Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Kurs Information Retrieval

Praktikumsbericht

Zusammenfassung:
Dieser Praktikumsbericht behandelt die Erstellung einer Suchmaschine im Rahmen der Information-Retrieval-Vorlesung. Die Domäne wurde auf die Universität Leipzig eingeschränkt.
Während der Bearbeitung der Praktikumsaufgabe konnten viele, für die Entwicklung einer Suchmaschine typischen Komponenten angefertigt und verwendet werden. Dieser Bericht beleuchtet entsprechend einer für Suchmaschinen üblichen Architektur [CMS09b] das Vorgehen dafür.
Dementsprechend beschreibt Kapitel 1 den Indexierungsprozess. Dieser ermöglichte die Erstellung eines Index über 390 000 Dokumente.
Kapitel 2 beschreibt den Anfrageprozess. Dabei werden die Bestandteile zur Nutzerinteraktion, zum Ranking und für das Logging besprochen.
Abgeschlossen wird die Ausarbeitung mit einer Auswertung der Effektivität der entstandenen Suchmaschine basierend auf einem Laborexperiment (Kapitel 3).
Auf Grundlage dieser Auswertung werden potentielle Verbesserungen identifiziert, welche im Anschluss an die Arbeit umgesetzt werden können.

Leipzig, März 2018

vorgelegt von
Maik Fröbe, Danilo Morgado, Sebastian Günther

Betreuer:

Jun.-Prof. Dr. Martin Potthast
Fakultät für Mathematik und Informatik
Text Mining und Retrieval

1 Von der Datenbeschaffung zum fertigen Index

Die in einer Dokumentensammlung enthaltenen Informationen tragen fundamental zum Erfolg einer Suchmaschine bei [CMS09a]. Dementsprechend stellt eine geeignete, mit vertretbarem Aufwand durchführbare Erzeugung dieser Dokumentensammlung den Anfang der Wertschöpfungskette dar.

Die im Kontext von Web-Suchmaschinen dafür benötigten Programme werden Crawler genannt [CMS09a]. Diese beschäftigen sich damit, Webseiten zu finden und herunterzuladen, um sie für spätere Verarbeitungsschritte lokal Verfügbar zu haben. Die in der Problembeschreibung dargestellte Beschränkung auf die Website der Universität Leipzig vereinfacht dieses, im Allgemeinen sehr anspruchsvolle Problem, und wird als Site Search bezeichnet [CMS09b].

Im Rahmen dieser Arbeit wurde zur Erzeugung der Dokumentensammlung die freie Software [wik18a] Apache Nutch[1] verwendet. Nutch übernahm dabei sowohl die Aufgaben eines Crawlers[2] als auch die unmittelbar folgenden Schritte bis einschließlich der Indexerzeugung. Im Folgenden werden verschiedene, während der Arbeit angewandten Anpassungen und Vereinfachungen des von Nutch bereitgestellten Workflows vorgestellt.

1.1 Domänenspezifische, manuell gepflegte Selectivity

Unter den Features, die ein Crawler umsetzen sollte [MRS09], findet sich auch die Notwendigkeit, die Qualität der zu crawlenden Seiten in die Crawling-Reihenfolge einfließen zu lassen. Am Beispiel von Nutch ist diese Funktionalität durch eine Bewertung der einzelnen Seiten auf Basis der aktuell bekannten Linkstruktur realisiert [nut13]. Dabei werden Seiten, welchen eine höhere Wichtigkeit vorhergesagt wird, zeitiger heruntergeladen.

Aufgrund begrenzter Ressourcen wurde diese Idee weiter verschärft, so dass bestimmte Seiten erst gar nicht gecrawlt wurden. Dies betrifft Seiten, welche nicht direkt zur Universität gehören, sowie Spider-Traps[3], umfangreiche Webanwendungen[4], oder auch vollständige Suchmaschinen[5] innerhalb der Universität.

[1] Nutch ist unter Apache Version 2.0 lizensiert.
[2] Der finale Crawling-Durchgang benötigte 14 Tage um 390 000 Dokumente in 50GB zu sammeln.
[3] Zum Beispiel ein Kalender mit dynamisch erzeugten Links.
[4] Zum Beispiel die Wortschatz-, verschiedene Wörterbuch-, oder auch Semantic-Web-Anwendungen.
[5] Zum Beispiel ein Dokumentenserver für Abschlussarbeiten.

Insbesondere für die Letzteren besteht die Herausforderung darin, genug Informationen über Existenz und Einsatzzweck einer solchen Seite zu sammeln, ohne gleichzeitig unnötig vielen dynamisch erzeugten Links zu folgen. Beispiel 1.1 beschreibt dies am Fall eines Dokumentenservers.

Beispiel 1.1: Dokumentenserver FMI

Der Dokumentenserver der Fakultät für Mathematik und Informatik stellt unter anderem folgende Funktionalitäten bereit:

Den Download von Dokumenten:
Ein Nutzer kann die im Dokumentenserver enthaltenen Dokumente herunterladen. Dazu besitzt jedes Dokument eine Übersichtsseite, welche Metadaten zum Dokument bereitstellt sowie einen Link zum vollständigen Dokument. Beispiele dafür sind:

- Eine Übersichtsseite: http://lips.informatik.uni-leipzig.de/pub/2017-0
- Ein Dokument: http://lips.informatik.uni-leipzig.de/files/thesis.pdf

Beide enthalten wichtige Informationen und sollten gecrawlt werden.

