و در ادامه به صـورت تفصیلی به آنها پرداخته خواهد شد.



شکل ۳-۱۸- نمونه‌ای از کاربرد مولفه‌های بخش ۳ شکل ۳-۱۷

۳-۵-۱- مولفه QUERY

این مولفه هسته اصلی فرآیند مدیریت درخواست‌ها بوده و وظیفه یافتن کلیه اسناد مرتبط با عبارت پرس‌وجوی کاربر را با توجه به نمایه‌های موجود در لوسن برعهده دارد. به عبارت دیگر، در این مولفه با کمک تجزیه‌گر عبارت پرس‌وجو[۱]، عبارت مورد نظر کاربر به واژگان مورد نیاز تبدیل شده و اسناد مرتبط بازیابی می‌شوند.

در هنگام استفاده از این مولفه، دو پارامتر q (عبارت پرس‌وجوی کاربر (query)) و fq (فیلترهای اعمال شده بر عبارت پرس‌وجو (filter query)) باید تعریف شود. تعریف این دو پارامتر به صورت مجزا مزیت‌هایی را به‌همراه دارد:

- هدف از پارامتر q محدود کردن نتایج جستجو به اسناد مرتبط و فراهم آوردن لیستی از واژگان مرتبط برای محاسبه ارتباط اسناد با عبارت پرس‌وجو بوده درحالی که هدف از پارامتر fq تنها محدود کردن نتایج جستجو به اسناد مرتبط است. به عبارت دیگر، اگر پرس‌وجوی کاربر به صورت زیر باشد

```
keywords: “solr in action”, category: “technology”
```

[۱] Query parser

پارامتر $q = solr\ in\ action$ و پارامتر $fq = technology$ خواهد بود و درواقع پارامتر q علاوه بر محدود کردن نتایج جستجو به اسناد مرتبط، لیستی از واژگان مرتبط را نیز فراهم می‌کند.

- □ اغلبِ کاربران از fqهای ثابتی در دفعات جستجو[۱]ی خود استفاده کرده و تنها q را تغییر می‌دهند. لذا جدا بودن این دو پارامتر باعث بهبود سرعت پردازش و کاهش حجم محاسبات در سنجش میزان ارتباط یک سند با عبارت پرس‌وجو خواهد شد.

ترتیب پردازش q و fq نیز اهمیت دارد؛ اول باید q را پردازش کرد و سپس fq را اعمال کرد؟ و یا برعکس؟. پاسخ دادن به این پرسش بستگی به زمینه کاری موتور جستجو و بافت مسئله دارد اما فرآیند کلی در سولر به صورت زیر است:

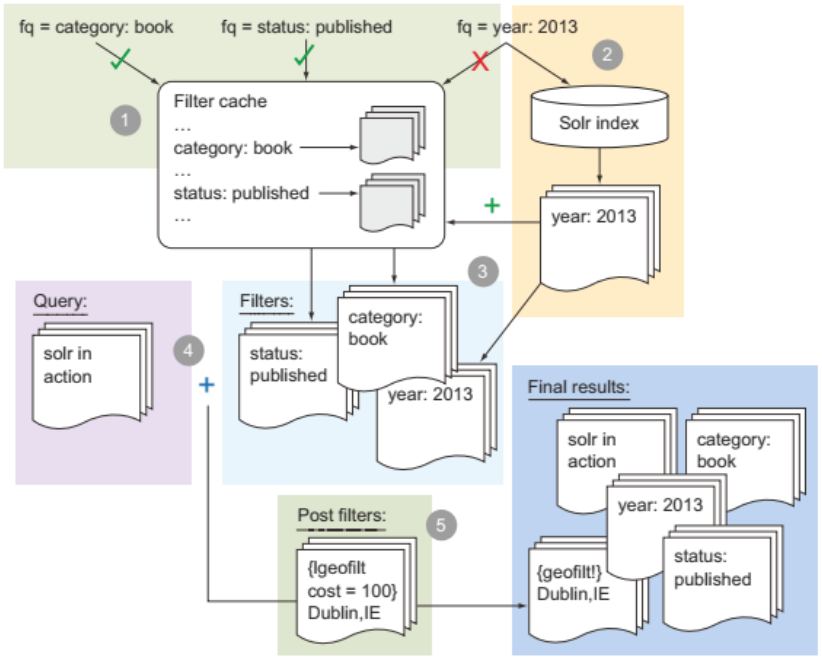
۱. ابتدا fqها به صورت مجزا بر مجموعه اسناد موجود در cache سیستم اعمال شده و مجموعه DocSet برای هر fq تشکیل می‌شود.
۲. اگر گام ۱ نتیجه‌ای دربرنداشت، fqها به صورت مجزا بر مجموعه اسناد موجود در پایگاه داده اعمال شده و مجموعه DocSet برای هر fq تشکیل می‌شود.
۳. نتایج برای هر fq با رابطه AND تجمیع شده و یک DocSet واحد تشکیل می‌شود.
۴. پارامتر q بر نتایج موجود در DocSet اعمال شده و امتیاز ارتباط محاسبه می‌شود.
۵. اگر کاربر فیلتر نهایی[۲] تعریف کرده باشد، این فیلتر در بازنمایی نتایج اعمال خواهد شد.

گام‌های فوق برای پارامترهای زیر در شکل ۳-۱۹ نمایش داده شده‌اند.

```
q=solr in action &
fq=category:book & fq=status:published & fq=year:2013 &
post filter= {!geofilt tag="Dublin,IE” cost=100 sfield=location}
```

[۱] Search session

[۲] Post filter



شکل ۳-۱۹- گام‌های پردازش q و fq

همانگونه که ذکر شد، لوسن و سولر برای تحلیل اسناد پایگاه داده و تبدیل عبارت پرس‌وجوی کابر به واژگان مورد نیاز از تجزیه‌گرها ا ستفاده می‌کنند. دو تجزیه‌گر پرکاربرد عبارتند از: لو سن query parser و eDisMax query parser.

LUCENE QUERY PARSER

سولر به صورت پیش‌فرض از Lucene query parser استفاده می‌کند که از طریق کلاس `LuceneQParserPlugin` بازخوانی می‌شود. خصوصیات اصلی این کلاس به شرح زیر است:

- اگر کاربر فیلد جستجو را مشخص نکرده باشد، جستجو به صورت پیش‌فرض در فیلد محتوا (content) انجام می شود. بنابراین دو پرس‌وجوی «solr» و «content:solr» یکسان بوده و نتایج دو پرس‌وجوی «title:apache solr » و «title:apache content:solr» متفاوت خواهند بود.
- عملگرهای +، AND و && برای تعیین کلمات ضروری در نتایج جستجو و عملگرهای OR و || برای کلمات اختیاری درنظر گرفته شده‌اند. همچنین در حالت پیش‌فرض وجود «فاصله» در بین کلمات جستجو به معنای OR است. این امر سبب می‌شود تا بازخوانی افزایش و دقت کاهش یابد (البته این حالت را می‌توان تغییر داد).
- جستجوی دقیق عبارت پرس‌وجو با " ..." انجام می‌شود.

مانند آنچه در بخش۳-۲-۳، تطابق فازی، اشاره شده نزدیکی دو واژه و یا اصلاح واژه براساس فاصله با کمک ~ تعیین می‌شود.

- □ جستجوی Wildcard با عملگرهای * و ؟ انجام می‌شود.

این کلاس دارای محدودیت‌هایی نیز هست:

- □ به عملگرهای مورد استفاده کاربر در پرس‌وجو بسیار حساس است و از آنجا که نمی‌توان انتظار داشت همه کاربران از عملگرهای درست استفاده کنند، در اغلب کاربردها کارایی پایین‌تر از انتظار دارد.
- □ این تجزیه‌گر قابلیت جستجو در تمامی فیلدها را نداشته و درنتیجه کارایی آن کاهش می‌یابد (البته می‌توان از فیلدهای کپی (بخش Schema۰) استفاده نمود).

به منظور رفع این کاستی‌ها و بکارگیری تجزیه‌گر قوی‌تر، سولر با ترکیب دو تجزیه‌گر Lucene query parser و disjunction maximum (DisMax) تجزیه‌گر جدیدی را با نام Extended disjunction maximum (eDisMax) ارائه نموده است که می‌توان آن را به صورت کلان و کلی در Schema و یا به صورت محلی فراخوانی نمود.

EDISMAX QUERY PARSER

تجزیه‌گر eDisMax نسخه توسعه یافته DisMax بوده و نسبت به Lucene query parser از حساسیت کمتری نسبت به عملگرهای پرس‌وجو برخوردار است. همچنین قابلیت جستجو در تمامی فیلدها را دارد و برای روشن‌تر شدن بحث فرض کنید بخواهیم عبارت «Solr in Action» را در تمامی اسناد پایگاه داده در سه فیلد title، description و author جستجو کنیم. در حالتی که از Lucene query parser استفاده شود عبارت پرس‌وجو به صورت زیر خواهد بود:

```
(((title:solr) OR (description:solr) OR (author:solr)) AND
((title:in) OR (description:in) OR (author:in)) AND ((title:action)
OR (description:action) OR (author:action)))
```

اما در حالتی که از eDisMax استفاده شود عبارت پرس‌وجو به صورت زیر خواهد بود و سه فیلد فوق را می‌توان در پارامتر فیلد عبارت پرس‌وجو[۱] (*qf*) تعریف کرد:

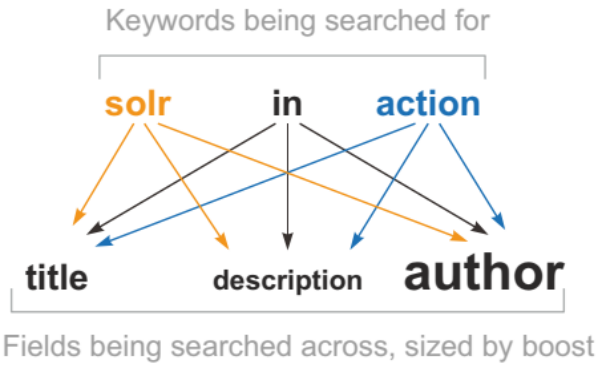
```
q=solr in action&qf=title description author
```

[۱] Query field

همچنین می‌توان برای هر فیلد ضریب اهمیت (وزن) جداگانه‌ای نیز تعریف نمود:

```
q=solr in action&qf=title^1.5 description author^2
```

عبارت پرس‌وجوی فوق به صورت نمادین در شکل ۳-۲۰ نشان داده شده است.



شکل ۳-۲۰- نمایش نمادین عبارت پرس‌و جو در eDisMax در حالت تفاوت وزن فیلدها

یکی از قابلیت‌های کلیدی eDisMax درنظرگیری نزدیکی کلمات پرس‌وجوی در محاسبه امتیاز سند بازیابی شده است. هرچه کلمات در اسناد بازیابی شده به یکدیگر نزدیک‌تر باشند امتیاز بیشتری به سند تعلق خواهد گرفت. این امتیازدهی با کمک پارامتر فیلد عبارت[۱] در سه نوع PF، $PF2$ و $PF3$ انجام می‌شود. درصورت تنظیم پارامتر PF، سولر دقیقاً عبارت پرس‌وجو را جستجو می‌کند (exact matching)، با تنظیم پارامتر $PF2$، سولر عبارت پرس‌وجو را به ۲-گرم شکسته و عملیات بازیابی را با کمک عبارت(های) جدید انجام می‌دهد. به طور مشابه با تنظیم پارامتر $PF3$، سولر عبارت پرس‌وجو را به ۳-گرم شکسته و عملیات بازیابی را با کمک عبارت(های) جدید انجام می‌دهد. به بیانی دیگر، درصورتی که پارامترهای جستجو به صورت زیر تنظیم شده باشند:

```
defType=edismax&
q=big data analytics&
qf=content&
pf=content^10
```

سولر به دنبال یافتن عبارت «big data analytics» در فیلد content بوده و ضریب وزنی آن برابر ۱۰ است. حال اگر:

```
Pf2=content^10
```

[۱] Phrase field

در این‌صورت سولر به دنبال یافتن عبارت‌های «big data» و «data analytics» در فیلد content خواهد بود و به طور مشابه اگر:

```
Pf3=content^10
```

سولر به دنبال یافتن عبارت «big data analytics» در فیلد content است.

پارامترهای دیگری چون فاصله کلمات[۱] ($PS, PS2, PS3$)، با هدف اصلاح املایی کلمات)، گره بازکن[۲] (TIE، با هدف تعیین نحوه امتیازدهی در صورتی که عبارت پرس‌وجو در بیش از یک فیلد یافت شود) و تقویت عبارت پرس‌وجو[۳] (bq)، با هدف دادن وزن بیشتر به اسنادی که با این پارامتر منطبق هستند) برای تابع eDisMax قابل تنظیم بوده و در صورت اعمال درست، کارایی تجزیه‌گر افزایش می‌یابد.

پارامتر حداقل انطباق[۴] (mm) پارامتری است که حداقل تعداد قطعی حضور عبارت پرس‌وجو در سند را برای اینکه به عنوان نتیجه برای کاربر نمایش داده شود مشخص می‌کند. با تنظیم این پارامتر می‌توان مقادیر بازخوانی و دقت را براساس نیاز تنظیم نمود.

نکته مهم دیگر، نحوه امتیازدهی به اسناد در این تجزیه‌گر در مقایسه با Lucene query parser است. اگر عبارت پرس‌وجو برای این دو تجزیه‌گر به صورت زیر باشد:

```
q=title:solr OR content:solr
q={!edismax qf=title content}solr
```

در تجزیه‌گر لوسن امتیاز نهایی یک سند برابر است با مجموع امتیاز آن سند برای فیلدهای title و content. در حالی‌که امتیاز نهایی یک سند برای eDisMax برابر است با بیشینه امتیاز آن سند در فیلدهای title و content. به بیانی دیگر، در تجزیه‌گر لوسن امتیاز یک سند بمعنای میزان ارتباط عبارت پرس‌وجو با آن سند، به عنوان یک موجودیت واحد، است درحالی که در تجزیه‌گر eDisMax، امتیاز یک سند بمعنای بیشینه ارتباط عبارت پرس‌وجو با یک یا چند فیلد مشخص شده سند است. مشکلی که این نوع امتیازدهی در روش eDisMax می‌تواند ایجاد کند آن است که فرض کنید جستجو برای فیلد عنوان و چکیده تنظیم شده باشد، حال اگر در سندی چکیده با متن سند تطابق مناسبی نداشته باشد، در صورت تطابق عبارت پرس‌وجو و چکیده سند امتیاز بالاتری

---

۱ Phrase slope

۲ Tie breaker

۳ Boost query

۴ Minimum match

خواهد گرفت نسبت به سندی که چکیده آن انطباق کم‌تر و متن آن منطبق‌تر با عبارت پرس‌وجو است. لذا در انتخاب بین این دو تجزیه‌گر، بافت و نوع اسناد موجود در پایگاه داده و نوع تحلیل مورد نیاز اهمیت بسیار دارد.

تجزیه‌گرهای دیگری نیز در سولر طراحی شده‌اند که می‌توان از آنها به صورت مجزا و یا ترکیبی استفاده کرد:

- Field query parser: این تجزیه‌گر با استفاده از فاصله بین کلمات و بزرگ یا کوچک بودن کلمات آنها را تجزیه می‌کند. به عنون مثال، `{!field f=myfield}Foo Bar` برابر با جستجوی `myfield:"Foo Bar"` در پایگاه داده است.
- Function Range query parsers: این تجزیه‌گر براساس تابع داده شده و در بازه‌ای خاص جستجو را انجام می‌دهد. به عنوان مثال،

```
fq={!frange l=0 u=2.2} sum(user_ranking,editor_ranking)
```

بدین معنا است که ابتدا جمع دو مقدار `user_ranking` و `editor_ranking` انجام شده و سپس در بازه مشخص شده نتایج بازیابی می‌شوند.

