

Abschnittsbasierte Textklassifikation in der Wikipedia

Magisterarbeit

im Studiengang
MAGISTER ARTIUM

der

Friedrich-Alexander-Universität
Erlangen-Nürnberg
in der Philosophischen Fakultät und Fachbereich Theologie

Betreuer:

Prof. Dr. Roland Hausser, Ph.D.
Prof. Dr. Dirk Riehle, M.B.A

vorgelegt von

Guido Leisker

aus

Berlin

Erlangen, im März 2011

Inhaltsübersicht

Zusammenfassung	ii
1 Einleitung	1
2 Grundlagen	7
3 Relevante Arbeiten	31
4 Wikitech Classify	36
5 Evaluation	59
6 Ausblick und abschließende Zusammenfassung	74
Literaturverzeichnis	80
A Bedienungsanleitung	86
B Nutzung der Wikitech-Classify-API (Beispiele)	91
C Lebenslauf	99
D Wahrheitsgemäße Erklärung	100

Zusammenfassung

Die Wikipedia hat das Ziel, eine global verfügbare, freie Informationsquelle in Form eines Online-Lexikons zu erstellen. Freiwillige aus der ganzen Welt erstellen und kategorisieren kollaborativ neue Artikel, prüfen, aktualisieren und verbessern bestehende Artikel. Diese Änderungen erfordern auch das Überarbeiten anderer Inhalte, um die Konsistenz der Wikipedia zu erhalten. Aufgrund des Umfangs der Wikipedia ist es für Autoren eine besondere Herausforderung, diese Textstellen ausfindig zu machen. Das in dieser Arbeit entwickelte System kann die Wikipedia automatisch nach ähnlichen Artikeln und Abschnitten durchsuchen und darauf aufbauend Kategorievorschläge zusammenstellen. Auf einer Volltextsuche basierend, skaliert es für den gesamten textuellen Inhalt der Wikipedia und liefert innerhalb kürzester Zeit Ergebnisse. Damit stellt es eine Verbesserung gegenüber anderen Ansätzen dar, die entweder auf die Linkstruktur der Wikipedia beschränkt sind oder nur für einen Teilbereich der Wikipedia getestet wurden. Die Möglichkeit, dabei auf Abschnittsebene zu arbeiten, ist neu. Mithilfe der Wikipedia-Kategorien wird das System qualitativ evaluiert.

Inhaltsverzeichnis

Zusammenfassung	ii
1 Einleitung	1
2 Grundlagen	7
2.1 Information Retrieval und Textklassifikation	7
2.1.1 Textverarbeitung	7
2.1.2 Boolesches Information Retrieval	9
2.1.3 Relevanzbasiertes Information Retrieval	11
2.1.4 Textklassifikation	13
2.1.5 Evaluation	15
2.2 Wikis	18
2.3 MediaWiki	18
2.4 Wikipedia	19
2.4.1 Projekt	20
2.4.2 Artikel	20
2.4.3 Andere Seiten	21
2.4.4 Kategorien	22
2.4.5 Dumps	23
2.5 MediaWiki-Parser	24
2.5.1 AST	25
2.5.2 Visitors	27
2.6 Lucene	27
2.6.1 Index	27
2.6.2 Suche	30
3 Relevante Arbeiten	31
3.1 Qualität der Wikipedia	32

3.2	Semantische Nähe und Textklassifikation	34
4	Wikitech Classify	36
4.1	Finden ähnlicher Artikel	37
4.1.1	Aufbau des Indexes	37
4.1.2	Suchanfrage	43
4.1.3	Zusammenfassung	48
4.2	Finden ähnlicher Abschnitte	49
4.2.1	Artikelbasierter Index	49
4.2.2	Abschnittsbasierter Index	51
4.2.3	Zusammenfassung	52
4.3	Kategorisierung	54
4.3.1	Kategorisierung auf Basis ähnlicher Abschnitte . .	57
4.4	Einfache Volltextsuche	58
5	Evaluation	59
5.1	Test-Indizes und Test-Datensätze	61
5.1.1	Indizes	61
5.1.2	Datensätze	62
5.2	Schlüsselwort-Algorithmus	63
5.3	Gewichtung nach Ähnlichkeit	64
5.4	Abschnitte	65
5.5	Kategorie-Ebenen	66
5.5.1	Finden allgemeinerer Kategorien	66
5.5.2	Kategorisierung mithilfe der Oberkategorien . . .	66
5.6	Vergleich zwischen verschiedenen Artikel-Qualitäten . . .	67
5.7	Vergleich zur Baseline	68
5.8	Vergleich zu Gantner/Schmidt-Thieme (2009)	69
5.9	Repräsentativität	71
5.10	Zusammenfassung	72
6	Ausblick und abschließende Zusammenfassung	74
6.1	Anwendungsmöglichkeiten	74
6.2	Verbesserungen	75
6.3	Einschränkungen und Erweiterungen	77
6.4	Zusammenfassung	78
	Literaturverzeichnis	80
A	Bedienungsanleitung	86
A.1	Inhalt der beigelegten CD	86

A.2	Installation	86
A.3	Bedienung	87
B	Nutzung der Wikitech-Classify-API (Beispiele)	91
B.1	Index	91
B.2	Similar	93
B.3	Categories	95
C	Lebenslauf	99
D	Wahrheitsgemäße Erklärung	100

Abbildungsverzeichnis

1.1	Schema des Systems (Aufbau des Indexes und Ähnlichkeitssuche)	4
2.1	Wikitext des Wikipedia-Artikels Cologne (Bildschirmaufnahme)	19
2.2	Wikipedia-Artikel Cologne (Bildschirmaufnahme)	21
2.3	Kategorie-Graph des Artikels Cologne (Ausschnitt)	23
2.4	Beispiel für einen AST (1/3)	25
2.5	Beispiel für einen AST (2/3)	26
2.6	Beispiel für einen AST (3/3)	26
4.1	Wikitext-Beispiel (Artikel Wallace Neff)	40
4.2	Einfacher Text (Artikel Wallace Neff)	41
4.3	Vektor von Token (Artikel Wallace Neff)	41
4.4	Vektor von gestemmtten Wörtern (Artikel Wallace Neff)	41
4.5	Vereinfachter AST (Referenzierung der Abschnitte)	45
4.6	TFIDF-Schlüsselwörter des Java-Artikels gewichtet nach Frequenz	46
4.7	Die beiden möglichen Vorgehensweisen zum Finden ähnlicher Abschnitte	50
4.8	Aufspalten eines Artikels an Section-Knoten	50
4.9	Erstellen einer Liste von Kategorievorschlägen nach der Ähnlichkeitssuche	54

Tabellenverzeichnis

2.1	Beispielindex mit E-Mails	10
3.1	Anzahl neuer Arbeiten im jeweiligen Jahr (mit Wikipedia im Titel, lt. Google-Scholar am 31.01.2010).	31
4.1	Datenquellen für Wikipedia-Artikel	37
4.2	Aufbau eines Indextokumentes	39
4.3	Klassen für die Schlüsselwort-Extrahierung	46
4.4	Ähnliche Artikel zu Java (programming language)	47
4.5	Ähnliche Abschnitte zum Artikel Java (programming lan- guage)	53
4.6	Ähnliche Abschnitte zum ersten Abschnitt aus Java (pro- gramming language)	53
4.7	Kategorievorschlge fur den Artikel Java (programming language)	55
4.8	Vorschlge fur allgemeinere Kategorien fur den Artikel Java (programming language)	56
4.9	Gewichtungsfaktoren fur die verschiedenen Gewichtungs- strategien (Beispiel Ebene 1-4)	57
5.1	Verteilung der Wikipedia-Seiten auf Namensrume	60
5.2	Anzahl der Dokumente im Index	62
5.3	Evaluation der Schlüsselwort-Algorithmen	63
5.4	Evaluation der Schlüsselwortgewichtung	64
5.5	Evaluation der Gewichtung nach Artikelahnlichkeit	64
5.6	Artikelklassifikation im Vergleich mit Abschnittsklassifi- kation	65
5.7	Kategorisierung auf verschiedenen Kategorie-Ebenen	66
5.8	R-Precision fur verschiedene Kategorie-Ebenen	67
5.9	Ergebnisse fur die verschiedenen Testsatze	67
5.10	Wikitech-Classify-Werte im Vergleich zu einer Baseline	68
5.11	Ergebnisse von Gantner/Schmidt-Thieme (2009)	69

5.12	Ergebnisse für Tests mit jeweils 1000 zufällig ausgewählten Artikeln	71
A.1	Inhalt der beigelegten CD	86

2.1	Einfacher Wikitext (Beispiel)	25
4.1	Konfiguration der SQL-Datenquelle	38
4.2	Konstruktor der Klasse NodeSourceSQL	38
4.3	Methode index der Klasse NodeIndexer	42
4.4	Konstruktor der Klasse NodeIndexer	42
4.5	Methode createDocument der Klasse NodeDocument . . .	43
4.6	CLI-Aufruf für das Indizieren	43
4.7	Beispiele für Referenzierungen von Abschnitten durch XPointer	44
4.8	Beispielsuchanfrage für die Ähnlichkeitssuche	47
4.9	Methode search der Klasse NodeSearcher	47
4.10	Methode find der Klasse FindSimilar	48
4.11	Methode find (2) der Klasse FindSimilar	48
4.12	Konstruktor der Klasse FindSimilar	48
4.13	CLI-Aufruf für Ähnlichkeitssuche	48
4.14	Konstruktor der Klasse NodeSplitterByElement	49
4.15	Methode split der Klasse NodeSplitterByElement	51
4.16	Konstruktor der Klasse NodeSplitterByLevel	51
4.17	Zweiter Konstruktor der Klasse FindSimilar	51
4.18	Methode index der Klasse NodeIndexer	52
4.19	Konstruktor der Klasse FindCategories	56
4.20	Methode find der Klasse FindCategories	56
4.21	CLI-Aufruf für Kategorievorschläge	57
4.22	Volltextsuche auf der Kommandozeile	58
5.1	Filtern der Namensräume	61
A.1	Aufruf des Wikitech-Classify-Archivs	87
A.2	Skript für den einfachen Aufruf des Wikitech-Classify- Archivs	87

A.3	Setzen der Umgebungsvariable für den Index	87
A.4	CLI: Indizieren zweier Artikel aus der englischsprachigen Wikipedia	88
A.5	CLI: Dokumente erneuern oder aus dem Index entfernen .	88
A.6	CLI: Finden ähnlicher Artikel und Abschnitte	88
A.7	CLI: PDF-Datei als Ausgangsdokument für die Ähnlich- keitssuche	88
A.8	CLI: Aufbau eines Indexes mit Abschnitten	89
A.9	CLI: Finden ähnlicher Abschnitte in einem Index mit Abschnitten	89
A.10	CLI: Kategorisierung	89
A.11	CLI: Schlüsselwörter und Volltextsuche	90
A.12	CLI: Zufällige Artikel ausgeben lassen	90
A.13	CLI: Evaluation	90
B.1	Index.java	91
B.2	Similar.java	93
B.3	Categories.java	95

Das Wikipedia-Projekt hat das Ziel, ein Online-Lexikon zu erstellen. Die Inhalte sollen weltweit und frei verfügbar für alle Menschen in ihrer Muttersprache angeboten werden.

Die Informationen werden dabei von freiwilligen Autoren aus der ganzen Welt zusammengetragen. Sie legen neue Artikel für fehlende Themen an, tragen zusätzliche oder fehlende Informationen in bestehenden Artikeln nach und korrigieren Fehler (Riehle, 2006).

Innerhalb von weniger als zehn Jahren hat die Wikipedia einen Umfang erreicht, der alle vorhandenen Enzyklopädien übertrifft. Die englischsprachige ist mit über 3,5 Mio. Artikeln die größte Wikipedia und wächst noch immer¹, wenn auch langsamer (Suh et al., 2009).

Thematisch ähnliche Artikel werden in Kategorien zusammengefasst. Die Kategorien sollen den Lesern, aber auch Autoren, eine einfache Möglichkeit bieten, sich innerhalb eines bestimmten Themengebietes einen Überblick zu verschaffen. Da Kategorien wiederum in Oberkategorien zusammengefasst werden können, entsteht eine Hierarchie von Kategorien. Weder die Kategorien noch deren Hierarchie ist festgelegt, sie entwickeln sich – wie die Wikipedia selbst – ständig weiter.

Durch das Hinzufügen zusätzlicher Informationen und das Ändern einzelner Abschnitte kann es zu Inkonsistenzen innerhalb der Wikipedia kommen. Die neuen Informationen können zum Beispiel schon in einem anderen Artikel vorhanden sein, wodurch Redundanzen entstehen. Änderungen an einzelnen Abschnitten können die Überarbeitung von anderen Abschnitten, gegebenenfalls in anderen Artikeln, erforderlich machen.

Es ist eine wichtige Aufgabe der Autoren, die Konsistenz der Wikipedia zu wahren. Um dieses Problem zu lösen, müssen die Autoren die anderen Abschnitte kennen, ebenfalls anpassen und gegebenenfalls löschen. Aufgrund des Umfangs der Wikipedia ist es für die Autoren schwer, immer einen kompletten Überblick über einen gesamten Themenbereich zu haben. Gelegenheitsautoren, die unter Umständen nur Details anpassen und kleine Fehler korrigieren, sind sich dieses Problems oft nicht bewusst.

1. <http://stats.wikimedia.org/EN/ChartsWikipediaEN.htm>

Diese Arbeit beschäftigt sich genau mit dieser Problematik. Ziel ist es, ein System zu entwickeln, das Autoren bei der genannten Aufgabe unterstützen kann. Es soll Autoren zu einem beliebigen Abschnitt eine Liste mit ähnlichen Abschnitten liefern, um eine anschließende manuelle Prüfung hinsichtlich der Konsistenz zu ermöglichen. Die Autoren sollen sofort während ihrer Arbeit davon profitieren, deshalb müssen die Antwortzeiten so gering wie möglich sein.

Ein Maß für Ähnlichkeit innerhalb der Wikipedia kann sich dabei grundsätzlich auf Folgendes stützen:

- Kategorien
- Linkstruktur
- textueller Inhalt.

Die Kategorien fassen allerdings, wie oben angemerkt, mehrere Artikel zusammen und können nicht den erforderlichen Detailgrad erreichen. Sie sind jedoch durchaus ein guter Einstiegspunkt, um manuell nach ähnlichen Abschnitten zu suchen. Für neue Artikel oder zusätzliche Abschnitte sind die Kategorien jedoch nur bedingt nutzbar. Im Fall ersterer müssen überhaupt erst Kategorien gefunden werden und die Kategorisierung von erweiterten Artikeln muss gegebenenfalls zunächst überarbeitet werden. Auch an dieser Stelle soll das System durch automatisch erstellte Kategorie-Vorschläge behilflich sein.

Die Linkstruktur der Wikipedia wurde bereits erfolgreich als Grundlage für das Finden semantisch ähnlicher Artikel herangezogen (Abschnitt 3.2). Dieses System wurde bisher nur für ganze Artikel angewandt, könnte sich aber durchaus für Abschnitte eignen. Die Links verweisen zwar meist auf komplette Artikel, statt auf Teile, der Ausgangspunkt lässt sich allerdings exakt bestimmen.

Um zu neuen Textstellen ähnliche zu finden, sind sie aber nur bedingt geeignet, denn eine ordentliche Verlinkung setzt bereits die Kenntnis der relevanten Artikel und Abschnitte voraus.

Für das Klassifizieren von Dokumenten auf Basis ihres textuellen Inhalts gibt es verschiedene Ansätze. Am weitesten verbreitet sind Naive Bayes und die Support Vector Maschine (SVM). Beide Verfahren können jedoch nur zwischen zwei Klassen Entscheidungen treffen. Für jede Kategorie muss somit ein separater Klassifizierer trainiert werden. Dabei werden die Trainingsdokumente der jeweiligen Klasse als positive Trainingsmenge und alle verbleibenden Dokumente als negative Trainingsmenge herangezogen. Jedes zu klassifizierende Dokument muss gegen alle Klassifizierer getestet werden. Im Falle einer Änderung im Categoriesystem oder der Artikelinhalte müssen die Klassifizierer neu trainiert werden. Die Verfahren wurden an vergleichsweise kleinen Textmengen mit wenigen Kategorien wie dem Reuters-Korpus² getestet.

2. Die häufig zur Evaluation von Textkategorisierungssystemen eingesetzten Reuters-Korpora mit ihren 21578 Dokumenten und 118 Kategorien (Reuters-21578 von 2000) bzw. 800000 Dokumenten und 103 Kategorien (Reuters-RCV1 von 2004) sind wesentlich kleiner als die Wikipedia und haben vor allem wesentlich weniger Kategorien (Lewis et al., 2004).

Bei einem Versuch, SVM für die Klassifizierung von Wikipedia-Artikeln einzusetzen, wurden die Tests lediglich auf einem kleinen Ausschnitt der Wikipedia durchgeführt, was die Komplexität wesentlich verringert. Die Menge der Kategorien wurde dadurch zum Beispiel von über 500 000 auf ca. 100 reduziert (Gantner und Schmidt-Thieme, 2009).

Bei der Suche nach einer geeigneten Vorgehensweise für diese Arbeit wurden diese Systeme zunächst in Betracht gezogen. Um das Problem der großen Anzahl von Texten und Kategorien zu umgehen, sollte der Suchraum zunächst durch eine Volltextsuche eingeschränkt werden, wodurch die Trainingsmenge so klein wie möglich gehalten wird. Für die Volltextsuche wurde ein externer Dienst³ verwendet, der entsprechende Wikipedia-Artikel zurück lieferte.

Anschließend wurden die gefundenen Artikel durch einen Bayesschen Klassifizierer⁴ nach Ähnlichkeit sortiert. Dabei wurde für jeden Artikel aus den Ergebnissen der Volltextsuche ein Klassifizierer trainiert, der den Ausgangsartikel als positive Trainingsmenge und alle verbleibenden Artikel aus den Ergebnissen als negative Trainingsmenge festlegt. Auf diese Weise war es möglich, für jeden Artikel einen Ähnlichkeitswert zum Ausgangsartikel zu berechnen.

Die Methode hat den entscheidenden Nachteil, dass die Klassifizierer jedes Mal zur Laufzeit trainiert werden müssen, was schon bei wenigen Ergebnissen aus der Volltextsuche viel Zeit beansprucht. Des Weiteren sollte das System keinesfalls von externen Diensten abhängig sein. Die Erweiterung dieses Systems, um einzelne ähnliche Abschnitte zu finden, erhöhte die Komplexität nochmals. Diese ersten Versuche zeigten jedoch, dass das Finden ähnlicher Abschnitte grundsätzlich möglich ist.

Die in dieser Arbeit entwickelte Lösung für die zuvor genannten Probleme ist es, die Volltextsuche selbst als zentralen Punkt der Ähnlichkeitssuche festzulegen und in das System zu integrieren. Die Wahl fiel dabei auf Apache Lucene⁵, eine weit verbreitete, funktionsreiche Open-Source-Volltextsuchmaschinen-Bibliothek. Auf deren Basis wurde ein System entwickelt, das zu beliebigen Ausgangstexten ähnliche Artikel und Abschnitte aus der gesamten Wikipedia, in kürzester Zeit und nach Ähnlichkeit sortiert, zurück liefert.

Dazu muss zunächst ein sogenannter Index aufgebaut werden, der den gesamten Inhalt der Wikipedia enthält. Danach wird dieser Index nach Artikeln oder Abschnitten durchsucht, die dem Ausgangsdokument ähnlich sind. Dieses Ausgangsdokument kann dabei selbst ein Artikel oder Abschnitt der Wikipedia sein.

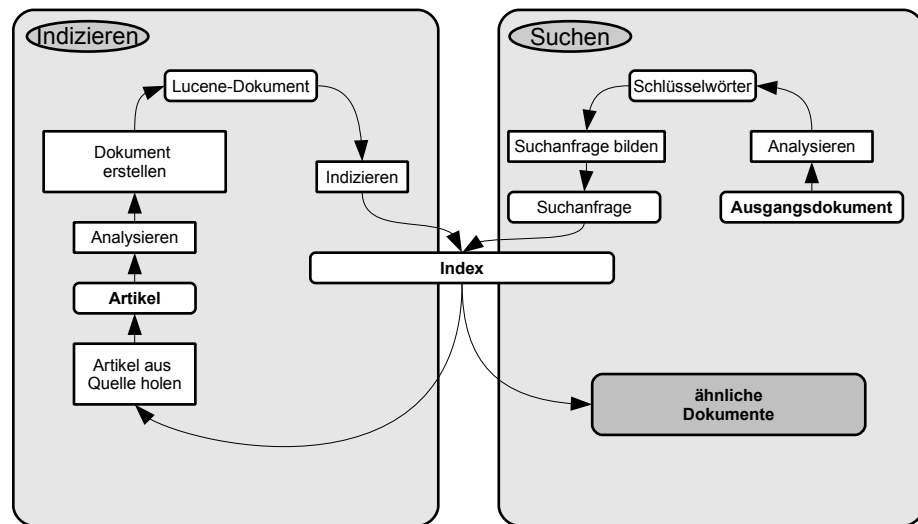
Die Abbildung 1.1 stellt die Vorgehensweise beim Erstellen eines Indexes schematisch dar. Jeder Wikipedia-Artikel wird einer Textanalyse unterzogen und der originale Artikelinhalt so aufbereitet, dass er für das Indizieren geeignet ist. Die Daten werden anschließend zu einem Lucene-Dokument zusammengefasst, welches letztendlich in den Index geschrieben wird.

3. <http://www.geonames.org/export/wikipedia-webservice.html>

4. <https://ci-bayes.dev.java.net/>

5. <http://lucene.apache.org/>

Abbildung 1.1: Schema des Systems (Aufbau des Indexes und Ähnlichkeitssuche)



Für die Ähnlichkeitssuche wird der Text des Ausgangsdokumentes zunächst auf die gleiche Weise analysiert wie beim Indizieren. Daraus werden im Anschluss Schlüsselwörter extrahiert, auf deren Basis eine Suchanfrage an den Index gestellt wird. Die Antwort für diese Anfrage ist eine Liste von relevanten Dokumenten, sortiert nach Ähnlichkeit.

Die Ähnlichkeitssuche wird mithilfe der Wikipedia-Kategorien evaluiert. Die Kategorien haben, wie bereits angesprochen, die Aufgabe, ähnliche Artikel zusammenzufassen. Für die Evaluation werden ähnliche Artikel zu einem bereits durch die Autoren kategorisierten Artikel gesucht. Aus den Kategorien der ähnlichen Artikel wird eine gewichtete Liste von Kategorievorschlägen für den Ausgangsartikel erstellt. Je besser die vorgeschlagenen Kategorien mit den tatsächlichen Kategorien übereinstimmen, desto besser funktioniert die Ähnlichkeitssuche.

Die Kategorievorschläge können neben der Evaluation der Ähnlichkeitssuche auch selbst als Hilfe für die Autoren dienen. Schließlich müssen wie oben angesprochen, neue Artikel kategorisiert werden und die Kategorien geänderter Artikel gegebenenfalls überarbeitet werden.

