Design Dokumentation

PSE Projekt Textmining

22. Mai 2014

Inhaltsverzeichnis

Einleitung

Aufgabe

Die Hauptaufgabe in unserem Projekt ist es, auf einem vorgegebenen Suchraum (in unserem Fall der deutschen Wikipedia) eine Suche zu implementieren, welche die relevantesten Seiten zum Thema Medizin ausgibt.

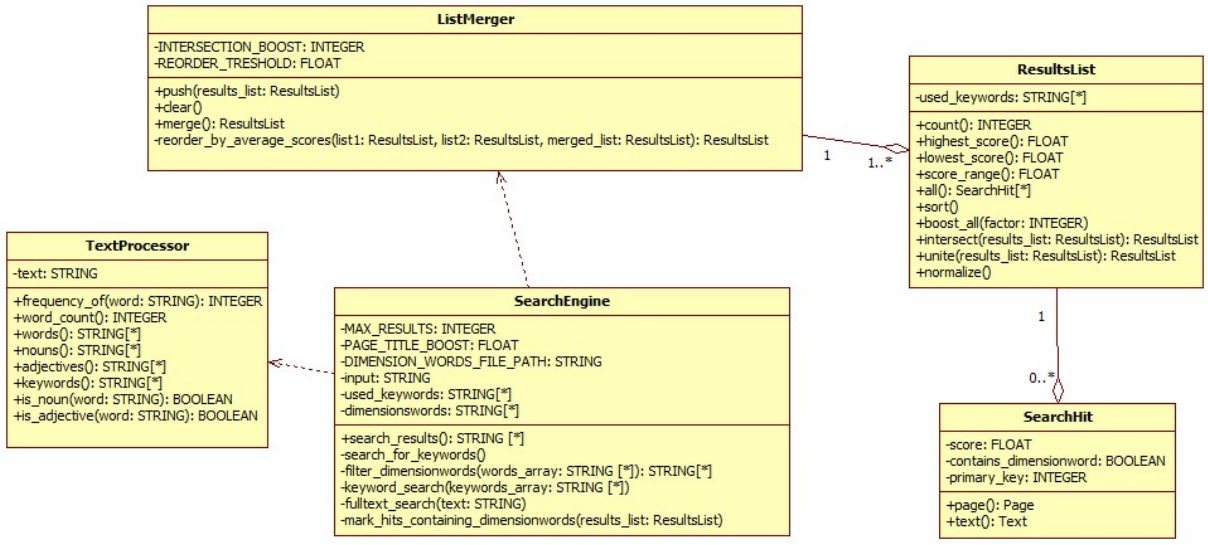
Installation

Das gesamte Projekt ist auf GitHub verfügbar. Es kann unter github.com/pse-group2/medminersolr geklont werden. Die Anweisungen zur Installation sind direkt auf der Homepage oder im Readme zu finden.

Übersicht über das System

Aufbau

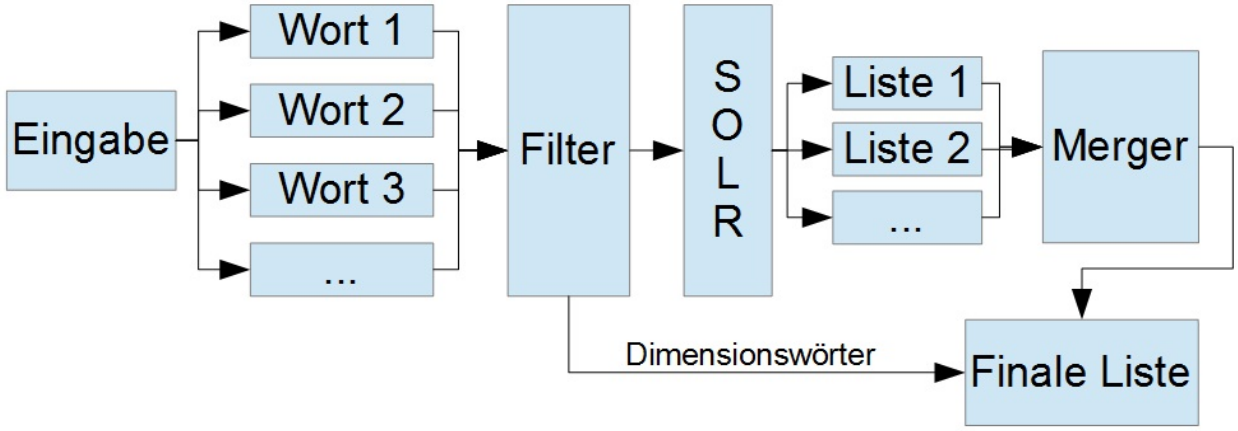
Die Grundlage dieses Projekts ist das Open-Source Web Framework Ruby on Rails. Zusätzlich verwenden wir den Suchserver Apache Solr, welchen wir auf die medizinische Suche zugeschnitten haben und mit dem wir den medizinischen Bereich der deutschen Wikipedia indexieren. Den zu indexierenden Text der Wikipedia speichern wir in einer MySQL Datenbank ab.