- Nested query parser: این تجزیه‌گر، عبارت پرس‌جوی تودرتو ایجاد می‌کند. به عنوان مثال،

```
q:"roses are red" AND _query_:"type:poems"
```

به معنای جستجوی عبارت «`roses are red`» است با این شرط که از نوع «`poems`» باشند.

- Boost query parser: این تجزیه‌گر، عبارت پرس‌وجو را با کمک تابع تعیین شده وزن‌دار می‌کند. به عنوان مثال،

```
{!boost b=log(popularity)}foo
```

به معنای آن است که عبارت پرس‌وجوی «`foo`» وزنی معادل «`log(popularity)`» در جستجو دارد.

- Prefix query parser: در این تجزیه‌گر عبارت پرس‌جوی کاربر به عنوان پیش‌وند برای جستجو بکار می‌رود. به عنوان مثال، `{!prefix f=myfield}foo` برابر با جستجوی `myfield:foo*` در پایگاه داده است.
- Join query parser: این تجزیه‌گر نتایج بازیابی اطلاعات را براساس ویژگی‌های تعیین شده با یکدیگر ترکیب می‌کند. به عنوان نمونه،

```
{!join from=manu_id_s to=id}ipod
```

بدین معنا است که جستجوگر تمامی اسناد حاوی عبارت ipod را یافته و براساس id تولید کننده آنها را ترکیب کند.

- Collapsing query parser: این تجزیه‌گر، نتایج جستجو را براساس ساختار سلسله مراتبی (درختی) با کمک میزان ارتباط بدست آمده نمایش می‌دهد. به عنوان نمونه،دستور fq={!collapse field=group_field}سبب می‌شود تا نتایج بازیابی شده براساس میزان ارتباط و نیز group_field به صورت سلسله مراتبی نمایش داده شود.

۳-۵-۲- مولفه FACET

دومین مولفه در بخش ❸ شکل ۳-۱۷ مولفه Facet است. این مولفه امکان برقراری جستجوی چندوجهی را در سولر مهیا می‌سازد. با کمک این مولفه کاربر قادر است نتایج جستجو را براساس نیاز خود فیلتر نموده و نمایی کلی از خصوصیات اسناد موجود در پایگاه داده را درخصوص موضوع مورد جستجو مشاهده نماید. به تعبیری دیگر، با کمک این مولفه فراداده مربوط به اسناد مرتبط با عبارت پرس‌وجو برای کاربر نمایش داده می‌شود (شکل ۳-۱۸). نکات مربوط به این مولفه به صورت مختصر عبارتند از:

- این مولفه به صورت پیش‌فرض فعال نیست و باید در Schema تنظیم شود.
- Facetها به صورت برخط و برای هر جستجو محاسبه می‌شوند. لذا، زمان‌بر بوده و درصورت طراحی نامناسب می‌تواند سبب کاهش کارایی شوند.
- در نسخه‌های فعلی سولر، این قابلیت در صورتی ممکن است فعال شود که سولر دارای یک هسته پردازشی فعال باشد.

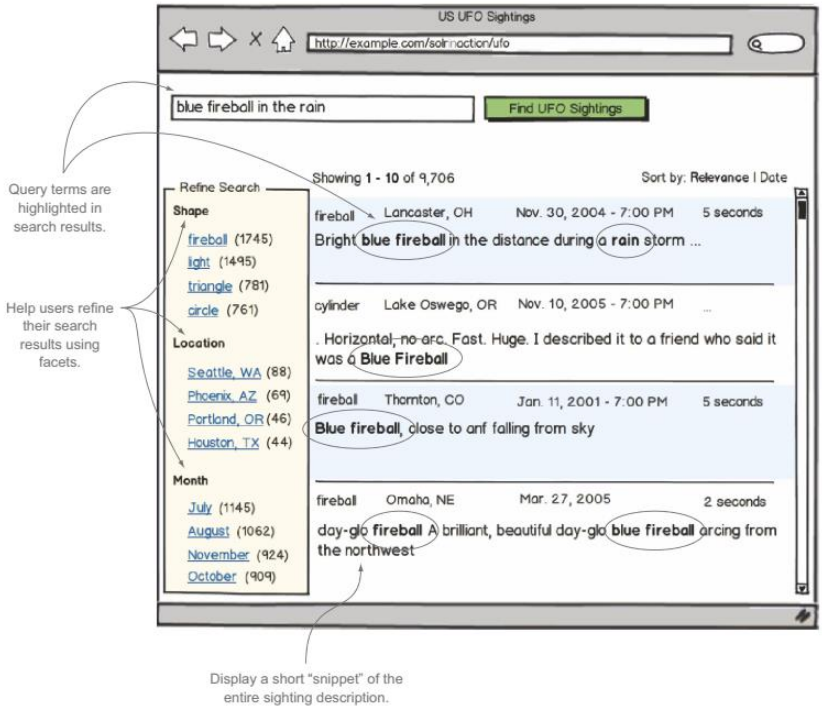
۳-۵-۳- مولفه MORE LIKE THIS

مولفه More like this سومین مولفه در بخش ❸ شکل ۳-۱۷ بوده و وظیفه نمایش اسناد مشابه با سند مورد انتخابی کاربر را برعهده دارد (شکل ۳-۱۸). شباهت در این مولفه براساس زاویه بین بردار و TF-IDF دو سند محاسبه می‌شود؛ این محاسبه می‌تواند به صورت برخط (درهنگام جستجو) و یا آفلاین باشد. تفاوت بین این دو حالت در میزان استفاده از حافظه و فضای ذخیره سازی و نیز زمان پردازش است. به تعبیری دیگر، در حالت برخط فضای ذخیره سازی کمتر و حافظه بیشتری مورد نیاز است درصورتی که در حالت آفلاین میزان استفاده از حافظه و سرعت پردازش افزایش می‌یابد.

این مولفه به صـــورت پیش‌فرض فعال نبوده و باید در Schema فعال شـــده و فیلدهایی که برای محاسبه شباهت استفاده می‌شود مشخص گردند. همچنین می‌توان لیستی از واژگان مهم برای یافتن شباهت را نیز به همراه وزن آنها تعیین نمود تا در محاسبه شباهت بین دو سند تغییراتی ایجاد شود (این مورد در بخش ۵-۴، شخصی‌سازی جستجو، بررسی خواهد شد).

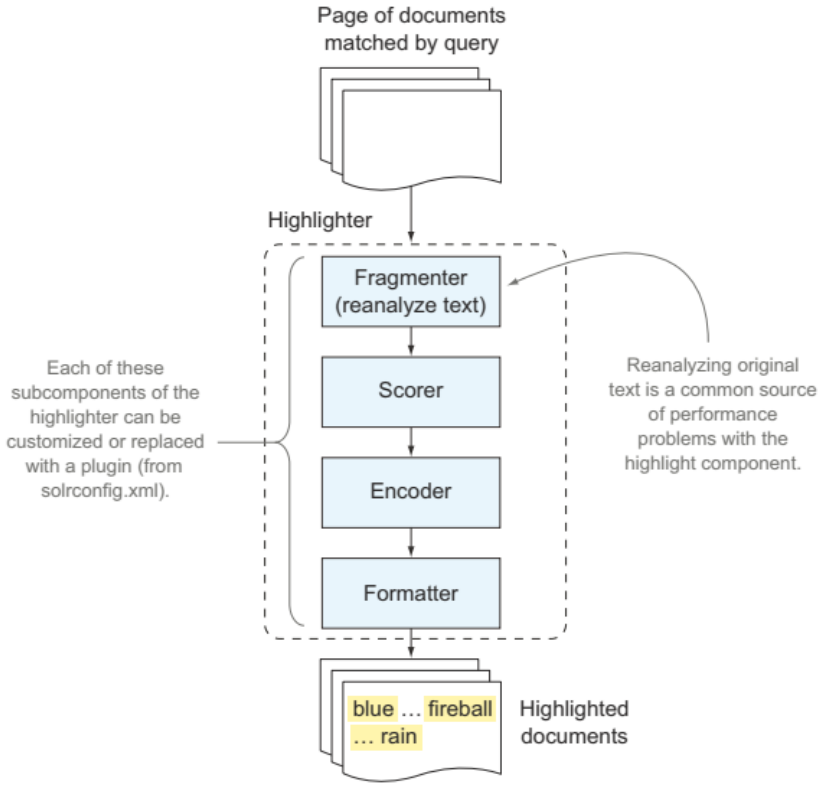
۴-۵-۳- **مولفه** HIGHLIGHT

پررنگ کردن بخش‌هایی از سند که با عبارت پرس‌وجوی کاربر منطبق هستند اتفاقی است که پس از فعال کردن این مولفه در فرآیند مدیریت درخواســـت جســـتجو (شـــکل ۳-۱۷) و در Schema می‌افتد. نمونه‌ای از نتایج جستجو پس از فعال‌کردن این مولفه در شکل ۳-۲۱ نشان داده شده است.



شکل ۳-۲۱- نمونه‌ای از نتایج جستجو پس از فعال کردن مولفه hithighlight

یکی از مواردی که در این شکل نشان داده شده نمایش snippet است. درواقع snippet بخشی از متن ســند اســت که با عبارت پرس‌وجوی کاربر مطابقت دارد و انتخاب آن از طریق فرآیند نشــان داده شده در شکل ۳-۲۲ انجام می‌شود.



شکل ۳-۲۲- فرآیند نمایش snippet **و پررنگ کردن بخش‌هایی از سند**

ورودی این فرآیند بخش‌هایی از سند ا ست که با عبارت پرس‌وجوی کاربر مطابقند و گام‌های آن عبارتند از:

□ گام ۱: قطعه‌بندی کننده[۱] (تحلیل مجدد متن):

برای پررنگ کردن بخش‌هایی از سند، واژگان سند باید قابل مقایسه با واژگان عبارت پرس‌وجو باشند و همچنین سولر باید از مکان دقیق واژگان در سند اطلاع داشته باشد. لذا باید فرآیند تحلیل متن روی سند اجرا شود (موارد مطرح شده در بخش ۴-۳، زیر سیستم تحلیل متن).

سپس برای تعیین snippetها برای نمایش، متن باید به قطعاتی تقسیم شود. از دو رویکرد برای قطعه‌بندی متن در سولر استفاده می‌شود: GapFragmenter و RegexFragmenter.

GapFragmenter رویکرد پیش‌فرض بوده و متن را به تعداد واژگان از پیش تعیین‌شده[۲] قطعه‌بندی می‌کند (مقدار پیش‌فرض (`h1.fragsize`) برابر ۱۰۰ کاراکتر است). در

---

[۱] Fragmenter

[۲] Token boundery

RegexFragmenter متن براساس الگوهای از پیش تعیین شده (مانند رسیدن به نشانه‌هایی چون . ، / و ...) قطعه‌بندی می‌شود.

گام ۲: امتیازدهنده: در این گام برای تمامی قطعات شناسایی شده در گام ۱ امتیاز ارتباط با عبارت پرس‌وجو محاسبه شده و قطعه دارای بالاترین امتیاز برای نمایش انتخاب می شود. تابع پیش‌فرض برای محاسبه این امتیاز QueryScorer است و بر اساس شمارش واژه‌ها در هر قطعه، امتیاز را محاسبه می‌کند.

گام ۳ و ۴: کدگذاری و فرمت‌دهی: در این گام واژه‌های تعیین شده برای پررنگ کردن کد گذاری شده و فرمت آنها به فرمت مشخص شده تغییر می‌یابد.

در این مولفه چند نکته وجود دارد:

- اگر کاربر در صفحه نمایش نتایج جستجو facetی را انتخاب کند، در این حالت به صورت پیش‌فرض، facet انتخابی کاربر پررنگ نمی‌شود مگر آنکه تغییراتی در تنظیمات مولفه Facet ایجاد شود. به عنوان مثال، اگر کاربر بخواهد نتایج جستجو را، درصفحه نمایش نتاج، به نام دانشگاه خاصی محدود کند، در حالت عادی نام این دانشگاه پررنگ نخواهد شد.
- فرض کنید عبارت جستجوی کاربر به صورت «blue fireball in the rain» باشد. در این حالت کد پرس‌وجوی لوسن به صورت زیر خواهد بود:

```
content: blue OR content: fireball OR content: rain
```

نتایج جستجو در پایگاه داده نیز به صورت زیر است:

1- and orange as well as **blue** light and what appeared to be a **fireball**. After that no more lights and no more **rain**

2- I saw a **blue fireball** in the sky, it was an awesome **fireball**.

حال اگر کاربر به دنبال نتایج دقیق عبارت پرس‌وجو باشد، نظیر "blue fireball" در این حالت کد پرس‌وجوی لوسن به صورت زیر است:

```
content: blue fireball
```

و نتایج جستجو در پایگاه داده نیز به صورت زیر خواهد بود:

1- I saw a **blue fireball** in the sky, it was an awesome fireball.

در این حالت واژه fireball دوم پررنگ نمی‌شود.

- از آنجا که در این مولفه باید متن سند تحلیل شود، عملکرد آن در اسناد حجیم (بیشتر از ۵۰ صفحه) کند است. برای این منظور می‌توان از توابع FastVectorHighlighter و یا PostingsHighlighter استفاده نمود.

در هردوی این توابع تغییراتی در فرآیند پررنگ کردن بوجود آمده است و هردو باید به اطلاعات شاخص معکوس دسترسی داشته باشند با این تفاوت که FastVectorHighlighter نیازمند چینش متفاوتی از اطلاعات شاخص معکوس بوده و لذا از ابتدا اطلاعات این شاخص را جداگانه ذخیره می‌کند، اما PostingsHighlighter مستقیماً از اطلاعات شاخص معکوس استفاده می‌کند.

در FastVectorHighlighter گام تحلیل متن حذف شده و ضروری است ویژگی های termVectors,termPositions و termOffsets برای فیلد هایی که باید پررنگ شوند، فعال شده باشد. اما بقیه تنظیمات مانند حالت پیش‌فرض است.

بر خلاف FastVectorHighlighter، تابع PostingsHighlighter از رویکرد جمله-آگاه [1] برای قطعه‌بندی متن استفاده کرده و قطعات با «...» (سه نقطه یا ellipsis) از هم جدا می‌شوند. همچنین PostingsHighlighter برای یافتن میزان ارتباط یک قطعه با عبارت پرس‌وجو از روش BM25 استفاده می‌شود.

برای استفاده از PostingsHighlighter وجود اطلاعات مربوط به offsetهای واژه‌ها و اضافه کردن پارامتر storeOffsetsWithPositions در Schema ضروری است و از آنجا برخی از ابزارهای واژه-واژه‌سازی برای برخی از زبان ها نمی‌توانند اطلاعات offsetهای را گزارش کنند استفاده از این تابع محدود خواهد بود.

پارامترهای تنظیم مولفه highlighter متعدد بوده و در بخش ضمیمه (جدول ۷-۷) به برخی از آنها اشاره می‌شود.

۵-۵-۳- مولفه LAST-COMPONENT

مولفه Last-Component مولفه دیگری در فرآیند مدیریت درخواست جستجو (شکل ۳-۱۷، بخش ❹) است که وظیفه پردازش نهایی فرآیند جستجوی کاربر را بر عهده دارد. این مولفه

[1] Sentence-aware approach

اختیاری بوده و شامل ابزارهایی است که زمان زیادی را برای پردازش به خود اختصاص می‌دهند و ممکن است نتایج آنها تاثیر چندانی در نتایج بازیابی شده نداشته باشد.