Das Filtern nach Dokumenten:
Ein Nutzer kann über eine Facettensuche [wik18b] innerhalb der Dokumente filtern. Beispiele dafür sind:

- Filterung nach „Organisation IfI“: http://lips.informatik.uni-leipzig.de/browse/results/taxonomy%3A912
- Filterung nach „Organisation IfI und Author Erhard Rahm“: http://lips.informatik.uni-leipzig.de/browse/results/taxonomy%3A912%20field_authors%3A%22Rahm%2C%20Erhard%22

Die enorme Menge an möglichen Filter-Permutationen, sowie teilweise auftretenden Endlosschleifen lassen den Aufwand für ein Crawling dieser Seiten als unangemessen erscheinen. Dies wird abgerundet durch den Umstand, dass die oben beschriebenen Zielseiten ohne Filterung erreichbar sind. Dazu ist eine Traversierung von folgenden Seiten sinnvoll:

- Die erste Seite einer Liste aller vorhandenen Dokumente: http://lips.informatik.uni-leipzig.de/browse/results/
- Für welche nachfolgende Seiten direkt über Links zugänglich sind: http://lips.informatik.uni-leipzig.de/browse/results?page=4

Um derartige Problemfälle zu behandeln müssen diese im ersten Schritt identifiziert werden. Diesbezüglich bietet es sich an, während kleinerer Probe-

Crawlings eine Auswertung der Logs vorzunehmen. Um dies in vereinfachter Form durchzuführen, wurde ein Statistik-Plugin für Nutch implementiert[6]. Dieses Plugin identifiziert mittels Map-Reduce[7] Bereiche für die der Crawler besonders aktiv wird. Dazu werden alle bekannten URLs durch Entfernung von Queries oder Fragmenten normalisiert, und in ihre hierarchischen Bestandteile zerlegt. Beispiel 1.2 zeigt dies für eine Auswahl der in Beispiel 1.1 besprochenen Links, wie durch Aufsummieren dieser Bestandteile die Ausgabe des Plugins erzeugt wird.

Beispiel 1.2: Berechnungsschritte des Statistik-Plugins

Die Map-Phase nimmt die Normalisierung der Links, sowie eine Umwandlung in die hierarchischen Bestandteile vor:

http://lips.informatik.uni-leipzig.de/pub/2017-0
→
[(http://lips.informatik.uni-leipzig.de, 1),
(http://lips.informatik.uni-leipzig.de/pub, 1),
(http://lips.informatik.uni-leipzig.de/pub/2017-0, 1)]

http://lips.informatik.uni-leipzig.de/files/thesis.pdf
→
[(http://lips.informatik.uni-leipzig.de, 1),
(http://lips.informatik.uni-leipzig.de/files, 1),
(http://lips.informatik.uni-leipzig.de/files/thesis.pdf, 1)]

Die Reduce Phase summiert alle eintreffenden Paare bestehend aus Link-Bestandteil und count:

[(http://lips.informatik.uni-leipzig.de, 2),
(http://lips.informatik.uni-leipzig.de/pub, 1),
...]

[6] Das Statistik-Plugin kann in dem entsprechenden Github-Repository eingesehen werden.
[7] Einführungen in das Map-Reduce-Paradigma können hier eingesehen werden: [wik18c], [EF11], [Lam11].

In der Ausgabe des Statistik-Plugins können schließlich sinnvoll Problemfälle identifiziert werden[8]. Um dies effizient durchführen zu können, bietet es sich an, dieses Plugin als Eingabe für existierende Standardwerkzeuge zu nutzen. So erlaubt die nach Anzahl sortierte sowie durch einen Mindest-Threshold und White-List gefilterte Ausgabe auf den ersten Blick potentielle Probleme zu identifizieren.

Im nächsten Schritt ist es notwendig, diese Problemfälle zu behandeln. Eine einfache, aber effiziente Möglichkeit, dies umzusetzen, wird in Nutch in Form von Regex-URL-Filtern bereitgestellt. Darunter versteht man eine Liste von regulären Ausdrücken, welche jeweils um eine positive oder negative Markierung erweitert sind. Für eine URL werden die regulären Ausdrücke dann der Reihe nach ausgewertet. Der erste passende Ausdruck entscheidet, ob eine URL gecrawlt werden darf (falls der zugehörige Ausdruck positiv markiert ist), oder nicht[9].

Die in Nutch vordefinierten Regex-URL-Filter verfolgen das Ziel, das Crawling auf sinnvolle Protokolle[10] und Dokumenttypen[11] zu beschränken. Um diese Filter systematisch korrekt zu erweitern wurde ein testgetriebenes Vorgehen gewählt[12]. Dieses zeichnet sich durch die Möglichkeit aus, einzelne Bereiche getrennt voneinander zu konfigurieren, ohne ungewünschte Seiteneffekte zu erzeugen.

Nach dem Teile-und-herrsche-Verfahren werden diesbezüglich für kleine Bereiche der zu crawlenden Webseiten einzeln anhand von Beispielen Konfigurationen vorgenommen. Dafür werden für diesen Bereich Positiv- und Negativ-Beispiele in Form von URLs zusammengetragen. Im nächsten Schritt werden die für diese Beispiele passenden Regex-URL-Regeln aufgestellt. Diese verfolgen das Ziel, dass sie sich für alle Positivbeispiele zu positiv auswerten, und für alle Negativbeispiele entsprechend zu negativ. Beispiel 1.3 verdeutlicht dies für die aus Beispiel 1.1 bekannten URLs.

Die Konfigurationen der einzelnen Bereiche werden schließlich Bottom-up zu einer einzigen URL-Filter-Liste vereinigt. Für diese Liste werden anschließend alle Positiv- und Negativ-Beispiele evaluiert. Nur wenn dabei für alle Beispiele

[8] Neben den in Beispiel 1.1 hervorgehobenen Problemen sei hier noch auf potentiell fehlende Seiten hingewießen. Diese können durch einen Vergleich der aufsummierten Linkbestandteile mit der Anzahl der von Google indexierten Seiten entdeckt werden.
[9] Aus diesem Grund ist es üblich, die Liste der Ausdrücke mit einem entsprechendem immer zutreffendem Eintrag abzuschließen.
[10] HTTP, kein mailto- oder file-Protokoll.
[11] HTML-Seiten, keine CSS-Resourcen, Bilder oder Videos.
[12] Die Implementierung befindet sich in dem Github-Repository check-nutch-regex-urlfilter.

das spezifizierte Verhalten beobachtet wird, ist diese URL-Liste valide und kann verwendet werden.