Verglichen mit bisherigen Arbeiten bringt das System folgende Verbesserungen:

- Das System arbeitet auf Basis des textuellen Inhalts der Wikipedia.
- Das System liefert die Antworten innerhalb sehr kurzer Zeit.
- Das System skaliert für den gesamten Inhalt der Wikipedia.
- Änderungen am Inhalt der Wikipedia können vergleichsweise einfach übernommen werden.
- Die Möglichkeit, dabei auf Abschnittsebene zu arbeiten, ist neu.

Zu Beginn dieser Arbeit (Kapitel 2) sollen die Grundlagen, die für das weitere Verständnis notwendig sind, erarbeitet werden. Dabei werden zunächst die Methoden des Information Retrieval als Grundlage der Volltextsuche und der Textklassifikation beschrieben sowie auf grundlegende Herangehensweisen, wie die Vorverarbeitung des Textes, eingegangen. Des Weiteren werden Möglichkeiten vorgestellt, mit denen die Leistungsfähigkeit dieser Systeme evaluiert werden kann.

Kapitel 3 gibt einen Überblick über eine Auswahl von relevanten wissenschaftlichen Arbeiten aus dem Umfeld der Wikipedia.

Die zentralen Kapitel dieser Arbeit sind die Kapitel 4 und 5. In Kapitel 4 wird die Vorgehensweise zum Finden ähnlicher Artikel und Abschnitte sowie der Kategorisierung detailliert beschrieben. Darüber hinaus wird dieses Kapitel der Anforderung gerecht, die wesentlichen Schnittstellen zum System vorzustellen. In Kapitel 5 wird eine Evaluation des Systems vorgenommen. Dafür wird zunächst die Vorgehensweise, wie die Indizes für die Tests aufgebaut wurden, erklärt. In den ersten Versuchen werden die optimalen Einstellungen für verschiedene Parameter ermittelt. Schließlich wird das System hinsichtlich seiner Leistungsfähigkeit bewertet und dafür mit einer Baseline und den Ergebnissen einer anderen Arbeit verglichen.

Danksagung

Ich danke meinen Betreuern Herrn Prof. Dr. Roland Hausser, Ph.D. und Prof. Dr. Dirk Riehle, M.B.A. sowie ihren Mitarbeitern Dipl.-Inf. Hannes Dohrn, Dipl.-Inf. Johannes Handl, Besim Kabashi, M.A., Carsten Kolassa, M.Sc., Thomas Proisl, M.A. und Carsten Weber, M.A. Die Unterstützung reichte von den ersten Gesprächen beim Finden des Themas über die Kritik, Anregungen und Hilfe während der Entwicklung des Systems bis hin zu wertvollen Tipps für die schriftliche Arbeit.

Ich bedanke mich weiterhin bei den Korrekturlesern – meinen Eltern, meiner Schwester und Hannes Dohrn – die sich in mühevoller Arbeit und ausdauernd auf die Suche nach Tipp- und Schreibfehlern gemacht haben.

Ein ganz besonderer Dank geht an meine kleine Tochter und an meine Freundin. Während der Entstehungszeit der Magisterarbeit habe ich durch ihre Rücksicht und ihr Verständnis sehr viel Unterstützung erfahren.

Konventionen

In dieser Arbeit wurden die folgenden Konventionen genutzt:

- Fachbegriffe werden bei ihrer ersten Nennung durch kursive Schreibweise hervorgehoben. Bestehen sie aus mehreren Wörtern, können sie Sätze schwer verständlich machen. In diesen Fällen werden die Fachbegriffe immer kursiv geschrieben. Das gleiche gilt für Begriffe, die sich auf Beispiele beziehen.
- Teile des Quellcodes, wie Klassenmamen, die im Fließtext auftreten, werden serifenlos geschrieben. Längere Listings werden in nichtproportionaler Schrift wiedergegeben und besonders gekennzeichnet.
- Literaturangaben werden innerhalb des Textes nur durch die Nennung des ersten Autors und dem Jahr des Erscheinens genannt. Am Ende der Arbeit befindet sich eine Literaturliste mit den vollständigen Angaben.
- Englische Fachbegriffe wurden nicht ins Deutsche übersetzt, wenn es entweder keine Übersetzung gibt oder die englischen Begriffe als gebräuchlich angesehen werden.
- Es wurde versucht, so weit wie möglich auf Abkürzungen zu verzichten. Nur wenige Begriffe werden gewöhnlich fast ausschließlich als Abkürzung gebraucht und auch in dieser Arbeit nicht ausgeschrieben. Auf ein Glossar wurde deswegen verzichtet. Stattdessen werden die Abkürzungen an den entsprechenden Stellen aufgelöst.

In diesem Kapitel werden die Grundlagen dargestellt, die für das weitere Verständnis dieser Arbeit notwendig sind.

Im Abschnitt 2.1 werden Information Retrieval für Textdaten und Textklassifikation im Allgemeinen vorgestellt. Der Abschnitt 2.2 gibt einen kurzen Überblick, was Wikis sind und wofür sie eingesetzt werden. Abschnitt 2.3 stellt die Wikisoftware der Wikipedia und Abschnitt 2.4 schließlich das Wikipedia-Projekt selbst vor. In Abschnitt 2.5 und 2.6 werden die beiden wichtigen Bibliotheken vorgestellt, auf denen diese Arbeit basiert: ein Parser für die Markup-Sprache der Wikipedia und die Volltextsuchmaschinen-Bibliothek Lucene.

2.1 Information Retrieval und Textklassifikation

Information Retrieval und Textklassifikation sind wichtige Teildisziplinen der Computerlinguistik. Information Retrieval beschäftigt sich mit dem (Wieder-)Auffinden von Informationen in großen Informationssammlungen. Einen Überblick geben die Abschnitte 2.1.2 und 2.1.3. In Abschnitt 2.1.4 wird auf die Textklassifikation eingegangen. Ihre Aufgabe ist es, Textdokumente in zuvor definierte Gruppen oder Klassen einzuordnen (Manning et al., 2008, Kapitel 1).

Textdaten müssen verschiedenen Vorverarbeitungsschritten unterzogen werden, bevor Information-Retrieval- oder Textklassifikations-Systeme sie verarbeiten können. Der nächste Abschnitt stellt die Techniken vor, die auch in dieser Arbeit angewandt wurden.

2.1.1 Textverarbeitung

Moderne Information-Retrieval-Systeme speichern die Daten in einer speziellen Struktur, um einen effizienten Zugriff zu ermöglichen. Dafür ist es notwendig, die Textdaten aus den ursprünglichen Dokumenten verschiedenen Vorverarbeitungsschritten zu unterziehen. In den folgenden Abschnitten wird eine Vorgehensweise vorgestellt, die in vielen Systemen angewandt wird.

Texteinheiten (Dokumente) bestimmen

Moderne Information-Retrieval-Systeme für Textdaten arbeiten oft mit einem sogenannten Index. Das ist eine spezielle Datenbank, die für eine Volltextsuche optimiert ist. In den Index werden die Ausgangsdokumente als Einheiten aufgenommen. Diese Einheiten werden entsprechend einer Anfrage an das System zurückgeliefert. Die Entscheidung was genau als Einheit angesehen werden soll, ist nicht immer einfach. Soll das System für eine Sammlung von Textdateien entwickelt werden, können diese direkt als Einheiten angesehen werden. Auch E-Mails können als Textdateien gelesen werden. Die Frage, wie die Anhänge der E-Mails behandelt werden sollen, gestaltet sich schon schwieriger. Sind die Anhänge selbständige Dokumente und sollen auch einzeln gefunden werden oder wäre es besser, sie zusammen mit dem eigentlichen Text der E-Mail zu einem Dokument im Index zusammenzufassen? Eine allgemeingültige Vorgehensweise kann nicht vorgeschlagen werden (Manning et al., 2008, Kapitel 1). Im Abschnitt 4.1.1 wird gezeigt, wie die Dokumente für das System dieser Arbeit aufgebaut sind.

Einfachen Text herausfiltern

Heute liegen die Daten nur noch selten als reine Textdateien vor. Das System muss deswegen eine Möglichkeit anbieten, den Text aus den Ausgangsdokumenten herauszufiltern. Je nach Art des Dokumentes kann sich diese Aufgabe unterschiedlich schwer gestalten. Die Qualität dieses Filters kann für die Leistungsfähigkeit des Systems entscheidend sein (Manning et al., 2008, Kapitel 2). In dieser Arbeit werden je nach Ausgangsdokument zwei verschiedene Verfahren eingesetzt. Sie werden in Abschnitt 2.6.1 und 4.1.1 vorgestellt.

Tokenisieren

Ein Textdokument ist zunächst eine Folge von Zeichen. Tokenisierung bezeichnet die Aufgabe, diesen Text in Teile, die Token, zu zerlegen. Eine weit verbreitete Methode ist es, den Text an allen Nicht-Buchstaben-Zeichen zu teilen. Komplexere Methoden, die zum Beispiel zusammengesetzte Wörter, Bindestriche oder Apostrophe beachten, sind wesentlich aufwendiger. Um Wörter zu erkennen, die ein Leerzeichen enthalten (wie Bad Reichenhall oder computer science im Englischen) können sich zum Beispiel Wörterbücher eignen. Grundsätzlich können komplexere Methoden die Leistungsfähigkeit des Systems verbessern, müssen aber gegebenenfalls für jede zu unterstützende Sprache überarbeitet werden. Das Ergebnis dieses Schrittes ist immer ein Vektor von Token (Manning et al., 2008, Kapitel 2).

Stopp-Wörter entfernen

Stopp-Wörter sind sehr häufig auftretende Wörter wie Artikel oder Konjunktionen. Diese Wörter haben selbst keine Aussagekraft für den Inhalt eines Textes und können auch nicht als Merkmale zur Abgrenzung von verschiedenen Texten dienen. Aufgrund ihrer Häufigkeit würden sie aber beim Indizieren einen erhöhten Arbeitsaufwand verursachen und letztendlich auch den Index stark vergrößern. Viele Systeme nehmen diese Stopp-Wörter nicht mit in den Index auf und entfernen sie aus

der Tokenliste. Sie greifen dafür auf eine vorgefertigte Wort-Liste für die jeweilige Sprache zurück. Die Länge solcher Listen schwankt ungefähr zwischen einem halben Dutzend und mehreren hundert Wörtern. Eine Alternative zu vorgefertigten Listen ist eine statistische Analyse des Textbestandes, um die Stopp-Wörter zu finden (Wilbur und Sirotkin, 1992).

Normalisierung und Stemming

Wörter treten im Text in verschiedenen Wortformen¹ auf. Sie sind zum Beispiel nach ihrem Kasus flektiert, stehen im Plural, sind gesteigert oder in der Vergangenheit. Wenn Anwender bestimmte Wortformen in eine Suchmaske eingeben, erwarten sie im Allgemeinen, dass das System auch andere Wortformen des gleichen Wortes findet. Ein weitverbreitetes Vorgehen ist es, zunächst alle Großbuchstaben in Kleinbuchstaben umzuwandeln. Um Probleme bei der Eingabe und bei verschiedenen Schreibweisen von nationalen Sonderzeichen zu umgehen, werden in deutschen Texten Umlaute oftmals einfach zu Buchstaben des englischen 26-Zeichen-Alphabets umgewandelt (ä, ß, é → a, ss, e) (Manning et al., 2008, Kapitel 2).

Durch Stemming werden die Wörter danach auf ihre Grundform zurückgeführt. Porter et al. (1980) schlagen einen heute sehr weit verbreiteten Algorithmus vor, zu dem auch freie Implementierungen in vielen Programmiersprachen erhältlich sind.² Neben der ursprünglichen Version für das Englische werden heute auch Versionen für viele andere Sprachen bereitgestellt³. Anhand einer Kette von Regeln werden solange Suffixe abgeschnitten (und gegebenenfalls ersetzt), bis eine Mindestanzahl von Vokal-Konsonant-Folgen erreicht ist. So werden zum Beispiel sowohl *connected*, *connecting*, *connection*, *connections* als auch *connect* selbst auf *connect* zurückgeführt. Ergebnis des Stemming ist nicht immer die grammatikalisch korrekte Grundform der Wörter (z.B. *germany* → *germani*), was allerdings auch nicht notwendig ist.

Eine Alternative zum Stemming ist es, ein Vollformenlexikon zu nutzen. Darin kann für jede Wortform die Grundform nachgeschlagen werden. Noch bessere Ergebnisse können von einer echten morphologischen Analyse erwartet werden. Sie ist nicht auf ein Vollformenlexikon angewiesen und kann auch Neologismen und Ad-Hoc-Bildungen erkennen (Hausser, 2000, Kapitel 13).

2.1.2 Boolesches Information Retrieval

Im weiteren Sinne kann jede Art der Informationsbeschaffung als Information Retrieval bezeichnet werden. Das Durchsuchen des Karteikartenkatalogs nach Büchern des Lieblingsautors in einer Bibliothek oder auch die Suche nach einem Schriftwechsel im Aktenschrank sind Aufgaben, die den meisten Menschen vertraut sind.

Im engeren Sinne bezieht sich der Begriff Information Retrieval aber auf das Auffinden von Informationen in großen, ungeordneten Samm-

1. „[Eine] Wortform [ist die] konkret realisierte gramm. Form eines Wortes im Kontext eines Satzes“ (Bußmann, 2002, S. 754).

2. <http://tartarus.org/~martin/PorterStemmer/>

3. <http://snowball.tartarus.org/>

lungen von Textdokumenten. Dieser Aufgabe mussten bis vor wenigen Jahren nur wenige Menschen, wie Journalisten und Rechtsanwaltsgehilfen, nachgehen.

Heute stehen den meisten Menschen im Internet riesige Datenmengen zur Verfügung. Fast jeder Internetnutzer ist damit vertraut, mithilfe von Suchmaschinen wie beispielsweise Google⁴, große Teile des Internets durchsuchen zu lassen (Manning et al., 2008, Kapitel 1).

Aber auch innerhalb von Firmen gibt es immer größer werdende Dokumentensammlungen. Laut Untersuchungen von Feldman (2004) verbringen Wissensarbeiter 15-35% ihrer Arbeitszeit mit der Suche nach Informationen. Nur in der Hälfte der Fälle können die gewünschten Informationen dann auch tatsächlich gefunden werden. Unternehmen entsteht durch die Suche und die zusätzliche Arbeit nach einer erfolglosen Suche ein Schaden von mehreren tausend Euro pro Mitarbeiter und Jahr.

Das Ziel der booleschen Suche ist es, Dokumente zu finden, die bestimmte Wörter enthalten und dabei gegebenenfalls auch Dokumente auszuschließen, die bestimmte andere Wörter enthalten. Liegen diese Dokumente als Textdateien vor, kann ein entsprechender regulärer Ausdruck⁵ verwendet werden. Das bedeutet aber, dass alle Dokumente für jede Suche komplett durchlaufen werden müssen, was bei einer großen Anzahl von Dokumenten eine sehr lange Zeit dauern kann.

Um dieses Problem zu umgehen, bauen Information-Retrieval-Systeme einen sogenannten Index auf. Ein einfacher Index kann so gestaltet sein, dass er für jedes zu suchende Wort abspeichert, in welchen Dokumenten es vorkommt. Die Tabelle 2.1 zeigt ein Beispiel für einen solchen Index, der die Suche von Namen in E-Mails ermöglichen soll.

Durch diesen Index kann nun viel schneller eine Liste von E-Mails, die einer Suchanfrage entsprechen, zurückgeliefert werden. Der Index liefert für jeden Namen einen Vektor, der angibt, in welchen Dokumenten der Name vorkommt. Als Beispiel soll eine Suchanfrage für E-Mails, welche die Namen Maria und Robert, nicht aber Allen enthalten, dienen. Für die Berechnung der Antwort werden die Vektoren dieser drei Namen in Betracht gezogen. Eine bitweise Und-Verknüpfung liefert die E-Mails, die der Suchanfrage entsprechen. Da Allen nicht vorkommen soll, wird der entsprechende Vektor invertiert.

$$\begin{array}{ccccc} 10110 & \& 10100 & \& \sim 11000 & = & 00100 \\ \text{(Maria)} & & \text{(Robert)} & & \text{(Allen)} & & \end{array}$$

Nur E-Mail 3 enthält die Namen Maria und Robert, nicht aber Allen. Der Index kann natürlich für alle in sämtlichen E-Mails vorkommenden Wörter erweitert werden.

*Tabelle 2.1:
Beispielindex mit
E-Mails*

Wort	E-Mail 1	E-Mail 2	E-Mail 3	E-Mail 4	E-Mail 5
Allen	1	1	0	0	0
Julia	0	0	0	0	1
Maria	1	0	1	1	0
Robert	1	0	1	0	0

4. <http://www.google.de/>

5. Reguläre Ausdrücke werden zum Beispiel in Friedl (2006) beschrieben.

Diese Art von Index wird auch invertierter Index genannt. Der Begriff leitet sich daraus her, dass nicht die Dokumente selbst hintereinander abgespeichert werden. Stattdessen speichert der Index alle Wörter aus allen E-Mails als Liste, zusammen mit der Information, in welchen Dokumenten sie auftreten.

Dadurch würden solche Tabellen in der Praxis sehr viele Nullen enthalten. Sie direkt zu speichern, würde sehr viel Speicherplatz in Anspruch nehmen. Deswegen besteht ein invertierter Index aus einem Verzeichnis, in dem zu jedem Wort eine Liste der Dokumente, in denen das Wort vorkommt, gespeichert wird (zum Beispiel Robert $\rightarrow$ 1, 3) (Manning et al., 2008, Kapitel 1).

2.1.3 Relevanzbasiertes Information Retrieval

Die Methode aus dem letzten Abschnitt liefert als Antwort einer Suche eine ungeordnete Menge von Ergebnissen. Je nach Anforderung wird das nicht zufriedenstellend sein. Vor allem wenn die Menge sehr groß ist, werden viele Anwender die Mühe scheuen, sich durch die gesamte Liste der Ergebnisse zu arbeiten. Von modernen Suchmaschinen wird also erwartet, dass sie eine nach Relevanz sortierte Liste der Ergebnisse zurückliefern. Die Relevanz eines Ergebnisses ist ein Maß für seine Ähnlichkeit mit der Anfrage. Die konkreten Relevanz-Werte sind dabei meistens von untergeordneter Wichtigkeit. Für den Anwender ist es viel interessanter, welche Ergebnisse relevanter sind als andere. Abhängig von seinen Anforderungen wird der Anwender wählen, ob er sich das erste, die ersten 10, 20 oder auch 100 Ergebnisse ansehen möchte (Manning et al., 2008, Kapitel 8).

In diesem Abschnitt wird das Vektorraum-Modell vorgestellt, das die zuvor geforderte Relevanzbewertung der Ergebnisse ermöglicht. Dabei wird davon ausgegangen, dass die Suchanfragen als freier Text eingegeben werden. Eine Suchanfrage entspricht demnach einer Liste von Wörtern oder Termen, nach denen die Dokumente im Index durchsucht werden sollen. Da das Vektorraum-Modell voraussetzt, dass Wörter nach ihrer Relevanz gewichtet werden, wird nun zunächst eine dafür geeignete Möglichkeit vorgestellt.

Term-Frequency und Inverse-Document-Frequency

Für das Ermitteln der *Term-Frequency* (*tf*) wird gezählt, wie oft ein Term in einem Dokument vorkommt⁶. Die Positionen sind dabei nicht relevant. Dieser Wert kann als Gewicht in den Vektor für das Dokument aufgenommen werden. Im Abschnitt 2.1.1 wurde schon anhand der Stopp-Wörter aufgezeigt, dass nicht alle Wörter gleich viel Bedeutung tragen. Es ist sogar so, dass besonders häufige (hochfrequente) Wörter weniger aussagekräftig sind. Das Maß für das Gewicht eines Terms muss daher so erweitert werden, dass die Häufigkeit über alle Dokumente im Korpus in das Gewicht eingeht. Das Ziel ist es, Termen, die im jeweiligen Dokument und im gesamten Korpus häufig vorkommen, niedriger zu gewichten. Terme, die im Dokument häufig vorkommen, aber insgesamt selten sind, sollen höher gewichtet werden. Statt die absolute Häufigkeit des jeweiligen Terms im gesamten Korpus zu bestimmen, hat sich die Methode etabliert,

6. Ein Term kann hier zum Beispiel ein Token aus dem Ausgangsdokument sein. Oft werden jedoch die Grundformen der Wörter betrachtet.

die Anzahl der Dokumente zu bestimmen, die diesen Term enthalten. Dieser Wert wird *Document Frequency* (df) genannt. Er ist allerdings für häufige Terme hoch. Um das Gewicht für hochfrequente Terme zu erniedrigen, wird die *Inverse Document Frequency* (idf) gebildet. Sie ist für einen bestimmten Term t wie folgt definiert (Manning et al., 2008, S. 118):

$$idf_t = \log \frac{N}{df_t}.$$

N ist die Anzahl der Dokumente im Korpus, df_t die Anzahl der Dokumente, in denen der Term t vorkommt.

Statt die Term-Frequency direkt als Gewicht für den Term anzunehmen, kann nun ein kombinierter Wert gebildet werden. Dieser $tfidf$ -Wert für einen Term t in einem Dokument d ergibt sich wie folgt (Manning et al., 2008, S. 119):

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t.$$

Vereinfacht ausgedrückt liefert $tfidf$ für einen Term t im Dokument d einen Wert, der

- hoch ist, wenn t häufig in d vorkommt, aber nur in wenigen anderen Dokumenten;
- niedrig ist, wenn t weniger häufig in d vorkommt, aber öfter in anderen Dokumenten;
- immer sehr niedrig ist, wenn t in sehr vielen Dokumenten vorkommt

(Manning et al., 2008, Kapitel 6).

In der oben angegebenen Formel für idf_t kann es zu Divisionen durch 0 kommen. Deshalb wird in der Praxis df_t immer um 1 inkrementiert. Aus $tf_{t,d}$ wird die Wurzel gezogen, um die $tfidf$ -Werte nicht zu hoch werden zu lassen. Dies verhindert auch, dass einzelne Wörter zu stark gewichtet werden. Für Terme, die nicht im Dokument vorkommen, ergibt sich immer ein $tfidf$ -Wert von 0.

Ein einfacher Wert für die Relevanz eines ganzen Dokumentes d für eine Anfrage kann dann aus der Summe der $tfidf$ -Werte aller Terme t aus der Anfrage q berechnet werden (Manning et al., 2008, S. 119):

$$score(q, d) = \sum_{t \in q} tfidf_{t,d}.$$

Vektorraum-Modell

Im Vektorraum-Modell werden die Dokumente als Vektoren dargestellt. Die Vektoren speichern dabei die Gewichtungen aller Terme, die im gesamten Korpus vorkommen. Sie werden in einem hochdimensionalen Raum abgebildet, bei dem jeder Term eine Achse ist. Die Ähnlichkeit zweier Dokumente ($sim(d_1, d_2)$) wird nun durch das Berechnen des Kosinus des Winkels zwischen deren Vektoren bestimmt. Für die beiden Dokumente d_1 und d_2 mit den beiden Vektoren $\vec{V}(d_1)$ und $\vec{V}(d_2)$ wird dafür folgende Formel angewendet (Manning et al., 2008, S. 121):

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| |\vec{V}(d_2)|}.$$

Im Zähler steht dabei das Skalarprodukt der beiden Dokumente und im Nenner das Produkt ihrer Längen. Durch die Division mit den Längen werden die Vektoren der Dokumente normalisiert, ihre Längen wirken sich nicht auf den Ähnlichkeitswert aus und das Ergebnis liegt immer zwischen 0 und 1.