در این بخش به دو ابزار مهم این مولفه شـــامل امکانات کنترل املاء[۱] و پیشـــنهاد خودکار[۲] عبارت پرس‌وجو پرداخته می‌شـود که می‌تواند موجب بهبود تجربه کاربر در اسـتفاده از سـیسـتم بازیابی اطلاعات شوند.

کنترل املاء به مسـئله کنترل نحوه نگارش عبارت پرس‌وجو پرداخته و درنتیجه سـبب می‌شـود تا نتایج نامناســب بازیابی نشــوند. پیشــنهاد خودکار نیز به دنبال فراهم آوردن امکان ارائه پیشــنهاد واژگانی جایگزین به کاربر براســاس نمایه‌های موجود در پایگاه داده اســت. به تعبیری دیگر با کمک امکان پیشنهاد خودکار بازخور فوری به کاربر درباره اعتبار عبارت پرس‌وجو داده می‌شود.

کنترل املاء

از منظر کنترل املاء چند سناریو ممکن است روی دهد:

- □ عبارت پرس‌وجو حاوی یک یا چند واژه نادرست بوده و تعداد اسناد بازیابی شده صفر است. در این حالت اگر سیستم بتواند واژه جایگزین پیدا کند، بهتر آن است که به صورت خودکار جستجو را با واژه جایگزین انجام داده و پیامی حاوی توضیحات مرتبط به کاربر ارائه نماید، نظیر «Searched for … instead of …».
- □ عبارت پرس‌وجو حاوی یک یا چند واژه نادرست بوده و تعداد اسناد بازیابی شده بسیار کم اسـت. در این حالت، اگر سـیسـتم بتواند واژه جایگزین بیابد، بهتر اسـت که توضـیحی به کاربر ارائه شود نظیر «Did you mean…».
- □ عبارت پرس‌وجو درست است و واژه جایگزینی نیز وجود دارد. در این حالت اگر تعداد نتایج بازیابی در دو حالت عبارت کاربر و واژه جایگزین تقریباً برابر باشـد، سـیسـتم نباید پیشنهاد(هایی) را به کاربر ارائه کند.
- □ عبارت پرس‌وجو حاوی واژه‌گانی است که در نمایه وجود نداشته و سیستم قادر نیست واژه جایگزینی را نیز پیشنهاد کند.

---

[۱] Spell-checking

[۲] Autosuggest

با کمک این سناریوها می‌توان دو نیازمندی را برای کنترل املاء استخراج کرد: (۱) روشی نیاز است تا واژگان جایگزین برای هر واژه در پرس‌وجو شناسایی شود و (۲) باید مشخص شود برای هر واژه پرس‌وجو و جایگزین چه تعداد نتایج بازیابی می‌شود.

سولر برای ارائه واژگان جایگزین از تابع Spellcheck و نمایه‌های موجود در پایگاه داده استفاده می‌کند. پارامترهای اصلی و کلان این تابع برای تنظیم در Schema در جدول ۷-۸ ضمیمه آمده است. تابع پیش‌فرض DirectSolrSpellChecker بوده و اطلاعات خود را مستقیماً از نمایه اصلی سیستم دریافت می‌کند. سه پارامتر اصلی این تابع عبارتند از:

- field: تعیین این که از اطلاعات مربوط به چه فیلدهایی برای ارائه پیشنهاد خودکار استفاده شود؛
- distanceMeasure: تعیین نحوه محاسبه فاصله بین واژه پرس‌وجوی کار بر و واژه پیشنهادی. فاصله پیش‌فرض فاصله دامرا-لونا شتاین است که در بخش تطابق فازی بدان پرداخته شد. البته می‌توان از فواصل دیگری نظیر موارد مطرح شده در جدول ۷-۲ نیز استفاده کرد؛
- accuracy: مقداری بین صفر و یک بوده و برای تعیین میزان دقت پیشنهادها تنظیم می‌شود.

این تابع از ابزارهای تحلیل متن StopFilterFactory برای حذف ایست‌واژه‌ها، UAX29URLEmailTokenizer برای حذف فاصله‌ها و نشانه‌ها و درعین حال نگهداری آدرس رایانامه‌ها و آدرس‌های اینترنتی، از LowerCaseFilter برای کوچک کردن حروف بزرگ، از ASCIIFoldingFilter برای تبدیل حروف به کدهای ASCII و از EnglishPossessiveFilter برای حذف مضاف الیه‌های زبان انگلیسی استفاده می‌کند.

تابع WordBreakSolrSpellchecker دیگر تابع مورد استفاده در کنترل املاء است و در مواردی کاربرد دارد که واژگان عبارت پرس‌وجو بدون فاصله درکنار یکدیگر قرار گرفته باشند. این ماژول این نوع واژگان را به بخش های مختلف تجزیه کرده و سپس آنها را برای ارائه پیشنهادهای جایگزین در نمایه جستجو می‌کند.

پیشنهاد خودکار

وظیفه اصلی ابزار پیشنهاد خودکار جلوگیری از اشتباهات کاربر در هنگام وارد کردن عبارت پرس‌وجو بوده و می‌تواند نقش موثری در کاهش هزینه‌های پردازشی ایفا نماید (پیاده‌سازی و بکارگیری آن در سیستم‌های مبتنی برتلفن همراه ضروری است). سرعت در ارائه پیشنهادها و درنظرگیری رتبه واژگان (بر اساس TF) برای پیشنهاد از نیازمندی‌های چنین ابزاری است.

یکی از توابع پرکاربرد در این ابزار (برای داده‌های حجیم) FSTLookupFactory است که از ساختار اطلاعات خودکارسازی وضعیت محدود[۱]، که نوعی نگاشت بین دو مجموعه نمادهاست، استفاده می‌کند و لغت‌نامه‌ای از پیشنهادها را می‌سازد.

علی‌رغم شباهت‌های بین دو ابزار کنترل املاء و پیشنهاد خودکار، تفاوت‌های کلیدی نیز وجود دارد: مولفه کنترل املاء نیازمند عبارت کامل پرس‌وجوی کاربر است درحالی‌که در پیشنهاد خودکار، در حین وارد کردن عبارت پرس‌وجو پیشنهادهایی به کاربر ارائه می‌شود. همچنین، نوع تابع فاصله یکی از پارامترهای تاثیرگذار در کارایی کنترل املاء است ولی مشهور بودن واژگان در پایگاه داده عامل تاثیرگذار در پیشنهاد خودکار است.

۶-۵-۳- **مولفه** RESPONSE WRITER

پس از بازیابی اطلاعات مرتبط، نتایج بایستی به کاربر نمایش داده شوند. این امر از طریق مولفه Response Writer (شکل ۳-۱۴ و شکل ۳-۱۷) انجام می شود. در این مولفه پاسخ به چند سوال ضروری است: «خروجی در چه فرمتی و از چه نوعی باید باشد؟»، «چه فیلدهایی باید برای کاربر نمایش داده شوند؟»، «هر صفحه باید شامل چه تعداد نتایج باشد؟» و «نحوه مرتب کردن نتایج چگونه باید باشد؟». تنظیمات مربوط به پاسخ‌های این سوالات در مولفه نمایش نتایج انجام می‌شود.

فرمت و نوع نتایج خروجی[۲] (wt) می‌تواند یکی از موارد گزارش شده در جدول ۳-۷ باشد.

جدول ۳-۷- پارامتر نوع نتایج خروجی

نوع نتایج خروجی (wt)	کلاس مورد استفاده در سولر
CSV	CSVResponseWriter
json	JSONResponseWriter
php	PHPResponseWriter

[۱] Finite state automaton (FST)

[۲] Writer type

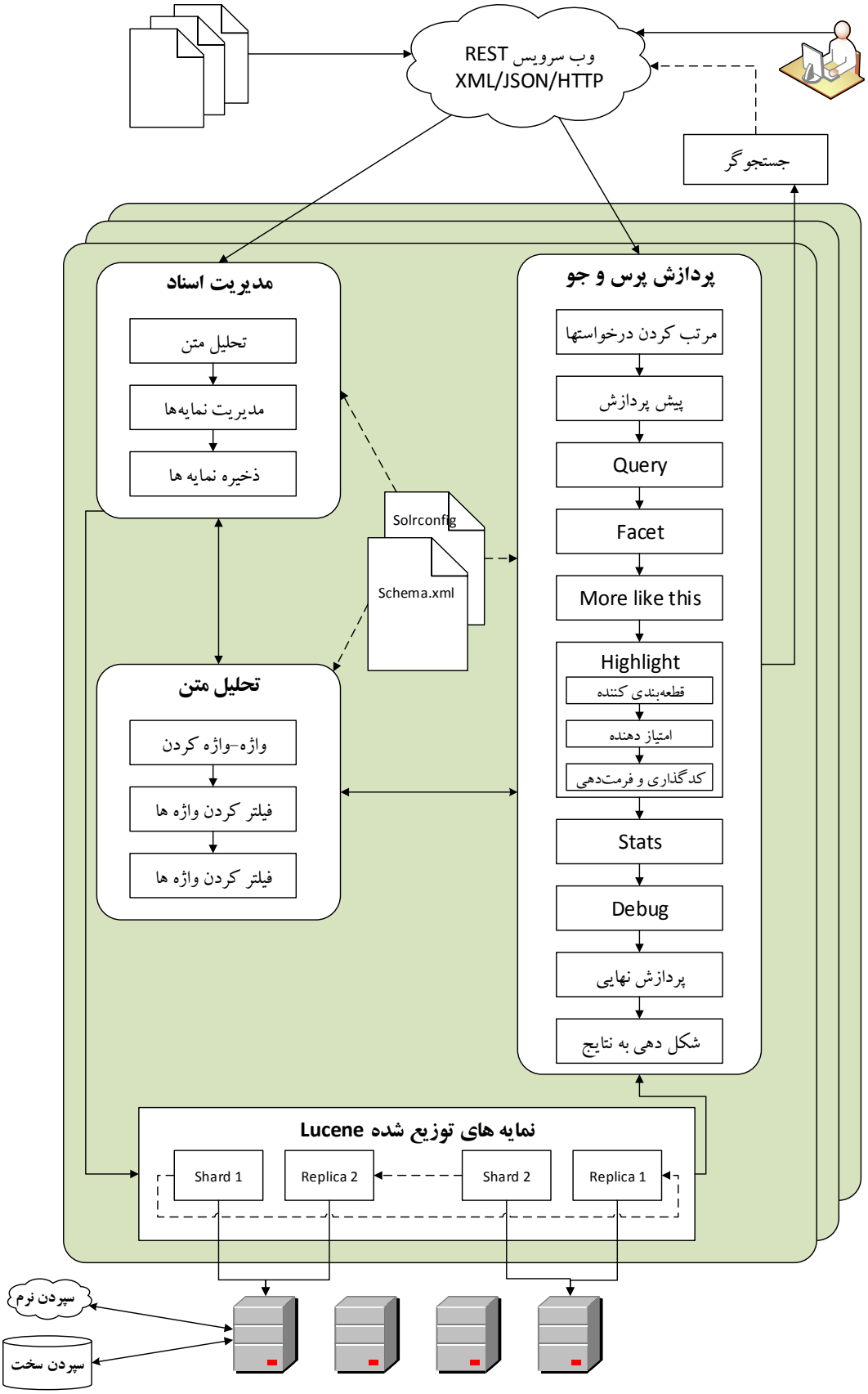
PHPSerializedResponseWriter	phps
PythonResponseWriter	python
RubyResponseWriter	ruby
XMLResponseWriter	xml
XSLTResponseWriter	xslt
BinaryResponseWriter	javabin

تعداد فیلدهایی که برای کاربر نمایش داده می شود بر سرعت نمایش نتایج و حجم اطلاعات تبادل شده موثر ا ست. یکی از راه‌های بهبود سرعت نمایش نتایج، ا ستفاده از صفحات متعدد ا ست. به عبارت دیگر، در صفحه نخست نمایش نتایج معمولاً نتایج با بالاترین امتیاز نمایش داده شده و کاربر با جابجا شدن در صفحات می‌تواند سایر نتایج را ببند. سولربرای مرتب کردن نتایج، از میزان انطباق هر سند با عبارت پرس‌وجو استفاده می کند و نتایج را در حالت پیش‌فرض به صورت نزولی نمایش می‌دهد. اگر دو سند دارای امتیازهای یکسان باشند، براساس ID سند در لوسن مرتب می‌شوند. البته برای مرتب کردن نتایج باید حافظه زیادی به جستجو تخصیص داد و این امر درصورتی که کاربران زیادی به صورت همزمان از سیستم استفاده کنند می‌تواند مشکل‌زا باشد.

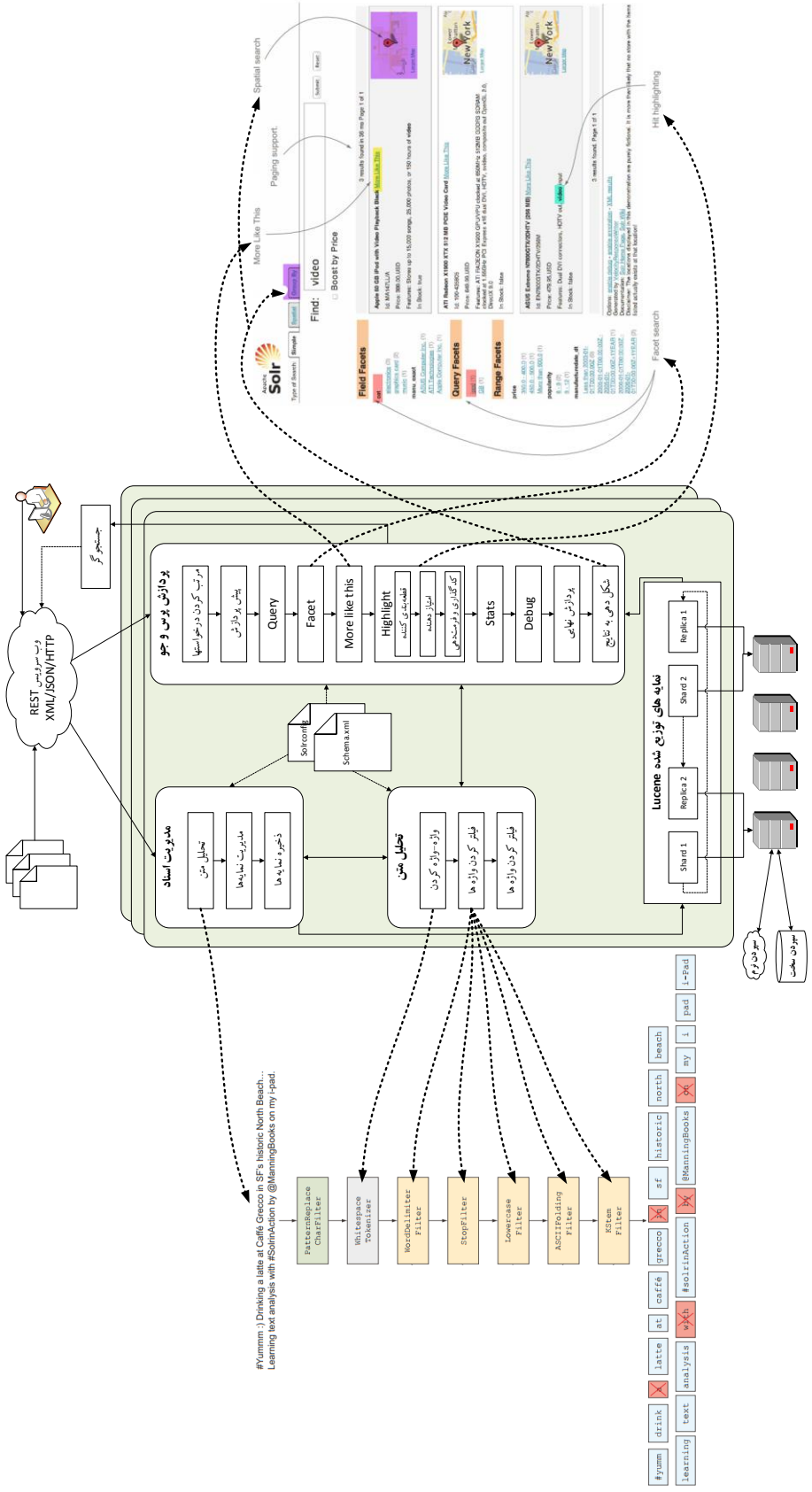
۳-۶ جمع‌بندی

فرآیند آنچه در سولر رخ می‌دهد را می‌توان در شکل ۳-۲۳ خلاصه نمود. در طی این فرآیند اسناد ورودی به پایگاه داده توسط لوسن نمایه شده و ذخیره می شوند. درخواست‌های جستجوی کاربر نیز تحلیل شده و نتایج براساس تنظیمات مشخص شده در Solrconfig و Schema بازیابی شده و به کاربر نمایش داده می‌شوند. فرآیند تحلیل متن جزء کلیدی سولر و لوسن بوده و به عنوان ابزاری در زیرسیستم‌های مدیریت اسناد و پردازش پرس‌وجو استفاده می‌شود. قابلیت تحلیل‌های چند زبانی نیز از دیگر قابلیت‌های سولر و لوسن است که برای اجرای آن سیستم نیازمند معماری خاصی است. شکل ۳-۲۴ نیز این فرآیند و عملکرد آن را در قالب یک مثال نمایش می‌دهد.