Beispiel 1.3: Erstellung der Regex-URL-Regeln für den Dokumentenserver

```
# Positivbeispiele: Stelle sicher, dass die Dokumente gecrawlt werden
http://lips.informatik.uni-leipzig.de/browse/results?page=4
http://lips.informatik.uni-leipzig.de/files/thesis.pdf
http://lips.informatik.uni-leipzig.de/pub/2017-0

# Negativbeispiele: Verhindere, dass die Filterfunktion genutzt wird
http://lips.informatik.uni-leipzig.de/browse/results/taxonomy

# Regex-URL-Regeln, welche diese Anforderungen erfüllen
+http://lips.informatik.uni-leipzig.de/(pub|files)/.*
+http://lips.informatik.uni-leipzig.de/browse/results.*
-http://lips.informatik.uni-leipzig.de.*
```

Durch diese Hilfswerkzeuge wurde im Rahmen der Arbeit die Website der Universität Leipzig anhand ihrer Fakultäten auf die einzelnen Bearbeiter aufgeteilt. Mit einer Reihe von Probe-Crawlings konnte die vollständige Regex-URL-Filter-Liste erstellt werden. Diese Vorbereitung unterstützte das unterbrechungsfreie Crawling der für den Index verwendeten Dokumente. Dafür konnten die entstandenen Positivbeispiele direkt als Seeds eingesetzt werden.

1.2 Reproduzierbare Erzeugung eines Lucene Index mit Docker und Solr

Um die gecrawlten Dokumente in der in Abschnitt 2.2 beschriebenen Suchmaschine verfügbar zu machen, müssen sie in einen Lucene Index überführt werden.

Dafür wird im ersten Schritt eine sogenannte Conversion durchgeführt [CMS09b]. Dabei werden im vorliegenden Fall gecrawlte Dokumente, welche in den unterschiedlichsten Formaten vorliegen können[13] in ein einheitliches Format umgewandelt. Auf eine mögliche Reduzierung der Dokumente auf deren Main-Content wurde verzichtet[14]. Realisiert wurde die Conversion mit Apache Tika.

[13] Häufigstes Format war HTML, aber auch PDF, Word oder Power Point waren vertreten.

[14] Die Extraktion des Main-Content ist mit Apache Tika unmittelbar möglich. Jedoch erschien es plausibel, dass für die gecrawlten Webseiten mit vergleichsweiße wenig Rauschen zu rechnen ist.

Anschließend wird mit Nutch eine Inversion der Links durchgeführt. Dabei werden für alle Dokumente die Texte der eingehenden Links in dedizierten Feldern an dem Dokument abgespeichert. Im nächsten Schritt wird eine Near Duplicate Detection durchgeführt, welche näher in Abschnitt 1.3 erläutert wird.

Nutch kann den Lucene Index nicht selbstständig erzeugen. Aus diesem Grund wird mit einem Solr Server ein Index über die gecrawlten Dokumente erzeugt. Diesbezüglich wird mit Solr eine Text Transformation [CMS09b] durchgeführt. Dabei wurde entsprechend der Standards auf eine Entfernung von Stoppwörtern sowie Stemming verzichtet[15]. Die eigentliche Indexerzeugung mit Solr schließt diesen Prozess ab. Dabei werden neben den Dokument-Statistiken [CMS09b] für das spätere Scoring von Dokumenten auch Vorbereitungen zur Erzeugung von Snippets vorgenommen. Der so erzeugte Index kann unmittelbar in Lucene eingesetzt werden.

Die hier beschriebene Erzeugung des Index aus den gecrawlten Dokumenten ist von vielen Parametern abhängig. Dadurch wird eine häufige Wiederholung dieses Prozesses sinnvoll, um die Auswirkung von Anpassungen an verschiedenen Parametern zu prüfen. Um den Aufwand dafür gering zu halten, wurde der komplette Workflow reproduzierbar automatisiert[16].

Dafür wird als Eingabe eine Menge von Nutch-Crawl-Directories erwartet. Diese werden im ersten Schritt bereinigt, um ein erneutes Parsing der rohen Dokumente durchzuführen. Danach werden die einzelnen, oben beschriebenen Arbeitsschritte durchgeführt. Sobald die Indexerzeugung beginnt, wird der dafür notwendige Solr-Server mit Docker bereitgestellt. Da die notwendige Konfiguration sowie das Index-Schema in den Container gemountet werden, ist auch dieser Schritt einfach wiederholbar.

1.3 Data Cleaning: Aggregation von Fast-Duplikaten

Duplikate und Fast-Duplikate treten in vielen Situationen auf [CMS09a]. Im Rahmen dieser Arbeit war die entsprechende Zielsetzung, diese transparent an den Nutzer zu kommunizieren. Abbildung 1 stellt dies an einem Beispiel dar.

Um dies zu ermöglichen, müssen zuerst die (Fast-)Duplikate identifiziert werden. Dafür wurde das TextProfileSignature-Verfahren verwendet. Dieser Algorithmus weist einem Dokument einen Fingerprint zu. Dabei werden ähnliche

[15] Jedoch gehört das testen unterschiedlicher Stoppwortlisten sowie verschiedener Stemmer zu dem geplanten Vorgehen, die Effektivität der Suchmaschine für das durchgeführte Laborexperiment zu optimieren (Siehe Kapitel 3).

[16] Das entsprechende Github-Repository mit den Scripten und der Konfiguration ist Nutch-Tools.

Abbildung 1: Ein Dokument mit 2 Fast-Duplikaten

Dokumenten auf den gleichen Wert abgebildet. Algorithmus 1 zeigt dies im Pseudocode. In Beispiel 1.4 wird der Fingerprint für einen Text berechnet.