In der obigen Abbildung wird das Modell des Projektes durch ein Klassendiagramm beschrieben. Die SearchEngine nimmt eine Sucheingabe entgegen und delegiert Aufgaben wie Filterung nach Schlüsselwörtern, Suche nach Schlüsselwörter oder Zusammensetzen der Ergebnislisten vom Suchserver an die zuständigen Entitäten.

Ablauf

Zunächst wird die aus mehreren Wörtern bestehende Eingabe in ihre Einzelwörter aufgeteilt. In einem ersten Filter werden Dimensionswörter, welche separat in einer Datei definiert sind, entfernt und zwischengespeichert. Die übrigen Wörter sind Schlüsselwörter oder Stoppwörter. Für diese wird dann einzeln eine Suchanfrage an Solr gemacht, wobei für die Stoppwörter keine Resultate zurückgegeben werden. Somit sind nur für Schlüsselwörter Ergebnislisten vorhanden. Bei diesen einzelnen Suchen werden durch Solr die vordefinierten Synonyme einbezogen (mehr dazu im Abschnitt zur Solr Konfiguration). In einem letzten Schritt werden nun die individuellen Ergebnislisten zu einer einzigen Liste zusammengeführt. Dies passiert im *ListMerger*. Dimensionswörter werden erst in der finalen Liste wieder behandelt.



Von der Eingabe zum Ergebnis

Vorfilterung

Der Benutzer kann natürlich jede Art von Eingabe generieren. Deshalb ist es wichtig, dass vor der Solr Suche eine grobe Filterung durchgeführt wird. Dazu wird ein NLP Gem namens Treat verwendet. Wir haben dieses Tool gewählt, da es sich sehr gut zur Aufspaltung der Eingabe in Wörter eignet (Tokenization). Diese Aufgaben sind praktischerweise in einer eigenen Klasse *TextProcessor* gebündelt (siehe Klassendiagramm).
Bei der Vorfilterung wird keine Aufspaltung der Wörter in Unterwörter gemacht. Dies ist direkt in der Solr Konfiguration eingebaut. Nach der Tokenization werden noch Dimensionswörter, welche im File *dimensionwords.txt* aufgelistet sind, herausgenommen. Diese sollen nicht in die Suche direkt miteinbezogen werden, da eine Gefahr der

Verfälschung besteht. Artikel, welche Dimensionswörter aus der Eingabe enthalten, werden dann am Schluss speziell markiert.

Suche

Nach der Filterung sind jetzt nur noch die Schlüsselwörter übrig. Auf diese Wörter wird nun Solr angesetzt, welcher zu jedem eine Liste mit Ergebnissen (Hits) generiert. In diesen Hits wird unter anderem ein von Solr berechneter Score hinterlegt, welcher im nachfolgend beschriebenen ListMerger verwendet wird.
Wir haben ausserdem mit einer zusätzlichen Klasse *SearchHit* den *Hit* von Sunspot Solr an unsere Bedürfnisse angepasst.

Bemerkung: Damit die Suche funktioniert, muss von Solr ein Index der Daten existieren. Dieser sollte einmalig bei der Installation oder bei jeder Änderung der Datenbank durchgeführt werden.

ListMerger

Der *ListMerger* bekommt von der *SearchEngine* eine Menge an Ergebnislisten (*ResultsList*). Die Aufgabe ist nun, diese Listen sinnvoll zusammenzuführen und dabei die Relevanz (Score) der Artikel einzubeziehen. Im Detail sind wir folgendermassen vorgegangen:

1. Es werden die zwei ersten Listen ausgewählt.
2. Auf diese wird eine Normierung angewendet, was bedeutet, dass alle Scores auf einen Wert zwischen 0 und 1 skaliert werden.
3. Die Schnittmenge der beiden Listen wird gebildet und die Scores der *SearchHits* in dieser Schnittmenge werden mit einem Boost versehen, welcher als Konstante vorgegeben ist.
4. Die übrigen *SearchHits* aus beiden Listen werden einfach an die temporäre Liste angehängt (ohne Boost).
5. Nachdem die Listen zusammgeführt sind, muss die Gewichtung der *SearchHits* neu gesetzt werden. Es kann sein, dass ein *SearchHit* in der einen Liste sehr weit oben ist und der gleiche *SearchHit* in der zweiten Liste sehr weit unten. Hier wird grundsätzlich das geometrische Mittel zur Gewichtung der Scores verwendet, mit einer Ausnahme: Wenn die Abweichung der Scores grösser als ein vordefinierter Schwellenwert ist (prozentual gewichtet), dann wird das Maximum der beiden ausgewählt. Der Gedanke dahinter ist, dass Artikel nicht zu stark durch “Messfehler” heruntergezogen werden sollen.
6. In einem letzten Schritt wird dann die temporäre Liste absteigend nach Scores sortiert.
7. Sind es mehr als zwei Listen, die zusammengefügt werden müssen, so wird das Vorgehen bei 1. mit der temporären und der nächsten ausstehenden Liste wiederholt.