علاوه بر این، به غیر از موارد مطرح شده در این فصل، ملاحظات و تنظیمات دیگری نیز در سولر وجود دارد که در این گزارش در قالب مباحث پیشرفته و در فصل بعد مورد بررسی قرار خواهند گرفت.



شکل ۳-۲۳- فرآیند پردازش اسناد و پرس‌وجو در سولر



شکل ۳-۲۴- فرآیند پردازش اسناد و پرس‌وجو در سولر در قالب مثال

فصل چهارم

مباحث پیشرفته در موتور جستجوی سولر

۴-۱ مقدمه

در فصل پیش توضیحاتی در مورد موارد و نکات پایه‌ای سولر ارائه شد. در این فصل به بررسی برخی نکات پیشرفته‌تر شامل نحوه فراخوانی و استفاده از داده‌های منابع بیرونی، عملیات جستجوی پیچیده، نحوه بهبود ارتباط سند با عبارت پرس‌وجو و شخصی‌سازی جستجواختصاص دارد.

۴-۲ استفاده از داده‌های منابع بیرونی

داده جزء بنیادین اغلب سیستم‌ها، خصوصاً سیستم‌های جستجو و بازیابی اطلاعات، است. این داده‌ها می‌توانند از منابع درونی سولر (ذخیره و نمایه شده در لوسن) و یا از منابع بیرونی تامین شوند.

سه مکانیزم اصلی برای استفاده از داده‌های منابع بیرونی (بیرون از نمایه سولر) وجود دارد: (۱) استفاده از دستوراتی در توابع پردازش پرس‌وجو برای دسترسی به منابع بیرونی؛ (۲) استفاده از ارتباطات بین اسناد در هسته‌های مجزای سولر فعال در یک سرور و یا (۳) استفاده از فیلد خاصی بانام `ExternalFileField` که می‌تواند مقادیر را از فایل بیرونی دریافت کند.

فیلد `ExternalFileField` امکان بروزرسانی یک فیلد خاص را مستقل از فرآیند نمایه‌سازی در سولر فراهم می‌کند. این امر با تعیین فایل متنی حاوی نگاشت `ID` هر سند در سرور سولر با مقدار آن در فایل بیرونی محقق می‌شود. به تعبیری دیگر سولر با منبع بیرونی از طریق مشخصه سند

ارتباط دارد. تعریف این فیلد زمانی مفید اســت که بخواهیم فیلد خاصــی را برای تعداد زیادی از اسناد بروزرسانی کنیم بدون آنکه بقیه فیلدها را تغییر دهیم.

ExternalFileField مانند سـایر فیلدها (البته با پارمترهایی بیشـتر) باید در Schema (به عنوان مثال به صورت زیر) تعریف شود:

```
<fieldType name="popularity" keyField="id" defVal="0"
stored="false" indexed="false" class="solr.ExternalFileField"
valType="pfloat"/>
```

در این مثال، فیلد popularity مقدار خود را برای هر ســند از فایل بیرونی می‌خواند که در دایرکتوری نمایه ســولر در ســرور قرار داده شــده اســت. اســم این فایل باید به صــورت external_{field} یا *.external_{field} باشد که در آن {field} نام ExternalFileField اســت که در Schema تعریف شــده اســت. بنابراین برای مثال فوق نام این فایل خارجی می‌تواند external_polularity.2013_11_03_09_00 و یا external_popularity.txt باشــد. این فایل می‌تواند دارای اطلاعاتی نظیر زیر باشـد که به عنوان مثال در آن مقدار معروفیت doc123 برابر ۲۲ است:

```
doc123=22
doc456=15.7
doc789=19.4
```

فیلد ExternalFileField خصوصیات مهمی دارد:

- جسـتجو براسـاس این فیلد امکان پذیر نیسـت. به بیانی دیگر، سـولر قادر نیسـت براسـاس مقادیر این فیلد جستجو کند.
- لزومی ندارد همه اسناد موجود در پایگاه داده دارای اطلاعاتی مخصوص به خود در فایل بیرونی باشــند و برعکس. به عنوان مثال، ســند doc709 ممکن اســت مقداری برای popularity نداشته باشد و یا سند doc456 در پایگاه اسناد وجود نداشته باشد.
- درحال حاضــر فیلد ExternalFileField تنها می‌تواند از نوع float باشــد و این امر می‌تواند محدودیت‌هایی را ایجاد نماید. برای رفع این مشــکل می‌توان پس از بازخوانی مقادیر خارجی توابعی را تعریف کرد تا نوع اطلاعات را تغییر دهد.
- مقادیر موجود در فایل بیرونی تنها یکبار و در هنگام اولین بار استفاده در حافظه بارگذاری می‌شوند. درنتیجه اگر تغییری در این مقادیر ایجاد شود سولر متوجه آن نخواهد شد. برای

رفع این مشـــکل می‌توان event listener تعریف کرد تا به محض رخداد تغییر، فایل بیرونی مجدداً بارگذاری شود.

۴-۳ عملیات جستجوی پیچیده

ســولر این امکان را برای کاربران خود فراهم می‌آورد تا علاوه بر انجام جســتجو براســاس عبارت پرس‌وجو، توابعی را (به صــورت از پیش طراحی شــده، تودرتو[1] و یا جدید) اجرا کرده و از نتایج آن به صـورت عبارت پرس‌وجوی جدید و یا پارامتری برای تعیین میزان ارتباط نتایج بازیابی شـده استفاده کند. برخی از توابع ریاضی در جدول ۷-۱ ضمائم، توابع فاصله در جدول ۷-۲ ضمائم و برخی از توابع منطق بولی در جدول ۷-۳ ضــمائم گزارش شــده اســت. اما به جهت اهمیت توابع محاســبه میزان ارتباط ســند و عبارت پرس‌وجو، برخی از مهمترین این توابع در جدول ۴-۱ آورده شده‌اند.

جدول ٤-١- توابع محاسبه میزان ارتباط سند و عبارت پرس‌وجو

دستور (syntax) تابع	توضیحات
`docfreq(fieldName, term)`	تعداد اسناد حاوی واژه term در فیلد fieldName
`idf(fieldName, value)`	محاسبه idf برای value در فیلد fieldName
`maxdoc()`	تعداد اسناد نمایه شده شامل اسناد حذف شده‌ای که از حافظه اصلی (durable storage) پاک نشده‌اند
`norm(fieldName)`	نرم محاســبه و ذخیره شــده در نمایه برای فیلد fieldName
`numdocs()`	تعداد اسـناد نمایه شـده بدون درنظرگیری اسـناد حذف شده‌ای که از حافظه اصلی پاک نشده‌اند
`query(subquery, defaultScore)`	امتیاز subquery برای همه اسناد مرتبط با subquery و امتیاز defaultScore برای اسناد نامرتبط
`sumtotaltermfreq(field)` `sttf(field)`	تعداد واژه‌های (token) نمایه شــده در field (این دو دستور عملکردی مشابه دارند)
`termfreq(fieldName, term)`	تعداد دفعاتی که واژه term در فیلد fieldname تکرار شده است

[1] Nested

`tf(fieldName, term)`	فراوانی کلمه (tf) برای واژه term در فیلد fieldname براساس شباهت آن فیلد
`totaltermfreq(fieldName, term)` `ttf(fieldName, term)`	تعداد دفعاتی که واژه term در کل نمایه تکرار می‌شود (این دو دستور عملکردی مشابه دارند)

نمونه‌ای از نحوه بکارگیری توابع فوق در برای یافتن عبارت «microsoft office» در فیلد content در ادامه آمده ا ست. در بخش ابتدایی این د ستور ابتدا فراوانی کلمه (TF) برای واژه «microsoft» در فیلد content و سپس فراوانی معکوس سند (IDF) برای همین واژه در فیلد content محاسبه شده و نتایج این دو محا سبه درهم ضرب می شوند. این عمل برای واژه «office» در فیلد content نیز تکرار شده و نتیجه کلی ضرب TF و IDF برای این دو واژه با هم جمع خواهد شد.

```
fq={!cache=false}content:"microsoft office”&
q={!func}sum(
            product(
                  tf(content, "microsoft"),
                  idf(content, "microsoft")
            ),
            product(
                  tf(content, "office"),
                  idf(content, "office")
            )
      )
```

۴-۴ ارتقاء ارتباط سند با عبارت پرس‌وجو

همانطور که در بخش۴-۲-۳، ارتباط، ذکر شد، محاسبه ارتباط سند و عبارت پرس‌وجو و رتبه‌دهی به اسناد بازیابی شده، از مزیت‌های موتورهای جستجو نسبت به پایگاه‌های داده است[۱]. در این بخش، روش معمول سولر برای این منظور بیان شد اما برای ارتقاء امتیاز محاسبه شده و بهبود نتایج بازیابی شده راهکارهایی وجود دارد:

- □ راهکار ۱- وزن‌دهی به فیلدی معین: اگر فیلدی حاوی اطلاعاتی باشد که کاربران اغلب به دنبال آن هستند می‌توان در محاسبه ارتباط به این فیلد وزن بیشتری اختصاص داد. برای این پیاده‌سازی این راهکار چند روش وجود دارد:

[۱] پایگاه‌های داده قادرند در بین داده‌های خود جستجو کنند اما نمی‌توانند نتایج را رتبه‌بندی نمایند.

- وزن‌دهی به فیلد خاص در هنگام نمایه‌سازی. مثال زیر را درنظر بگیرید:

```
<add>
<doc>
<field name="id">1</field>
<field name="restaurant_name" boost="10.0">Red Lobster</field>
```

مشکل این روش آن است که اگر بخواهیم مقدار وزن را با سعی و خطا تعیین کنیم نیاز است تمامی اسناد مجدداً نمایه شوند و لذا این روش در اغلب موارد مناسب نخواهد بود.

- وزن‌دهی به فیلد خاص در زمان پردازش جستجو: این راهکار انعطاف‌پذیری بیشتری نسبت به روش نخست دارد. مثال زیر را درنظر بگیرید:

```
q=red lobster&
qf=restaurant_name^10 description
```

در این مثال عبارت پرس‌وجو «red lobster» بوده و وجود این عبارت در فیلد نام رستوران وزنی ۱۰ برابر وجود آن در توضیحات خواهد داشت.

□ راهکار ۲- وزن‌دهی به واژگان خاص: در این راهکار به واژگان خاص در عبارت پرس‌وجوی کاربر وزن بیشتری داده می‌شود بجای آنکه به کل عبارت و یا فیلد مورد جستجو وزن داده شود. البته این روش را می‌توان با راهکار ۱ نیز ادغام نمود. به عنوان مثال:

```
q=red^2 lobster^8&
qf=restaurant_name^10 description
```

در مثال فوق وزن واژه «red» اگر در فیلد نام رستوران قرار داشته باشد برابر خواهد بود با $2\times10=20$ و وزن واژه «lobster» اگر در فیلد نام رستوران قرار داشته باشد برابر خواهد بود با $8\times10=80$ (ضرب وزن‌ها بدین صورت برای تمامی راهکارهای وزن‌دهی در سولر برقرار است).

□ راهکار ۳- وزن‌دهی به واژگان سند: در این راهکار، واژگان سند و نقش آنها در جمله شناسایی شده و به نقش‌های خاص (مثلاً اسم) وزن بیشتری در هنگام تحلیل متن داده می‌شود. این امر از طریق Payloads در سولر محقق می‌شود که به صورت پیش‌فرض

فعال نیست. Payloads آرایه‌ای است که در کنار واژگان نمایه شده در سند قرار گرفته و می‌توان اطلاعاتی را درباره آن واژه در آن ذخیره نمود[۱].

- □ راهکار ۴- وزن‌دهی با کمک تابع: استفاده از توابع در وزن‌دهی می‌تواند سبب تغییر در امتیازهای اسناد شود. به عنوان مثال:

```
q=restaurant_name:(Burger King) AND
_query_:"{!func}recip(geodist(location,37.765,-122.43),1,10,1)"
```

در این تابع رستوران‌هایی که به مکان مشخص شده توسط geodistنزدیک‌تر باشند وزنی ۱۰ داده می‌شود و بنابراین رستوران‌های نزدیک‌تر امتیاز بالاتری دریافت خواهند کرد. بنابراین تابع امتیازدهی به صورت زیر محاسبه می‌شود:

```
score("Burger") + score("King") + LocationProximityBoost
```

- □ راهکار ۵- وزن‌دهی به نزدیکی واژگان: اگر در عبارت پرس‌وجوی کاربر دو واژه در کنار هم قرار گرفته باشند، اسنادی باید امتیاز بالاتری بگیرند که در آنها این دو واژه نزدیک‌تر به هم هستند. برای اعمال این روش دو راهکار وجود دارد:
  - بکارگیری eDisMax query parser و استفاده از پارامتر pf نظیر:

```
defType=edismax&
q=big data analytics&
qf=content&
pf=content~3^10
```

از پارامتر های دیگر این تجزیه‌گر،pf2 و pf3، نیز می‌توان استفاده کرد. اگر pf2=content، اسناد دارای عبارت‌های «big data» و یا «data analytics» وزن بیشتری می‌گیرند و اگر pf3=content، اسناد دارای عبارت « big data analytics» وزن بیشتری می‌گیرند.

  - بکارگیری Lucene query parser و استفاده از دستور “term1 term 2 … termN”~slop^boost. به عنوان مثال، عبارت پرس‌وجوی "customer service"~2^10 اسنادی را بازیابی می‌کند که در آنها دو واژهcustomer و service با ۲ فاصله از هم قرار گرفته باشد و وزن این اسناد برابر ۱۰ خواهد بود. ممکن است کاربر بخواهد اسناد حاوی این دو واژه را بیابد اما اسنادی اهمیت

[۱] برای اطلاعات بیشتر به https://issues.apache.org/jira/browse/SOLR-1485 مراجعه نمایید.