Algorithmus 1 : Text Profile

Input : Document $\mathcal{D}$, $min \in \mathbb{N}^+$, $\mathcal{Q} \in \mathbb{R}$
Output : Fingerprint for Document $\mathcal{D}$

1 $\mathcal{D} \leftarrow$ removeNonAlphanumericCharacters($\mathcal{D}$)
2 $\mathcal{D} \leftarrow$ toLowerCase($\mathcal{D}$)
3 $\mathcal{T} \leftarrow$ tokenizeWithFrequencyCount($\mathcal{D}$)
4 $\mathcal{T} \leftarrow$ removeTokensShorterOrEqualThan(min, $\mathcal{T}$)
5 $\mathcal{T} \leftarrow$ roundFreqeunciesDownToNearestMultiple($\mathcal{Q}$, $\mathcal{T}$)
6 $\mathcal{T} \leftarrow$ removeTokensWithTooSmallFrequency($\mathcal{Q}$, $\mathcal{T}$)
7 return md5Hash(sort($\mathcal{T}$))

Nach der Zuweisung des Fingerprints werden entsprechend dem Discovery Scenario alle Paare von Fast-Duplikaten gesucht [CMS09a]. Durch dieses Vorgehen entstehen Gruppen ähnlicher Dokumente. Für jede solche Gruppe wird ein Repräsentant gewählt[17]. Dieser wird um die URLs der anderen Gruppenmitglieder angereichert. Abschließend werden die restlichen Dokumente einer Gruppe gelöscht[18].

[17] Das längste Dokument wird der Vertreter.
[18] Die Implementierung ist in dem Modul Custom Index Cleaning zu finden.

Beispiel 1.4: Berechnung des Fingerprints mittels TextProfileSignature

Eingabe:
$\mathcal{D}$ = „Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.“
$min = 2$
$\mathcal{Q} = 1$

Normalisierung des Dokuments:
Für $\mathcal{D}$ werden die Satzzeichen „.“und „,“entfernt. Gleichzeitig wird „Tropical“am Satzanfang zu „tropical“.

Tokenisierung mit Token-Frequenz und Filterung nach Wortlänge:
$\mathcal{T}$ = { (include, 1), (including, 1), (salt, 1), (environments, 1), (around, 1), (water, 1), (both, 1), (tropical, 2), (the, 1), (found, 1), (world, 1), (species, 1), (and, 1), (fish, 2), (freshwater, 1) }

Passe $\mathcal{Q}$ an die maximale Tokenfrequenz 2 an:
$\mathcal{Q} = 2 \times \mathcal{Q}$

Runde Frequenzen auf Vielfache von $\mathcal{Q}$ ab:
Alle Frequenzen von 1 werden auf 0 abgerundet, 2 bleibt bestehen.

Entferne Einträge, deren Frequenz unter $\mathcal{Q}$ liegt:
$\mathcal{T}$ = { (tropical, 2), (fish, 2) }

Berechne Signatur:
sort($\mathcal{T}$)
textProfileSignature = md5Hash($\mathcal{T}$)

2 Verarbeitung von Anfragen

Die Verarbeitung von Anfragen[19] wird im Allgemeinen durch drei Komponenten realisiert [CMS09b]. Auch die im Rahmen dieser Arbeit erstellte Suchmaschine bildet davon keine Ausnahme. Dementsprechend werden in den folgenden Abschnitten die Bestandteile User Interaction, Ranking und Evaluation vorgestellt.

2.1 User Interaction [CMS09b]

Die Komponente zur Nutzerinteraktion bietet eine Schnittstelle[20] zwischen dem Benutzer und der Suchmaschine. Um diesem Nutzer eine gewohnte Usability inklusive intuitiver Bedienung zu ermöglichen, wurden die in der Praxis verbreiteten Standards eingehalten.

Dementsprechend muss einem Nutzer das Absenden von Anfragen ermöglicht werden. Dafür ist es üblich, eine Landing Page mit prominent mittig platzierter Eingabemaske auszuliefern [BYRN10a]. Abbildung 2 zeigt dies für die im Rahmen dieser Arbeit entwickelte Suchmaschine.

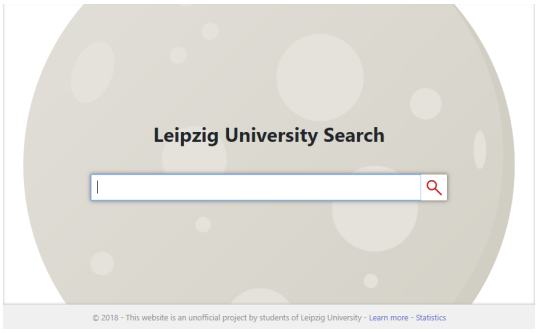

Abbildung 2: Die Landing Page der Suchmaschine

Nachdem ein Nutzer seine Anfrage spezifiziert hat, werden ihm die Ergebnisse präsentiert. Dafür erhält die Schnittstelle eine für die Query gerankte Liste von Dokumenten von der Ranking-Komponente. Diesbezüglich ist es gängig, einem Benutzer für kleinere Ausschnitte aus der Dokumentliste jeweils ausgewählte Metadaten in Verbindung mit einem für die Anfrage besonders relevanten Textausschnitt[21] zu präsentieren [BYRN10a]. Relevante Wörter werden dabei speziell hervorgehoben. Die unter Berücksichtigung dieser Punkte entstandene Search Engine Result Page[22] wird in Abbildung 3 vorgestellt.

In der Regel wird einem Nutzer die Formulierung und Spezialisierung seiner Anfragen durch verschiedene Hilfsmittel erleichtert. In dem vorliegenden Projekt wurde eine Query Suggestion implementiert, welche Vervollständigungs-

[19] Auch Query Processing
[20] Der Quellcode ist in dem Modul search-engine-backend enthalten.
[21] Sogenannte Snippets
[22] SERP

