Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Kurs Information Retrieval

Praktikumsbericht

Zusammenfassung:
Dieser Bericht ...

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 Einführung

Blaaa

2 Von der Datenbeschaffung zum fertigen Index

Die in einer Dokumentensammlung enthaltenen Informationen bestimmen die Sinnhaftigkeit einer Suchmaschine basierend auf diesen Dokumenten in besonderem Maße [CMS09b]. 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 [CMS09b]. 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 [CMS09a].

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.

2.1 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] TBA, Kalender mit []
[4] Wortschatz, Wörterbücher, Semantic-Web-Anwendungen
[5] 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 2.1 beschreibt dies am Fall eines Dokumentenservers.

Beispiel 2.1: 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 Bestandteile wie Schema, Host und Pfad zerlegt. Beispiel 2.2 zeigt für die in Beispiel 2.1 besprochenen Links, wie durch aufsummieren dieser Bestandteile die Ausgabe des Plugins erzeugt wird.

Beispiel 2.2: Berechnungsschritte des Statistik-Plugins

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

Die Reduce Phase summiert alle eintreffenden Paare bestehend aus Link-Bestandteil und count: **TODO MAIK BEISPIEL**

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 requlären 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 Dokument-Typen[11] zu beschränken. Um diese Filter systematisch korrekt zu erweitern wurde ein Testgetriebenes Vorgehen

[6] Das Statistik-Plugin kann in dem entsprechenden Github-Repository eingesehen werden.
[7] Einführungen in das Map-Reduce-Paradigma können hier eingesehen werden [], [], []
[8] Neben den in Beispiel 2.1 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...
[11] Html-Seiten, keine CSS-Resourcen, Bilder oder Videos

gewählt[12]. Dieses zeichnet sich durch die Möglichkeit aus, einzelne Bereiche getrennt voneinander zu konfigurieren, ohne ungewünschte Seiteneffekte zu erwarten.

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 2.3 verdeutlicht dies für die aus Beispiel 2.1 bekannten Urls.

Beispiel 2.3: 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.*
```

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

Durch diese Hilfswerkzeuge wurde im Rahmen der Arbeit die Site 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. Mithilfe dieser Vorbereitung konnten die für den Index verwendeten Dokumente unterbrechungsfrei gecrawlt werden. Dafür konnten die entstandenen Positiv-Beispiele direkt als Seeds eingesetzt werden.

[12] Die Implementierung findet sich in dem Github-Repository check-nutch-regex-urlfilter

2.2 Reproduzierbare Erzeugung eines Lucene Index mit Docker und Solr

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

Dafür wird im ersten Schritt eine sogenannte Conversion durchgeführt [CMS09a]. 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.

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 2.3 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 [CMS09a] durchgeführt. Diese umfasst die Entfernung von deutschen und englischen Stoppwörtern, sowie ein Stemming mit dem Porter Stemmer. Die eigentliche Indexerzeugung mit Solr schließt diesen Prozess ab. Dabei werden neben den Dokument-Statistiken [CMS09a] 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[15].

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 Ar-

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

beitsschritte 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.

2.3 Data Cleaning? Aggregation von Fast-Duplikaten

Duplikate und Fast-Duplikate treten in vielen Situationen auf [CMS09b]. Im Rahmen dieser Arbeit war die entsprechende Zielsetzung, diese transparent an den Nutzer zu kommunizieren. Abbildung 1 stellt dies an einem Beispiel dar.

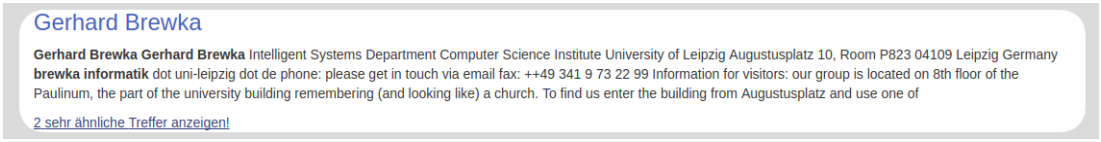

Abbildung 1: Ein Dokument mit 2 Fast-Duplikaten

Um dies zu ermöglichen, müssen zuerst die (Fast-)Duplikate identifiziert werden. Dafür wurde das TextProfile-Signature -Verfahren verwendet. Dieser Algorithmus weißt einem Dokument einen Fingerprint zu. Dabei werden ähnliche Dokumenten auf den gleichen Wert abgebildet. Algorithmus 1 zeigt dies im Pseudocode. In Beispiel 2.4 wird der Fingerprint für einen Text berechnet.

Algorithmus 1 : Text Profile

Input : Document $\mathcal{D}$
Output : Fingerprint for Document $\mathcal{D}$