بیشتر داشته باشند که در آنها این دو واژه در فاصله نزدیک به هم قرار دارند. در این حالت، نمونه‌های پرس‌وجوی زیر می‌توانند نتایج مورد نظر کاربر را حاصل کنند:

```
1. q=+customer +service OR "customer service"
2. q=+customer +service OR "customer service"~2^10
3. q=customer OR service +{some other terms}+(*:* OR "customer
   service"^100)
4. q=+customer +service +representative OR "customer service
   representative"~10000^10
```

در حالت ۱، اسنادی که حاوی این دو واژه باشند بازیابی می شوند و اسنادی که در آنها این دو واژه در کنار یکدیگر قرار داشته باشند وزنی دو برابر دریافت می‌کنند. در حالت ۲، اسنادی که حاوی این دو واژه باشند بازیابی می‌شوند و اسنادی که در آنها این دو واژه با ۲ فاصله از هم قرار داشته باشند وزنی ۱۰ برابر دریافت می‌کنند. در حالت ۳، اسنادی که حاوی یکی از این دو واژه باشند بازیابی می‌شوند و اسنادی که در آنها این دو واژه در کنار یکدیگر قرار داشته باشند وزن ۱۰۰ برابر دریافت می‌کنند. در حالت ۴ نیز اسنادی بازیابی می‌شوند که حاوی سه واژه customer، service و representative بوده و مکان قرارگیری و فاصله آنها از یکدیگر اهمیت چندانی ندارد. اما هرچه به هم نزدیک‌تر باشند امتیاز بیشتری دریافت خواهند کرد.

- □ راهکار ۶- ارتقاء ارتباط اسناد مهم: یکی از راه‌های بهبود امتیاز ارتباط سند و عبارت پرس‌وجوی کاربر، درنظر گرفتن اهمیت اسناد در بازیابی اطلاعات است. یکی از راهکارها برای اجرای این روش تخصیص وزن خاص به سند در زمان نمایه سازی است (این وزن برای تمامی فیلدهای آن سند اعمال می شود). اشکال این راهکار، مانند راهکار نخست روش ۱،آن است که اگر بخواهیم مقدار وزن را با آزمایش تعیین کنیم نیاز است تا تمامی اسناد مجدداً نمایه شوند. راهکار دیگر تخصیص وزن به اسناد در زمان جستجو است که این امر می‌تواند از طریق درنظرگیری محبوبیت سند اجرایی شود.

۴-۵ شخصی‌سازی جستجو

شخصی سازی سیستم‌های بازیابی اطلاعات با ارائه پیشنهادهای مفید به کاربران کمک می‌کند تا تعامل بهتر و کاراتری با سیستم داشته باشند. سیستم‌های بازیابی اطلاعات را می‌توان نوعی از سسیتم‌های پیشنهاددهنده دانست که در آنها براساس عبارت پرس‌وجوی کاربر و یا مشخصات سند انتخاب شده، اسناد مشابه به کاربران ارائه می‌شود. سولر این قابلیت را دارد تا از طریق راهکارهای زیر به عنوان یک سیستم جستجوی پیشنهاددهنده به کاربران خود کمک کند:

- راهکار ۱- تطابق براساس ویژگی‌ها[۱]: برای پیاده‌سازی این راهکار، اسناد در پایگاه داده باید دارای برچسب‌هایی بوده و سولر بر اساس پروفایل رفتاری کاربر و برچسب‌های اسناد پیشنهادهای لازم را به وی ارائه می‌کند.
- راهکار ۲- تطابق سلسله مراتبی[۲]: در این روش ویژگی‌های اسناد و پروفایل رفتاری کاربر به صورت سلسله‌مراتبی و براساس اهمیت مرتب شده و پیشنهادها براساس این ترتیب ارائه می‌شود.
- راهکار ۳- مشابه این...[۳]: در این راهکار نیازی به دسترسی به پروفایل کاربر و یا برچسب‌های از قبل تعیین شده نیست. بلکه در این روش تشابه دو سند به یکدیگر محاسبه شده و مشابه‌ترین سند با سند انتخابی کاربر به وی نمایش داده می‌شود. More Like This Handler در سولر وظیفه استخراج واژگان مهم سند را بر اساس روش TF-IDFبر عهده داشته و به صورت خودکار از این واژگان به عنوان کلیدواژه برای یافتن اسناد مشابه استفاده می‌کند. برای استفاده از این handler گام‌های زیر باید طی شوند:

  گام ۱: فعالسازی در Schema با دستور:

```
<requestHandler name="/mlt" class="solr.MoreLikeThisHandler" />
```

  گام ۲: تعیین فیلدهایی که این handler برای مشابهت‌یابی در آنها باید واژگان مهم را استخراج کند توسط پارامتر mlt.fl(اگر جزئیات و اطلاعات آماری این واژگان نیز نیاز باشد با کمک دستور mlt.interestingTerms=details اطلاعات مربوطه بازگردانده می‌شوند).

---

[۱] Attribute-based matching

[۲] Hierarchical matching

[۳] Like this …

گام ۳: تنظیم کردن مقدار true برای پارامتر های termVectors و stored در Schema برای فیلدهایی که برای مشابهت یابی استفاده می‌شوند.

گام ۴: تعیین وزن برای واژگان ا ستخراج شده. این وزن‌ها به صورت پیش‌فرض برابر ۱ هستند اما می‌توان این وزن را از طریق پارامتر `mlt.boost=true` به مقدار TF-IDF هر واژه تغییر داد.

ممکن است شرایطی وجود داشته باشد که در طی آن لیست واژگان مهم دارای نویز باشد (مانند ایست‌واژه). برای جلوگیری از این اتفاق می‌توان از تحلیل ادات متن[۱] و استخراج اسم‌ها از سند استفاده کرد. همچنین به منظور افزایش سرعت می‌توان میزان شباهت اسناد به یکدیگر را به صورت آفلاین محاسبه کرد، هرچند که در این حالت فضای زیادی برای ذخیره‌سازی مورد نیاز خواهد بود.

□ راهکار ۴- تطابق براساس مفهوم[۲]: روش‌های قبلی بر اساس ویژگی‌های یک سند و یا واژگان مهم آن سند، پیشنهادهایی را به کاربر ارائه می‌کردند. اما سولر این قابلیت را دارد که اسناد را خوشه‌بندی نماید (این خوشه بندی در سولر به صورت offline انجام شده و دربازه‌های زمانی مشخصی بروزرسانی می‌شود). این امر با کمک مولفه clustering محقق می‌شود. پس از بازیابی نتایج بر اساس عبارت پرس‌وجوی کاربر، سولر خوشه‌هایی که اسناد با بالاترین امتیاز بدان‌ها تعلق دارند را یافته و سایر کلید واژگان را شناسایی می‌کند. به عنوان مثال ممکن است کاربر عبارت «Net Jobs. » را جستجو کند اما در نتایج، علاوه بر Net Jobs.، نتایج مربوط به واژگان software engineer، developer، C# و net developer. نیز دیده می شود (البته باید دقت داشت اسنادی که حاوی عبارت پرس‌وجوی کاربر هستند وزن بالاتری داشته باشند). در این روش تعداد نتایج بازیابی شده تغییر نخواهد کرد بلکه ترتیب نمایش آنها متفاوت است. استفاه از این روش باعث بهبود دقت شده و بازخوانی نیز بهبود یافته و یا ثابت می‌ماند.

برای اجرای این روش باید ابتدا ClusteringComponent و روش خوشه‌بندی مورد استفاده در Schema تعریف شده و سپس در Search Handler فعال شود (سولر به

---

[۱] Part of speech (POS)

[۲] Concept-based matching

صورت پیش‌فرض از موتور خوشه‌بندی Carrot[2] برای خوشه‌بندی اسناد بازیابی شده استفاده می‌کند[۱]):

```
<searchComponent name="clustering" enable="true"
      class="solr.clustering.ClusteringComponent">
    <lst name="engine">
   <str name="name">default</str>
    <str name="carrot.algorithm">
      org.carrot2.clustering.lingo.LingoClusteringAlgorithm
    </str>
    <str name="carrot.resourcesDir">clustering/carrot2</str>
   </str>
</searchComponent>
<requestHandler name="/clustering" enable="true"
      class="solr.SearchHandler">
   <lst name="defaults">
      <bool name="clustering">true</bool>
      <str name="clustering.engine">default</str>
      <bool name="clustering.results">true</bool>
      <str name="fl">*,score</str>
   </lst>
   <arr name="last-components">
      <str>clustering</str>
   </arr>
</requestHandler>
```

این راهکار زمانی مناسب است که عبارت پرس‌وجوی کاربر محدود بوده و موتور جستجو قصد داشته باشد نتایج مناسب‌تری را به کاربر نمایش دهد.

□ راهکار ۵- انطباق جغرافیایی[۲]: از این روش برای مواقعی استفاده می‌شود که موقعیت مکانی فعلی کاربر مهم است و یا نتایج باید بر اساس موقعیت مکانی مورد نظر کاربر مرتب شوند.

□ راهکار ۶- پالایش همکارانه[۳]: این روش یکی از پرکاربردترین روش‌ها در حوزه یادگیری ماشین برای ارائه پیشنهاد(ها) است. یکی از فرض‌های اصلی در این روش آن است که کاربران مشابه، علائق مشابهی نیز دارند و بنابراین می‌توان اسناد مشابه و یا مکمل را به کاربر فعلی، بر اساس انتخاب‌ها و یا ترجیحات سایر کاربران، پیشنهاد داد. درواقع، این روش نیازی به اطلاع از موضوع و محتوای سند ندارد و صرفاً براساس رفتار کاربران

[۱] https://project.carrot2.org/

[۲] Geographical matching

[۳] Collaborative filtering

پیشـنهادهایی را ارائه می‌کند. شـکل ۴-۱ و شـکل ۴-۲ نمایی از سـیسـتم پیشـنهاد دهنده وب‌سایت آمازون را برای کالاهای مشابه و مکمل نشان می‌دهند.

شکل ٤-١- سیستم پیشنهاد دهنده آمازون برای کالاهای مشابه

شکل ٤-٢- سیستم پیشنهاد دهنده آمازون برای کالاهای مکمل

در شکل ۴-۱، کاربری که تمایل به خرید فصل نخست سریال The Big Bang Theory را دارد با احتمال زیاد فصـل دوم و سـوم این سـریال را نیز خواهد خرید. به عبارت دیگر مشاهده رفتار کاربران ارتباط یا همبستگی بین اسناد را مشخص می‌کند. در شکل ۴-۲ نیز پیشـنهاد کالاهای مکمل بر اسـاس کالاهای انتخابی کاربر به وی ارائه می‌شـود (کالاها با کالای انتخابی کاربر ارتباط دارند اما متفاوتند).

به تعبیری دیگر، این روش از دانش جمعی و اطلاعات سایر کابران برای تنظیم پارامترهای جستجو در الگوریتم‌های پیشنهاددهنده استفاده کرده و با کمک امتیاز ارتباط کسب شده توسـط روش TF-IDF، رتبه اسـناد بازیابی شـده را تغییر می‌دهد. نکته دیگر در این روش آن اسـت که با کمک آن می‌توان با اسـتفاده از رفتار سـایر کاربران پیشـنهاداتی را درباره اسناد مشابه به کاربر فعلی ارائه داد و یا آنکه بر اساس اسناد و نحوه مشاهده آنها توسط کاربران، کاربران مشـابه را به هم معرفی نمود. برای این منظور باید دو جدول مانند شـکل

۴-۳ برای ارتباط بین کاربران و اسناد محاسبه شده و با کمک آنها پیشنهادهای لازم ارائه شود.

Document	"Users who bought this product" field
doc1	user1, user4, user5
doc2	user2, user3
doc3	user4
doc4	user4, user5
doc5	user4, user1
...	...

Term	Documents
user1	doc1, doc5
user2	doc2
user3	doc2
user4	doc1, doc3, doc4, doc5
user5	doc1, doc4
...	...

شکل ٤-۳- جدول ارتباط بین کاربران و اسناد

□ راهکار ۷- راهکار ترکیبی: ترکیب راهکارهای موجود می‌تواند سبب بهبود ارتباط بین سند و عبارت پرس‌جوی کاربر شود. به عنوان نمونه:
با ترکیب راهکار «مشابه این...» و راهکار «تطابق براساس مفهوم» می‌توان شباهت‌های بین اسناد را براساس کلمات کلیدی خوشه‌ها محاسبه نمود. این ترکیب می‌تواند نتایج false positive زیادی را به همراه داشته باشد اما اگر از فیلتر مبتنی بر دسته‌بندی (وزن‌دهی) استفاده شود نتایج بهتر خواهد شد.
همچنین ترکیب راهکار «پالایش همکارانه» با راهکار «تطابق براساس ویژگی‌ها» یا به عبارتی ترکیب اطلاعات مربوط به رفتار کاربران با ویژگی های اسناد، سیستم پیشنهاددهنده قدرتمندتری را می‌توان ایجاد نمود. علاوه‌براین، اضافه کردن ویژگی‌های جغرافیایی به هریک از روش ها نتایج بهتری را فراهم آورده و می‌تواند سبب افزایش کارایی شود.

۴-۶ جمع بندی

هدف از نگارش این فصل، بررسی مباحث پیشرفته‌تر در موتور جستجوی سولر بود. بدین منظور مباحثی در خصوص منابع داده بیرونی و نحوه فراخوانی آن ارائه شد. همچنین عملیات جستجوی پیچیده و نحوه بهبود ارتباط سند با عبارت پرس‌وجو بررسی گردید و از آنجا که سیستم‌های جستجو نوعی از سیستم‌های پیشنهاددهنده هستند، درخصوص شخصی‌سازی آنها نیز مباحثی مطرح شد.

فصل پنجم

موتور جستجوی گنج

۵-۱ مقدمه

در فصل‌های پیشین به نکات پایه‌ای سولر و نیز برخی نکات پیشرفته‌تر شامل نحوه فراخوانی و استفاده از داده‌های منابع بیرونی، عملیات جستجوی پیچیده، نحوه بهبود ارتباط سند با عبارت پرس وجو و شخصی سازی جستجواختصاص دا شت. در این فصل نگاهی خواهیم دا شت به آنچه در موتور جستجوی گنج رخ می‌دهد. برای این منظور از دو سند اصلی این سامانه، Schema و Solrconfig استفاده خواهد شد و توابع مورد استفاده در هریک تشریح می گردد.

۵-۲ سند SOLRCONFIG سامانه گنج

یکی از اسناد مهم برای تنظیم سیستم بازیابی اطلاعات، سند Solrconfig است. این سند برای سامانه گنج در ادامه تشریح می‌شوند.

متن سند Solrconfig سامانه گنج به شرح زیر ا ست. به جهت تسهیل در تفسیر سند، تو ضیحات مربوط به هر بخش در ادامه آن آمده است. همچنین اصل سند در ضمائم آورده خواهد شد[1].

۵-۳ سند SCHEMA سامانه گنج

[1] به جهت اهمیت این سند، متن آن در این گزارش آورده نشده است.

همانگونه که ذکر شد، ساختار نمایه سازی و نوع-فیلد و فیلدهای مورد استفاده در سند Schema تعیین و ذخیره می‌شود. متن این سند برای سامانه گنج و تشریح آن در ادامه آورده شده است[1].