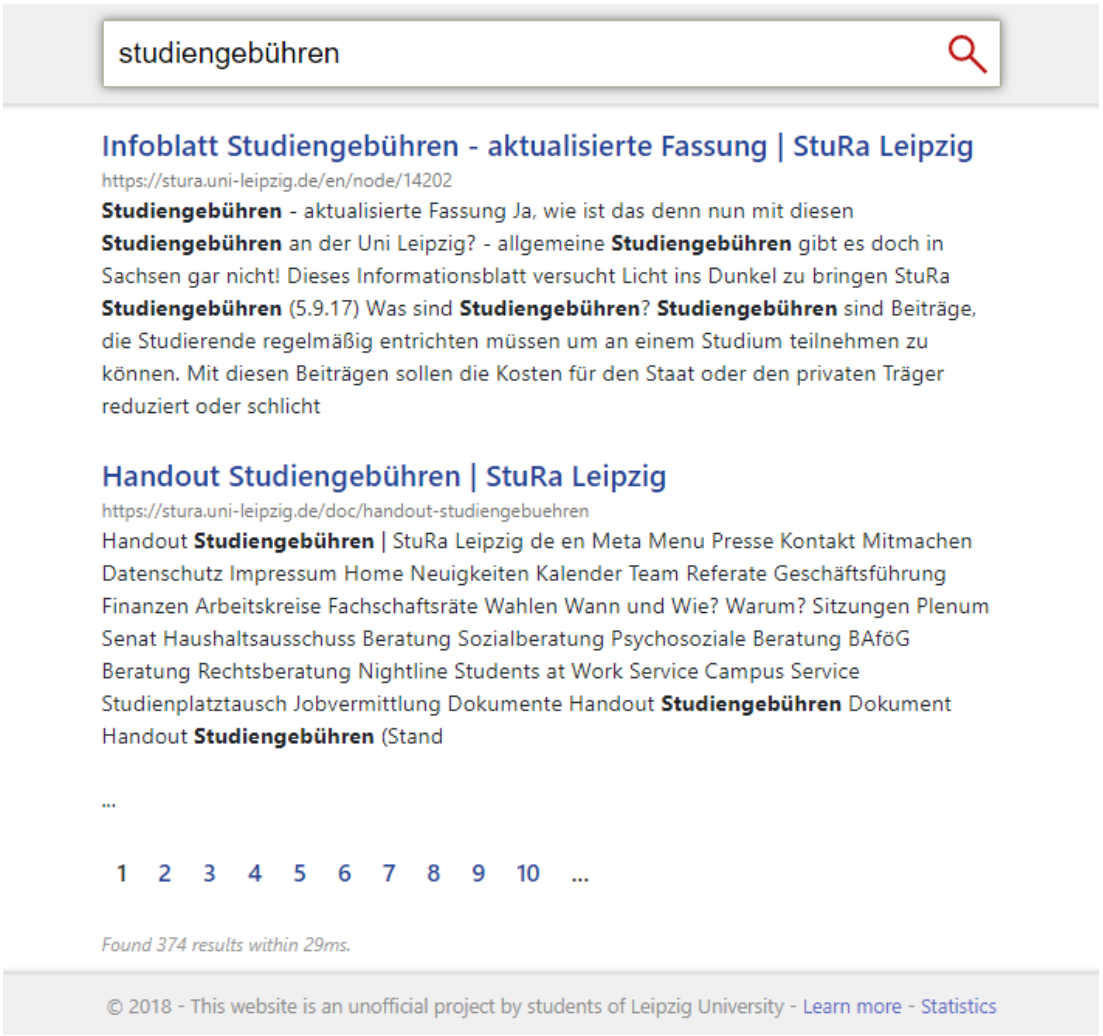

Abbildung 3: Die SERP für die Anfrage „studiengebühren“

vorschläge auf Basis populärer Anfragen liefert. Die dafür notwendige, dynamische Benutzerschnittstelle wird in Abbildung 4 gezeigt. Verwandte Maßnahmen zur Verbesserungen der Benutzbarkeit wie Spell Checking oder Query Refinement wurden zugunsten anderer Features nicht umgesetzt.

Um den Suchmaschinenservice einem möglichst breiten Nutzerkreis zuzuführen, sind die Komponenten zur Benutzerinteraktion in Form eines Webfrontends realisiert. Für die Entwicklung werden die jeweils aktuellen Standards HTML5, CSS3 sowie JavaScript eingesetzt. Ausgeliefert werden diese Bestandteile durch einen Express-Webserver, welcher im Rahmen einer NodeJS-Anwendung betrieben wird. Aus Kompatibilitätsgründen werden Teile der Bootstrap-Library eingebunden. Falls ein Browser die History-API unterstützt,

Abbildung 4: Query Suggestions für die Eingabe: „studien“

wird diese zur Bereitstellung von Deep-Links ohne einen Full-Page-Refresh geeigneter Seiten[23] verwendet.

Der Zugriff auf die Ranking-Komponente (siehe Abschnitt 2.2) und Query-Suggestions wird durch entsprechende REST-Endpunkte ermöglicht. Bereitgestellt werden diese durch eine Spring-Boot-Anwendung. Die Query-Suggestions unterscheiden dabei zwischen nutzerbezogenen und globalen Vorschlägen. Durch nutzerbezogene Vorschläge ist es einem Nutzer möglich, von ihm bereits getätigte Anfragen zu wiederholen. Globale Vorschläge identifizieren innerhalb der Suchmaschine populäre Anfragen mit Hilfe von Logdaten. Für eine sinnvolle initiale Menge von globalen Vorschlägen wurden semantisch passende Vorschläge von Google gecrawlt und eingepflegt[24].

2.2 Ranking [CMS09b]

Die Ranking-Komponente ist der Kern jeder Suchmaschine. Sie erzeugt für eine Query aus der User-Interaction-Komponente eine gerankte Liste von Dokumenten.

[23] Zum Beispiel die About-Page.
[24] Der Quellcode für den Crawler ist in dem Modul initial_suggestion_crawling. Damit wurden 2639 Vorschläge generiert

Dabei beeinflussen deren Effizienz[25] und Effektivität[26] die Nützlichkeit der Suchmaschine wesentlich. Um für beide Anforderungen einen sinnvollen Ausgangspunkt zu schaffen, wurde für das Ranking Lucene verwendet. Diese Software stellt einen geeigneten Einstieg dar, da darauf andere, in dieser Domäne verbreitete Softwarekomponenten aufbauen[27]. Verschiedene Ranking-Algorithmen [28] sind in Lucene verfügbar.

Davon wurde BM25F eingesetzt, da es als Baseline für modernere Ranking-Algorithmen fungiert und für allgemeine Dokumentsammlungen bessere Ergebnisse[29] erzielt als die verfügbaren, klassischen Vektor-Modelle [BYRN10b]. Dieses Retrieval-Modell besitzt die Fähigkeit, mehrere Felder[30] in die Berechnung des Scores einfließen zu lassen. Entsprechend wurden alle verfügbaren Felder einbezogen[31]. Da eine sinnvolle Wahl der Attribute[32] und deren Gewichtung im Allgemeinen in Verbindung mit der User-Relevanz steht, wurden alle Parameter bei ihren Standards belassen.