Eine Suchanfrage kann ebenfalls als Vektor in diesem Raum dargestellt werden. Die Dokumente, deren Vektoren die kleinsten Winkel zu dem Vektor der Suchanfrage haben, sind ihr am ähnlichsten (Manning et al., 2008, Kapitel 6).

2.1.4 Textklassifikation

Dokumente, die einer Suchanfrage entsprechen, können als eine Klasse oder Kategorie von Dokumenten betrachtet werden. Die einfachste Form der Kategorisierung von Dokumenten ist es, die Dokumente in zwei Klassen aufzuteilen: (i) solche Dokumente, die der Suchanfrage entsprechen und (ii) solche, die der Suchanfrage nicht entsprechen.

Das Beispiel der indizierten E-Mails aus Abschnitt 2.1.2 kann so erweitert werden, dass neu eingehende E-Mails klassifiziert werden. Die Klassen sind: (i) E-Mails von Freunden und (ii) andere E-Mails. Als Klassifizierer wird die eine Suchanfrage als freier Text formuliert:

Allen Julia Maria Robert.

Jedes Dokument, das mindestens eines dieser Begriffe enthält, entspricht der Suchanfrage und wird folglich in die Klasse E-Mails von Freunden aufgenommen.

Dies wird auch als vertikale Textklassifikation bezeichnet, da der Suchraum auf eine bestimmte Domäne – die E-Mails des Anwenders – „vertikal“ eingeschränkt ist. Würde der Suchraum auch E-Mails anderer Anwender oder sogar Zeitungsartikel und Internet-Seiten enthalten, würden viele Dokumente gefunden, die nicht der Intention des Anwenders entsprechen. Die Suchanfrage müsste im zweiten Fall so erweitert werden, dass sie die Einschränkung selbst festlegt.

Diese Methode definiert Klassen durch Suchanfragen. Das ist für Anwender bequemer, als jedes Dokument von Hand einer Klasse zuzuweisen. Der Nachteil dieses Verfahrens ist es, dass für jede Klasse ein Experte für das jeweilige Fachgebiet (in diesem einfachen Beispiel jemand, der die Namen der Freunde kennt) die Anfragen formulieren muss. Zudem werden sich die Anforderungen mit der Zeit ändern. In diesem Beispiel dann, wenn neue Freunde hinzukommen.

Die Alternative zu dieser manuellen Definition der Klassifizierer ist das maschinelle Lernen. Die Kriterien für die Klassifikation sollen dabei automatisch aus vorhandenen Trainingsdaten erlernt werden. Voraussetzung ist daher eine Vorgabe, welche Klassen es gibt und eine gewisse Menge an Beispieldokumenten, die diesen Klassen manuell zugeordnet werden (Manning et al., 2008, Kapitel 13).

Die Textklassifikation lässt sich wie folgt definieren (Manning et al., 2008, S. 256): Gegeben ist

- eine Menge von Dokumenten (X),
- ein Dokument d aus X ($d \in X$),
- eine festgelegte Menge von Klassen $C = \{c_1, c_2, \dots, c_J\}$ und
- eine Menge von manuell klassifizierten Trainingsdokumenten $D = \{(d, c) \mid (d, c) \in X \times C\}$

Die Klassifikation ist eine Funktion (γ), die Dokumenten Klassen zuordnet:

$$\gamma : X \rightarrow C.$$

Dies wird auch als beaufsichtigtes Lernen (englisch: supervised learning) bezeichnet, da, wie oben angemerkt, jemand die Trainingsdaten manuell erstellen muss. Eine automatische Lernfunktion leitet dann aus den Trainingsdaten die Funktion γ ab.

Klassifikationen werden an einer Menge von Testdokumenten getestet. Diesen Testdokumenten muss zuvor ebenfalls manuell eine Klasse aus C zugeordnet worden sein. Die Menge der Trainingsdokumente und die Menge der Testdokumente dürfen sich nicht überschneiden.

Die Leistungsfähigkeit einer Klassifikation kann dann danach bestimmt werden, ob sie den Testdokumenten die gleichen Klassen zuordnet wie bei der manuellen Zuweisung. Wird jedes Dokument genau einer Klasse zugeordnet (one-of), ist die Auswertung einfach – die Klassifikation ist korrekt oder falsch. In der Praxis werden Dokumente aber meistens mehreren Klassen zugeordnet (any-of). Abschnitt 2.1.5 gibt einen Überblick der Möglichkeiten, die Leistungsfähigkeit solcher Systeme zu messen.

Weitverbreitete Methoden zur Textklassifikation, die auf dem beschriebenen Verfahren basieren, sind Naive Bayes und SVM (Stützvektormethode, englisch: Support Vector Maschine).

Bei den meisten Klassifizierungsanwendungen gibt es mehr als zwei Klassen. Dokumente können einer oder mehreren dieser Klassen zugeordnet werden. Die Standardvorgehensweise, die oben genannten Methoden bei any-of-Problemen einzusetzen, ist für jede Klasse einen eigenen Klassifizierer zu trainieren. Bei der Klassifikation muss dann jeder Klassifizierer für das jeweils zu klassifizierende Dokument angewandt werden.

Allen gemeinsam ist, dass Änderungen der Klassifizierungen teuer sind. Werden neue Klassen hinzugefügt oder vorhandene Dokumente neu klassifiziert, müssen die Entscheidungsfunktionen neu gelernt werden. Bei SVM entsteht auch ein nicht zu vernachlässigender Aufwand, wenn neue Dokumente in die Trainingsmenge aufgenommen werden.

Allgemein wird SVM heute als die Textklassifikationsmethode mit den besten Ergebnissen angesehen (Manning et al., 2008, Kapitel 13).

Neben der Textklassifikation gibt es noch das Textclustering. Dabei muss das System selbst Klassen finden, die die Dokumente sinnvoll gruppieren. Textclustering ist im Gegensatz zur Textklassifikation eine unbeaufsichtigte (englisch: unsupervised) Methode, um Texte Klassen zuzuordnen. Wie bereits angemerkt, ist die Textklassifikation nicht vollständig unbeaufsichtigt, da ein Fachexperte erst die Klassen festlegen und Dokumente für das Training zuweisen muss (Manning et al., 2008, Kapitel 16).

2.1.5 Evaluation

Das Beispiel der Suche nach Namen in E-Mails hat auch den Vorteil, dass der Nutzer immer eine exakte Anfrage stellen kann und immer eine richtige Antwort im Sinne der Anfrage erhält.

In praktischen Anwendungen ist es aber eher so, dass nach Dokumenten gesucht wird, die von einem bestimmten Thema handeln. Hausser (2000, Kapitel 2) führt als Beispiel eine Anwaltskanzlei auf, die für einen Fall nach Fahrradunfällen von Minderjährigen recherchiert, bei denen die Vorfahrt nicht beachtet wurde. In die Suchmaske für eine Datenbank, die sämtliche Gerichtsurteile enthält, wird eine Suchanfrage mit charakteristischen Begriffen eingegeben: minderjährig, Fahrrad, Unfall und Vorfahrt. Die Datenbank liefert als Ergebnis eine Liste aller Urteile, die all diese Begriffe enthalten. Trotzdem ist es so, dass nicht alle Urteile tatsächlich relevant im Sinne des ursprünglichen Anliegens sind. Zum Beispiel kann ein Fahrradunfall von einem minderjährigen Zeugen beobachtet worden sein.

Andererseits werden wahrscheinlich relevante Dokumente nicht gefunden, da in den Texten anderes Vokabular verwendet wird: 12-jährig, statt minderjährig zum Beispiel.

Recall und Precision

Um die Qualität der Suchergebnisse zu bewerten, werden die beiden Maße *Recall* und *Precision* herangezogen. Sie sind wie folgt definiert (Hausser, 2000, S. 38):

- „*Recall* misst, wie viele der relevanten Texte im Verhältnis zum Gesamtbestand gefunden werden.“
- „*Precision* misst, wie viele relevante Texte im Suchergebnis enthalten sind.“

In der Praxis sind diese Werte abhängig voneinander. Durch das Verwenden von zusätzlichen Suchbegriffen kann versucht werden, die Menge der relevanten Ergebnisse zu erhöhen. Da dadurch aber gleichzeitig die Menge der Ergebnisse insgesamt stark ansteigen wird, wird auch die Precision fallen. Wird dagegen versucht die Precision durch genauere Abfragen zu erhöhen, wirkt sich das negativ auf den Recall aus. Die Werte verhalten sich also umgekehrt proportional zueinander (Hausser, 2000, Kapitel 2).

Systeme anhand dieser beiden Werte zu bewerten, ist insofern sinnvoll, als unterschiedliche Anwender verschiedene Erwartungen haben. Nutzer einer Internet-Suchmaschine werden zum Beispiel erwarten, dass die erste Seite der Ergebnisse möglichst nur relevante Dokumente enthält. Dagegen werden Anwender auf der Suche nach bestimmten E-Mails auch akzeptieren, dass die Suche einige irrelevante E-Mails findet. Wichtig ist ihnen, dass alle relevanten E-Mails gefunden wurden. Ein Recall von 1 bzw. 100% lässt sich jedoch immer dadurch erreichen, alle Dokumente in die Liste der Ergebnisse aufzunehmen. Das ist natürlich auch für die Anwender der zuletzt genannten Gruppe nicht zumutbar.

Ergebnisse, die auch tatsächlich für die Anfrage relevant sind, werden auch *true positives (tp)* genannt. Ergebnisse, die nicht relevant sind, entsprechend *false positives (fp)*. Nicht gefundene Dokumente, die auch nicht relevant sind, heißen demnach *true negatives (tn)*, nicht gefundene, aber relevante *false negatives (fn)*.

Formeln für die Berechnung von Recall und Precision können somit wie folgt angegeben werden (Manning et al., 2008, S. 155):

$$\text{Recall} = \frac{tp}{tp+fp} = \frac{\text{Anzahl relevanter Ergebnisse}}{\text{Anzahl Ergebnisse}},$$

$$\text{Precision} = \frac{tp}{tp+fn} = \frac{\text{Anzahl relevanter Ergebnisse}}{\text{Anzahl relevanter Dokumente}}.$$

Ein anderer Wert, die *Accuracy*, teilt die Dokumente in nur zwei Klassen: (i) relevant und (ii) nicht relevant. Die Leistungsfähigkeit der Suche wird dann danach berechnet, wie viele Dokumente der richtigen Klasse zugeordnet wurden. Dieser Wert ist aber nicht geeignet, um die Ergebnisse in dieser Aufgabenstellung auszuwerten. Die Menge der relevanten Ergebnisse ist meistens sehr gering im Verhältnis zu der Menge aller Dokumente. Bei einem zum Beispiel schon eher hohen Anteil von relevanten Dokumenten von 1% würde selbst ein System, das einfach alle Dokumente als nicht relevant einstuft, stets eine Accuracy von 99% erreichen.

Ein solches System wäre natürlich überhaupt nicht zufriedenstellend, da Anwender immer erwarten, dass ein Teil der Dokumente als relevant klassifiziert wird (besonders dann, wenn sie wissen, dass ein relevantes Dokument vorhanden sein muss). Dafür wird auch in Kauf genommen, einige nicht relevante Dokumente in der Liste der Ergebnisse zu haben.

Recall und Precision haben den Nachteil, dass es nicht immer auf den ersten Blick gelingt, die Ergebnisse aus verschiedenen Experimenten miteinander zu vergleichen. Ein höherer Recall und eine niedrigere Precision können sowohl eine Verbesserung als auch eine Verschlechterung der Leistungsfähigkeit bedeuten. Das arithmetische Mittel der beiden Werte zu nutzen, um für jedes Experiment einen einzigen Ergebniswert zu erhalten, ist keine Lösung. Wie bereits zuvor ausgeführt, ist es immer möglich, einen Recall von 1 zu erreichen, indem das System einfach nur alle Dokumente zurückliefert. Der Mittelwert von mindestens 0,5 täuscht dann über das eigentlich nicht zufriedenstellende Ergebnis hinweg (Manning et al., 2008, Kapitel 1).

F-Measure

Ein einziger Wert, der Recall und Precision gegeneinander abwägt, ist der *F-Measure* (gewichtetes harmonisches Mittel) (Manning et al., 2008, S. 156):

$$F = \frac{(\beta^2+1) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}.$$

Für $\beta = 1$ werden Recall und Precision gleich gewertet. Ist der Recall für eine Anwendung wichtiger, wird ein Wert $\beta > 1$ gewählt. Ist dagegen die Precision wichtiger, wird ein Wert $0 < \beta < 1$ gewählt. Wie Precision und Recall selbst, liegt der Wert F-Measure immer zwischen 0 und 1, bzw. 0 und 100%.

Für $\beta = 1$ vereinfacht sich die Formel wie folgt (Manning et al., 2008, S. 156):

$$F = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Die oben angeführte Strategie, einfach alle Dokumente zurückzuliefern, führt nun bei einem Satz von 10 000 Dokumenten mit nur einem relevanten Dokument zu einem F-Measure von 0,0002, der die schlechte Leistungsfähigkeit ausdrückt (Manning et al., 2008, Kapitel 1).

Precision at k

In modernen Suchmaschinen werden den Ergebnissen Werte zugeordnet. Diese Werte geben an, wie sehr die gefundenen Dokumente der Suche entsprechen, beziehungsweise, wie relevant sie für die Anfrage sind. Die Liste der Ergebnisse ist entsprechend geordnet. Unter den ersten 10 Ergebnissen sind Recall und Precision normalerweise höher als unter den zweiten 10 Ergebnissen. Werden die ersten 10 Ergebnisse mit den ersten 20 Ergebnissen verglichen, steigt der Recall und sinkt die Precision. Eine Suchmaschine, die ihre Ergebnisse geordnet zurück liefert, wird also im Allgemeinen so konfiguriert sein, dass sie möglichst viele Ergebnisse zurück liefert. Der Anwender kann dann je nach Anforderung entscheiden, ob er sich die ersten 10, 20 oder sogar 100 Ergebnisse ansehen möchte. Da Recall, Precision und F-Measure aber mengen-basierte Werte sind, können sie nicht direkt für die Evaluation solcher Systeme herangezogen werden.

Für den Anwender ist es also interessant, wie hoch die Precision unter den ersten k (10, 20, 100, ...) Dokumenten ist. Um diesen Bedürfnissen in den Auswertungen gerecht zu werden, kann die jeweilige Precision für eben diese Werte für k berechnet werden. Das wird allgemein als *Precision at k* bezeichnet. Der Wert ist also besonders interessant, um die gefühlte Leistungsfähigkeit für den Anwender abzuschätzen. Er ist allerdings stark abhängig von der Menge der relevanten Dokumente im System. Sogar ein perfekt funktionierendes System kann nur einen Wert von 0,5 für die Precision at 20 erreichen, wenn nur 10 relevante Dokumente vorhanden sind (Manning et al., 2008, Kapitel 13).

R-Precision

Viel besser lassen sich die Werte für verschiedene Anfragen mithilfe der *R-Precision* vergleichen. Hierfür muss die Menge R der relevanten Dokumente im System bekannt sein. Statt fester Werte für k bei Precision at k wird nun die Precision bei R Ergebnissen berechnet. Ein perfektes System kann für dieses Maß immer einen Wert von 1 erreichen. Eine Mittelwertbildung über mehrere Anfragen ist hier also wesentlich sinnvoller als bei Precision at k . Auch Recall und F-Measure können bei $k = R$ Ergebnissen berechnet werden. Per Definition sind aber alle drei Werte immer gleich (Manning et al., 2008).

Buckley und Voorhees (2000) untersuchen verschiedene Maße zur Bewertung der Qualität von Information-Retrieval-Systemen hinsichtlich ihrer Stabilität. Die R-Precision ist dabei das Maß mit der geringsten Fehlerrate. Sie eignet sich besonders, um Versuche mit verschiedenen Einstellungen und über verschiedene Datensätze zu vergleichen.

In der Praxis haben all diese Maße das Problem, dass bekannt sein muss, welche beziehungsweise wie viele Dokumente für eine Anfrage relevant sind. Um die Precision zu berechnen, könnte noch nach jeder Anfrage manuell durch die Liste der Ergebnisse gegangen werden, um deren Relevanz zu bewerten. Ob alle relevanten Dokumente gefunden wurden, kann praktisch nicht mehr manuell ermittelt werden. Um ein System zu testen, wäre es also wünschenswert, dass vorher alle Dokumente entsprechend den Anfragen zugeordnet wurden, was nicht immer mit vertretbarem Aufwand möglich ist (Hausser, 2000, Kapitel 2).

Bei der Textklassifikation werden häufig Testdaten verwendet, um die Systeme zu evaluieren. Hierzu werden alle Dokumente von Hand klassifiziert und die Berechnung von Precision und Recall ist ohne weiteres möglich.

2.2 Wikis

Ein Wiki ist eine Serversoftware, die es Anwendern erlaubt, Webseiten online, nur durch die Verwendung eines Webbrowsers, zu erstellen und zu bearbeiten. Demzufolge sind auch keine zusätzlichen Programme auf dem PC des Anwenders notwendig. Die Daten werden immer zentral gespeichert und stehen direkt nach dem Erstellen zur Verfügung.

Eine einfache Auszeichnungssprache ermöglicht auch technisch eher unerfahrenen Anwendern das Anlegen neuer Seiten, das Einfügen von internen und externen Links und die Strukturierung des Textes (Cunningham und Leuf, 2001).

Das wohl größte und bekannteste Wiki ist die Wikipedia. Sie basiert auf der Wiki-Software MediaWiki⁷, welche sicherlich nicht zuletzt durch den Erfolg der Wikipedia zu einem der bekanntesten Wiki-Systeme geworden ist (Barrett, 2008). Auf MediaWiki wird im nächsten Abschnitt (2.3) genauer eingegangen.

Wikis werden aber auch unternehmens-, organisations- oder projektintern als Wissensmanagementplattform eingesetzt. Viele Arbeiten, die ihre Untersuchungen zwar speziell auf die Wikipedia ausrichten, lassen sich somit auch generell für Wikis anwenden. Die Wikipedia wird vor allem aufgrund ihres Bekanntheitsgrades, der freien Verfügbarkeit und des umfangreichen Inhalts verwendet. Darüber hinaus sind die Ergebnisse von verschiedenen Arbeiten vergleichbar und können von interessierten Lesern nachvollzogen werden.

2.3 MediaWiki

MediaWiki ist eine Wiki-Software. Sie wurde unter der Open-Source-Lizenz GPL veröffentlicht und ursprünglich für die Wikipedia entwickelt. Heute ist sie aber auch in anderen Bereichen weit verbreitet (Barrett, 2008).

Wichtig für diese Arbeit ist, dass MediaWiki eine Schnittstelle (API) zur Verfügung stellt, um anderen Programmen einen einfachen Zugriff auf die Daten zu ermöglichen und eine eigene Auszeichnungssprache

7. <http://www.mediawiki.org/wiki/MediaWiki>

Abbildung 2.1: Wikitext
des Wikipedia-Artikels
Cologne
(Bildschirmaufnahme)

```
'''Cologne''' ({{lang-de|Köln}}, {{IPA-de|ˈkœln|pron|De-Köln.ogg}}; [[Kölsch  
dialect]]: ''Kölle'' {{IPA-dia|ˈkœlə}}) is Germany's [[List of cities in  
Germany with more than 100,000 inhabitants|fourth-largest city]] (after  
[[Berlin]], [[Hamburg]] and [[Munich]]), and is the largest city both in the  
German Federal State of [[North Rhine-Westphalia]] and within the [[Rhine-  
Ruhr|Rhine-Ruhr Metropolitan Area]], one of the major European metropolitan  
areas with more than ten million inhabitants. It is one of the oldest cities in  
Germany, having been founded by the Ubii in the year 38 BC. The name is derived  
from that of the Roman settlement, ''[[Colonia Claudia Ara Agrippinensium]]''.
```

(Wikitext) bereitstellt. Die Abbildung 2.1 zeigt einen Ausschnitt aus dem Wikitext für den Artikel Cologne⁸ aus der englischsprachigen Wikipedia. In Abschnitt 4.1.1 wird ein genauerer Blick auf den Wikitext geworfen und erläutert, wie daraus einfacher Text gewonnen wird.

MediaWiki ist im Laufe der Zeit immer weiter gewachsen und eine komplexe Software mit vielen Eigenschaften und Möglichkeiten geworden. Diese Details sind aber für das Verständnis dieser Arbeit nicht relevant. Ausführlichere Beschreibungen können zum Beispiel auf der Projektseite von MediaWiki oder in Barrett (2008) nachgelesen werden.

2.4 Wikipedia

Das Wikipedia-Projekt wurde 2001 gestartet. Das Ziel ist es, eine freie Enzyklopädie in allen Sprachen der Welt zu erschaffen. Jeder der möchte, kann mit seinem Wissen beitragen. Zugangsbeschränkungen werden nur in Ausnahmefällen verhängt und Kriterien wie Relevanz und Korrektheit demokratisch geregelt. Das Wissen soll global, frei und kostenlos für jedermann verfügbar sein (Riehle, 2006).

Wikipedia übertrifft alle vorhandenen Enzyklopädien hinsichtlich ihres Umfangs. In den Anfängen, aber auch noch heute, gab es große Skepsis gegenüber dem Projekt (Denning et al., 2005). Dabei gibt es durchaus Arbeiten, die zeigen, dass Wikipedia mit ihren großen Konkurrenten mithalten kann.

Giles (2005) untersucht die Korrektheit von zufällig ausgewählten Wikipedia-Artikeln wissenschaftlicher Themen und vergleicht sie mit den entsprechenden Einträgen der Encyclopædia Britannica. In den 41 untersuchten Artikeln findet er 162 kleinere Fehler in der Wikipedia und 123 in der Britannica. Beide Quellen hatten keine groben Fehler wie eine völlig falsche Darstellung von Sachverhalten.

Lorenz und Türp (2010) untersuchten die Qualität zahnmedizinischer Einträge in der Wikipedia im Vergleich zu denen in medizinischer Fachliteratur. Sie kommen zu dem Ergebnis, dass über die Hälfte der untersuchten Einträge inhaltlich korrekt waren, auch wenn die Qualität der Aufbereitung nicht immer überzeugte. In nur 16% der Artikel wurden inhaltliche Fehler gefunden.

Andere Untersuchungen kommen zu ähnlichen Ergebnissen. Mühlhauser und Oser (2008) untersuchten die Qualität von Gesundheitsthemen in der Wikipedia und schlussfolgerten, dass nur wenige Angaben falsch sind. Allerdings fehlten viele wichtige Angaben, so dass die Wikipedia die Beratung durch einen Arzt nicht ersetzen kann.