۵-۴ جمع‌بندی

این فصل به بررسی دو سند اصلی سامانه گنج، Schema و Solrconfig اختصاص داشت و تلاش شد تا آنچه در موتور جستجوی گنج رخ می‌دهد بررسی شود. براساس مطالب عنوان شده، کاستی‌هایی وجود دارد که برای رفع آنها و بهبود عملکرد سامانه فعالیت‌ها و یا پژوهش‌هایی باید برنامه‌ریزی شوند. به این موارد در فصل بعد پرداخته خواهد شد.

[1] به جهت اهمیت این سند، متن آن در این گزارش آورده نشده است.

فصل ششم

راهکارها و روش‌های بهبود

۶-۱ مقدمه

با بهره‌گیری از مباحث مطرح شده در فصل قبل (وضعیت فعلی) می‌توان پیشنهادها و راهکارهایی را به منظور بهبود عملکرد سامانه گنج و بازیابی اطلاعات در زبان فارسی و برای این سامانه ارائه داد. این فصل به این موارد اختصاص دارد.

۶-۲ راهکارهای بهبود بازیابی اطلاعات در سامانه گنج

طبق نیازمندی‌های عمومی سامانه‌های بازیابی اطلاعات اشاره شده در (Manning and Schütze ۱۹۹۹) یک سیستم بازیابی اطلاعات برای پردازش اسناد موجود در پایگاه داده خود بایستی دارای اجزا و قابلیت‌های زیر باشد:

- □ شاخص معکوس به همراه اطلاعات مکانی نمایه‌ها و فراوانی واژگان در هر سند
- □ فهرستی از ایست‌واژه‌ها
- □ قابلیت تحلیل متن (شامل واژه–واژه سازی و ریشه‌یابی و یا بُن‌واژه‌سازی)
- □ قابلیت محاسبه ارتباط بین عبارت پرس‌وجوی کاربر و اسناد بازیابی شده

همچنین با بررسی وضعیت فعلی دو فایل Solrconfig و Schema سامانه گنج نکات زیر را می‌توان ازجمله کاستی‌های بازیابی اطلاعات سامانه گنج دانست:

- درحال حاضر برای برای ریشه‌یابی واژگان در د اسناد در زمان نمایه سازی (بخش ۴ سند Shema) و تحلیل عبارت پرس‌و جو (بخش ۵ سند Schema) از دستور solr.PorterStemFilterFactory (روش پورتر) استفاده می‌شود که روشی برای ریشه‌یابی در زبان انگلیسی است.

- در حال حاضر در سامانه گنج از روش solr.StandardTokenizerFactory برای واژه-واژه سازی متن و عبارت پرس‌وجو استفاده می‌شود. این روش مانند روش solr.KeywordTokenizerFactory (بخش ۲ سند Schema) برای زبان انگلیسی طراحی شده و بکارگیری آن برای اسناد موجود در سامانه گنج و یا عبارت‌های پرس‌وجو که اغلب به زبان فارسی هستند نتیجه‌ای را دربر نخواهد داشت.

- علاوه‌بر موارد فوق، در بخش ۲ سند Schema به نوع-فیلد autosuggest با وظیفه ارائه پیشنهادهای عبارت پرس‌وجو به کاربر در زمان ورود عبارت پرس‌وجو ا شاره شده که از کلاس solr.LowerCaseFilterFactory برای کوچک‌سازی واژگان استفاده می‌شود. این روش نیز برای زبان انگلیسی طراحی و توسعه داده شده و استفاده از آن برای واژگان فارسی کارایی ندارد. همچنین در حال حاضر برای سامانه گنج پیشنهادهای عبارت پرس‌وجو با کمک نمایه‌های موجود در Lucene و با انطباق ۳ کاراکتر وارد شده توسط کاربر با نمایه‌های ذخیره شده انجام می‌شود و در عمل از واژه-واژه کردن و کوچک‌سازی واژگان استفاده‌ای نمی‌شود.

- در سامانه گنج از دستور solr.PatternReplaceCharFilterFactory برای جایگزین کردن الگوهای نو شتاری ا ستفاده می شود. الگوی مورد ا ستفاده در سامانه گنج چه در زمان نمایه سازی و چه در زمان تحلیل عبارت پرس‌وجو یکسان بوده و به صورت تجربی استخراج شده‌اند.

- در سامانه گنج برای جایگزین کردن کاراکترهای متن با کاراکترهای از قبل تعیین شده از دستور solr.MappingCharFilterFactory استفاده می‌شود. فایل مورد استفاده

برای این منظور همان فایل اصلی طراحی شده برای سولر برای زبان انگلسی بوده و تغییری در آن اعمال نشده است.

- روش محاسبه ارتباط بین عبارت پرس‌وجوی کاربر و اسناد بازیابی شده در سولر به صورت پیش‌فرض و در سامانه گنج روش TF-IDF است که روشی پایه‌ای برای بازیابی اطلاعات است (هرچند که تغییرات و بهبودهایی در فرمول آن انجام شده است (رابطه (1-3)). همچنین، ازآنجا که فرآیند تحلیل متن چندانی بر روی اسناد و عبارت‌های پرس‌وجوی کاربر در سامانه گنج انجام نمی‌شود، نمی‌توان انتظار کارایی بالایی از روش TF-IDF داشت.

- در سامانه گنج عبارت پرس‌وجوی کاربر تجزیه نمی‌شود چرا که در بخش ۸ سند Solrconfig، مقدار پارامتر `handleSelect` برابر false قرار داده شده است و این سبب می‌شود تا عبارت پرس‌وجو به تجزیه‌گر فرستاده نشود. به بیانی دیگر، در سامانه گنج عبارت پرس‌وجوی کاربر تجزیه نشده و فرآیند تحلیل متن بر آن انجام نمی‌شود.

- طبق بخش ۹ و ۱۸ سند Solrconfig، سامانه گنج برای کنترل املاء و پیشنهاد خودکار از لغت‌نامه پیش‌فرض سولر استفاده می‌کند که در عمل برای زبان فارسی غیرقابل استفاده است. به همین دلیل است که درصورتی که کاربر اشتباه املایی در عبارت پرس‌وجو داشته باشد، هیچ پیشنهادی به وی برای اصلاح ارائه نمی‌شود.
چنین کاستی در زمان نمایه‌سازی و سپردن اسناد نیز وجود دارد. بدین معنا که فرآیند کنترل املاء برای اسنادی که وارد فرآیند نمایه سازی می شوند انجام نمی شود. لذا، بسیار محتمل است در فیلدهای نمایه شده اسناد پایگاه گنج اشتباهات املائی وجود داشته و این امر سبب می‌شود تا این سامانه عملکرد نامناسبی در بازیابی اطلاعات داشته باشد.

- از قابلیت‌های More like this، Facet و Highlight در سامانه گنج استفاده نمی‌شود.
با کمک قابلیت More like this سولر می‌تواند اسناد مشابه با سند مورد انتخابی کاربر را به او نمایش دهد. با این حال، این قابلیت برای سامانه گنج غیرفعال است.
با کمک قابلیت Facet کاربر قادر است نتایج جستجو را براساس نیاز خود فیلتر نموده و نمایی کلی از خصوصیات اسناد موجود در پایگاه داده را درخصوص موضوع مورد

جستجو مشاهده نماید (شکل ۳-۱۸). درحال حاضر این قابلیت برای سامانه گنج غیرفعال است.

پررنگ کردن بخش‌هایی از سند که با عبارت پرس‌وجوی کاربر منطبق هستند اتفاقی است که پس از فعال کردن مولفه Highlight می‌افتد (شکل ۳-۲۱). درحال حاضر این قابلیت برای سامانه گنج غیرفعال است و آنچه با عنوان پررنگ کردن در صفحه نتایج نمایش داده می‌شود قابلیتی است که طراحان سامانه گنج آن را طراحی و پیاده‌سازی نموده‌اند و مجزا از این قابلیت سولر است.

بنابراین بر اساس نیازمندی‌های عمومی سامانه‌های بازیابی اطلاعات و کاستی‌های فوق راهکارهای بهبود بازیابی اطلاعات در زبان فارسی و برای سامانه گنج را می‌توان به صورت زیر بیان نمود. این راهکارها در دو دسته راهکارهای نیازمند پژوهش بیشتر و عمیق‌تر و راهکارهای فنی و عملیاتی دسته‌بندی شده‌اند.

۱-۲-۶- راهکارهای نیازمند پژوهش

- استفاده از روش‌های تحلیل متن برای زبان فارسی ؛
  - درحال حاضر در سامانه گنج برای واژه-واژه سازی متن و عبارت پرس‌وجو از روش استانداردی که برای زبان انگلیسی طراحی شده استفاده می‌شود و لذا بکارگیری آن برای اسناد موجود در سامانه گنج و یا عبارت‌های پرس‌وجو که اغلب به زبان فارسی هستند نتیجه‌ای را دربر نخواهد داشت. بنابراین بکارگیری و یا توسعه روش(هایی) که بتوانند فعالیت واژه-واژه سازی را در سامانه گنج انجام دهند ضروری است.

  - درحال حاضر در سامانه گنج برای ریشه‌یابی از روش پورتر استفاده می شود و ازآنجا که روش پورتر روشی برای ریشه‌یابی در زبان انگلیسی است، توسعه و پیاده‌سازی روش‌های دیگر برای زبان فارسی از ضروریات است.

  - در حال حاضر در سامانه گنج از تجزیه‌گر استفاده نشده و عبارت پرس‌وجوی کاربر به تجزیه‌گر فرستاده نمی‌شود. به بیانی دیگر، در سامانه گنج عبارت پرس‌وجوی کاربر تجزیه نشده و فرآیند تحلیل متن بر آن انجام نمی‌شود. این

موضوع چالشی جدی برای بازیابی اطلاعات بوده و ضروری است برای بهبود نتایج بازیابی اطلاعات در سامانه گنج بدان پرداخته شود.

- در سامانه گنج برای کنترل املاء و پیشنهاد خودکار از لغت‌نامه پیش‌فرض سولر استفاده می‌شود که در عمل برای زبان فارسی غیرقابل استفاده است. به همین دلیل است که درصورتی که کاربر اشتباه املایی در عبارت پرس‌وجو داشته با شد، هیچ پیشنهادی به وی برای ا صلاح ارائه نمی شود. چنین کا ستی در زمان نمایه‌سازی و سپردن اسناد نیز وجود دارد. بدین معنا که فرآیند کنترل املاء برای اسنادی که وارد فرآیند نمایه‌سازی می‌شوند انجام نمی‌شود. بنابراین ضروری است با بررسی تجارب گذشته و نیز نمایه‌های موجود، لغت‌نامه‌ای برای کنترل املای کاربران و اسناد نمایه شده تدوین گردد.

- در سامانه گنج از دستور `solr.PatternReplaceCharFilterFactory` برای جایگزین کردن الگوهای نو شتاری ا ستفاده می شود. الگوی مورد ا ستفاده در سامانه گنج چه در زمان نمایه‌سازی و چه در زمان تحلیل عبارت پرس‌وجو یکسان بوده و به صورت تجربی استخراج شده‌اند. به نظر می‌رسد تعیین دقیق‌تر و علمی‌تر این الگوها می‌تواند سبب شود تا داده‌های ورودی بهتری برای فرآیند تحلیل متن حاصل شوند.

- در سامانه گنج برای جایگزین کردن کاراکترهای متن با کاراکترهای از قبل تعیین شده از دستور `solr.MappingCharFilterFactory` استفاده می‌شود. فایل مورد استفاده برای این منظور همان فایل اصلی طراحی شده برای سولر بوده و تغییری در آن اعمال نشده است و از آنجا که سولر برای زبان انگلسی طراحی شده است، تدوین آن برای زبان فارسی ضروری است.

□ ارتقاء روش محاسبه ارتباط بین عبارت پرس‌وجوی کاربر و اسناد بازیابی شده

روش محاسبه ارتباط بین عبارت پرس‌وجوی کاربر و اسناد بازیابی شده در سامانه گنج روش TF-IDF است که روشی پایه‌ای برای بازیابی اطلاعات بوده و روش‌های جایگزین دیگری با کارایی بالاتر در ادبیات برای این منظور ارائه شده‌اند. همچنین، ازآنجا که

فرآیند تحلیل متن چندانی بر روی اسناد و عبارت‌های پرس‌وجوی کاربر در سامانه گنج انجام نمی شود، نمی‌توان انتظار کارایی بالایی از روش TF-IDF دا شت. بنابراین پیشنهاد می‌شود در پژوهش‌های آتی علاوه بر توسعه روش‌های تحلیل متن برای زبان فارسی، روش‌های جدیدتر بازیابی اطلاعات نیز توسعه یابند.

علاوه بر دو راهکار فوق ضروری ا ست برخی از قابلیت‌های سولر برای زبان فار سی و در سامانه گنج فعال شوند. این قابلیت‌ها عبارتند از:

- فعال‌سازی قابلیت More like this: با کمک قابلیت More like this سولر می‌تواند اسناد مشابه با سند مورد انتخابی کاربر را به او نمایش دهد. درحال حاضر این قابلیت برای سامانه گنج غیرفعال است و از آنجا که این قابلیت برای کاربران جذاب است پیشنهاد می‌شود پژوهش‌های جدی در این زمینه انجام شود.

- فعال سازی قابلیت Facet: با کمک این قابلیت کاربر قادر ا ست نتایج جستجو را برا ساس نیاز خود فیلتر نموده و نمایی کلی از خصوصیات اسناد موجود در پایگاه داده را درخصوص موضوع مورد جستجو مشاهده نماید. به تعبیری دیگر، با کمک این مولفه فراداده مربوط به اسناد مرتبط با عبارت پرس‌وجو برای کاربر نمایش داده می‌شود. درحال حاضر این قابلیت برای سامانه گنج غیرفعال است و از آنجا که این قابلیت برای کاربران بسیار کاراست پیشنهاد می‌شود بدان پرداخته شود.

- فعال‌سازی قابلیت Highlight: با استفاده از این قابلیت بخش‌هایی از سند که با عبارت پرس‌وجوی کاربر منطبق هستند پررنگ می‌شود. درحال حاضر این قابلیت برای سامانه گنج غیرفعال است و از آنجا که این قابلیت نیازمند فرآیند تحلیل متن است (شکل ۳-۲۲) پیشنهاد می‌شود برای فعال‌سازی آن پژوهش‌های بیشتری صورت پذیرد.

۲-۲-۶- راهکارهای فنی و عملیاتی

- فعال کردن JMX برای کنترل و ردیابی عملکرد سولر؛
  این می‌تواند ابزاری مناسب برای پشتیبانی از سامانه گنج باشد.

- جایگزینی سیاست‌های مدیریت Cache؛

در حال حاضـر، روش مورد اسـتفاده برای مدیریت Cache، نگهداری نتایج جسـتجوهای قبلی، نگهداری اسـناد برای جسـتجوهای بعدی و مدیریت حافظه کوتاه‌مدت در سـامانه گنج روش پیش‌فرض LRU اسـت. سـیاسـت Least Frequently Used (LFU) نیز در سولر تعریف شده است که در طی آن اسناد یا نتایج بازیابی شده با کمترین استفاده (غیر جذاب‌ترین) از حافظه پاک می‌شـود. بنابراین می‌توان روش LFU را در مواردی امتحان کرده، نتایج و ر ضایت کاربران را با حالت پیش‌فرض سنجیده و در صورت نیاز تنظیمات را تغییر داد.