Anzeige

Als Rückgabe der *SearchEngine* erhält der Controller nun die finale Liste der relevantesten Artikel. Diese wird an die View weitergegeben um dem Benutzer die Ergebnisse zu präsentieren. Es werden die Schlüsselwörter angezeigt, die bei der Suche verwendet werden und die Artikel, welche Dimensionswörter enthalten, werden hervorgehoben.

Konfiguration von Solr

Um bestmögliche Resultate zu erzielen, wurde die Grundkonfiguration von Solr etwas abgeändert. Dazu haben wir im *schema.xml* das Feld zu den *Analyzern* bearbeitet:

1. Das Analyzer Feld wurde in Query und Index unterteilt.
2. Filter und Tokenizer wurden abgeändert, um ein besseres Resultat zu erhalten.

Im Folgenden wird das Solr Konfigurationsfile *schema.xml* beschrieben, in welchem der Analyzer abgeändert wurde. Die Filterung wurde auf den Index und die Queries aufgeteilt. Dies haben wir erreicht, indem wir das Analyzer-Feld in zwei Felder mit jeweiliger Typenspezifizierung aufgeteilt haben:

Index

WhitespaceTokenizerFactory:

Ersetzt den StandardTokenizer und ist dafür zuständig, durch Whitespace getrennte Wörter in einzelne Tokens aufzuteilen, um diese nachher so indexieren zu können.

WordDelimiterFilterFactory:

Dieser Filter folgt direkt auf den WhitespaceTokenizer und bearbeitet die Tokens so, dass Trennzeichen wie Bindestriche und ähnliches eliminiert werden, wobei das Originalwort auch als Token erhalten bleibt.

SynonymFilterFactory:

Wird benutzt, um selbsterstellte Synonyme aus dem *synonyms.txt* File mit zu indexieren. So wird beispielsweise bei der Indexierung von “Mumps” das Synonym “Ziegenpeter” mit einbezogen. Um möglichst alle Synonyme und Teile davon zu bekommen, wird dieser Filter zwei mal mit jeweils anderen Tokenizer-Factories angewendet.

KeywordTokenizerFactory:

Benutzt ganze Keywords im Volltext als Tokens.

LetterTokenizerFactory:

Teilt Keywords nach Sonderzeichen auf und entfernt diese.

StopFilterFactory:

Dieser Filter ist verantwortlich für den Ausschluss der im *stopwords.txt* definierten Wörter von der Indexierung. Für Genaueres zu den Stoppwörtern siehe Glossar.

EdgeNGramFilterFactory:
Der Filter bezieht N-Gramme von Wörtern, die indexiert werden sollen, mit ein. In diesem Fall sind das nur N-Gramme, die von links nach rechts gebildet werden, mit einer Minimallänge von drei Buchstaben. Zum Beispiel wird das Wort “Asthma” aufgeteilt in: Ast, Asth, Asthm, Asthma.
Der Vorteil daran ist: Wenn “Asthma” bei der Suche nicht vollständig eingegeben wird, dann werden Asthma Artikel trotzdem in der Ergebnisliste erscheinen.

CommonGramsFilterFactory:
Dieser Filter entfernt nachträglich noch die N-Gramme, welche neu als Stoppwörter hinzugekommen sind.

Query

Wie bei der Indexierung werden diese Filter in folgender Reihenfolge verwendet:

WhitespaceTokenizerFactory
WordDelimiterFilterFactory
LowerCaseFilterFactory
Gross- und Kleinschreibung sollte bei der Eingabe keine Rolle spielen. Darum ändert dieser Filter jede Eingabe auf Kleinschreibung.
StopFilterFactory

Für eine genauere Ansicht der verwendeten Parameter siehe *schema.xml*.

Einbezug der Synonyme

Damit Solr gewünschte Synonyme in die Indexierung einbezieht, müssen diese in einem gewissen Format im *synonyms.txt* stehen. Um dieses Textfile zu befüllen, wurde ein Rake Task geschrieben, der aus einem vom Kunden mitgeliefertes .json File die Synonyme ausliest.