Das entsprechende Vorgehen für ein Tuning basierend auf Nutzer-Feedback wird in Abschnitt 3 vorgestellt.

2.3 Evaluation [CMS09b]

Die Aufgabe der Evaluationskomponente ist es, Effizienz und Effektivität zu monitoren. Dafür muss eine Aufzeichnung des Nutzerverhaltens sowie ausgewählter Systemmetriken vorgenommen werden.

Insbesondere zur Aufzeichnung des Nutzerverhaltens ist eine Identifikation der Nutzer notwendig. Darauf aufbauend kann protokolliert werden, welcher Nutzer welche Anfragen ausgeführt hat, welche Ergebnisseiten oder Query-Suggestions einem Nutzer präsentiert wurden, sowie gegebenenfalls welche Ergebnisse oder Query Suggestions ausgewählt wurden[33].

[25] Die Verarbeitung vieler Anfragen in kurzer Zeit.
[26] Die Qualität des Rankings: Kann die Suchmaschine relevante Informationen finden?
[27] Insbesondere die verbreiteten Tools [dbe] Elasticsearch und Solr basieren auf Lucene. Deren erweiterter Funktionsumfang, wie beispielsweise eine horizontale Skalierung, wurde im Rahmen dieser Arbeit nicht benötigt.
[28] Und damit auch Retrieval-Modelle
[29] Entsprechendes Tuning der von BM25F verwendeten Parameter vorrausgesetzt [BYRN10b].
[30] Felder werden auch als Attribute oder Features bezeichnet.
[31] Alle Felder mit unmittelbarem Bezug zu dem Inhalt der Dokumente. Dies sind der vollständige Text eines Dokuments, sowie deren Titel, URL und Anchor-Texte eingehender Links.
[32] Einschließlich eventuell abgeleiteter Features.
[33] Das Erweitern dieser Informationen um die entsprechenden Event-Zeitpunkte erlaubt eine sinnvolle Rekonstruktion des Nutzerverhaltens.

Bei einer technischen Realisierung dieser Punkte fällt auf, dass es sich um sogenannte Cross-Cutting Concerns handelt. Ein bewährtes Mittel derartige Funktionalitäten zu realisieren, ohne die Komplexität der betroffenen Systemteile unnötig zu erhöhen, stellt die Aspektorientierte Programmierung[34] dar [Wal14a]. Die darin verwendeten Aspekte definieren, was[35], wann[36] und wo[37] auszuführen ist [Wal14b].

In dem für die Interaction-Komponente verwendeten Framework werden alle Endpunkte über speziell annotierte Methoden bereitgestellt. Diesen ist gemein, dass sie ein POJO[38] als Modell für die Antwort zurückgeben. Dieses Modell wird je nach Endpunkt später in ein HTML-Template[39] gerendert oder JSON-serialisiert.

Unter diesen Voraussetzungen eignen sich diese Methoden hervorragend, um sie durch Aspekte um die gewünschten Funktionalitäten zu erweitern[40]. Dazu wird ein Effizienz-, ein User-Identification-, sowie ein Logging-Aspekt eingesetzt. Der Effizienz-Aspekt misst dabei beispielhaft für weitere Systemmetriken die Bearbeitungszeit von Anfragen, und erweitert das zurückgegebene Modell entsprechend. Der User-Identification-Aspekt stellt vor dem Aufruf jeder Endpunkt-Methode sicher, dass der Request mit einer eindeutigen Identifikation des Nutzers versehen ist. Dies wird über einen Cookie realisiert. Der Logging-Aspekt protokolliert ausnahmslos alle verfügbaren Informationen. Dazu notiert er für jeden Endpunkt den Request des Clients zusammen mit dem zurückzugebenden POJO-Model[41].

Die vom Logging-Aspekt protokollierten Informationen können sowohl Online[42] als auch Offline analysiert werden. Um beides zu ermöglichen, wurde mit Apache Kafka eine Streaming Platform mit der Möglichkeit zur persistenten Da-

[34] AOP

[35] In der zugehörigen AOP Terminologie definiert der zu einem Aspekt gehörende Advice die Aufgabe, also was von dem Aspekt zu erledigen ist [Wal14b].

[36] Der Advice eines Aspekts definiert neben dem was auch zu welchen Zeitpunkten ein Aspekt auszuführen ist. Unter anderem kann vor, nach oder das Wrappen einer Zielmethode spezifiziert werden [Wal14b].

[37] Durch sogenannte Pointcuts werden eine oder mehrere Methoden definiert, an die der Advice gewoben wird [Wal14b].

[38] Diese sind entsprechend einer gewählten Konvention in dem Backend-Modul in zugehörigen dto Paketen definiert.

[39] Spezifiziert innerhalb des templates-Ordner in dem Backend-Modul.

[40] Schließlich lassen sie sich eindeutig und generisch über die notwendigen Endpunkt-Annotationen identifizieren.

[41] Mit diesem Vorgehen lassen sich alle angesprochenen Informationen zur Rekonstruktion des Nutzer-Verhaltens erheben, da insbesondere die Links zu den Dokumenten auf dem Server durch Weiterleitungen aufgelöst werden.

[42] Beispielhaft wurde das im Rahmen dieser Arbeit für die Query-Suggestions vorgenommen.

tenhaltung [NSP17] eingesetzt. Dazu publisht der Logging-Aspekt seine Daten als Events nach Kafka. Um diesen Logging-Aspekt so einfach wie möglich zu halten, bereitet ein Stream Processor diese Daten weiter auf[43]. Online-Analysen wie die Query-Suggestion können sich nun unmittelbar an die für sie interessanten Events subscriben. Offline Analysen können die persistierten Events gleichzeitig in einem Batch-Vorgang verarbeiten. Als Beispiel dafür stellt Abbildung 5 einen mit Python realisierten Import[44] der Events nach Neo4j zur Visualisierung dar.