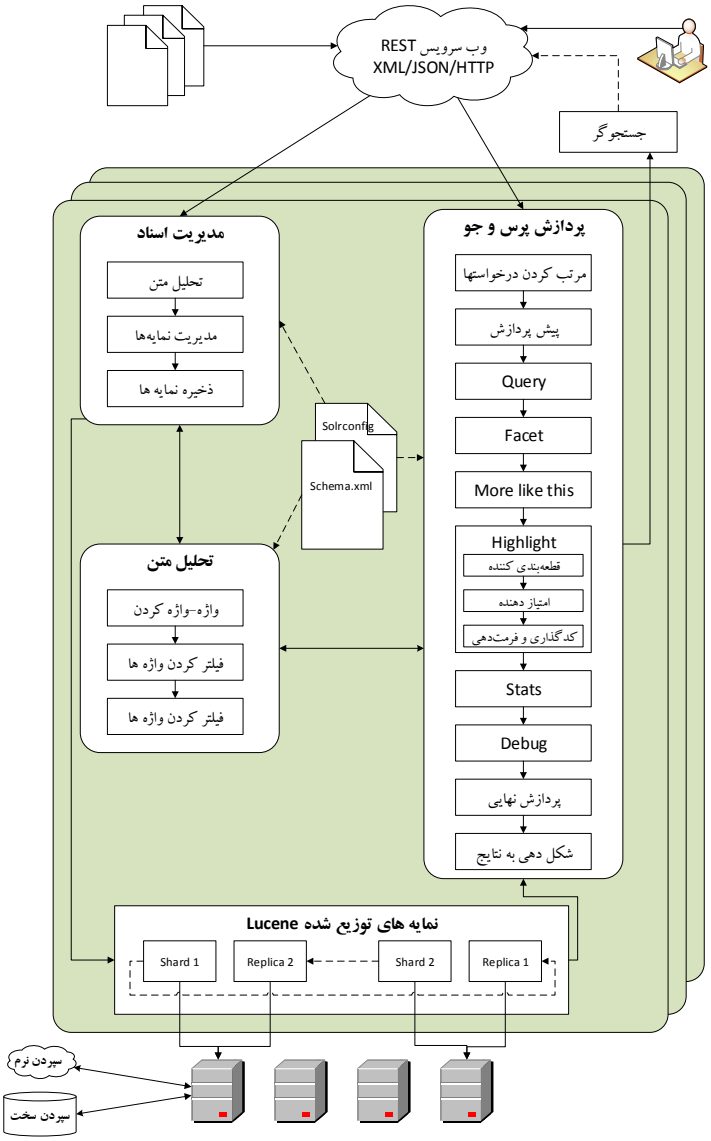
- □ فعال کردن قابلیت جستجوی جغرافیایی.

یکی از امکانات و قابلیت‌های گنج پشـتیبانی از جسـتجوی جغرافیایی اسـت که در حال حاضر برای سامانه گنج فعال نیست.

البته موارد دیگری نیز در این رابطه وجود دارند که به علت عدم تمرکز طرح حاضـر بر کدهای اجرایی سولر بدان پرداخته نشده است.

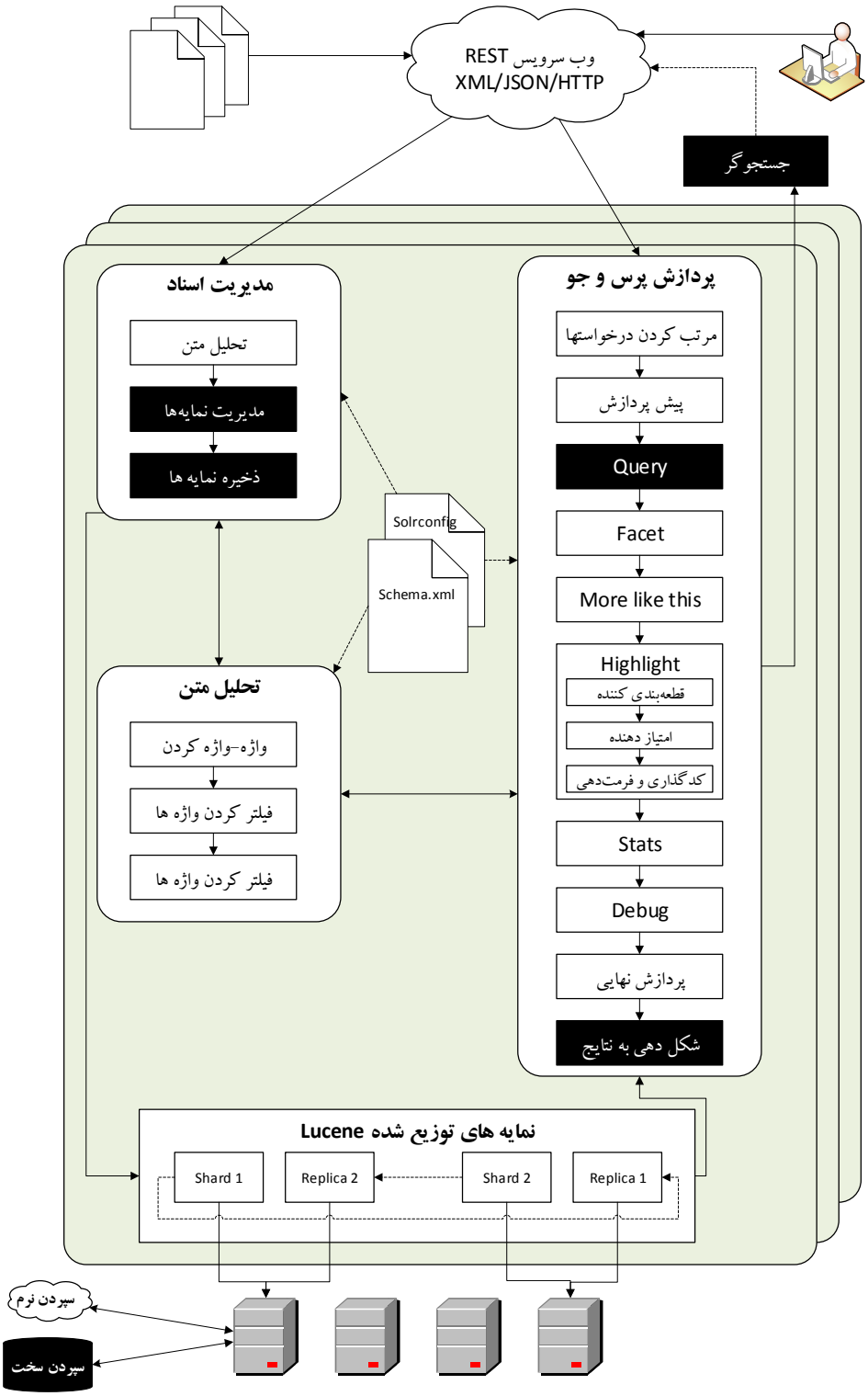
جمع‌بندی و نتیجه‌گیری

فرآیند پردازش اسناد و پرس‌وجو در سولر به صورت خلاصه در شکل زیر نمایش داده شده است. در طی این فرآیند اسناد ورودی به پایگاه داده توسط لوسن نمایه شده و ذخیره می‌شوند. درخواست‌های جستجوی کاربر نیز تحلیل شده و نتایج براساس تنظیمات مشخص شده در Solrconfig و Schema بازیابی شده و به کاربر نمایش داده می‌شوند. فرآیند تحلیل متن جزء کلیدی سولر و لوسن بوده و به عنوان ابزاری در زیرسیستم‌های مدیریت اسناد و پردازش پرس‌وجو استفاده می‌شود.



فرآیند پردازش اسناد و پرس‌وجو در سولر

سامانه گنج و موتور جستجوی آن نیز از چنین ساختاری برخوردار است. با بررسی‌های انجام شده می‌توان عنوان داشت که تنها مراحل پررنگ شده در شکل زیر در سامانه گنج و برای زبان فارسی انجام شـــده و بقیه مراحل همچنان غیرفعال اســـت (برخی نیازمند پژوهش‌های عمیق‌تر و توســـعه و پیاده‌سازی روش‌های مبتنی بر زبان فارسی و برخی نیز نیازمند فعال‌سازی فنی و عملیاتی است).



از سویی دیگر، با توجه به ضرورت وجود قابلیت تحلیل متن (شامل واژه–واژه سازی و ریشه‌یابی و یا بُن‌واژه‌سازی) و قابلیت محاسبه ارتباط بین عبارت پرس‌وجوی کاربر و اسناد بازیابی شده در سامانه‌های بازیابی اطلاعات در (Manning and Schütze ۱۹۹۹)، نکات زیر را می‌توان ازجمله راهکارهای ضروری و نیازمند پژوهش بیشتر و عمیق‌تر برای بهبود بازیابی اطلاعات در زبان فارسی در سامانه گنج دانست.

- □ استفاده از روش‌های جایگزین برای ریشه‌یابی در زبان فارسی و پیاده سازی آن در سامانه گنج (درحال حاضر از روش پیش‌فرض پورتر استفاده می‌شود که در زبان فارسی کارایی ندارد).
- □ بکارگیری و پیاده‌سازی روش‌های برای واژه–واژه سازی متن و عبارت پرس‌وجو در سامانه گنج (درحال حاضر از روش‌های StandardTokenizer و KeywordTokenizer برای واژه–واژه سازی متن و عبارت پرس‌وجو استفاده می شود که برای زبان انگلیسی طراحی شده و بکارگیری آن برای اسناد موجود در سامانه گنج و یا عبارت‌های پرس‌وجو که اغلب به زبان فارسی هستند نتیجه‌ای را دربر نخواهد داشت).
- □ استفاده از تجزیه‌گر در سامانه گنج (درحال حاضر عبارت پرس‌وجوی کاربر تجزیه نشده و فرآیند تحلیل متن بر آن انجام نمی‌شود).
- □ توسعه و ارتقاء روش بازیابی اطلاعات در سامانه گنج (روش جستجوی مورد استفاده، روش TF-IDF است و ازآنجا که فرآیند تحلیل متن چندانی بر روی اسناد و عبارت‌های پرس‌وجو در سامانه گنج انجام نمی‌شود، نمی‌توان انتظار کارایی بالایی از روش TF-IDF داشت).

راهکارهای نیازمند پژوهش دیگری نیز وجود دارند که توجه به آنها می‌تواند بهبود مناسبی را در بازیابی اطلاعات در سامانه گنج ایجاد نماید.

- □ تعیین دقیق‌تر و علمی‌تر جایگزینی الگوهای نوشتاری برای سامانه گنج (درحال حاضر الگوی مورد استفاده چه در زمان نمایه‌سازی و چه در زمان تحلیل عبارت پرس‌وجو یکسان بوده و به صورت تجربی استخراج شده‌اند. به نظر می‌رسد تعیین دقیق‌تر و علمی‌تر این الگوها می‌تواند سبب شود تا داده‌های ورودی بهتری برای فرآیند تحلیل متن حاصل شوند).

- □ تدوین فایل مورد نیاز برای برای جایگزین کردن کاراکترهای متن با کاراکترهای از قبل تعیین شده در سامانه گنج (درحال حاضر فایل مورد استفاده همان فایل اصلی طراحی شده برای سولر بوده و تغییری در آن اعمال نشده است و از آنجا که سولر برای زبان انگلسی طراحی شده است، تدوین آن برای زبان فارسی ضروری است).

- □ فعال‌سازی و پیاده‌سازی کنترل املاء در سامانه گنج (در حال حاضر از لغت‌نامه پیش‌فرض سولر برای کنترل املاء و پیشنهاد خودکار استفاده می شود که در عمل برای زبان فارسی غیرقابل استفاده است. به همین دلیل است که درصورتی که کاربر اشتباه املایی در عبارت پرس‌وجو داشته باشد، هیچ پیشنهادی به وی برای اصلاح ارائه نمی‌شود. چنین کاستی در زمان نمایه‌سازی و سپردن اسناد نیز وجود دارد. بدین معنا که فرآیند کنترل املاء برای اسنادی که وارد فرآیند نمایه سازی می شوند انجام نمی شود. لذا، بسیار محتمل است در فیلدهای نمایه شده اسناد پایگاه گنج اشتباهات املائی وجود داشته و این امر سبب می‌شود تا این سامانه عملکرد نامناسبی در بازیابی اطلاعات داشته باشد).

علاوه‌بر موارد فوق، موتور جستجوی سولر دارای قابلیت‌هایی است که در حال‌حاضر در از آنها در سامانه گنج استفاده نمی‌شود و فعال‌سازی و پیاده‌سازی آنها می‌تواند در جلب نظر کاربران این سامانه موثر باشد. این قابلیت‌ها عبارتند از:

- □ قابلیت More like this
- □ قابلیت Facet
- □ قابلیت Highlight
- □ قابلیت جستجوی جغرافیایی

همچنین، در حال حاضر، روش مورد استفاده برای مدیریت Cache سامانه گنج، روش پیش‌فرض LRU است که می‌توان آن را با روش LFU جایگزین کرده، نتایج و رضایت کاربران را با حالت پیش‌فرض سنجیده و درصورت نیاز تنظیمات را تغییر داد.

ضمائم

۷-۱ توابع

سولر این امکان را برای کاربران خود فراهم می‌آورد تا علاوه بر انجام جستجو براساس عبارت پرس‌وجو، توابعی را اجرا کرده و از نتایج آن به صورت عبارت پرس‌وجوی جدید و یا پارامتری برای تعیین میزان ارتباط نتایج بازیابی شده استفاده کند. برخی از این توابع در ادامه آورده شده‌اند.

۷-۱-۱- توابع ریاضی

برخی از توابع ریاضی قابل استفاده در سولر به شرح زیر هستند:

جدول ۷-۱- توابع ریاضی در سولر

دستور (syntax) تابع	توضیحات
`abs(x)`	قدرمطلق مقدار x
`sin(x), asin(x), sinh(x)`	سینوس، آرک‌سینوس و سینوس هیپربولیکx
`cos(x), acos(x), cosh(x)`	کسینوس، آرک‌کسینوس و کسینوس هیپربولیک x
`tan(x), atan(x), tanh(x)`	تانژانت، آرک تانژانت و تانژانت هیپربولیک x
`exp(x)`	تابع نمایی x
`ln(x)`	لگاریتم طبیعی x
`log(x)`	لگاریتم x
`rad(x)`	تبدیل x به رادیان
`sqrt(x)`	ریشه دوم x

x/y	`div(x,y)`
$x-y$	`sub(x,y)`
x^y	`pow(x,y)`
مقدار تابع $f(x)=a/mx+b$	`recip(x,m,a,b)`
محاسبه عدد پی	`pi()`
ضرب تمامی ورودی‌ها	`product(x,...n)` `mul(x,...n)`
جمع تمامی ورودی ها	`add(x,...n)`

۲-۱-۷- توابع فاصله

تابع فا صله پیش‌فرض در سولر تابع فا صله دامرا-لونا شتاین ا ست اما توابع دیگری نیز وجود دارد که لیستی از آنها در جدول ۷-۲ آمده است.

جدول ۷-۲- توابع فاصله در سولر

دستور (syntax) تابع	توضیحات
`dist(power,x1…n1,x2…n2)`	محاسبه فاصله دو بردار/نقطه در فضای n بعدی براساس معیار فاصله تعیین شده با power: $power=0\rightarrow$ Sparseness calculation $power=1\rightarrow$ Manhattan distance $power=2\rightarrow$ Euclidean distance $power=\text{infinity}\rightarrow$ Infinit norm
`sqdist(x1…n1,x2…n2)`	محاسبه فاصله اقلیدسی
`hsin(radius INKM, isDegrees,x1,y1,x2,y2)`	محاسبه فاصله Haversine

`geohash(lat,lon)`
`geodis(sfield,lat,lon)`
`strdis(s1,s2,disType)`
`strdis(s1,s2,disType,ngramSize)`
	محاسبه فاصله جغرافیایی
	محاسبه شباهت یا فاصله کاراکترها در یک رشته. نتیجه محاسبه بین 0 (بدون شباهت) تا 1 (کاملا مشابه) تغییر می کند.
	disType می‌تواند مقادیر زیر را داشته باشد:

jw → Jaro-Winkler *edit* → Levenshtein distance ngram → NGram distance سایر توابع مورد نیاز برای فاصله را می‌توان در رابط کاربری StringDistance تعیین و یا تعریف کرد.	