Erstellen der Datenbank

Die fertige Datenbank enthält alle Artikel aus der deutschen Wikipedia in der Kategorie “Medizin”, ausser Personen. Zum Erstellen dieser Datenbank wird wie folgt vorgegangen:

Zuerst wird mithilfe des Tools “Category tree intersection” von CatScan ein JSON file generiert, das Metadaten zu allen Artikeln, die in irgendeiner Unterkategorie (rekursiv) der Kategorie “Medizin” sind, enthält. Dies wird durch das Aufrufen folgender URL erreicht:

http://tools.wmflabs.org/catscan2/quick_intersection.php?lang=de&project=wikipedia&cats=Medizin&ns=0&depth=-1&max=100000&start=0&format=json&redirects=&callback=

Danach werden die für die Datenbank relevanten Informationen (*page_id*, *text_id* und *page_title*) herausgelesen und in einem Array als Hashes gespeichert. Dieses Array wird dann aufgeteilt und jedes Subarray wird an eine Instanz der Klasse “Downloader” gegeben, welche alle in unterschiedlichen Threads laufen. Dadurch nimmt der Prozess deutlich weniger Zeit in Anspruch.

Ein Downloader iteriert durch “sein” Array - wenn ein Artikel mit der entsprechenden *page_id* bereits in der Datenbank vorhanden ist und die *text_id* die gleiche ist (was bedeutet, dass der Artikel auf Wikipedia nicht geändert wurde), überspringt er den Artikel. Sonst wird der Inhalt des Artikels über eine Wikipedia-URL heruntergeladen und mittels Nokogiri geparsed und schliesslich in der Datenbank abgespeichert. Gegebenenfalls wird der alte Eintrag (falls einer vorhanden ist) gelöscht.

In einem zweiten Schritt werden die Schnittmengen der Kategorien “Medizin” und “Mann” sowie “Medizin” und “Frau” mit demselben Tool heruntergeladen und alle Artikel, deren *page_id* darin enthalten ist, gelöscht. Auf diese Weise kann ein sehr grosser Teil der irrelevanten Artikel über Personen entfernt werden. “Mann” und “Frau” werden verwendet, weil in der Kategorie “Person” auch Artikel wie z.B. “Chirurg” enthalten sind, welche relevant sein können.

Die Kategorisierung von Wikipedia ist aber für unsere Bedürfnisse nicht optimal.

Fazit

In diesem Projekt hatten wir die Möglichkeit, uns ausführlich mit der Problematik des NLP auseinanderzusetzen. Die Entscheidung, einen Suchserver zu verwenden hat uns sehr viel Zeit gespart, vorallem nach unserem ersten Versuch eine eigene Indexierungsmethode zu entwickeln.

Wir haben indes viele Fortschritte gemacht, es aber immer noch zahlreiche Vorschläge zur Verbesserung.

- Ausarbeitung der Datenbankstruktur, so dass andere Quellen eingebunden werden können. Dazu müsste man eine vordefinierte Art der Dokumentspeicherung verwenden.
- Unterstützung mehrerer Sprachen.
- Dimensionswörter direkt im Merger miteinbeziehen. Dazu muss eine neue Anpassung der Scores gemacht werden.
- Unterstützung für Synonymterme bestehend aus mehreren Wörtern.
- Diverse Verbesserungen am GUI, auch in Verbindung mit den anderen Verbesserungsvorschlägen.

Glossar

Stoppwort

Wort welches bei einer Volltextindexierung nicht beachtet wird, da es im Sprachgebrauch zu häufig auftritt und keine Relevanz für die Suche besitzt. Beispiele dazu sind “der”, “ein”, “bei” etc.

Schlüsselwort

Wort, welches kein Dimensionswort ist.

Dimensionswort

Wort, welches bei einer Volltextindexierung nicht beachtet wird, aber im Gegensatz zum Stoppwort noch weiterhin verwendet wird. Dimensionswörter sind meistens Wörter, welche im Suchgebiet sehr häufig vorkommen und deswegen von der Indexierung ausgeschlossen werden sollten. Beispiel für eine Dimension ist: “Therapie”, “therapieren”, “Behandlung” etc.

Hit

Einzelnes Suchergebnis zu einem Artikel, welches zusätzliche Informationen über Relevanz enthält. Siehe dazu auch die Dokumentaion zu *Sunspot::Search::Hit*.

Score

Punktzahl, welche vom Suchserver vergeben wird. Er zeigt an, wie relevant ein Artikel für die Sucheingabe ist.

Boost

Eine Erhöhung der Punktzahl, um einen Artikel, welcher eine gewisse Eigenschaft (wie Vorkommnis im Titel) besitzt.

Tokenizer, Token

Ein Tokenizer zerlegt eine Eingabe in Einheiten (Tokens), die nicht mehr weiter zerlegt werden können. Zum Beispiel können die Token eines Satzes die Wörter sein.