8. <http://en.wikipedia.org/w/index.php?title=Cologne&oldid=405461331>

Nur wenige qualitativ hochwertige Artikel werden nach einem Abstimmungsprozess auch durch die Attribute *exzellent* (engl. *featured*) und *lesenswert* (engl. *good*) kenntlich gemacht. In der englischsprachigen Wikipedia sind das nur etwa 14 000 der über 3,5 Mio. Artikel (Wikipedia contributors (2011b) und Wikipedia contributors (2011c)).

Für einzelne Bereiche schließen sich auch Autoren zusammen, um gemeinschaftlich an der Verbesserung der Qualität von Artikeln aus „ihrem“ Bereich zu arbeiten und die Qualität der Artikel nach eigenen Kriterien zu bewerten. Ziel ist es, gute Artikel als solche kenntlich zu machen und aufzuzeigen, wo noch Verbesserungen notwendig sind. Ein Beispiel hierfür ist das *WikiProject Germany*⁹. Darin werden auf Deutschland bezogene Artikel gesammelt und bewertet. Die Bewertungen werden hierbei wesentlich differenzierter vorgenommen, sind aber auch auf diesen Bereich beschränkt.

Metriken für eine automatische Bewertung der Artikelqualität wurden in verschiedenen Arbeiten vorgeschlagen und evaluiert. Einen Überblick gibt Abschnitt 3.1.

2.4.1 Projekt

In traditionellen Enzyklopädien sind die Einträge alphabetisch angeordnet. Es gibt interne Verweise zu anderen Einträgen und externe Verweise zu Belegen und weiterführender Literatur.

Diese grundlegenden strukturellen Eigenschaften wurden von den Wikipedia-Entwicklern für die spezielle Umgebung angepasst. In den folgenden Abschnitten soll zunächst beschrieben werden, was ein Artikel ist und welche weiteren Seiten es in der Wikipedia gibt. Sowohl für Autoren als auch Leser ist die Möglichkeit, Artikel in Kategorien zu gruppieren, besonders hilfreich. Darauf wird in Abschnitt 2.4.4 eingegangen.

2.4.2 Artikel

Das wichtigste Strukturelement sind die Artikel. Artikel sind Wikipedia-Seiten, die einen Eintrag in der Enzyklopädie beinhalten. Allein die englischsprachige Wikipedia besteht derzeit schon aus über 3.5 Mio. Artikeln.

Jeder Artikel handelt von einem Konzept¹⁰ und jedes Konzept soll nur in einem Artikel behandelt werden. Die Titel sind Wörter oder kurze Phrasen, wie sie auch in Wörterbüchern verwendet werden. Für Synonyme gibt es keinen eigenen Artikel. Für sie werden Weiterleitungsartikel angelegt, die lediglich auf den eigentlichen Artikel verweisen.

Begriffserklärungsseiten (disambiguation pages) listen verschiedene Artikel, deren Konzepte denselben Namen haben. Da Artikelnamen immer eindeutig sein müssen, wird zur Unterscheidung eine kurze Anmerkung, durch Klammern gekennzeichnet, in die Artikelnamen aufgenommen.

Jeder Artikel soll mit einer Definition (Abriss) beginnen – der Autor soll nie davon ausgehen, dass der Leser bereits weiß, wovon der Artikel

9. http://en.wikipedia.org/wiki/Wikipedia:WikiProject_Germany

10. „[Ein Konzept oder] Begriff [ist ein] durch Abstraktion gewonnenes gedankliches Konzept, durch das Gegenstände oder Sachverhalte auf Grund bestimmter Eigenschaften und/oder Beziehungen klassifiziert werden. [...] Auf der intensionalen Begriffsdefinition beruht die in der Semantik geläufige Gleichsetzung von B. mit Bedeutung und Freges Sinn“ (Bußmann, 2002, S. 119).

Abbildung 2.2:
Wikipedia-Artikel
Cologne
(Bildschirmaufnahme)

Cologne

From Wikipedia, the free encyclopedia

"Köln" redirects here. For other uses, see *Köln (disambiguation)*.

This article is about the German city. For other uses, see *Cologne (disambiguation)*.

Cologne (German: *Köln*, pronounced [ˈkœln] (listen); *Kölsch* dialect: *Kölle* [ˈkœlə]) is Germany's **fourth-largest city** (after *Berlin*, *Hamburg* and *Munich*), and is the largest city both in the German Federal State of *North Rhine-Westphalia* and within the *Rhine-Ruhr Metropolitan Area*, one of the major European metropolitan areas with more than ten million inhabitants. It is one of the oldest cities in Germany, having been founded by the Ubii in the year 38 BC. The name is derived from that of the Roman settlement, *Colonia Claudia Ara Agrippinensium*.



handelt. Die Artikel enthalten interne Verweise (Hyperlinks) zu anderen Artikeln und Verweise zu externen Seiten, Quellen und weiterführenden Literaturangaben. Externe Verweise können, je nach Ziel, sowohl als Hyperlinks als auch in Textform auftreten (Medelyan et al., 2009).

Die Abbildung 2.2 zeigt den Beginn eines charakteristischen Wikipedia-Artikels. Ganz oben steht der Artikelname, unter dem der Artikel gespeichert wurde: *Cologne*. Es existieren aber noch weitere Artikel mit dem Namen Cologne. Sie werden unter einer Begriffserklärungsseite gelistet, auf die direkt nach dem Artikelnamen verwiesen wird („[...] see Cologne (disambiguation)“). In Fällen, in denen eine bestimmte Bedeutung eines Lemmas¹¹ die anderen nicht so sehr überwiegt, wie das bei Cologne der Fall ist, kann der Leser auch direkt zur Begriffserklärungsseite weitergeleitet werden und selbst entscheiden, welcher Artikel seiner Suche entspricht.

Im Beispielartikel trifft des Weiteren der seltene Fall ein, dass ein anderes Lemma (*Köln*), welches auf diesen Artikel weiterleitet, selbst noch Einträge in der Wikipedia hat. Auch in dieser Situation wird auf die entsprechende Begriffserklärungsseite verwiesen.

Neben der Behandlung von Synonymen werden Weiterleitungen auch genutzt, um häufige Schreibfehler (*Adress* → *Address*) abzufangen und Wortformen auf ihre Grundform (*Trees* → *Tree*) verweisen zu lassen (Wikipedia contributors, 2011e).

Die Artikel sind intern durch Absätze und Überschriften strukturiert. In Abschnitt 2.5.1 wird ein etwas genauerer Blick auf die Artikelstruktur geworfen.

2.4.3 Andere Seiten

Neben den eigentlichen Artikeln und Weiterleitungsartikeln gibt es in der Wikipedia viele weitere Seiten. Den größten Anteil stellen dabei die Autoren-Seiten. Sie werden von den Autoren genutzt, um sich selbst vorzustellen und Notizen oder Entwürfe zu sammeln. Des Weiteren gibt es spezielle Seiten für Bilder, zur Kategoriebeschreibung oder Hilfeseiten für MediaWiki und Wikipedia. Zu jeder Seite kann jeweils noch eine Diskussionsseite angelegt werden.

Um all diese verschiedenen Seiten von den eigentlichen Artikeln zu unterscheiden, werden sie in sogenannte Namensräume unterteilt. Die Namen der Namensräume werden den Namen der Seiten vorangestellt. Die Seite der Kategorie Cologne wird zum Beispiel durch *Category:Cologne* referen-

11. „[Ein Lemma ist ein] Eintrag bzw. einzelnes Stichwort in einem Lexikon oder Wörterbuch.“ (Bußmann, 2002, S. 399)

ziert, wobei der Teil links des Doppelpunktes den Namensraum angibt. Die eigentlichen Artikel sind im Namensraum Main, der nicht extra angegeben werden muss. Die Diskussionsseiten werden zusätzlich durch das Wort *talk* im Namensraum kenntlich gemacht. Die Diskussionsseite zum Artikel Cologne wird zum Beispiel durch *Talk:Cologne* und die der Kategorie Cologne durch *Category talk:Cologne* referenziert (Wikipedia contributors, 2011d).

Die Tabelle 5.1 in Abschnitt 5.1.1 listet alle Namensräume, die in der englischsprachigen Wikipedia genutzt werden und gibt an, wie viele Seiten jeweils in diesen Namensräumen enthalten sind.

2.4.4 Kategorien

Wegen der immer größer werdenden Anzahl von Artikeln stellten die Entwickler der Wikipedia die Möglichkeit zur Verfügung, Artikel zu kategorisieren. Artikel, die thematisch zusammengehören, sollen gleichen Kategorien zugewiesen werden. Auf diese Weise bieten die Kategorien sowohl Lesern als auch Autoren eine einfache Möglichkeit, sich einen Überblick über ein bestimmtes Themengebiet zu verschaffen. Die Kategorien können somit auch als Ergänzung zu den internen Links gesehen werden.

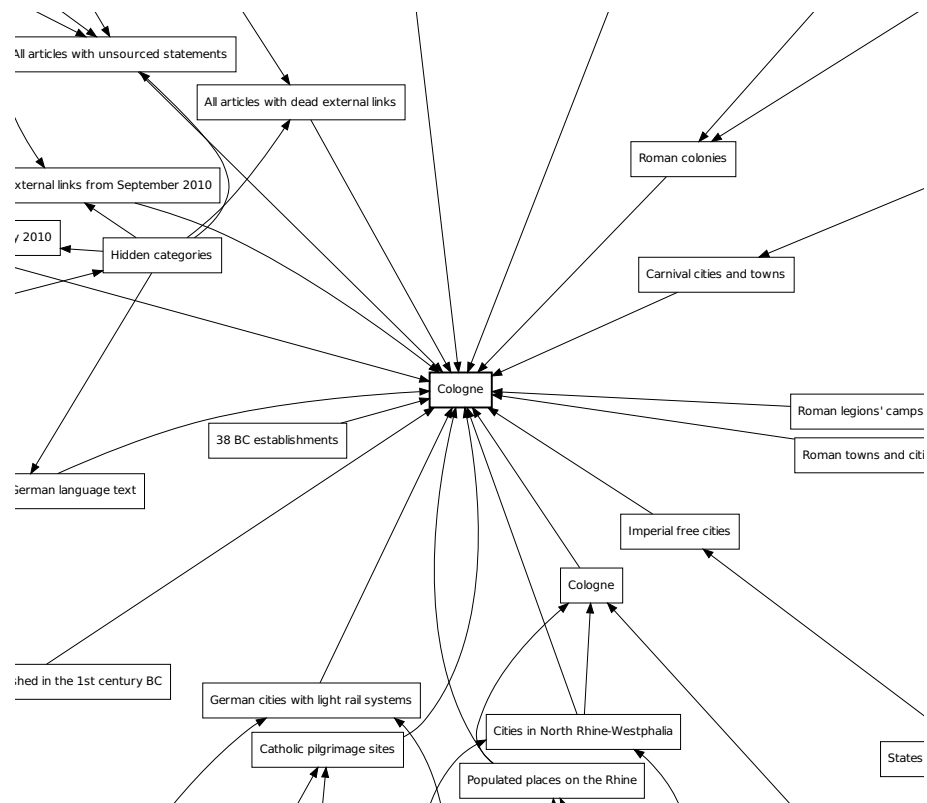
Das Categoriesystem und die Menge der zur Verfügung stehenden Kategorien ist nicht festgelegt. Es entwickelt sich, genau wie die Wikipedia selbst, ständig weiter. Technisch gesehen entstehen neue Kategorien dadurch, dass sie einfach einem Artikel zugeordnet werden. Jeder Autor sollte jedoch zunächst versuchen, auf bestehende Kategorien zurückzugreifen. Sind zu viele Artikel in einer Kategorie, kann es sinnvoll sein, die Kategorie aufzuteilen und feinere Kategorien zu verwenden.

Die meisten Artikel sind mindestens einer Kategorie zugeordnet, viele mehreren Kategorien. Kategorien können wiederum kategorisiert werden, wodurch sich Oberkategorien ergeben. Auch die Oberkategorien können wiederum (Ober-)Oberkategorien zugeordnet werden. Das setzt sich so lange fort, bis die Kategorie *Main topic classifications* erreicht ist, die die oberste Kategorie in der Wikipedia ist (Wikipedia contributors, 2011a).

Die Hierarchie der Wikipedia-Kategorien kann nicht als einfacher Baum angesehen werden, da Kategorien mehreren Oberkategorien zugeordnet werden können (Schönhofen, 2009).

In der Wikipedia selbst wird die Kategoriestructur als gerichteter azyklischer Graph bezeichnet (Wikipedia contributors, 2011a). Gerichtet bedeutet hier, dass festgelegt ist, welche die Oberkategorie und welche die Unterkategorie ist. Bei einer Darstellung des Graphen können deshalb Pfeile verwendet werden, um die Beziehung zwischen den Kategorien zu beschreiben. Azyklisch heißt, dass innerhalb des Graphen keine Zyklen auftreten. Ein Zyklus tritt auf, wenn zum Beispiel die Kategorie *A* die Kategorie *B* enthält, die wiederum Kategorie *C* enthält und Kategorie *C* wiederum Kategorie *A* enthält. Beide Graph-Eigenschaften sind zwar als Richtlinien angegeben, werden aber nicht durch technische Einschränkungen erzwungen. So haben zum Beispiel Zesch und Gurevych (2007) in der deutschsprachigen Wikipedia mehrere Zyklen ausfindig gemacht.

Abbildung 2.3:
Kategorie-Graph des
Artikels Cologne
(Ausschnitt)



Ein weiteres Problem ist, dass ein Artikel einer Kategorie sowohl direkt als auch indirekt zugeordnet werden kann. Der Artikel Cologne ist zum Beispiel der Kategorie Cologne direkt zugeordnet, aber auch indirekt über die Kategorie *Cities in North Rhine-Westphalia* erreichbar. Die Abbildung 2.3 bildet einen Ausschnitt aus dem Kategorie-Graph des Artikels Cologne ab.

Weiterhin sind Kategorien problematisch, in denen nur ein einziger Artikel enthalten ist und zudem keine Oberkategorien zugeordnet wurden. Der Kategoriegraph ist in diesem Fall noch unvollständig. Die Kategorie *38 BC establishments* des Artikels Cologne ist ein Beispiel dafür. In Abschnitt 6.3 wird diskutiert, wie sich diese Probleme auf Wikitech Classify auswirken.

Neben den bereits vorgestellten Kategorien zur inhaltlichen Strukturierung der Wikipedia gibt es noch sogenannte Wartungskategorien. Beispiele hierfür sind *All articles with dead external links* oder *Date of Birth missing* (Wikipedia contributors, 2011a).

Zurzeit gibt es in der englischsprachigen Wikipedia über 500 000 Kategorien, zuzüglich der Wartungskategorien.

2.4.5 Dumps

Das Wikipedia-Projekt bietet komplette Abzüge aller Daten an. Diese Daten können zum Beispiel verwendet werden, um Spiegelseiten aufzusetzen. Da sie komplett lokal vorgehalten werden können, bieten sich die Dumps auch als effizientere Methode an, um mit dem Wikipedia-Inhalt zu arbeiten.

Der für diese Arbeit verwendete Dump der Wikipedia ist eine komprimierte XML-Datei, die alle Seiten in ihren zur Zeit der Erstellung

aktuellen Versionen und einige Zusatzinformationen enthält. Durch die XML-Struktur ist es möglich, auf einzelne Artikel und ihre Zusatzinformationen, wie den Titel, zuzugreifen. Die Artikel selbst liegen innerhalb eines einzigen XML-Elements als Wikitext vor (Wikipedia contributors, 2010).

Erste Tests für diese Arbeit erfolgten direkt auf Basis dieser Datei. Später wurde eine von der Open Source Research Group zur Verfügung gestellte PostgreSQL-Datenbank¹² verwendet. In diese Datenbank wurde das komplette Abbild der Wikipedia vom 30. Januar 2010¹³ eingespielt.

Die Alternative zur Verwendung eines Abbilds ist das direkte Arbeiten auf den aktuellen Seiten in der Online-Version. Das hat aber neben den Geschwindigkeits-Problemen auch den Nachteil, dass Ergebnisse zwischen verschiedenen Versuchen gegebenenfalls nicht vergleichbar sind. Die Daten könnten vor einem späteren Test bereits geändert worden sein.

2.5 MediaWiki-Parser

In der Open Source Research Group wird zurzeit ein Parser für die MediaWiki-Markup-Sprache entwickelt. Dieses Projekt ist eine wichtige Grundlage für diese Arbeit.

Die Bibliothek stellt Funktionen bereit, um einen beliebigen Wikitext in einen sogenannten abstrakten Syntaxbaum (englisch: abstract syntax tree, kurz: AST) zu überführen. Mit ihm kann auf einfache Weise auf die Struktur der Artikel zugegriffen werden. Außerdem ist es möglich, auf einzelne Elemente, wie beispielsweise Links, zuzugreifen.

MediaWiki selbst ist in PHP geschrieben und übersetzt den Wikitext in HTML, indem wiederholte Ersetzungen auf Basis von regulären Ausdrücken durchgeführt werden. Dadurch gibt es keine festgeschriebene Syntax für die Markup-Sprache. Sie definiert sich durch die Implementierung des originalen Übersetzers. Dieser ist kulant, was die Eingaben der Autoren betrifft, und versucht oft, trotz ungenauer Angaben das gewünschte Ergebnis zu erzeugen. Aus diesen Gründen musste die verwendete Parser-Bibliothek zum Teil durch Reverse Engineering entwickelt werden und trifft immer wieder auf Stellen, die aufgrund von Syntaxfehlern nicht analysiert werden können.

Die Bibliothek kann in zwei verschiedenen Modi genutzt werden, wobei der erste versucht, möglichst korrekt zu sein. Der zweite versucht, für möglichst viele Artikel ein Ergebnis zu liefern. In dieser Arbeit ist es wichtiger, auf möglichst alle Artikel zugreifen zu können, als eine möglichst korrekte Darstellung zu erhalten, die dann gegebenenfalls ohnehin nicht möglich ist.

Aus diesem Grund wurde entschieden, in dieser Arbeit den zweiten Modus zu verwenden. Die Bibliothek kann fast alle Artikel aus dem verwendeten Dump erfolgreich analysieren (Abschnitt 5.1.1).

Die Funktionalität der Parser-Bibliothek kann durch das Schreiben eigener Erweiterungen ergänzt werden (Abschnitt 2.5.2).

12. <http://www.postgresql.org/>

13. enwiki-20100130-pages-meta-current.xml.bz2

2.5.1 AST

Anhand eines Beispiels soll gezeigt werden, wie der Syntaxbaum für einen Wikitext aussieht. In Listing 2.1 ist ein kurzer Wikitext gegeben, der als Grundlage dienen soll. Er besteht aus einem einleitenden Absatz, einer Überschrift erster Ebene und einer Überschrift zweiter Ebene. Im Wikitext werden die Überschriften durch mehrere Gleichheitszeichen markiert. Zwei Gleichheitszeichen markieren dabei eine Überschrift ersten Grades und drei Gleichheitszeichen eine Überschrift dritten Grades und so weiter. Unter jeder Überschrift ist wiederum ein Absatz, dessen Ende durch eine oder mehrere Leerzeilen markiert ist. Die Wörter *Erlangen* im ersten Absatz und *Bayreuth* im letzten Absatz sollen fett geschrieben werden. Dafür werden sie von jeweils drei Hochkommata (Ticks) umschlossen. Der Artikel enthält auch einen internen Link, markiert durch doppelte eckige Klammern, zum Artikel *Germany*.

Der Parser durchläuft nun den Wikitext und bildet seine Struktur auf einen AST ab (Abbildungen 2.4 bis 2.6). Die Elemente des Wikitextes werden dafür in Knoten des AST abgebildet. Für jedes mögliche Element aus dem Wikitext gibt es einen entsprechenden AST-Knotentyp. Der Baum beginnt immer mit dem Wurzelknoten *Page*, der wiederum über genau einen Kindknoten *Section* verfügt. Dieser bildet den Umstand ab, dass in den gerenderten HTML-Seiten der Wikipedia der Artikelname als oberste Überschrift – noch über der ersten Überschrift des Wikitextes – erscheint.

Jedem Knoten können Attribute angefügt werden. Schon nach dem initialen Parsen verfügt zum Beispiel jeder *Text*-Knoten über ein *Content*-Attribut, das den eigentlichen Textinhalt dieses Knotens enthält. In den Abbildungen werden diese Attribute durch kursive Schreibweise hervorgehoben.