۳-۱-۷- توابع منطق بولی

جدول ۷-۳- توابع منطق بولی در سولر

دستور (syntax) تابع	توضیحات
`and(x,y)`	true اگر هردوی x و y مقدار true داشته باشند
`exists(x)`	true اگر مقداری برای x وجود داشته باشد
`if(x, trueValue, falseValue)`	مقدار trueValue را بر می‌گرداند اگر x درست باشد، درغیر این‌صورت مقدار falseValue را بر می‌گرداند
`not(x)`	false اگر x، true باشد و true اگر x، false باشد
`or(x,y)`	true اگر x یا y یا هردو true داشته باشند
`xor(x,y)`	true اگر x یا y، true داشته باشند و false اگر هردو true داشته باشند

۷-۲ عوامل محاسبه ارتباط

جدول ۷-٤- عوامل محاسبه ارتباط در سولر

دستور (syntax) عامل	توضیحات
`docfreq(fieldName,term)`	تعداد اسنادی حاوی واژه مورد نیاز در فیلد مشخص شده
`idf(fieldName,value)`	محاسبه idf برای ارزش خاص در فیلد مشخص شده
`maxdoc()`	تعداد کل اسناد (شامل تعداد اسنادی که از حافظه کوتاه مدت حذف شده‌اند اما همچنان در durable storage قرار دارند)
`sumtotaltermfreq(field)`	تعداد واژگان (token) نمایه شده
`sttf(field)`	

`termfreq(fieldname,term)`	تعداد تکرار واژه خاص در فیلد مشخص شده در یک سند
`tf(fieldname,term)`	تعداد تکرار واژه خاص در فیلد مشخص شده
`totaltermfreq(fieldname,term)` `ttf(fieldname,term)`	تعداد دفعات تکرار واژه در کل اسناد نمایه شده

۳-۷ کلاس‌های محاسبه ارتباط

سولر به صورت پیش‌فرض از `DefaultSimilarity` برای محاسبه شباهت بین عبارت پرس‌وجوی کاربر و اسناد موجود استفاده می‌کند اما کلاس‌ها و روش‌های دیگری نیز وجود دارند که می‌توان از آنها بجای کلاس پیش‌فرض استفاده کرد (جدول ۷-۵). این کلاس‌ها را می‌توان به صورت کلی برای کل سیستم جستجو در Schema فعال کرد و یا به صورت محلی و برای فیلدی خاص (خصوصیت similarity برای هر فیلد به صورت مجزا قابل تعریف است).

جدول ۷-۵- کلاس‌های محاسبه ارتباط در سولر

دستور (syntax) عامل	توضیحات
`DefaultSimilarity`	محاسبه براساس رابطه (*1-3*)
`BM25Similarity`	محاسبه براساس مدل BM25
`DFRSimilarity`	محاسبه براساس مدل Divergence from randomness (DFR)
`IBSimilarity`	محاسبه براساس مدل Information based (IB) که مشابه DFR است اما ورودی‌های ساده‌تری دارد
`LMDirichletSimilarity`	استفاده از مدل‌های زبانی برای اصلاح وزن واژه در پیکره واژگانی
`LMJelinekMercerSimilarity`	استفاده از مدل‌های زبانی مانند LMDirichletSimilarity اما با پیاده‌سازی آسان‌تر
`SweetSpotSimilarityFactory`	توسعه یافته DefaultSimilarity با قابلیت تنظیم پارامترهای tf و lenghtNorm

۷-۴ خصوصیات UPDATE HANDLER

درخواست اضافه‌کردن اسناد از طریق واسط‌هایی به سولر ارائه شده و از طریق Update Handler اجرا می‌شود. خصوصیات این Handler به شرح جدول ۷-۶ است.

جدول ۷-٦- خصوصیات Update Handler

نوع درخواست‌ها	توضیحات
Add	اضافه کردن یک یا چند سند
Delete	حذف یک سند با استفاده از ID
Delete by query	حذف سندی که با پرس‌وجوی لوسن مطابقت دارد
Atomic update	بروزرسانی یک یا چند فیلد در یک سند
Commit	سپردن اسناد به نمایه‌سازی
Optimize	بهینه کردن نمایه‌ها با ادغام بخش‌ها و حذف موارد حذف شده

۷-۵ پارامترهای مولفه HIGHLIGHT

برخی از پارامترهای مولفه highlight برای تنظیم در Schema به شرح است.

جدول ۷-۷- برخی از پارامترهای مولفه highlight

پارامتر	توضیحات	مقدار پیش فرض
h1	فعالسازی مولفه highlight	False
h1.snippets	بیشینه تعداد snippetها برای نمایش	1
h1.f1	فیلدهایی که snippetها براساس آن ساخته شود	--
h1.fragmenter	روش قطعه‌بندی کردن متن	Gap
h1.fragsize	اندازه قطعات	100
h1.q	تعیین عبارت برای پررنگ کردن	--
h1.alternateField	اگر سیستم قادر نباشد snippet بسازد، چه مواردی را در صفحه نتایج نمایش دهد	--
h1.formatter	تعیین فرمت عبارت پررنگ شده	simple
h1.maxAnalyzedChars	بیشینه تعداد کاراکترهای مورد بررسی در مواردی که اسناد حجیم باشند	51200

۷-۶ پارامترهای مولفه SPELLCHECK

برخی از مهمترین پارامترهای مولفه Spellcheck برای تنظیم در Schema به شرح است.

جدول ۷-۸- برخی از پارامترهای مولفه Spellcheck

پارامتر	توضیحات
spellcheck	فعالسازی مولفه Spellcheck
spellcheck.build	اگر true، دستور به سولر برای ساخت دایرکتوری spell-check (درهنگام بکارگیری تابع DirectSolrSpellChecker فعالسازی این پارامتر ضروری است)
spellcheck.count	تعیین بیشینه تعداد پیشنهادها برای اصلاح
spellcheck.dictionary	معرفی نام و آدرس لغت‌نامه‌ای که پیشنهادها بر اساس آن ارائه می‌شوند
spellcheck.onlyMorePopular	اگر true، محدود کردن پیشنهادها به مواردی که بازخوانی بالاتری دارند
spellcheck.maxResultsForSuggest	تعیین حداقل تعداد اسناد بازیابی شده برای فعال‌سازی Spellcheck
spellcheck.collate	اگر true، جستجو براساس عبارت پرس‌وجوی کاربر انجام می‌شود و پیام Did you mean…? برای کاربر نمایش داده می‌شود.
spellcheck.maxCollations	تعیین بیشینه تعداد پرس‌وجوهای تولید شده توسط سولر
spellcheck.maxCollationTries	تعیین بیشینه تعداد دفعات جستجوی جایگزین

منابع

ارشـادی م.، رجبی ت.، شـیرانی ف.، رضـایی ن. (۱۳۹۴). کاربرد تکنیک تحلیل ریشـه در حل مشـکلات کیفی سامانه‌های اطلاعاتی تحقیقاتی: مطالعه موردی سامانه اشاعهٔ اطلاعات پایان‌نامه‌ها/ رساله‌های دانش‌آموختگان داخل کشور (گنج). *مدیریت اطلاعات*، ۱ (۲)، ۷۵–۸۹.

زرین بال م. (۱۳۹۷). *طراحی مدل مفهومی برای معنابخشی به خدمات ارائه شده در سامانه گنج*. تهران.

فتاحی س. (۱۳۹۶). *تحلیل رفتار جست‌وجوی اطلاعات پژوهشگران در موتور جست‌وجوی سامانه گنج*. تهران.

Ai, Qingyao, Liu Yang, Jiafeng Guo, and W Bruce Croft. 2016. "Analysis of the Paragraph Vector Model for Information Retrieval." In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, 133–42. ACM.

Aly, Robin, and Thomas Demeester. 2011. "Towards a Better Understanding of the Relationship between Probabilistic Models in IR." In *Conference on the Theory of Information Retrieval*, 164–75. Springer.

Amati, Gianni, and Cornelis Joost Van Rijsbergen. 2002. "Probabilistic Models of Information Retrieval Based on Measuring the Divergence from Randomness." *ACM Transactions on Information Systems* 20 (4): 357–89. https://doi.org/10.1145/582415.582416.

Bookstein, Abraham. 1980. "Fuzzy Requests: An Approach to Weighted Boolean Searches." *Journal of the American Society for Information Science* 31 (4): 240–47. https://doi.org/10.1002/asi.4630310403.

Brown, Peter J, and Gareth J F Jones. 2001. "Context-Aware Retrieval: Exploring a New Environment for Information Retrieval and Information Filtering." *Personal and Ubiquitous Computing* 5 (4): 253–63.

Bruza, Peter, and Dawei Song. 2003. "A Comparison of Various Approaches for Using Probabilistic Dependencies in Language Modeling." In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, 419–20. ACM.

Bush, Vannevar. 1945. "As We May Think." *The Atlantic*, 1–10.

Chen, Hsinchun. 1995. "Machine Learning for Information Retrieval: Neural Networks, Symbolic Learning, and Genetic Algorithms." *Journal of the American Society for Information Science* 46 (3): 194–216. https://doi.org/10.1002/(SICI)1097-4571(199504)46:3<194::AID-ASI4>3.0.CO;2-S.

Damerau, Fred J. 1964. "A Technique for Computer Detection and Correction of Spelling Errors." *Communications of the ACM* 7 (3): 171–76.

Dumais, S T, G W Furnas, T K Landauer, S Deerwester, and R Harshman. 1988.

"Using Latent Semantic Analysis to Improve Access to Textual Information." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 281–85. CHI '88. New York, NY, USA: ACM. https://doi.org/10.1145/57167.57214.

Fang, Hui, and ChengXiang Zhai. 2005. "An Exploration of Axiomatic Approaches to Information Retrieval." In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 480–87. ACM.

Grainger, Trey, and Timothy Potter. 2014. *Solr in Action*. New York: Manning Publications Co. https://doi.org/10.1017/CBO9781107415324.004.

Grossman, David A, and Ophir Frieder. 2012. *Information Retrieval: Algorithms and Heuristics*. Vol. 15. Springer Science & Business Media.

Hiemstra, Djoerd. 2000. "A Probabilistic Justification for Using Tf× Idf Term Weighting in Information Retrieval." *International Journal on Digital Libraries* 3 (2): 131–39.

Hofmann, Katja, Shimon Whiteson, and Maarten de Rijke. 2013. "Balancing Exploration and Exploitation in Listwise and Pairwise Online Learning to Rank for Information Retrieval." *Information Retrieval* 16 (1): 63–90.

Klouche, Khalil, Tuukka Ruotsalo, Luana Micallef, Salvatore Andolina, and Giulio Jacucci. 2017. "Visual Re-Ranking for Multi-Aspect Information Retrieval." In *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*, 57–66. ACM.

Kwok, K L. 1989. "A Neural Network for Probabilistic Information Retrieval." In *12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 23:21–30. New York, NY, USA: ACM. https://doi.org/10.1145/75335.75338.

Liu, Xiaodong, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. 2015. "Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval."

Manning, Christopher D., Prabhakar Ragahvan, and Hinrich Schutze. 2009. *An Introduction to Information Retrieval*. Cambridge, England: Cambridge University Press. https://doi.org/10.1109/LPT.2009.2020494.

Manning, Christopher D, and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT press.

Maron, M E, and J L Kuhns. 1960. "On Relevance, Probabilistic Indexing and Information Retrieval." *Journal of the ACM* 7 (3): 216–44. https://doi.org/10.1145/321033.321035.

McCandless, Michael, Erik Hatcher, and Otis Gospodnetic. 2010. *Lucene in Action*. 2nd ed. Stamford: Manning Publications Co.

Metzler, Donald, and W Bruce Croft. 2004. “Combining the Language Model and Inference Network Approaches to Retrieval.” *Information Processing & Management* 40 (5): 735–50.

Oard, Douglas W, and Anne R Diekema. 1998. “Cross-Language Information Retrieval.” *Annual Review of Information Science and Technology (ARIST)* 33: 223–56.

Ponte, Jay M, and W Bruce Croft. 1998. “A Language Modeling Approach to Information Retrieval.” In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 275–81. ACM.

Ribeiro, Berthier A N, and Richard Muntz. 1996. “A Belief Network Model for IR.” In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 253–60. ACM.

Rijsbergen, C J Van. 1986. “A Non-Classical Logic for Information Retrieval.” *The Computer Journal* 29 (6): 481–85. http://dx.doi.org/10.1093/comjnl/29.6.481.

Robertson, S., and K. Sparck-Jones. 1976. “Relevance Weighting of Search Terms.” *Journal of the American Society for Information Science* 27 (3): 129–46. https://doi.org/10.1002/asi.4630270302.

Robertson, S., S. Walker, S. Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1995. “Okapi at TREC-3.” *Nist Special Publication Sp* 109: 109.

Robertson, S., S Walker, and M Hancock-Beaulieu. 1995. “Large Test Collection Experiments on an Operational, Interactive System: Okapi at TREC.” *Information Processing & Management* 31 (3): 345–60. https://doi.org/https://doi.org/10.1016/0306-4573(94)00051-4.

Rocchio, Joseph John. 1971. “Relevance Feedback in Information Retrieval.” *The SMART Retrieval System: Experiments in Automatic Document Processing*, 313–23.

Salton, G. 1971. *The SMART Retrieval System-Experiments in Automatic Document Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Salton, G, A Wong, and C S Yang. 1975. “A Vector Space Model for Automatic Indexing.” *Communications of the ACM* 18 (11): 613–20. https://doi.org/10.1145/361219.361220.

Shen, Yelong, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. 2014. “A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval.” In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 101–10. ACM.

Zaragoza, Hugo, Djoerd Hiemstra, and Michael Tipping. 2003. “Bayesian Extension to the Language Model for Ad Hoc Information Retrieval.” In

Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, 4–9. ACM.

Zhai, ChengXiang. 2008. "Statistical Language Models for Information Retrieval." *Synthesis Lectures on Human Language Technologies* 1 (1): 1–141.

ABSATRACT

Information Retrieval (IR) is generally defined as finding document(s) (unstructured data) from dataset(s) to satisfy users' informational needs. Open-source search engine platforms are designed to enable the websites to find the related documents or data to users' query. Apache Lucene and Apache Solr are among the most well known open-source search platforms. Ganj search engine uses these two open-source platforms to search among the documents in its dataset. However, no research has ever been conducted in IranDoc to thoroughly investigate Apache Lucene and Solr. Thus, in order to improve Ganj search engine, especially its capabilities in Persian, the main structure of Apache Lucene and Solr should be investigated. This is the aim of the following research. Here, the main structure of Apache Lucene and Solr and Ganj modifications has been investigated and some recommendations for improving the results are proposed. These recommendations are divided in two categories: recommendations needing further research and functional recommendations.

Keywords: Information retrieval; Apache Lucene; Apache Solr; Ganj search engine

Iranian Research Institute
for Information Science and Technology
(I r a n D o c)

Research Project

Investigating Ganj search engine to improve Persian information retrieval

By:

Marzieh Zarinbal Masouleh

September 2019