```
1 TBA
2 TBA
3 TBA
```

Beispiel 2.4: Berechnung des Fingerprints mittels TextProfileSignature

TODO MAIK

Nach der Zuweißung des Fingerprints werden entsprechend dem Discovery Scenario alle Paare von Fast-Duplikaten gesucht [CMS09b]. Durch dieses Vorgehen entstehen Gruppen ähnlicher Dokumente. Für jede solche Gruppe wird ein Repräsentant gewählt[16]. Dieser wird um die Urls der anderen Gruppenmitglieder angereichert. Abschließend werden die restlichen Dokumente einer Gruppe gelöscht[17].

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

3 Verarbeitung von Anfragen

Die Verarbeitung von Anfragen[18] wird im Allgemeinen durch drei Komponenten realisiert [CMS09a]. 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.

3.1 User Interaction [CMS09a]

Die Komponente zur Nutzerinteraktion bietet eine Schnittstelle[19] zwischen dem Benutzer und der Suchmaschine.

Die erste Funktionalität, mit der ein Nutzer dabei in Berührung kommt stellt das Absenden der Anfragen dar. Entsprechend Baeza-Yates wird dem Nutzer dafür eine typische Eingabemaske bereitgestellt. Diese ist in Abbildung **??** im Kopf der Seite dargestellt. Index-terme

Wie die während der Interaktion eines Nutzers mit der Suchmaschine anfallenden Daten evaluiert werden ist Bestandteil von abschnitt **??**. Realisiert wurde mit Hilfe von

Technisch wurde ... und ... mit ... realisiert. Dabei...

3.2 Ranking [CMS09a]

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

Dabei beeinflussen deren Effizienz[20] und Effektivität[21] 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 Do-

[18] Auch Query Processing
[19] Das entsprechende Modul ist TODO MAik LINK
[20] Die Verarbeitung vieler Anfragen in kurzer Zeit.
[21] Die Qualität des Rankings: Kann die Suchmaschine relevante Informationen finden?

mäne verbreitete Softwarekomponenten aufbauen[22]. Verschiedene Ranking-Algorithmen [23] sind in Lucene verfügbar.

Davon wurde BM25F eingesetzt, da es als Baseline für modernere Ranking-Algorithmen fungiert und für allgemeine Dokumentsammlungen bessere Ergebnisse[24] erzielt als die verfügbaren, klassischen Vektor-Modelle [BYRN10]. Dieses Retrieval-Modell besitzt die Fähigkeit, mehrere Felder[25] in die Berechnung des Scores einfließen zu lassen. Entsprechend wurden alle verfügbaren Felder einbezogen[26]. Da eine sinnvolle Wahl der Attribute[27] 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 **??** vorgestellt.

3.3 Evaluation [CMS09a]

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 worden, sowie gegebenenfalls welche Ergebnisse oder Query Suggestions ausgewählt wurden [28].

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 ehöhen, stellt die Aspektorientierte Programmierung[29] dar []. Die

[22] Insbesondere die verbreiteten Tools [] Elasticsearch und Solr basieren auf Lucene. Deren erweiterter Funktionsumfang, wie Beispielsweiße eine horizontale Skalierung, wurde im Rahmen dieser Arbeit nicht benötigt.
[23] Und damit auch Retrieval-Modelle
[24] Entsprechendes Tuning der von BM25F verwendeten Parameter vorrausgesetzt [BYRN10].
[25] Felder werden auch als Attribute oder Features bezeichnet.
[26] 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.
[27] Einschließlich eventuell abgeleiteter Features.
[28] Das erweitern dieser Informationen um die entsprechenden Event-Zeitpunkte erlaubt eine sinnvolle Rekonstruktion des Nutzerverhaltens.
[29] AOP

darin verwendeten Aspekte definieren, was[30], wann[31] und wo[32] auszuführen ist [].

In dem für die Interaction-Komponente verwendeten Framework werden alle Endpunkte über speziell anotierte Methoden bereitgestellt. Diesen ist gemein, dass sie ein POJO[33] als Model für die Antwort zurückgeben. Dieses Modell wird je nach Endpunkt später in ein HTML-Template[34] gerendert oder JSON-serialisiert.

Unter diesen Vorraussetzungen eignen sich diese Methoden hervorragend, um sie durch Aspekte um die gewünschten Funktionalitäten zu erweitern[35]. 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 Model 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[36].

Die vom Logging-Aspekt protokollierten Informationen können sowohl Online[37], als auch Offline analysiert werden. Um beides zu ermöglichen, wurde mit Apache Kafka ein ... mit persistenter Datenhaltung 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[38]. 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 verarbei-

[30] In der zugehörigen AOP Terminologie definiert der zu einem Aspekt gehörende Advice die Aufgabe, also was von dem Aspekt zu erledigen ist. []
[31] 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 [].
[32] Durch sogenannte Pointcuts werden eine oder mehrere Methoden definiert, an die der Advice gewoben wird [].
[33] Jeweils definiert in...
[34] Spezifiziert in ....
[35] Schließlich lassen sie sich eindeutig und generisch über die notwendigen Endpunkt-Anotationen identifizieren
[36] 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.
[37] Beispielhaft wurde das im Rahmen dieser Arbeit für die Query-Suggestions vorgenommen
[38] auf die richtige bahn werfen, Siehe modul...

ten. Als Beispiel dafür stellt Abbildung **??** einen mit Python realisierten Import der Events nach Neo4j zur Visualisierung dar.

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

4 Analyse der Logdaten

Bla

4.1 Durchführung eines Laborexperiments

Bla

Literatur

[BYRN10] 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 2: Architecture of a Search Engine, pages 13–30. Pearson, 2009.

[CMS09b] 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.

[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.

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

[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].