Listing 2.1: Einfacher Wikitext (Beispiel)

```
'''Erlangen''' is a city in [[Germany]].  
  
== History ==  
  
Erlangen was first mentioned in 1002.  
  
=== University ===  
  
The University was founded in 1742 in '''Bayreuth'''.
```

Abbildung 2.4: Beispiel für einen AST (1/3)

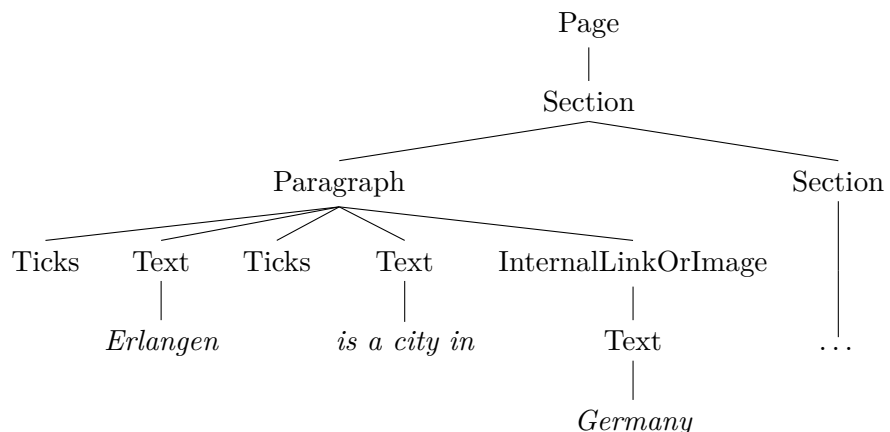


Abbildung 2.5: Beispiel
für einen AST (2/3)

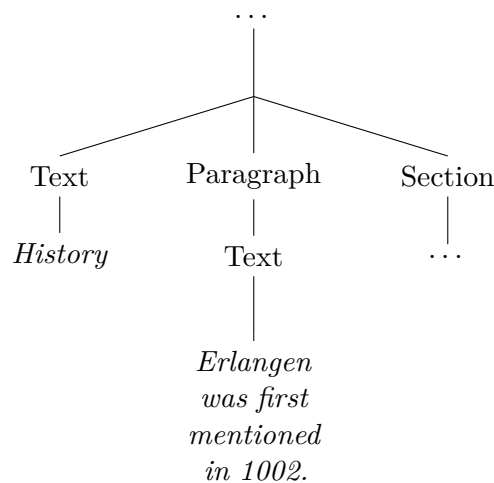
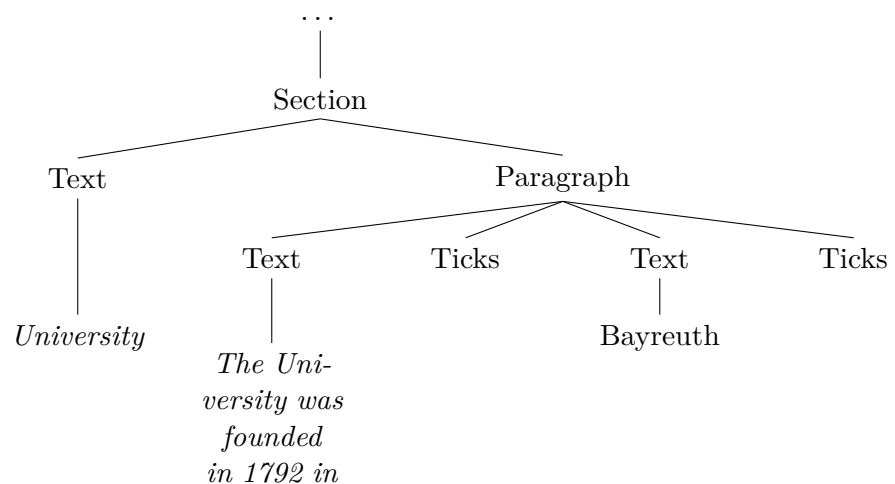


Abbildung 2.6: Beispiel
für einen AST (3/3)



Die erste Section hat in diesem Beispiel zwei Kindknoten. Das ist zum einen der erste Abschnitt (*Paragraph*) des Artikels und zum anderen eine untergeordnete Überschrift. Der Paragraph enthält die Kindknoten *Ticks*, *Text*, *Ticks* und *InternalLinkOrImage*. Der Kindknoten von *InternalLinkOrImage* ist ein *Text*-Knoten und enthält als Attribut den eigentlichen Textinhalt des Knotens. Da in diesem Fall der Anzeigetext mit dem Linkziel übereinstimmt, gibt es keinen weiteren Knoten, der das Linkziel enthält. Der Knoten heißt übrigens *InternalLinkOrImage*, da interne Links auf die gleiche Weise ausgedrückt werden, wie das Einbinden von Bildern. Um hier eine Unterscheidung zwischen Link und Bild zu treffen, müsste auch das Ziel untersucht werden. Dies würde aber über die Analyse des Wikitextes eines Artikels hinausgehen. Komplexere Artikel können schnell mehrere tausend Knoten enthalten. Die hier gewählte Darstellungsweise kann deswegen schnell unübersichtlich werden.

Die folgenden Knotentypen sind in dieser Arbeit am wichtigsten:

- Page
- Section
- Paragraph
- Text
- InternalLinkOrImage
- AstNode

AstNode ist der Knotentyp, von dem alle anderen Knoten abgeleitet sind.

2.5.2 Visitors

Visitor (deutsch: Besucher) ist ein Entwurfsmuster. Er dient dazu, einen Algorithmus unabhängig von der konkreten Datenstruktur, auf der er arbeitet, zu entwerfen.

Konkret kann in diesem Fall die Funktionalität der Parser-Bibliothek durch das Schreiben eigener Visitors erweitert werden. Die Visitors können die Struktur des AST verändern oder einfach nur den Knoten neue Attribute hinzufügen. Ein neuer Visitor wird durch das Schreiben einer Java-Klasse, die von der Klasse Visitor (definiert innerhalb der Bibliothek) abgeleitet wurde, erstellt. Diese Klasse kann eine oder mehrere visit-Methoden enthalten, die für bestimmte Ast-Knoten aufgerufen werden.

Alle Knotentypen sind von der Oberklasse AstNode abgeleitet. Wenn eine bestimmte Operation unabhängig vom konkreten Knotentyp ist, kann eine visit-Methode für den Knotentyp AstNode geschrieben werden. Für alle Knotentypen, für die keine eigene visit-Methode vorhanden ist, wird nun diese Standard-Methode aufgerufen.

In dieser Arbeit werden Visitors eingesetzt, um den Text aus den Wikipedia-Artikeln zu extrahieren (Abschnitt 4.1.2) oder die Artikel in Abschnitte aufzuteilen (Abschnitt 4.2).

2.6 Lucene

Lucene ist eine mächtige Java-Bibliothek, mit deren Hilfe Suchanwendungen geschrieben werden können. Im Open-Source-Bereich ist sie die am weitesten verbreitete. So basieren zum Beispiel die Volltextsuche der Wikipedia und einige Open-Source-Desktop-Suchmaschinen auf Lucene. Neben der Standard-Version in Java gibt es auch Portierungen für C++, C#, Ruby, Perl, Python, PHP und andere.

Aufgrund ihrer Stabilität, Skalierbarkeit und nicht zuletzt der unternehmensfreundlichen Apache-Lizenz wird sie auch in kommerziellen Projekten eingesetzt¹⁴.

Lucene stellt eine Schnittstelle zur Verfügung, um Textdokumente zu indizieren und den dabei entstandenen Index wiederum zu durchsuchen. Unter der Oberfläche nutzt Lucene moderne Techniken, um sowohl mit Qualität als auch Geschwindigkeit zu überzeugen (McCandless et al., 2010, Kapitel 1).

Das Erstellen der Index-Dokumente, das Wählen verschiedener Parameter und die genaue Festlegung, wie gesucht werden soll, muss dabei in der Anwendung, die Lucene nutzt, festgelegt werden.

2.6.1 Index

Ein Lucene-Index besteht aus einer Menge von Dokumenten. Der Kern der Dokumente ist stets der zu durchsuchende Inhalt. Lucene selbst kennt nur einfachen Text. In den meisten Anwendungsfällen liegen die Ausgangsdokumente nicht in einfachen Textdateien vor, sondern enthalten wie im Falle von XML oder Wikitext, für die Indizierung störende Markup-Daten. Es ist also die Aufgabe der Suchanwendung, die Da-

14. <http://www.lucidimagination.com/>

ten zunächst in einer geeigneten Form zur Verfügung zu stellen. Wie die Wikipedia-Artikel für das Indizieren mithilfe des Wikitech-Parsers (Abschnitt 2.5) aufbereitet werden, wird in Abschnitt 4.1.1 beschrieben.

Tika

Tika, als Unterprojekt von Lucene entstanden, stellt eine einfache Möglichkeit dar, aus verschiedenen Dokumententypen wie HTML und XML, oder sogar Office- und PDF-Dokumenten¹⁵, den Text zu extrahieren (McCandless et al., 2010, Kapitel 7). Tika ermöglicht es dem während dieser Arbeit entwickelten System, ähnliche Wikipedia-Einträge zu allen von Tika unterstützten Dokumenten zu suchen.

Filtern und Analysieren

Nachdem aus einem zu indizierenden Dokument einfacher Text extrahiert wurde, muss er zunächst in eine zum Indizieren geeignete Form überführt werden. Dieser Vorgang wird bei Lucene als Analysieren bezeichnet. Dafür werden Klassen genutzt, die von der Basisklasse *Analyzer* abgeleitet wurden.

Der Text wird dafür in eine Sequenz von Token zerlegt. Grob lässt sich sagen, dass jede Wortform in dem Text ein Token ist. Bei diesem wichtigen Schritt ist es zum Beispiel auch möglich, die sogenannten Stopp-Wörter herauszufiltern. Des Weiteren können die Token auch nach Kriterien wie der Länge gefiltert werden. Die hierbei herausgefilterten Token können später bei der Suche natürlich nicht mehr gefunden werden. Die Filterung beeinflusst möglicherweise auch die Ordnung der Suchergebnisse. Da Lucene bei einer Suche aber allen Token Relevanzwerte zuordnet, ist dieses Problem zu vernachlässigen. Die gefilterten Wörter hätten ohnehin sehr niedrige Relevanzwerte. Der größte Vorteil der Filterung ist eine geringere Größe des Indexes und eine höhere Suchgeschwindigkeit.

Anschließend können diese Token entweder direkt in den Index geschrieben oder noch zuvor bearbeitet werden. Normalerweise werden zumindest noch alle Großbuchstaben zu Kleinbuchstaben umgewandelt, damit später die Suche unabhängig von Groß- und Kleinschreibung funktioniert (McCandless et al., 2010).

Durch Stemming können die Token auf ihre Grundform zurückgeführt werden (Abschnitt 2.1.1).

Indizieren

Um die Quelldokumente zu indizieren, muss ein Lucene-konformes Dokument daraus erstellt werden. Lucene stellt dafür die Klasse *Document* zur Verfügung. Jedes Dokument besteht wiederum aus einem oder mehreren Feldern (Klasse *Field*), die Text enthalten. Häufig wird der Titel des Dokumentes in einem vom Inhalt separaten Feld gespeichert. Weitere Kandidaten für eigene Felder sind Zeiger (zum Beispiel eine URL oder ein Dateiname) auf das ursprüngliche Dokument, Metadaten und gegebenenfalls sogar das ursprüngliche Dokument selbst.

15. PDF (Portable Document Format, http://www.adobe.com/devnet/pdf/pdf_reference.html) wurde für den plattformunabhängigen Austausch von Dokumenten entwickelt.

Die Entscheidung, wie die Dokumente im Index aufgebaut sein sollen, ist vor allem für die spätere Suche wichtig. Es besteht die Möglichkeit, eine Suche nur auf bestimmte Felder oder sogar nur ein einziges Feld zu beschränken. Des Weiteren muss die Anwendung gegebenenfalls Informationen bereithalten, die bei den Suchergebnissen mitgeliefert werden sollen. Hier steht der Entwickler zum Beispiel vor der Entscheidung, statt den Titel des Dokumentes mit abzuspeichern, nur einen eindeutigen Zahlenwert, der auf einen Eintrag in einer SQL-Datenbank verweist, in das Dokument aufzunehmen. Oft ist es allerdings gewünscht, dass Anwender auch nach dem Titel selbst suchen können. Die Suchanfrage kann dafür entsprechend eingegrenzt werden. Hierzu ist das Speichern des Titels in einem separaten Feld obligatorisch.

Für jedes Feld kann einzeln angegeben werden, welche Filter angewandt werden und wie der Text analysiert werden soll. Für die oben angesprochene Suche nach dem Titel, soll dieser oft unverändert abgespeichert werden, so dass die Suche nur genau das Dokument mit dem exakten Titel findet. Der Hauptinhalt dagegen soll zerlegt, gefiltert und analysiert werden, um eine flexible Suche zu ermöglichen.

Metadaten wie eine URL zum ursprünglichen Dokument müssen natürlich immer in der originalen Form abgespeichert werden. Häufig werden auch Metadaten gespeichert, die erst während der Analyse des Dokumentes gewonnen werden. Das Dateiformat, die Kodierung oder die Textlänge sind Beispiele dafür. In manchen Umgebungen kann es auch notwendig sein, Zugriffsberechtigungen anzugeben. Dokumente, auf die bestimmte Anwender keinen Zugriff haben, können dann durch Lucene automatisch von den Suchergebnissen ausgeschlossen werden.

Ein Spezialfall sind Felder, die mehrmals im Dokument vorkommen können: sogenannte *Multi-Value-Fields*. Sie bieten sich zum Beispiel an, um mehrere Autoren eines Dokumentes auf dieselbe Weise, aber getrennt voneinander, abzuspeichern.

Um ein Dokument letztendlich dem Index hinzuzufügen, ist nur der Aufruf einer einzigen Lucene-Methode notwendig. Dabei muss zuvor lediglich angegeben werden, wo auf der Festplatte (oder im Arbeitsspeicher) der Index abgelegt werden soll.

Die indizierten Dokumente stehen aber erst nach dem Aufruf einer commit-Methode für die Suche bereit. Diese Methode veranlasst Lucene, die neuen Dokumente zu einem sogenannten Segment zusammenzufügen und an den eigentlichen Index anzuhängen. Da häufige Commits den Index stark fragmentieren, muss die Anwendung entscheiden, wann ein Commit durchgeführt werden soll. Die Commits stellen zudem sicher, dass der Index immer konsistent ist. In der Folge ist es beim Indizieren einer großen Menge von Dokumenten sinnvoll, nicht nach jedem Dokument einen Commit durchzuführen. Andererseits sind alle bereits indizierten Dokumente, für die noch kein Commit durchgeführt wurde, verloren, falls zum Beispiel das Programm durch ein Netzwerkproblem nicht mehr auf die Quelldokumente zugreifen kann und das Indizieren mit einem Fehler abbricht.

Ein zu stark fragmentierter Index kann sich negativ auf die Suchzeiten auswirken. Lucene stellt deshalb eine Möglichkeit bereit, den Index zu optimieren. Die Optimierung eines Indexes kann sehr viel Zeit in Anspruch nehmen und benötigt temporär ein Vielfaches des Speicherplatzes des Indexes.

Lucene nutzt einen invertierten Index, wie er in Abschnitt 2.1.2 vereinfacht beschrieben wurde. Es speichert dabei alle Daten, die für das Vektorraum-Modell notwendig sind (McCandless et al., 2010, Kapitel 2).

Da Wikitech Classify von den Einzelheiten des Indizierens abstrahiert, soll an dieser Stelle nicht weiter darauf eingegangen werden. In Abschnitt 4.1.1 wird beschrieben, wie Wikipedia-Artikel mithilfe des in dieser Arbeit entwickelten Systems indiziert werden.

2.6.2 Suche

Lucene stellt Klassen zur Verfügung, mit deren Hilfe eine Suchanfrage abgebildet werden kann. Einzelne Bedingungen, zum Beispiel welche Wörter enthalten sein sollen und welche ausgeschlossen werden sollen, boolesche Operatoren und Einschränkungen auf bestimmte Dokumente und Felder müssen zu einer einzigen Suchanfrage zusammengefügt werden.

Diese Suchanfrage wird an Lucene übergeben, die dann den Index nach relevanten Dokumenten durchsucht. Dabei wird der Suchraum zunächst durch eine boolesche Suche eingeschränkt. Für die verbliebenen Dokumente wird mithilfe des Vektorraum-Modells ein normalisierter Ähnlichkeitswert berechnet. Schließlich liefert Lucene eine Liste der Dokumente, sortiert nach Ähnlichkeit, zurück. Der Ähnlichkeitswert ist unabhängig von den anderen Dokumenten, die gefunden wurden, und wird für jedes Dokument einzeln berechnet. Höhere Werte drücken dabei eine größere Ähnlichkeit aus. Die Berechnung des Ähnlichkeitswertes basiert auf der Formel, die in Abschnitt 2.1.3 für das Vektorraum-Modell angegeben wurde. Lucene beachtet noch weitere Faktoren, die aber in dieser Arbeit keine Rolle spielen.

Anfragen können, wie oben angedeutet, durch das Kombinieren von Teilanfragen zusammengesetzt werden. Lucene stellt auch die Möglichkeit zur Verfügung, Suchanfragen als Text in einer bestimmten Syntax auszudrücken. Beide Möglichkeiten können wiederum verbunden werden, um zum Beispiel die Texteingabe von Nutzern mit weiteren Einstellungen zu kombinieren. Wie Wikitech Classify Wikipedia-Artikel und Textdokumente in eine Lucene-Suchanfrage überführt, wird in Abschnitt 4.1.2 beschrieben.

Im Normalfall sollen die Ergebnisse ausgegeben werden. Hierzu muss in der Anwendung festgelegt sein, welche Felder zurückgeliefert werden sollen. Der Titel des Dokumentes, die ersten Zeilen des Inhalts und ein Verweis zum Ursprungsdokument sind Kandidaten dafür (McCandless et al., 2010, Kapitel 3).

Alternativ können die Ergebnisse auch intern weiterverarbeitet werden. Bei der Kategorisierung nutzt Wikitech Classify zum Beispiel die gefundenen Dokumente, um Kategorien zu finden, statt die Ergebnisse direkt auszugeben.

3

Relevante Arbeiten

Das wissenschaftliche Interesse an der Wikipedia ist seit ihrer Gründung stetig gestiegen. Die Arbeiten reichen dabei von Untersuchungen über die Wikipedia selbst und Versuchen, bestehende Systeme mithilfe der Wikipedia zu erweitern, bis hin zu Bestrebungen, die Wikipedia selbst zu verbessern.

Die Tabelle 3.1 listet die Anzahl der Suchergebnisse von Google Scholar¹ für Arbeiten, die das Wort Wikipedia im Titel beinhalten. Dabei werden nur die im jeweiligen Jahr erschienenen Arbeiten gezählt. Aus der Tabelle kann abgelesen werden, dass die Anzahl der neuen Veröffentlichungen jedes Jahr zugenommen hat. Es ist davon auszugehen, dass die Zahl für 2010 noch ansteigen wird². Bei der Betrachtung der absoluten Zahlen ist zu beachten, dass Google nicht auf alle Publikationen Zugriff hat und viele relevante Arbeiten das Wort Wikipedia nicht im Titel aufführen³.

Die folgenden Abschnitte stellen eine Auswahl von Arbeiten vor, die mit dem hier bearbeiteten Thema im Zusammenhang stehen. Eine Verbesserung der Qualität der Wikipedia ist das übergeordnete Ziel dieser Arbeit. Sie wird in einer Vielzahl wissenschaftlicher Arbeiten aufgegriffen. Einige Arbeiten beschäftigen sich mit der semantischen Nähe von Konzepten der Wikipedia, die auch als Ähnlichkeitswert angesehen werden kann. Die Klassifikation von Textinhalten der Wikipedia selbst fand bisher weniger Beachtung.

Die Arbeiten von Voss (2005), Medelyan et al. (2009) und Strube (2010) geben einen weiter reichenden Überblick zur Forschung im Umfeld der Wikipedia. Nicht zuletzt wird in der Wikipedia eine Liste mit relevanten wissenschaftlichen Arbeiten gepflegt⁴.

Tabelle 3.1: Anzahl neuer Arbeiten im jeweiligen Jahr (mit Wikipedia im Titel, lt. Google-Scholar am 31.01.2010).

Jahr	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
Artikel	3	3	4	9	40	131	185	296	340	316

1. <http://scholar.google.de/>

2. Die Zahl ist noch im Januar 2011 innerhalb von 2 Wochen um 18 angestiegen.

3. Als Beispiel sei „Measuring qualities of articles contributed by online communities“ ((Lim et al., 2006)) genannt.

4. http://en.wikipedia.org/wiki/Wikipedia:Wikipedia_in_academic_studies

3.1 Qualität der Wikipedia

Wie bereits in 2.4 angerissen, gibt es für die Leser der Wikipedia nur zwei zuverlässige Merkmale, um die Qualität von Artikeln zu erkennen. Exzellente und lesenswerte Artikel werden durch einzelne oder mehrere Nutzer vorgeschlagen. Die tatsächliche Auszeichnung wird aber erst nach einem Abstimmungsprozess verliehen. Allerdings werden nur sehr wenige Artikel ausgezeichnet (Wikipedia contributors (2011b) und Wikipedia contributors (2011c)).

Eine allgemeine Diskussion über die Qualität der Wikipedia wird in Hammwöhner (2007) und Fuchs et al. (2007) vorgenommen. Sie zeigen zum Beispiel, dass viele Kategorien zu allgemein andere hingegen zu spezifisch sind und darüber hinaus wichtige Kategorien fehlen. Wegen der losen Struktur wurde das Categoriesystem auch als willkürlich, unvollständig, inkonsistent und voller Redundanzen kritisiert (Chernov et al., 2006). Auch dieses Thema ist in der vorliegenden Arbeit relevant und wird später nochmals aufgegriffen (Abschnitt 6.3).

Lorenz und Türp (2010) unterteilten eine Auswahl von über 200 Artikeln aus dem Bereich Zahnmedizin durch manuelle Auswertung in drei Gruppen: (i) lehrbuchtauglich, (ii) bedingt lehrbuchtauglich und (iii) nicht lehrbuchtauglich. Für alle Artikel wurden dafür unter anderem Statistiken über Rechtschreibung und Ausdruck erstellt. Es zeigt sich, dass diese beiden Kriterien zwar zur Unterscheidung zwischen den Gruppen (ii) und (iii) dienen können, allerdings nicht zuverlässig sind.

Nielsen et al. (2007) zeigten die Zunahme von wissenschaftlichen Zitaten in der Wikipedia allgemein auf. Es ist anzunehmen, dass Artikel, deren Inhalt durch wissenschaftliche Zitate belegt ist, durchschnittlich eine höhere Qualität aufweisen. Ob dies tatsächlich so ist, wurde allerdings nicht untersucht.

In einer der ersten Arbeiten zur Qualität der Wikipedia untersuchte Lih (2003) unter anderem die Anzahl der Links, die von außerhalb auf die Wikipedia verweisen. Neben anderen Metriken werden in Stvilia et al. (2005b) auch die Diskussionsseiten der Artikel in Betracht gezogen. Druck et al. (2008) versuchten, die Qualität einzelner Änderungen in Wikipedia-Artikeln zu bewerten. Das System von Adler et al. (2008) ist im Ansatz ähnlich zu anderen bereits hier vorgestellten. Es kann die Qualität jedoch anhand verschiedener Farben für einzelne Textfragmente leicht nachvollziehbar machen. Hu et al. (2007b) haben einen Ansatz vorgeschlagen, wie die Wikipedia-Suche unter Berücksichtigung der Artikelqualität verbessert werden könnte.

Trotz der zum Teil positiven Untersuchungen werden in wissenschaftlichen Arbeiten Wikipedia-Belege nach wie vor nicht akzeptiert. Das liegt allerdings auch daran, dass die Wikipedia in ihrer Eigenschaft als Enzyklopädie keine Primärquelle ist. Sie kann jedoch als Einstiegspunkt für eine genauere Recherche genutzt werden (Lorenz, 2011).

Es gibt eine Reihe früherer Arbeiten, die sich mit der Qualität kollaborativ erstellter Inhalte auseinandersetzen. Schon vor der Zeit der Wikipedia wurde versucht, die Qualität von Internetseiten automatisch zu evaluieren. Motro und Rakov (1998) stellen Metriken für die Qualitätsmessung von Datenbankinhalten und Zhu und Gauch (2000) solche für Webseiten vor.

In zahlreichen Arbeiten wurden Vorschläge entwickelt, um speziell die Qualität von Wikipediaartikeln anhand einfacher Metriken abzuschätzen. Hu et al. (2007a) schlagen mehrere Metriken vor, die mit qualitativ hochwertigen Artikeln korrelieren sollen. Die einfachste vorgeschlagene Metrik (*Basic*) misst die Anerkennung der Autoren. Die Annahme ist, dass Artikel mit einem hohen Anteil anerkannter Autoren eine höhere Qualität aufweisen. Dabei werden anerkannte Autoren darüber definiert, dass sie besonders häufig an lesenswerten oder exzellenten Artikeln mitgearbeitet haben.