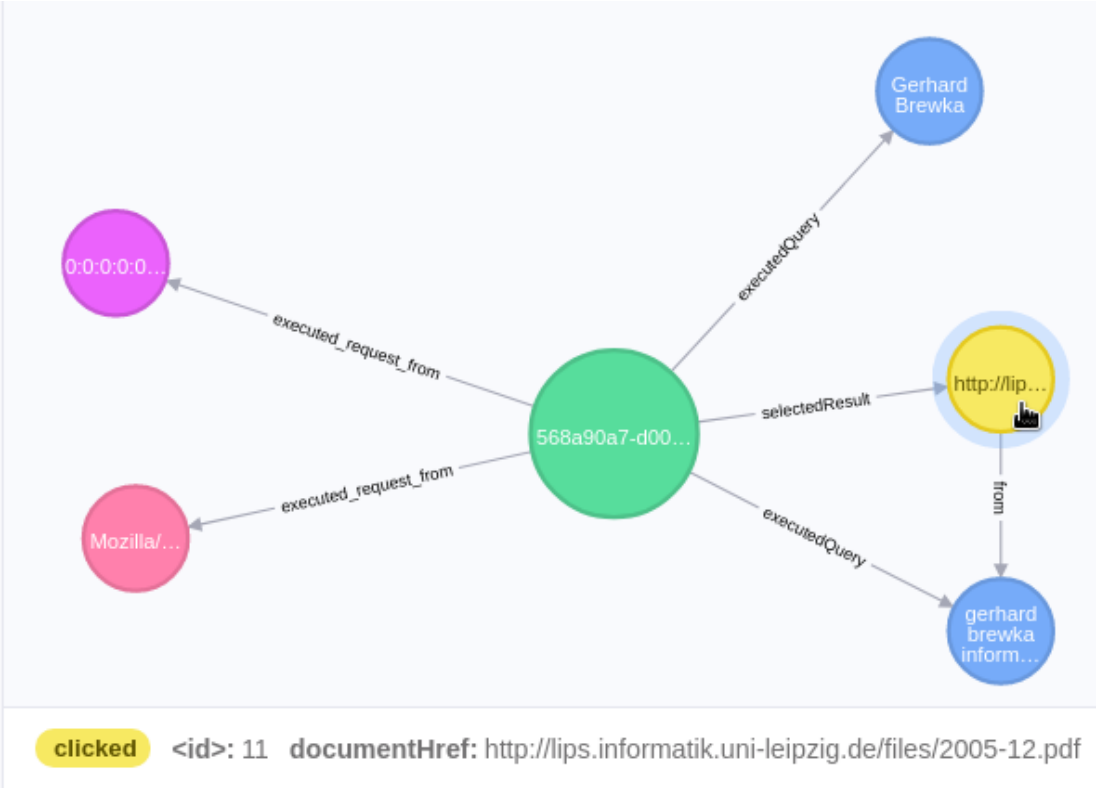

Abbildung 5: Beispielgraph im Neo4j-Browser einer User-Session bestehend aus 2 Anfragen, bei denen bei einer Anfrage ein Ergebniss angeklickt wurde.

Die hier vorgestellte Evaluationskomponente funktioniert für alle Clients, welche Cookies erlauben und den Referrer-Header korrekt setzen.

[43] Dieser Stream Processor reichert die Events an und leitet sie auf sinnvolle Topics um. Der Quellcode ist in dem Modul search-engine-kafka-streams enthalten.
[44] Der Quellcode ist in dem Modul visualize_events.

3 Analyse der Logdaten

Wie in Abschnitt 2.3 beschrieben, werden Nutzerinteraktionen durch das Loggingsystem erfasst. Das so gesammelte Feedback kann auf zweierlei Weise analysiert werden. Mittels Online-Analyse kann der Index um neue Daten erweitert oder Einfluss auf das Ranking ausgeübt werden. So kann die Menge der Query Suggestions mit Hilfe der durch Nutzer gestellten Anfragen erweitert werden. Außerdem kann die Häufigkeit von Suchbegriffen auch mit Bezug zum zeitlichen Kontext das Ranking beeinflussen, um populäre Vorschläge weiter vorn zu listen. Da die Links der SERP zudem Tracking erlauben, kann ermittelt werden, welche Ergebnisse tatsächlich von den Nutzern angeklickt werden, und welche Ergebnisse eher irrelevant sind. Für das Tuning der Suchmaschine ist im Gegensatz dazu eine Offline-Auswertung vorzuziehen. Zu diesem Zweck können die Sessions ausgewählter Testnutzer ausgiebig protokolliert werden, um Metriken wie Precision und Recall zu ermitteln. Die in diesem Kontext durchgeführte Evaluation wird im folgenden Abschnitt beschrieben.

3.1 Durchführung eines Laborexperiments

Um Schwächen der Suchmaschine zu identifizieren sowie die Effektivität von Parametertuning bewerten zu können, muss die Ergebnisqualität mit Hilfe verschiedener Metriken erfasst werden. Dabei ist es vergleichsweise einfach, die Precision zu bestimmen, indem die Relevanz einzelner Treffer in den Ergebnismengen beurteilt wird. Da allerdings die Menge der tatsächlich relevanten Dokumente zu einer Suchanfrage nicht bekannt sind, kann der Recall-Wert nur approximativ ermittelt werden.

Aus diesem Grund wird ein initiales Experiment durchgeführt, bei dem Testnutzer die ersten 30 Suchergebnisse zu vorgegebenen Topics nach deren Relevanz bewerten. Zu Beginn werden 50 entsprechende Suchanfragen sowie deren Information Need und die Relevanzkriterien vorbereitet. Im Anschluss werden die Suchen durchgeführt und für jedes Ergebnis auf den Rängen 1 bis 30 erfasst, ob ein Ergebnis irrelevant oder relevant ist bzw. ob dieses exakt das gewünschte Dokument darstellt. Auf diese Weise wird zu jedem Topic ein Korpus relevanter Dokumente erstellt, welcher zudem manuell um noch nicht enthaltene Ergebnisse erweitert werden kann. In nachfolgenden Experimenten kann diese Suche sowie die Bewertung dann automatisiert auf Basis der so erhobenen Daten durchgeführt werden. Damit ist eine schnelle Bewertung von Parameteränderungen möglich.