Die Metrik *PeerReview* basiert auf der Annahme, dass die trotz Änderungen am Artikel selbst nicht veränderten Textabschnitte indirekt vom Autor bestätigt wurden. Erfolgt diese Bestätigung durch Autoren mit einer hohen Anerkennung, sind diese Abschnitte von hoher Qualität. Diese Methode wurde in *ProbReview* noch bis auf Wortebene verfeinert. Nicht als Metrik sondern als Vergleichswert wird die Textlänge der Artikel (*Naive*) vorgeschlagen. Dabei wird angenommen, dass mit zunehmender Artikellänge auch die Qualität der Artikel steigt.

Das Ergebnis ist, dass nur *PeerReview* und *ProbReview* ein wenig besser zur Beurteilung der Artikelqualität geeignet sind als *Naive*. *Basic* ist sogar schlechter.

Schon zuvor wurden von Stvilia et al. (2005a) eine Reihe Metriken vorgeschlagen, die besser mit der Artikelqualität korrelieren. Anders als bei Hu et al. (2007a) wird nicht nur die Änderungsgeschichte der Artikel untersucht, sondern auch der Inhalt selbst. Die Metriken auf Basis der Bearbeitungsgeschichte setzen sich aus Werten wie Anzahl der Autoren, Bearbeitungen oder abgelehnten Änderungen, Aktualität (Zeit seit der letzten Änderung) und Änderungshäufigkeit zusammen. Auch wurden Metriken auf Basis der Autoren berechnet (Schrieben die Autoren auch in anderen Artikeln? Wie hoch ist der Anteil von Administratoren an den Artikeln?). Bezüglich des Inhalts des Artikels wurden Metriken wie Anzahl der internen und externen Links, Anzahl von Bildern und die Anzahl der verwaisten Links verwendet. In der Auswertung wurde aufgezeigt, dass die Metriken in der Lage sind, die von den Nutzern ausgezeichneten Artikel von anderen, zufällig ausgewählten Artikeln, zu unterscheiden.

Stein und Hess (2007) untersuchten, ob die Qualität und Quantität der Autoren der Artikel mit der Artikelqualität korrelieren. Sie teilten die Autoren dafür in gute Autoren und andere Autoren ein. Im Ergebnis lässt sich von der Qualität, nicht aber von der Quantität der Autoren auf die Artikelqualität schließen.

Erst Blumenstock (2008) wertete tatsächlich die Leistungsfähigkeit der einfachsten Metrik aus: der Artikellänge. Es wird gezeigt, dass diese einzelne Metrik zum einen ausgezeichnete Artikel sehr gut von nicht ausgezeichneten Artikeln unterscheidet und zum anderen besser als die Metriken aus vorherigen Arbeiten ist.

Zusammenfassend kann gesagt werden, dass eine automatische Messung der Artikelqualität nicht zuverlässig möglich ist. Die vorgeschlagenen Metriken können aber durchaus Hinweise geben.

Halavais und Lackaff (2008) zeigten, dass verschiedene Themenbereiche entgegen anderer Behauptungen doch relativ ausgeglichen behandelt werden. Dazu verglichen sie den Inhalt der Wikipedia zum Beispiel mit dem in anderen Enzyklopädien. Daraus kann geschlossen werden, dass sich die Daten der Wikipedia domänenunabhängig nutzen lassen.

Die Inhalte der Wikipedia werden in verschiedenen Forschungsfeldern verwendet. In mehreren Arbeiten wurden speziell aufbereitete Korpora aus den Daten der Wikipedia erstellt, um diese für Forschungszwecke nutzbar zu machen. Dabei ging es vor allem darum, einfacher auf die Daten zugreifen zu können, indem sie in ein XML-Format umgewandelt werden. Beispiele sind *Wikipedia XML Corpus* (Denoyer und Gallinari, 2006), *WikiPrep*⁵ (Gabrilovich und Markovitch, 2007) und *WikiXML*⁶. Zesch et al. (2008) stellen eine objektorientierte Bibliothek für die Arbeit mit Wikipedia-Daten, die dafür in einer Datenbank gespeichert werden, zur Verfügung. Der in Abschnitt 2.5 vorgestellte Parser für den MediaWiki-Wikitext stellt zwar selbst kein Korpus bereit, seine Funktionalität geht aber weit über die hier aufgeführten Systeme hinaus. So ist es beispielsweise möglich, auf die Struktur der Artikel zuzugreifen.

Die Arbeiten über die Qualität stellen den größten Anteil der Arbeiten über die Wikipedia. Daraus kann geschlossen werden, dass dieses Thema von besonders großem Interesse ist.

3.2 Semantische Nähe und Textklassifikation

Die Ähnlichkeit zweier Konzepte wird auch als semantische Nähe (englisch *semantic relatedness*) bezeichnet. Es gibt eine Reihe von Arbeiten, in denen mithilfe der Wikipedia die semantische Nähe zwischen zwei Begriffen ermittelt werden sollte. Strube und Ponzetto (2006) nutzten dafür die Links und Weiterleitungsseiten innerhalb der Wikipedia. Ponzetto und Strube (2007) entwickelten auf Basis der Wikipedia-Kategorien eine Taxonomie⁷. Sie verglichen ihre Ergebnisse unter anderem mit den Werten von WordNet (Fellbaum, 1998), welches manuell erstellt wurde und konnten zeigen, dass ihre automatische Methode vergleichbare Ergebnisse liefert. Die semantische Nähe zweier Konzepte kann auch als Ähnlichkeitswert der Artikel, die diese Konzepte beschreiben, angesehen werden.

Mihalcea und Csomai (2007) bauten darauf auf und machten das Verfahren für Wikipedia selbst nutzbar. Durch die Kenntnis der eigentlichen Bedeutung von Wörtern und Phrasen können automatisierte Vorschläge für Links auf andere Wikipediaartikel erbracht werden.

Die in den Arbeiten vorgestellten Ansätze sind jedoch nur bedingt für das Finden ähnlicher Textstellen zu neuen Texten geeignet. Gerade neue Textstellen werden anfangs noch schlecht mit anderen Artikeln verlinkt sein.

5. <http://www.cs.technion.ac.il/~gabr/resources/code/wikiprep/>

6. <http://ilps.science.uva.nl/WikiXML/>

7. „[Eine Taxonomie ist ein] Terminus zur Bezeichnung von sprachwissenschaftlichen Analysen, die sich überwiegend oder ausschließlich auf klassifikatorische Merkmale oder Distribution von sprachlichen Einheiten, Merkmalen oder Strukturen stützen.“ (Bußmann, 2002, S. 680)

Die Inhalte der Wikipedia und ihre Kategorie-Hierarchie wurden in mehreren Arbeiten genutzt, um die Leistungsfähigkeit von Anwendungen wie Textklassifikationssystemen zu verbessern. Wang et al. (2007) durchsuchten die Wikipedia nach Synonymen und Oberbegriffen, um die ursprünglichen Dokumente damit anzureichern. Sie konnten bessere Ergebnisse erreichen als vergleichbare Systeme, bei denen die Dokumente nicht durch Informationen aus der Wikipedia angereichert wurden. Die zugrundeliegende Idee wurde bereits von Gabrilovich und Markovitch (2006) verwendet. Dort wird auch darauf hingewiesen, dass insbesondere die Klassifikation von sehr kurzen Dokumenten verbessert werden kann.

Gantner und Schmidt-Thieme (2009) versuchten dagegen, die Textklassifikation für die Wikipedia selbst zu nutzen. Ziel war es, die Kategorien der Artikel anhand ihres Inhalts vorherzusagen. Dazu griffen sie auf LIBSVM (Chang und Lin, 2001) zurück. Die Entscheidung, ob eine Kategorie zu einem Dokument zugewiesen wird, ist ein binäres Klassifikationsproblem. Für jede Kategorie muss somit jeweils ein Klassifizierer trainiert werden. Im Gegensatz zu Klassifizierungsmethoden, bei denen jede Kategorie mit jeder anderen verglichen werden muss, steigt der Arbeitsaufwand hier immerhin nur linear.

Um die grundsätzliche Funktionsweise zu demonstrieren, wurde das System auf zwei Teilmengen der deutschsprachigen Wikipedia evaluiert. Die erste besteht aus allen Artikeln, die in der Kategorie *Eishockey* und deren Unterkategorien vorkommen. Kategorien, die nicht selbst Unterkategorien von Eishockey sind, wurden ausgeschlossen. Von den ursprünglich 886 000 Artikeln und 64 000 Kategorien der Deutschen Wikipedia von Ende 2008 blieben somit nur noch 5 037 Artikel und 159 Kategorien übrig. In einem weiteren Versuch wurden alle Artikel und Kategorien unterhalb der Kategorie *Philosoph* herangezogen. Hier reduzierte sich die Anzahl der Artikel auf 2 445 und die Anzahl der Kategorien auf 55.

Die Ergebniswerte sind besser als die Werte, die ein statischer Klassifizierer liefert, der einfach immer die häufigsten Kategorien zuweist. Leider werden keine Angaben über die Geschwindigkeit gemacht. Allgemein skaliert LIBSVM aber nicht für größere Datenmengen⁸. Zudem müssen die Klassifizierer bei Änderungen am Korpus neu trainiert werden, was sich bei der großen Datenmenge und der Häufigkeit der Änderungen der Wikipedia im praktischen Einsatz sehr negativ auswirken dürfte. Die Ergebnisse der Arbeit von Gantner und Schmidt-Thieme (2009) werden in Abschnitt 5.8 mit den Ergebnissen dieser Arbeit verglichen.

Müller und Gurevych (2008) versuchten, die Leistungsfähigkeit von Information-Retrieval-Systemen durch die Nutzung semantischen Wissens zu verbessern. Neue und aufwendige Methoden wie die Beschreibung der Bedeutung eines Dokumentes durch zuvor aus Wikipedia abgeleitete Konzepte (Gabrilovich und Markovitch, 2007), wurden mit dem vergleichsweise einfachem statistischen Modell von Lucene verglichen. Die Ergebnisse zeigen, dass diese Methoden den Lucene-Ansatz (noch) nicht erreichen konnten. Allerdings könne die Leistungsfähigkeit des Lucene-Ansatzes durch eine Kombination mit den neuen Methoden verbessert werden. Dieser Ansatz ist eventuell auch interessant, um das in dieser Arbeit entwickelte System zu erweitern.

8. <http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html>

4

Wikitech Classify

In diesem Kapitel soll das im Rahmen dieser Arbeit entwickelte System vorgestellt werden. Sein Name ist *Wikitech Classify*.

Wikitech Classify wurde als Java-Bibliothek entwickelt. Zusätzlich stellt es aber auch ein einfaches CLI (Command Line Interface) zur Verfügung. Dieses CLI kann über das Shell-Skript `classify` aufgerufen werden. Es stellt für die verschiedenen Anwendungsmöglichkeiten von Wikitech Classify sogenannte Aufgaben (`index`, `similar`, `categories`, ...) zur Verfügung. Der Name der Aufgabe muss als erstes Argument übergeben werden. Die Aufgaben werden durch verschiedene Schalter und Argumente konfiguriert. Im Anhang A befindet sich eine Bedienungsanleitung.

Im Mittelpunkt dieses Kapitels stehen die Vorgehensweise und die Möglichkeiten der Konfiguration. Auf eine detaillierte Beschreibung der Implementation wird deswegen verzichtet. Stattdessen werden nur die wichtigsten Klassen und Methoden vorgestellt, die ein auf Wikitech Classify aufbauendes System nutzen kann.

Das System kann grob in zwei Hauptaufgaben unterteilt werden. Zum einen das Finden der ähnlichen Artikel und Abschnitte und zum anderen die darauf aufbauende Kategorisierung. Die Suche nach ähnlichen Artikeln wird in Abschnitt 4.1 und die Suche nach ähnlichen Abschnitten unter 4.2 beschrieben. In Abschnitt 4.3 wird beschrieben, wie die Klassifikation funktioniert. Unter 4.4 wird dargestellt, wie Wikitech Classify auch für eine Volltextsuche genutzt werden kann.

4.1 Finden ähnlicher Artikel

Die Suche nach ähnlichen Artikeln funktioniert auf Basis eines Volltextindexes. Bevor Wikitech Classify für die Suche genutzt werden kann, müssen zunächst alle Artikel in den Index geschrieben werden. Anschließend kann das System Anfragen entgegennehmen und eine Liste der ähnlichen Artikel liefern.

4.1.1 Aufbau des Indexes

Eine Volltextsuchmaschine soll als Ergebnisse für eine Suchanfrage eine Liste von Dokumenten liefern, die für die Suchanfrage relevant sind. Dafür müssen die zu durchsuchenden Daten zunächst in den Index der Suchmaschine aufgenommen werden.

Datenquelle

Wikitech Classify holt sich alle Daten, die für den Aufbau des Indexes benötigt werden, selbst aus einer zuvor durch den Anwender konfigurierten Quelle. Um einen oder mehrere Artikel in den Index zu schreiben, müssen dann nur noch deren Namen (Titel) angegeben werden.

Das System stellt die in Tabelle 4.1 aufgelisteten Datenquellen zur Verfügung. `NodeSourceOnline` holt Artikel aus einem beliebigen MediaWiki über HTTP¹. Dabei wird auf die MediaWiki-API² zurückgegriffen. Zur Konfiguration muss lediglich die URL³ festgelegt werden. `NodeSourceWikipediaEN` und `NodeSourceWikipediaDE` sind einfache Erweiterungen von `NodeSourceOnline`, bei denen die URL auf die jeweilige API der englischsprachigen Wikipedia beziehungsweise der deutschsprachigen Wikipedia festgelegt ist.

Der Index, eigentlich für die Volltextsuche erstellt, enthält alle wichtigen Daten eines Artikels und kann deshalb selbst als Datenquelle genutzt werden (`NodeSourceLuceneIndex`). Dadurch besteht auch die Möglichkeit, einen Artikel aus dem Index als Ausgangsdokument für die Ähnlichkeitssuche zu verwenden. Des Weiteren bietet sich ein gegebenenfalls lokal vorhandener Index als Datenquelle für einen neuen Index an, wenn zum Beispiel neue Einstellungen für die Textaufbereitung während des Indizierens getestet werden sollen.

*Tabelle 4.1:
Datenquellen für
Wikipedia-Artikel*

Name	Beschreibung
<code>NodeSourceOnline</code>	Beliebiges MediaWiki
<code>NodeSourceWikipediaEN</code>	Englischsprachige Wikipedia
<code>NodeSourceWikipediaDE</code>	Deutschsprachige Wikipedia
<code>NodeSouceLuceneIndex</code>	Lucene-Index
<code>NodeSourceDirectory</code>	Einfache Textdateien
<code>NodeSourceSQL</code>	SQL-Datenbank
<code>NodeSourceCache</code>	lokaler Cache

-
1. HTTP ist ein Protokoll zur Datenübertragung in Netzwerken (Irvine et al., 1999).
 2. http://www.mediawiki.org/wiki/API:Main_page
 3. zum Beispiel <http://en.wikipedia.org/w/fürdieenglischsprachigeWikipedia>

Die von WikiMedia bereitgestellten Dumps (Abschnitt 2.4.5) sind eine weit verbreitete Methode, um lokal mit den Daten der Wikipedia zu arbeiten. Der Zugriff auf einzelne Artikel innerhalb einer solchen mehrere Gigabyte großen Datei ist allerdings sehr ineffizient, wenn die Artikel nicht der Reihe nach abgearbeitet werden. Die Datei wird deswegen meist in viele kleine Dateien, eine für jeden Artikel, aufgeteilt. `NodeSourceDirectory` stellt die Möglichkeit zur Verfügung, die Artikel aus einfachen Textdateien zu lesen. Dabei muss jeder Artikel in einer Textdatei gespeichert sein, deren Namen dem Artikelnamen in URL-kodierter Form (Berners-Lee et al., 2005) entspricht. Die Artikel sollten auf mehrere Unterverzeichnisse verteilt werden, da die meisten Dateisysteme Probleme mit einer sehr großen Anzahl von Dateien in einem einzigen Verzeichnis haben.

`NodeSourceSQL` kann Artikel aus einer SQL-Datenbank lesen. Grundsätzlich werden dabei alle Datenbanken unterstützt, für die ein entsprechender JDBC-Treiber⁴ für Java zur Verfügung steht. Um eine SQL-Datenbank als Datenquelle zu nutzen, müssen SQL-URL, -Nutzer und -Passwort sowie eine SQL-Abfrage in eine einfache Konfigurationsdatei geschrieben werden:

Listing 4.1:
Konfiguration der
SQL-Datenquelle

```
url = jdbc:postgresql://db.example.com:5432/wikitech
user = user
password = secret
query = SELECT wikitext FROM table WHERE title='%s';
```

Der Platzhalter `%s` wird später vom System durch den Artikelnamen ersetzt. Die SQL-Anfrage muss den Wikitext für genau diesen Artikel liefern.

`NodeSourceCache` verbindet `NodeSourceDirectory` mit einer anderen Datenquelle und stellt dadurch einen lokalen Cache zur Verfügung. Um den Cache zu nutzen, muss zunächst eine der oben genannten Datenquellen angelegt und an den Cache übergeben werden. Danach versucht Wikitech Classify stets, einen Artikel zuerst aus dem Cache abzurufen, statt zum Beispiel direkt auf eine entfernte, wesentlich langsamere Datenquelle zuzugreifen. Kann der Artikel nicht im Cache gefunden werden, kümmert sich Cache darum, die Daten aus der alternativen Quelle zu beziehen. Somit muss jeder Artikel nur einmal aus der entfernten Datenquelle gelesen werden und steht später lokal zur Verfügung. `NodeSourceCache` stellt allerdings keine Möglichkeit bereit, veraltete Artikel-Versionen im Cache zu erkennen oder sogar zu entfernen.

Alle hier vorgestellten Klassen sind von der abstrakten Klasse `NodeSource` abgeleitet, die es ermöglicht, die Datenquellen beliebig auszutauschen. Listing 4.2 gibt als Beispiel den Konstruktor der Klasse `NodeSourceSQL` an. Diesem Konstruktor muss die Datei übergeben werden, aus der die Konfiguration gelesen werden kann.

Listing 4.2: Konstruktor
der Klasse
`NodeSourceSQL`

```
public NodeSourceSQL(File configfile)
```

4. <http://download.oracle.com/javase/6/docs/technotes/guides/jdbc/getstart/GettingStartedTOC.fm.html>

Index-Dokument

Eine Lucene-Index besteht aus Dokumenten. Ein Dokument ist ein Container, der ein oder mehrere Felder enthält. In diesen Feldern wird wiederum der eigentliche Inhalt gespeichert, der später durchsucht werden soll.

Eine wichtige Aufgabe beim Entwurf einer Volltextsuchmaschine ist es, die Ausgangsdokumente auf diese Container mit einer sinnvollen Aufteilung in Felder abzubilden. Hinter der Aufteilung in Felder steht die Notwendigkeit, verschiedene Teile des Ausgangsdokumentes auf eine jeweils den Anforderungen entsprechende Weise zu verarbeiten. Kandidaten für verschiedene Felder sind zum Beispiel der Titel des Dokumentes, der exakt abgespeichert werden muss, da er später als eindeutiger Verweis auf das originale Dokument verwendet werden soll. Oder auch der eigentliche textuelle Inhalt des Ausgangsdokumentes, der spezielle Verarbeitungsschritte durchlaufen muss, um eine leistungsfähige Volltextsuche zu ermöglichen. Des Weiteren sind die einzelnen Felder später getrennt voneinander durchsuchbar. Unter anderem können Anwender ihre Suche auf den Titel des Dokumentes einschränken.

In dieser Arbeit werden die Lucene-Dokumente in die Felder aufgeteilt, die in Tabelle 4.2 aufgelistet sind.

Die Felder `title`, `wikitext` und `category` werden nicht für die Volltextsuche genutzt. Der Titel des Dokumentes ist notwendig, um später die Suchergebnisse eindeutig zu identifizieren. In das Feld `wikitext` wird der Wikitext des Ausgangsartikels unverändert übernommen. Dieser wird später benötigt, wenn die Artikel zur Suchzeit in Abschnitte aufgespalten werden sollen.

Das Feld `category` ist ein sogenanntes Multi-Value-Field und kann eine beliebige Anzahl von Kategorien aufnehmen. Für jede Kategorie des Ausgangsartikels wird während der Indizierung ein solches Feld angelegt. Sowohl `category`, als auch `wikitext` könnten auch nach der Suche aus der originalen Datenquelle geholt werden. Wie beschrieben sind sie für die Funktionalität der Volltextsuche unerheblich. Die Entscheidung, beide in den Index aufzunehmen, hat jedoch den Vorteil, dass neben den Daten für die Volltextsuche auch die Daten für die Kategorisierung im Index vorhanden sind und durch den gespeicherten Wikitext jeder Artikel als Ausgangsdokument für die Ähnlichkeitssuche genutzt werden kann. Folglich ist die Suche vollkommen unabhängig von der ursprünglichen Datenquelle.

Kern des Dokumentes ist das Feld `text`, worauf die eigentliche Volltextsuche basiert. Der nächste Abschnitt gibt eine detaillierte Beschreibung, wie aus dem Wikitext des Quellartikels die Text-Daten für den invertierten Index gewonnen werden.

Tabelle 4.2: Aufbau eines Indextokumentes

Feldname	Inhalt
<code>title</code>	Name des Wikipedia-Artikels
<code>text</code>	Artikelinhalt (aufbereitet für Volltextsuche)
<code>wikitext</code>	Wikitext des Wikipedia-Artikels
<code>category</code>	Kategorien des Wikipedia-Artikels.

Abbildung 4.1:
Wikitext-Beispiel
(Artikel Wallace Neff)

```
'''Wallace Neff''' ([[1895]] &ndash; June 8, 1982)
was an [[architect]] based in [[Southern California]]
and was largely responsible for developing the
region's distinct architectural style referred to
as "California" style.
```

Textanalyse

Der wichtigste Schritt des Indizierens ist die Analyse des Textes des Ausgangsdokumentes. Technisch gesehen ist der Ausgangstext zunächst lediglich eine lange Kette von Zeichen. Wikitech Classify muss diese Quelldaten zunächst so aufbereiten, dass Lucene sie in den Index schreiben kann. Im ersten Schritt wird dafür aus dem originalen Wikitext einfacher Text, ohne das für die Volltextsuche störende Wiki-Markup, gewonnen. Danach werden die Zeichenketten in einzelne Token zerlegt. Diese Token werden noch in eine für die Suche optimierte Form gewandelt, bevor sie schließlich in den Index geschrieben werden.

Wikitext enthält Auszeichnungselemente, die für die Suche nicht genutzt werden können und sogar störend sind. Abbildung 4.1 zeigt den ersten Paragraph aus dem Artikel *Wallace Neff*⁵. Das Beispiel enthält Hochkommata, um Fettschreibung zu erzwingen und doppelte eckige Klammern für Wiki-interne Links. Im Wikitext können darüber hinaus Elemente für Tabellen, Bilder, Aufzählungen, farbliche Hervorhebungen, externe Verweise und Referenzen und vieles mehr enthalten sein. Sogar HTML-Elemente und spezielle XML-Elemente dürfen die Nutzer einfügen, um beispielsweise die Gestaltungsmöglichkeiten, die über die Möglichkeiten der Vorgaben von Wikitext hinausgehen, zu nutzen.

Diese Elemente müssen zunächst aus dem ursprünglichen Wikitext entfernt werden. Das Ziel ist eine Version des Artikels, die ausschließlich aus einfachem Text besteht. Um dieses Ziel zu erreichen, wurde der Wikitech-Parser (Abschnitt 2.5) um einen Visitor erweitert, der den textuellen Inhalt der Knoten sammelt. Der Visitor hängt dabei an jeden Knoten den gesamten gesammelten Text der jeweiligen Kindknoten an. Daraus ergibt sich, dass jeder Paragraph-Knoten den gesamten Text des jeweiligen Paragraphen enthält. Jeder Section-Knoten enthält den gesamten Text aller unter ihm enthaltenen Paragraph- und Section-Knoten. Der Wurzelknoten beinhaltet schließlich den gesamten Text des Artikels. Einige Knoten, wie Dateinamen, Attribute oder Optionen werden ignoriert, da ihr Text nicht zum Text des Artikels gehört.

Das Ausgangsdokument liegt nun als einfacher Text vor (Abbildung 4.2), ist aber noch immer eine lange Kette von Zeichen. Im nächsten Schritt wird diese Zeichenkette in Wörter zerlegt und für das Indizieren vorbereitet.

Der *Tokenizer* zerlegt die Ausgangszeichenkette in Token. Dafür trennt er die Zeichenkette an Zeichen, die keine Buchstaben sind. Das Ergebnis ist ein Vektor von Token. Im Anschluss kommt ein Filter zum Einsatz, der aus diesem Vektor hochfrequente Token, sogenannte Stopp-Wörter, entfernt. Die Stopp-Wörter sind als Liste von Wörtern innerhalb dieses Filters

5. http://en.wikipedia.org/w/index.php?title=Wallace_Neff&oldid=397862911

Abbildung 4.2:
Einfacher Text (Artikel
Wallace Neff)

Wallace Neff (1895 June 8, 1982) was an architect based in Southern California and was largely responsible for developing the region's distinct architectural style referred to as "California" style.

Abbildung 4.3: Vektor
von Token (Artikel
Wallace Neff)

wallace	neff	june	architect	based	southern
california	largely	responsible	developing		
region	distinct	architectural	style	referred	
california	style				

festgelegt⁶. Zudem werden alle Wörter mit weniger als 3 Buchstaben als Stopp-Wörter behandelt. Ein weiterer Filter wandelt Sonderzeichen wie beispielsweise Umlaute in Buchstaben des englischen Alphabets und alle Großbuchstaben in Kleinbuchstaben um. Abbildung 4.3 zeigt den Token-Vektor für das Beispiel aus Abbildung 4.2. Abschließend werden alle Wörter durch das Stemming nach dem Porter-Algorithmus (Abschnitt 2.6.1) auf ihre Grundform zurückgeführt. Das Ergebnis hiervon wird in Abbildung 4.4 dargestellt.

Diese Vereinfachungen sollen später die Suche verbessern. Dabei soll es egal sein, ob Wörter groß oder klein, in Einzahl, Mehrzahl oder anders flektiert vorkommen. Aus linguistischer Sicht wird nach Wörtern statt nach konkreten Wortformen gesucht (Hausser, 2000, Kapitel 13).

Lucene stellt sogenannte *Analyzer* für diese Aufgaben bereit⁷. Sie können auf verschiedene Weise konfiguriert und kombiniert werden. Wikitech Classify abstrahiert einen weiteren Schritt davon und stellt die beiden Klassen `NodeAnalyzerSimple` und `NodeAnalyzerStemmer` zur Verfügung. Beide sind von der abstrakten Klasse `NodeAnalyzer` abgeleitet und daher austauschbar. Ersterer beinhaltet lediglich den Tokenizer und nimmt keine weiteren Änderungen am Ausgangstext vor. Er wurde nur für erste Tests verwendet. Später kam ausschließlich `NodeAnalyzerStemmer` zum Einsatz. Die beinhaltet alle oben aufgeführten Verarbeitungsschritte.

Abbildung 4.4: Vektor
von gestemmtten
Wörtern (Artikel
Wallace Neff)

wallac	neff	june	architect	base	southern
california	larg	respons	develop	region	
distinct	architectur	style	refer	california	
style					

6. In dieser Arbeit wurde die von Armand Brahaj vorgeschlagene Liste (<http://armandbrahaj.blog.al/2009/04/14/list-of-english-stop-words/>) genutzt. Sie wurde unter anderem in der Humboldt Digital Library (<http://www.avhumboldt.net/>) erfolgreich eingesetzt.

7. Die entsprechenden Klassen, die bereits für den nicht Lucene-basierten Prototypen von Wikitech Classify geschrieben wurden, finden deswegen keine Anwendung mehr und werden hier auch nicht beschrieben.

Vorgehensweise

Wikitech Classify stellt für das Indizieren die Klasse `NodeIndexer` zur Verfügung. Kern dieser Klasse ist die Methode `index` (Listing 4.3). An sie muss eine Liste von `NodeIdentifiers` übergeben werden. Durch diese können die Artikel eindeutig referenziert werden. Weiterhin muss festgelegt werden, ob der Index nach dem Schreibvorgang intern optimiert werden soll. Die Optimierung ist eine Lucene-interne Aufgabe und ermöglicht später eine schnellere Suche. Sie ist sinnvoll, wenn der Index schrittweise, durch wiederholten Aufruf von `index`, aufgebaut wird. Dabei ist zu beachten, dass der Vorgang temporär ein Vielfaches des Speicherplatzes benötigt, den der Index auf der Festplatte einnimmt und sehr viel Zeit in Anspruch nehmen kann⁸ (McCandless et al., 2010, Kapitel 2). Über den Schalter `redirect` wird festgelegt, ob auch Weiterleitungsartikel (Abschnitt 2.4.2) in den Index aufgenommen werden sollen. Eine Alternative dazu ist es, Weiterleitungsartikel gar nicht erst in die Liste der zu indizierenden Artikel aufzunehmen. Das kann aber je nach Datenquelle schwierig sein, da ein Artikel erst durch eine Betrachtung seines Inhalts⁹ als Weiterleitungsartikel identifiziert werden kann. Da Weiterleitungsartikel keinen Text-Inhalt haben, sollten sie grundsätzlich nicht in den Index aufgenommen werden. Sofern Sicherheit besteht, dass keine Weiterleitungsartikel in der Liste der zu kategorisierenden Artikel enthalten sind, kann die Überprüfung abgeschaltet werden, was das Indizieren ein wenig beschleunigt.

Listing 4.3: Methode `index` der Klasse `NodeIndexer`

```
public void index(List<NodeIdentifier> nodes,
                 boolean optimize, boolean redirect)
```

Listing 4.4: Konstruktor der Klasse `NodeIndexer`

```
public NodeIndexer(NodeSource nodeSource, File target,
                  Analyzer defaultAnalyzer)
```

An den Konstruktor der Klasse `NodeIndexer` (Listing 4.4) muss eine Instanz der Klasse `NodeSource` (Abschnitt 4.1.1) übergeben werden. Da beim Aufruf der Methode `index` nur die Artikelnamen übergeben werden, muss `NodeIndexer` wissen, woher es den Inhalt der Knoten holen kann. Das zweite Argument gibt den Speicherort des Lucene-Indexes an¹⁰.

Für das Argument `NodeAnalyzer` kommen Instanzen der beiden Klassen `NodeAnalyzerSimple` und `NodeAnalyzerStemmer`, die im letzten Abschnitt vorgestellt wurden, infrage.

Intern nutzt `NodeIndexer` die Klasse `NodeDocument`. Sie stellt die statische Methode `createDocument` (Listing 4.5) zur Verfügung, die aus den vier Einheiten `title`, `wikitext`, `text`, und `category` ein Lucene-Dokument mit den entsprechenden Feldern erstellt.

8. Bei den ersten Tests hat das Optimieren des Indexes über 3 Stunden gedauert. Auf der Workstation im CIP-Pool brach das Optimieren für den ca. 30 GB großen Index ab, da die 100GB große Festplatten-Partition nicht ausreichend war.

9. Weiterleitungsartikel enthalten den String `#REDIRECT` im Wikitext.

10. Wikitech Classify verwendet dabei die Standard-Möglichkeit `SimpleFSDirectory` zum Speichern des Indexes. Über einen alternativen Konstruktor können auch alle anderen Speichermöglichkeiten für den Index genutzt werden, die Lucene zur Verfügung stellt (McCandless et al., 2010, Kapitel 2).

Listing 4.5: Methode `createDocument` der Klasse `NodeDocument`

```
public static Document createDocument(
    NodeIdentifier identifier, String wikitext,
    String plaintext, List<String> categories)
```

Wikitech Classify legt nicht fest, woher die Liste der zu indizierenden Artikel kommt. Das CLI stellt aber folgende beiden Möglichkeiten zur Verfügung: Zum einen kann ein einzelner Artikel als Argument, zum anderen eine Datei mit einer Liste von Artikelnamen als Argument übergeben werden. Darin muss in jeder Zeile genau ein Artikelname aufgelistet sein.

Listing 4.6 zeigt, wie ein einzelner Artikel (*Java (Programming language)*) mithilfe der Wikitech-Classify-CLI indiziert wird. Dabei werden der Titel des Artikels, der indiziert werden soll, und die Datenquelle angegeben. Da keine weiteren Angaben gemacht wurden, wird der Artikel in den Index im aktuellen Verzeichnis geschrieben. Sofern noch kein Index existiert, wird ein neuer angelegt. Das Argument `-help` gibt eine Liste aller möglichen Schalter und Argumente an.

Listing 4.6: CLI-Aufruf für das Indizieren

```
classify index -online http://en.wikipedia.org/w/ \
    -identifizier "Java_(programming_language)"
```

Die Aufgabe `index` der Wikitech-Classify-CLI ist in der Klasse `Index` implementiert. Diese Klasse dient auch als Beispiel, wie das Indizieren mithilfe der Wikitech-Classify-API funktioniert (Anhang B.1).

Nach dem Aufruf von `index` stehen die neuen Artikel direkt für die Suche bereit, da Wikitech Classify automatisch die `commit`-Funktion von Lucene aufruft.

Während der ersten Tests für das Erstellen eines Indexes mit allen Wikipedia-Artikeln brach Wikitech Classify gelegentlich wegen Netzwerk-Problemen ab. Deswegen wurde die Artikelliste in Teile zu je 100 000 Artikeln aufgespalten und diese Teile jeweils einzeln indiziert. Obwohl der Index wegen des nicht ausreichenden Speicherplatzes anschließend nicht optimiert werden konnte, waren die Antwortzeiten für die Tests ausreichend schnell. Insofern konnte diese Vorgehensweise beibehalten werden.

4.1.2 Suchanfrage

Aus Sicht des Anwenders (beziehungsweise des übergeordneten Systems) besteht die Suchanfrage aus einem Ausgangsdokument. Der Anwender übergibt das Dokument, genauer einen Zeiger auf das Dokument, an Wikitech Classify. Das System liefert als Antwort eine Liste von ähnlichen Wikipedia-Artikeln.

Technisch gesehen, muss aus dem Ausgangsdokument zunächst eine Lucene-konforme Suchanfrage gebildet werden. Eine Suchanfrage besteht aus einer Reihe von Schlüsselwörtern. Dafür wird das Ausgangsdokument zunächst auf die gleiche Weise analysiert wie die Wikipedia-Artikel während des Indizierens. Wikitech Classify stellt verschiedene Algorithmen zur Verfügung, um aus dem Vektor von Wörtern eine Liste von Schlüsselwörtern zu erstellen. Diese Schlüsselwörter können in der Suchanfrage nach ihrer Relevanz gewichtet werden.

Ausgangsdokument

Als Ausgangsdokumente kommen

- ein Wikipedia-Artikel,
- ein Abschnitte eines Wikipedia-Artikels oder
- ein beliebiges Textdokument

infrage.

Zu Beginn muss aus dem Ausgangsdokument einfacher Text herausgefiltert werden. Wie das für Wikipedia-Artikel funktioniert, wurde bereits in Abschnitt 4.1.1 beschrieben. Für Textdokumente, die bereits als einfacher Text vorliegen, entfällt dieser Schritt. Die Wikitech-Classify-CLI bietet zusätzlich die Möglichkeit an, alle von Tika (Abschnitt 2.6.1) unterstützten Dokumente als Ausgangsdokument zu nutzen. In diesem Fall übernimmt Tika die Aufgabe, Text aus den Ausgangsdokumenten herauszuholen. Im Anschluss werden die gleichen *Analyzer* auf den Text angewendet, wie auch bei Wikipedia-Artikeln.

Es ist außerdem möglich, einen Wikipedia-Artikel als Ausgangsdokument auf einen einzelnen Abschnitt einzuschränken. Als Abschnitt kann dabei jeder Knoten mit seinem Unterbaum innerhalb des AST (Abschnitt 2.5.1) eines Artikels herangezogen werden. Die Knoten werden dabei durch XPointer (DeRose et al., 2000) referenziert. XPointer ist eine Abfragesprache, um Teile eines Dokumentes zu referenzieren. Der Knotentyp *Paragraph* entspricht einem Absatz in einem Text. Er wird durch eine oder mehrere Leerzeilen vom folgenden Text abgegrenzt. Der Knotentyp *Section* kann ebenfalls als Abschnitt angesehen werden. Er fasst den Text zusammen, der unterhalb dieser Überschrift steht, inklusive der Unter-Überschriften.

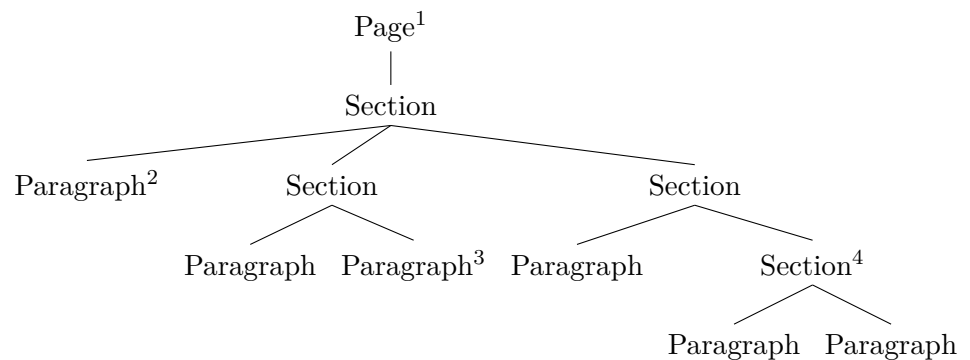
An dieser Stelle soll erklärt werden, wozu die Klasse *NodeIdentifier* eingeführt wurde. Wie im letzten Abschnitt beschrieben, benötigt das System eine Möglichkeit, um Teile eines Wikipedia-Artikels zu referenzieren und verwendet dafür XPointer. Eine Instanz der Klasse *NodeIdentifier* kann neben dem Artikelnamen auch den Pfad (XPath, Berglund et al. (2003)) zu einem Abschnitt enthalten. Im XPath werden die Knoten nacheinander aufgelistet, die vom Wurzelknoten aus durchlaufen werden müssen, um das Ziel zu erreichen. Der Name des Artikels und der XPath zu einem Knoten ergeben, getrennt durch #, den XPointer. Der einfachste Fall ist, das gesamte Dokument zu referenzieren, wobei der XPath-Teil wegfällt und der XPointer lediglich aus dem Artikelnamen besteht.

Abbildung 4.5 gibt ein Beispiel für einen einfachen AST, der auf die hier interessanten Knoten reduziert wurde. Listing 4.7 gibt die XPointer an, um die mit hochgestellten Zahlen markierten Knoten zu referenzieren. *Wallace Neff* ist der Name des Artikels. 1 referenziert in diesem Beispiel den gesamten Artikel, 2 den ersten Absatz im Artikel, 3 den zweiten Absatz unter der ersten Überschrift und 4 die erste Unter-Überschrift unterhalb der zweiten Hauptüberschrift.

Listing 4.7: Beispiele für Referenzierungen von Abschnitten durch XPointer

```
1: Wallace Neff
2: Wallace Neff#/Section[0]/Paragraph[0]/
3: Wallace Neff#/Section[0]/Section[0]/Paragraph[1]/
4: Wallace Neff#/Section[0]/Section[1]/Section[0]/
```

Abbildung 4.5:
Vereinfachter AST
(Referenzierung der
Abschnitte)



Textanalyse

Bei der Textanalyse für das Ausgangsdokument ist das Vorgehen analog zur Analyse für das Indizieren. Allerdings soll das Ergebnis, ein Vektor der Wörter, danach nicht in einen Index geschrieben werden. Stattdessen dient er als Ausgangspunkt für die Liste der Schlüsselwörter, die für das Bilden der Suchanfrage benötigt wird. Wichtig in diesem Schritt ist, dass wirklich der gleiche **NodeAnalyzer**, beispielsweise **NodeAnalyzerStemmer**, wie beim Indizieren genutzt wird. Nur so ist gewährleistet, dass Lucene die Wörter der Suche mit den Wörtern im Index abgleichen kann.

Aus dieser Liste aller Wörter des Ausgangsdokumentes muss eine Auswahl von Wörtern, die für das Dokument besonders relevant sind, genannt Schlüsselwörter, getroffen werden. Diese Auswahl wird benötigt, weil Volltextsuchmaschinen den Suchraum zunächst durch eine boolesche Suche (Abschnitt 2.1.2) einschränken und nur für die restlichen Dokumente die Ähnlichkeitswerte bestimmen. Bei der booleschen Suche werden zunächst alle Dokumente ausgeschlossen, die nicht mindestens ein Wort aus der Suche enthalten. Ohne eine Begrenzung auf eine bestimmte Menge von Schlüsselwörtern würde die boolesche Suche wesentlich mehr Dokumente zurückliefern und Lucene müsste für all diese Dokumente den Ähnlichkeitswert zur Suchanfrage berechnen. Die Suchzeit würde sich somit unnötig verlängern. Darüber hinaus kann eine Gewichtung der Schlüsselwörter in der Suchanfrage die Suchergebnisse verbessern.

Wikitech Classify stellt die folgenden drei Möglichkeiten, Schlüsselwörter zu extrahieren, zur Verfügung:

- Die ersten n Wörter
- Die n häufigsten Wörter
- Die n relevantesten Wörter nach TFIDF.

Die erste Möglichkeit (**Simple**) bestimmt die ersten n Wörter aus dem Ausgangsdokument zu den Schlüsselwörtern. Bei diesem Algorithmus ist es nicht möglich, die Wörter bei der Suchanfrage sinnvoll zu gewichten. Wörter, die mehrmals vorkommen, haben jedoch implizit ein höheres Gewicht, da sie auch mehrmals in die Suchanfrage aufgenommen werden. **Simple** soll vor allem als Vergleich für die Leistungsfähigkeit der anderen Algorithmen dienen.

Im zweiten Algorithmus (**Frequency**) werden die häufigsten Wörter als Schlüsselwörter genutzt und im dritten (**GlobalTfidf**) die relevantesten. Jedes Schlüsselwort wird dabei nur einmal in die Liste der Schlüsselwörter aufgenommen und tritt demzufolge auch nur einmal in der Suchanfrage

Tabelle 4.3: Klassen für die Schlüsselwort-Extrahierung

Klasse	Beschreibung
Simple	Die ersten n Wörter, ohne Gewichtung
Frequency	Die n häufigsten Wörter, gewichtet nach Frequenz
GlobalTfidf	Die n relevantesten Wörter, gewichtet nach TFIDF
GlobalTfidfFrequency	Die n relevantesten Wörter, gewichtet nach Frequenz

auf. Die Relevanz wird dabei nach TFIDF (Abschnitt 2.1.3) berechnet. Die Frequenz (TF) ergibt sich bei Beiden aus der Häufigkeit der Wörter im Ausgangsdokument. Die für TFIDF benötigte inverse Dokumentenhäufigkeit (IDF) wird über den gesamten Index, der durchsucht werden soll, berechnet. Lucene stellt dafür eine Methode zur Verfügung, die für ein bestimmtes Wort die Dokumentenhäufigkeit (DF) im Index liefert. Das Gewicht der Schlüsselwörter ergibt sich dann jeweils aus den Werten, die der jeweilige Algorithmus liefert. Für den Häufigkeitsalgorithmus bedeutet das zum Beispiel, dass ein 5 mal vorkommendes Wort das Gewicht 5 hat. Ergibt die TFIDF-Berechnung für ein Schlüsselwort 4,2, so ist auch das Gewicht für dieses Wort 4,2.

Für den TFIDF-Algorithmus stellt Wikitech Classify noch eine weitere Implementierung bereit, bei der zwar die Auswahl der Schlüsselwörter nach dem TFIDF-Werten erfolgt, deren Gewichtung aber nach der Häufigkeit vorgenommen wird (GlobalTfidfFrequency). Dieser Algorithmus soll der Tatsache Rechnung tragen, dass TFIDF in ersten Tests subjektiv subjektiv die besten Schlüsselwörter gefunden hat. Lucene berechnet die Relevanzwerte für die Suchergebnisse allerdings selbst auf Basis von TFIDF-Werten und benötigt dafür streng genommen die Wörthäufigkeit (TF) des Ausgangsdokuments. Es ist somit anzunehmen, dass es sinnvoller ist, die Worthäufigkeit, statt der TFIDF-Werte selbst, als Gewichtung in die Suchanfrage aufzunehmen. In Tabelle 4.3 werden die Klassen mit ihren Bedeutungen aufgelistet.

Abbildung 4.6 gibt die ersten 10 Schlüsselwörter mit ihrer Gewichtung für den Artikel *Java (programming language)* an. Die Schlüsselwörter wurden nach dem TFIDF-Algorithmus bestimmt und nach der Häufigkeit gewichtet (GlobalTfidfFrequency). Die Reihenfolge ergibt sich aus den ursprünglichen TFIDF-Werten, ist aber für die Suchanfrage nicht relevant.

Abbildung 4.6:
TFIDF-Schlüsselwörter
des Java-Artikels
gewichtet nach Frequenz

java, 246	oddeven, 17	applet, 28	japplet, 8
hello, 31	class, 83	method, 57	object, 51
joptionpan, 4	sun, 38		

Suche

Aus den Schlüsselwörtern muss noch eine Lucene-konforme Suchanfrage gebildet werden. Die einfachste Methode dafür ist, die Schlüsselwörter zu einer langen Zeichenkette zusammenzufügen. Dabei müssen die Schlüsselwörter durch Leerzeichen voneinander getrennt werden. Das Gewicht der Schlüsselwörter wird als Zahl, getrennt durch ein Zirkumflex, an die Wörter angehängt. Listing 4.8 stellt die Suchanfrage dar, wie sie für die Schlüsselwörter aus Abbildung 4.6 gebildet wird.

Listing 4.8:
Beispielsuchanfrage für
die Ähnlichkeitssuche

```
java^246 oddeven^17 applet^28 japplet^8 hello^31 class^83  
method^57 object^51 joptionpan^4 sun^38
```

In Wikitech Classify steht die Klasse `NodeSearcher` mit der Methode `search` (Listing 4.9) zur Verfügung. Diese Methode nimmt eine wie oben gebildete Suchanfrage entgegen und bildet daraus eine Lucene-Suchanfrage, die an die Dokumentstruktur des Indexes angepasst ist.