Sportwissenschaftliche Fakultät	Ich möchte mich darüber informieren was die Sportwissenschaftliche Fakultät macht	Relevant sind seiten die Erklären woran die Sportwissenschaftliche Fakultät forscht oder wie sie sich definiert oder was das studium umfasst	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	1
Medizinische Fakultät Stellenangebote	Ich suche Stellenangebote an der Medizinischen Fakultät.	Relevant sind Seiten, die Stellenangebote der Fakultät sowohl wissenschaftlicher Art (Ärzte, Wissenschaftler), als auch pflegerisch (Krankenpfleger, Zivildienst, ...) bieten. Auch Seiten die sagen dass momentan leider keine Stellen ausgeschrieben sind sind ok.	0	0	0	0	1	0	1	1	1	1	1	0	1	1	1	1	0	0	0	1	1	0	0	0	1	0	0	0	0	0	13	0,486687074

Abbildung 6: Überblick über einen Teil der Topics des Laborexperiments

Im Rahmen der Durchführung der Bewertungen haben sich einige Auffälligkeiten gezeigt, die im Folgenden kurz erläutert werden:

In den Ergebnissen tauchen häufig Seiten bestimmter Fakultäten beziehungsweise Institute auf, weil Wortwahl und Struktur der Webpräsenz das Ranking der Seite besonders positiv beeinflussen. Im Gegensatz dazu stehen andere Websites, welche über verschiedene URLs erreichbar sind oder für jedes Semester dieselben Informationen in abgewandelter Form veröffentlichen und damit die Bewertung ihrer einzelnen Seiten herabsetzen.

Weiterhin ist anhand der Ergebnisse ersichtlich, dass die Suchmaschine der Reihenfolge der Suchbegriffe innerhalb der Query stark überbewertet. So werden Dokumente, welche das erste Wort mehrfach enthalten noch vor jenen gelistet, welche alle Suchbegriffe enthalten.

Darüber hinaus wird klar, dass häufig aufgrund anderer Schreibweisen, Abkürzungen oder einfach nur Synonymen das gewünschte Ergebnis nicht gefunden werden kann. Da es sich häufig um fachspezifische Terminologien handelt, ist die Standardkonfiguration der Suchmaschine nicht ausreichend.

Ein Fall, bei dem der gesamte Inhalt eines PDF-Dokuments als Bild eingebettet ist, zeigt außerdem die Limitierungen der zur Indexierung verwendeten Software. Da diese kein OCR anwendet, ist der gesamte enthaltene Text nicht im Index zu finden.

Die Auswertung der Metriken zeigt, dass in den MRR-Szenarien bei rund 63% der Anfragen das gesuchte Ergebnis innerhalb der Top 10 zu finden ist. Bei einer weiteren Anfrage wird Rang 14 erreicht. Bei allen weiteren Anfragen (etwa ein Drittel) kann das gesuchte Dokument nicht innerhalb der ersten 30 Suchergebnisse gefunden werden.

In den 24 nicht-MRR-Szenarien, bei denen nicht nur ein bestimmtes Dokument gesucht wird, zeigen sich ähnlich unterschiedliche Ergebnisse. Neben einzelnen Anfragen, die die meisten Spitzenplatzierungen mit relevanten Dokumenten be-

setzen, existieren auch drei Szenarien, die in ihrer Ergebnismenge kein einziges als wichtig erachtetes Dokument enthalten. Die meisten Resultate liegen dagegen jedoch im Mittelfeld, sodass sich insgesamt eine Mean Average Precision von 0,535 ergibt.

Literatur

[BYRN10a] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The concepts and technology behind search*, chapter Chapter 2.3: Search Interfaces Today, pages 25–40. Pearson, 2010.

[BYRN10b] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The concepts and technology behind search*, chapter Chapter 3.5: Alternative Probabilistic Models, page 107. Pearson, 2010.

[CMS09a] W. Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines Information Retrieval in Practice*, chapter Chapter 3: Crawls and Feeds, pages 31–73. Pearson, 2009.

[CMS09b] W. Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines Information Retrieval in Practice*, chapter Chapter 2: Architecture of a Search Engine, pages 13–30. Pearson, 2009.

[dbe] Db-engines ranking of search engines. [Online erhältlich unter https://db-engines.com/en/ranking/search+engine. Abgerufen am 29.03.2018].

[EF11] Stefan Edlich and Achim Friedland. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*, chapter Chapter 2.1: Map/Reduce, pages 12–30. Carl Hanser Verlag GmbH & Co. KG, 2011.

[Lam11] Chuck Lam. *Hadoop In Action*, chapter Chapter 1.5: Understanding MapReduce, pages 8–14. Wiley India, 2011.

[MRS09] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*, chapter Chapter 20: Web crawling and indexes, pages 443–459. Cambridge University Press, 2009.

[NSP17] Neha Narkhede, Gwen Shapira, and Todd Palino. *Kafka: The Definitive Guide*, chapter Foreword, pages xi–xiii. O'Reilly UK Ltd., 2017.

[nut13] Apache nutch documentation: invertlinks, 2013. [Online erhältlich unter https://wiki.apache.org/nutch/bin/nutch%20invertlinks. Abgerufen am 25.03.2018].

[Wal14a] Graig Walls. *Spring In Action*, chapter Chapter 1.1: Simplifying Java development, pages 4–18. Manning, 2014.

[Wal14b] Graig Walls. *Spring In Action*, chapter Chapter 4.1: What is aspect-oriented programming, pages 98–103. Manning, 2014.

[wik18a] Free software license — Wikipedia, the free encyclopedia, 2018. [Online erhältlich unter https://en.wikipedia.org/wiki/Free_software_license. Abgerufen am 22.03.2018].

[wik18b] Free software license — Wikipedia, the free encyclopedia, 2018. [Online erhältlich unter https://de.wikipedia.org/wiki/Facettensuche. Abgerufen am 25.03.2018].

[wik18c] Mapreduce — Wikipedia, the free encyclopedia, 2018. [Online erhältlich unter https://de.wikipedia.org/wiki/MapReduce. Abgerufen am 22.03.2018].