Listing 4.9: Methode
search der Klasse
`NodeSearcher`

```
public List<Result> search(String queryString, int numDocs)
```

Innerhalb dieser Klasse ist auch die Klasse `Result` definiert. Instanzen dieser Klasse können genau ein Ergebnis enthalten. Ein Ergebnis besteht aus dem `NodeIdentifier`, der das gefundene Dokument eindeutig referenziert, und aus dem Ähnlichkeitswert, den die Lucene-Suche für jedes Ergebnis mitliefert. Auf die Werte kann mittels der Methoden `getIdentifizier` und `getScore` zugegriffen werden.

Die `search`-Methode liefert letztlich eine Liste dieser Ergebnisse. Die Anzahl der Ergebnisse in dieser Liste kann durch das zweite Argument der Methode begrenzt werden.

Tabelle 4.4 listet die ersten 10 Ergebnisse für die Ähnlichkeitssuche nach *Java (programming language)* mit ihren Ähnlichkeitswerten. Als erstes Ergebnis wird das Ausgangsdokument selbst gefunden. Es hat auch den mit Abstand größten Ähnlichkeitswert.

Tabelle 4.4: Ähnliche
Artikel zu Java
(*programming language*)

Artikel	Ähnlichkeits-Wert
Java (programming language)	4,129594
Java (software platform)	0,5384701
Comparison of Java and C++	0,50830543
Programming language	0,49797016
Java Platform, Standard Edition	0,49787718
Comparison of Java and C Sharp	0,4972087
Pascal (programming language)	0,48900205
Java version history	0,47248083
Swing (Java)	0,4181688
C Sharp (programming language)	0,4133601

4.1.3 Zusammenfassung

Die Klasse `FindSimilar` stellt zwei verschiedene Varianten der Methode `find` zur Verfügung, die jeweils die oben genannten Aufgaben zusammenfassen. Die erste Methode (Listing 4.10) nimmt einen beliebigen Text als Java-String (erstes Argument) entgegen und sucht nach ähnlichen Artikeln im Index. Die zweite Methode (Listing 4.11) nimmt einen `NodeIdentifier`, also eine Referenz auf einen Artikel oder Artikel-Abschnitt, als erstes Argument. Der Artikel muss in dem Index, der durchsucht werden soll, enthalten sein.

Bei beiden Methoden kann festgelegt werden, wie viele Ergebnisse maximal zurückgeliefert werden sollen (Argument `int maxResults`) und ob die Gewichtungen der Schlüsselwörter in die Suchanfrage eingebaut werden sollen (`useTermWeight`). Die zweite Methode stellt zusätzlich den Schalter `removeOrigin` zur Verfügung. Über diesen kann bestimmt werden, dass das Ausgangsdokument nicht in der Liste der Ergebnisse enthalten sein soll.

Listing 4.10: Methode `find` der Klasse `FindSimilar`

```
public List<Result> find(String text,  
                        int maxResults, boolean useTermWeight)
```

Listing 4.11: Methode `find (2)` der Klasse `FindSimilar`

```
public List<Result> find(NodeIdentifier identifier,  
                        int maxResults, boolean useTermWeight,  
                        boolean removeOrigin)
```

Um die `find`-Methoden aufzurufen, muss zunächst ein Objekt der Klasse `FindSimilar` angelegt werden. Die Klasse stellt mehrere Konstruktoren zur Verfügung, wobei an dieser Stelle nur der in Listing 4.12 von Interesse ist. Der Konstruktor verlangt einen Schlüsselwort-Algorithmus (Tabelle 4.3), einen `NodeAnalyzer` (Abschnitt 4.1.1) und ein Verzeichnis, in dem der Lucene-Index liegt.

Listing 4.12: Konstruktor der Klasse `FindSimilar`

```
public FindSimilar(KeywordExtractor keywordExtractor,  
                  NodeAnalyzer analyzer, File directory)
```

Listing 4.13 zeigt, wie ähnliche Artikel mithilfe der `Wikitech-Classify-CLI` gefunden werden können. Der Ausgangsartikel muss dabei angegeben werden. `Wikitech Classify` durchsucht automatisch den Index, der in der Umgebungsvariablen `INDEXDIR` festgelegt wurde. Das Argument `-help` gibt eine Liste aller möglichen Schalter und Argumente an.

Listing 4.13: CLI-Aufruf für Ähnlichkeitssuche

```
classify similar -identifier "Java_(programming_language)"
```

Die Aufgabe `similar` der `Wikitech-Classify-CLI` ist in der Klasse `Similar` implementiert (Anhang B.2).

4.2 Finden ähnlicher Abschnitte

Im Abschnitt 4.1 sind die Ergebnisse für die Ähnlichkeitssuche auf ganze Artikel beschränkt. Das Ausgangsdokument kann jedoch bereits auf einen Abschnitt eingeschränkt werden (Abschnitt 4.1.2).

Wikitech Classify stellt zwei Möglichkeiten zur Verfügung, um die Suche so zu erweitern, dass ähnliche Abschnitte anstatt ganzer Artikel zurückgeliefert werden. Die erste und allgemeinere Methode (Abschnitt 4.2.1) basiert zunächst auf der Suche nach ähnlichen Artikeln. Diese Artikel werden anschließend in Abschnitte aufgeteilt. Diese Abschnitte werden in einen temporären Index geschrieben, über welchen wiederum eine zweite Suche mit demselben Ausgangsdokument durchgeführt wird. Die zweite Suche liefert letztendlich ähnliche Abschnitte.

Alternativ gibt es die Möglichkeit, bereits den Index aus Artikelabschnitten aufzubauen (Abschnitt 4.2.2). Bei dieser Vorgehensweise liefert die initiale Suche direkt die gewünschten Ergebnisse und ist damit wesentlich schneller. Dieser Vorteil muss allerdings durch eine geringere Flexibilität erkauft werden, da bereits zum Zeitpunkt des Indizierens festgelegt werden muss, wie die Artikel in Abschnitte aufgeteilt werden sollen.

Die Abbildung 4.7 stellt die beiden möglichen Vorgehensweisen übersichtlich dar. In der oberen Hälfte sind die Schritte angegeben, die beim Indizieren notwendig sind. Die Schritte in der unteren Hälfte müssen bei jeder Suche durchlaufen werden. In den folgenden Abschnitten werden die beiden Möglichkeiten detailliert beschrieben.

4.2.1 Artikelbasierter Index

Wie bereits beschrieben, sucht Wikitech Classify zunächst nach ähnlichen Artikeln und spaltet diese dann in Abschnitte auf. Für die Suche werden dabei die gleichen `find`-Methoden der Klasse `FindSimilar` verwendet, die auch zuvor bei der Suche nach ähnlichen Artikeln angewandt wurden (Abschnitt 4.1).

Um Artikel in Abschnitte aufzuspalten, stellt das System zwei verschiedene Möglichkeiten bereit. Einerseits können Artikel an beliebigen Knoten-Typen gespalten werden. Diese Aufgabe übernimmt die Klasse `NodeSplitterByElement`. Dem Konstruktor (Listing 4.14) muss lediglich der Name des Elements (`Section`, `Paragraph`, ...) übergeben werden. Die Methode `split` (Listing 4.15) nimmt den Wurzelknoten eines Artikels und liefert eine Liste der Knoten, die dem Element, welches im Konstruktor angegeben wurde, entsprechen. Die einzelnen Knoten in dieser Liste enthalten noch immer ihre Kindknoten und repräsentieren somit den entsprechenden Unterbaum. Je nachdem für welchen Knotentyp gespalten wird, können sich die Unterbäume überschneiden. Die Unterbäume der `Paragraph`-Elemente zum Beispiel überschneiden sich per Definition nicht. Die der `Section`-Elemente können sich allerdings überschneiden, da die Unterbäume der Überschriften möglicherweise auch Unter-Überschriften enthalten, welche selbst aber auch wiederum in der Liste der Knoten sind. Wird der AST in Abbildung 4.8 nach `Section`-Knoten aufgespalten, enthält `Section 2` den Text der `Paragraph`-Knoten `b` und `c`. Da `Section 2` eine Unter-Überschrift von `Section 1` ist, enthält sie sowohl den Text von `Paragraph a` als auch den Text von `b` und `c`.

Abbildung 4.7: Die beiden möglichen Vorgehensweisen zum Finden ähnlicher Abschnitte. Links: Artikel-Index, Aufspaltung zur Suchzeit. Rechts: Abschnitts-Index, Aufspaltung vor dem Indizieren.

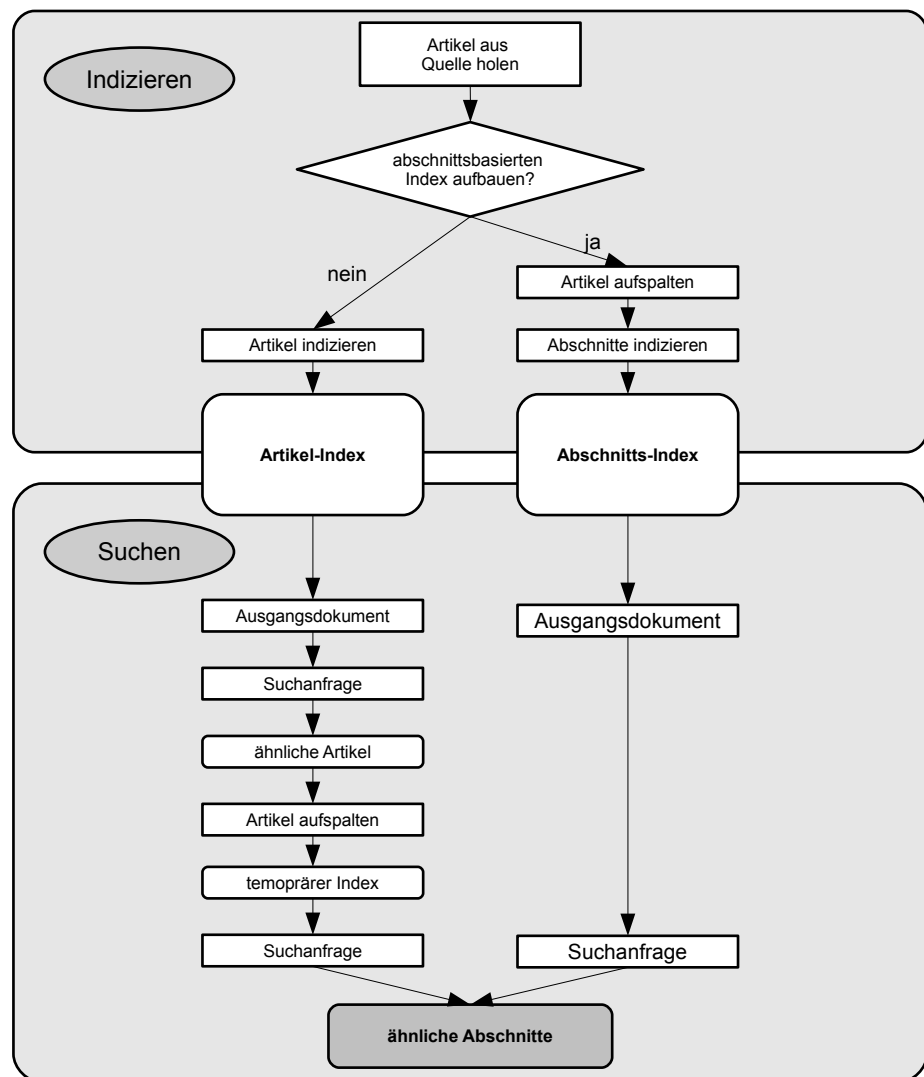
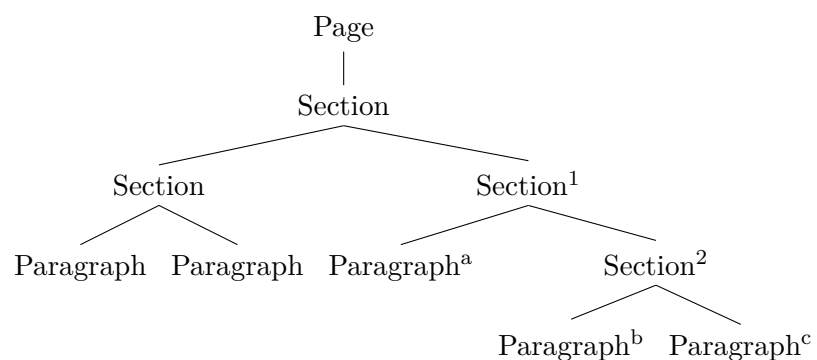


Abbildung 4.8: Aufspalten eines Artikels an Section-Knoten



Listing 4.14:
Konstruktor der Klasse
NodeSplitterByElement
Listing 4.15: Methode
split der Klasse
NodeSplitterByElement

```
public NodeSplitterByElement(String elementname)
```

```
public List<AstNode> split(AstNode node)
```

Die zweite Möglichkeit stellt *NodeSplitterByLevel* bereit. Dieser kann Artikel an den Knoten einer Ebene (oder Level), unabhängig von ihrem Typ, spalten. Auf welcher Ebene gespalten werden soll, wird dem Konstruktor (Listing 4.16) als Argument übergeben. Die *split*-Methode dieser Klasse ist identisch zur *split*-Methode von *NodeSplitterByElement* (Listing 4.15).

Listing 4.16:
Konstruktor der Klasse
NodeSplitterByLevel

```
public NodeSplitterByLevel(int level)
```

Sowohl *NodeSplitterByElement* als auch *NodeSplitterByLevel* sind von der abstrakten Klasse *NodeSplitter* abgeleitet. Um find aus *FindSimilar* anzuweisen, ähnliche Abschnitte zu finden, wird dem Konstruktor (Listing 4.17) zusätzlich ein *NodeSplitter* übergeben.

Listing 4.17: Zweiter
Konstruktor der Klasse
FindSimilar

```
public FindSimilar(KeywordExtractor keywordExtractor,  
                  NodeAnalyzer analyzer, Directory directory,  
                  NodeSplitter splitter)
```

Intern sucht *FindSimilar* wieder zunächst nach ähnlichen Artikeln. Für alle ähnlichen Artikel ruft es dann die *split*-Methode des *NodeSplitters* auf. Daraus resultieren die Knoten, welche die jeweiligen Abschnitte repräsentieren. *FindSimilar* schreibt diese Abschnitts-Knoten nun in einen temporären Index und wendet dabei die gleichen Regeln an, wie auch beim Indizieren ganzer Artikel. Dieser Index wird für die Dauer der Anfrage im Arbeitsspeicher angelegt und danach direkt gelöscht, die Funktionalität ist identisch. Über den temporären Index wird danach die gleiche Suchanfrage nochmals ausgeführt. In diesem Fall liefert die Suche ähnliche Abschnitte.

Dem Kommandozeilenaufwurf aus Listing 4.13 für das Finden ähnlicher Artikel muss hier zusätzlich das Argument *-levelsplitter <N>*, wobei *<N>* die Ebene ist, an der aufgespalten werden soll, übergeben werden. Für das Aufspalten nach Knoten-Typen wird dagegen das Argument *-elementsplitter <ELEMENT>* genutzt. *<ELEMENT>* muss dabei durch den Namen eines Elements, zum Beispiel Paragraph, ersetzt werden.

4.2.2 Abschnittsbasierter Index

Bei der zweiten Methode, um ähnliche Abschnitte zu finden, wird bereits der Index auf Basis von Abschnitten aufgebaut. Dafür wird genau wie für einen Artikel-basierten Index ein *NodeIndexer* benötigt. Der Methode *index* (Listing 4.18) wird nun allerdings ein *NodeSplitter* übergeben. *index* holt sich wieder die Artikel aus der angegebenen Quelle. Sie werden allerdings nicht direkt in den Index geschrieben, sondern zuvor mithilfe des *NodeSplitters* in Abschnitte aufgespalten. Statt des gesamten Artikels wird nun nur jeweils ein Abschnitt auf ein Lucene-Dokument abgebildet. Die Anzahl der Dokumente im Index wird folglich wesentlich größer sein. Solange sich die Knoten nicht überschneiden, ist der Index insgesamt aber nur unwesentlich größer.

Listing 4.18: Methode
index der Klasse
NodeIndexer

```
public void index(List<NodeIdentifier> nodes, boolean optimize,  
                 boolean redirect, NodeSplitter splitter)
```

Der Aufruf der CLI-Aufgabe index (Listing 4.6) muss ebenfalls so erweitert werden, dass eine Methode für das Aufteilen der Artikel angegeben wird. Dafür stehen die Argumente `-elementsplitter` und `-levelsplitter` bereit.

4.2.3 Zusammenfassung

In den beiden letzten Abschnitten wurden die Möglichkeiten vorgestellt, nach ähnlichen Abschnitten zu suchen. Beide haben Vor- und Nachteile. Die wesentlich schnellere Anfragezeit der zweiten Möglichkeit wird durch den höheren Aufwand beim Indizieren und nicht zuletzt durch die Festlegung auf die Art der Aufteilung in Abschnitte erkauft. Soll die Suche verschiedene Arten von Abschnitten unterstützen, wäre für jede Art der Aufbau eines zusätzlichen Indexes notwendig.

Bei der ersten Möglichkeit kann die Art der Aufteilung dagegen für jede Suche neu festgelegt werden. Alle möglichen Varianten können auf einem einzigen Index, der nur Artikel enthält, durchgeführt werden.

Für die Suche nach ähnlichen Abschnitten ist es bei der ersten Möglichkeit notwendig, einen temporären Index aufzubauen, der die entsprechenden Abschnitte der ähnlichen Artikel enthält. Durch dieses Vorgehen wird der Suchraum für ähnliche Abschnitte somit auch entsprechend eingeschränkt. Es ist durchaus möglich, dass sehr ähnliche Abschnitte nicht gefunden werden, da die entsprechenden Artikel zuvor schon als nicht ähnlich genug aussortiert wurden. Neben den bereits zuvor aufgeführten Vor- und Nachteilen muss also auch beachtet werden, dass sich die Ergebnisse der beiden Methoden durchaus unterscheiden können.

Die gefundenen Abschnitte werden wieder durch XPointer referenziert. Die Tabelle 4.5 listet als Beispiel die ersten zehn ähnlichen Paragraph-Knoten für die Ähnlichkeitssuche mit *Java (programming language)* als Ausgangsdokument und Tabelle 4.6 die ersten zehn ähnlichen Paragraph-Knoten zu dem ersten Paragraph-Knoten¹¹, der Text enthält, aus dem Java-Artikel (*Java (programming language)#/Section[0]/Paragraph[2]/*).

11. Je nach Aufbau eines Artikels ist es möglich, dass dieser aus technischer Sicht Paragraph-Knoten enthält, die aber keinen Text enthalten. In diesem Fall enthält erst der dritte Paragraph-Knoten den ersten Abschnitt

Tabelle 4.5: Ähnliche Abschnitte zum Artikel Java (programming language)

Abschnitt	Wert
Java (programming language) #/Section[0]/Section[12]/Paragraph[4]/	0,4142118
Java (programming language) #/Section[0]/Section[3]/Section[1]/Paragraph[6]/	0,23516893
Programming language #/Section[0]/Section[9]/Paragraph[4]/	0,21958461
Comparison of Java and C++ #/Section[0]/Section[1]/Section[1]/Paragraph[0]/	0,18921988
Java (programming language) #/Section[0]/Section[5]/Paragraph[0]/	0,18174432
Pascal (programming language) #/Section[0]/Section[10]/Paragraph[4]/	0,13581617
Java version history #/Section[0]/Section[5]/Paragraph[0]/	0,11086359
Comparison of Java and C++ #/Section[0]/Section[1]/Section[2]/Paragraph[0]/	0,10071209
Pascal (programming language) #/Section[0]/Section[4]/Section[0]/Paragraph[0]/	0,09732283
C Sharp (programming language) #/Section[0]/Section[3]/Paragraph[3]/	0,0966209

Tabelle 4.6: Ähnliche Abschnitte zum ersten Abschnitt aus Java (programming language)

Abschnitt	Wert
Java (programming language) #/Section[0]/Paragraph[2]/	5,545354
Java (disambiguation) #/Section[0]/Section[1]/Paragraph[0]/	1,1171597
Write once, run anywhere #/Section[0]/Paragraph[0]/	0,8758534
Java Virtual Machine #/Section[0]/Section[0]/Paragraph[0]/	0,8670965
Java Virtual Machine #/Section[0]/Paragraph[0]/	0,78873
Juice (JVM) #/Section[0]/Paragraph[0]/	0,73829806
Java (software platform) #/Section[0]/Section[0]/Section[0]/Paragraph[2]/	0,72197896
Oracle Database #/Section[0]/Section[2]/Section[0]/Paragraph[0]/	0,7092593
Java (software platform) #/Section[0]/Paragraph[3]/	0,7085904
Java (programming language) #/Section[0]/Section[5]/Paragraph[0]/	0,70666885

4.3 Kategorisierung

Die Kategorisierung wird als eigenständige Aufgabe von Wikitech Classify angesehen. Sie ist aber auch ein Maß für die Qualität der Ähnlichkeitssuche.

Das Ergebnis der Ähnlichkeitssuche ist eine Liste von Artikeln. Die Wikipedia-Autoren haben diesen Artikeln Kategorien zugewiesen. Auf Grundlage der Kategorien der ähnlichen Artikel sollen Vorschläge für die Kategorisierung des Ausgangsdokumentes erstellt werden. Statt eine bestimmte Menge von Kategorien für die Vorschläge auszuwählen, ist es sinnvoll, eine Liste aller gefundenen Kategorien, geordnet nach Relevanz, zu liefern. Dieses Vorgehen ist, genau wie bei den Suchergebnissen für ähnliche Artikel und Abschnitte, für Anwender, welche die Vorschläge später nutzen, von Vorteil (Sebastiani, 2002, Kapitel 2).

Abbildung 4.9 zeigt die Vorgehensweise beim Erstellen der Kategorievorschläge auf. $A_?$ ist der Artikel oder Text, der kategorisiert werden soll. Die Suche nach ähnlichen Artikeln hat die Artikel $A_1, A_2, \dots$ als Ergebnisse geliefert. Die Liste der ähnlichen Artikel kann sehr lang sein. Für dieses Beispiel sollen aber nur die ersten fünf Artikel herangezogen werden. Jeweils rechts neben den Artikeln A_1 bis A_5 sind die Kategorien dieser Artikel gelistet. Die Reihenfolge der Kategorien eines Artikels ist willkürlich und stellt keine Gewichtung dar.

Im ersten Schritt zählt Wikitech Classify die Anzahl der Kategorien aller ähnlichen Artikel. Hier zeigt sich, warum es sinnvoll ist, dabei nur eine bestimmte Anzahl der besten Ergebnisse der Ähnlichkeitssuche in Betracht zu ziehen. Anderenfalls läuft das System Gefahr, immer nur die häufigsten Kategorien aus dem Index zuzuweisen.

Die Zählung für dieses Beispiel ergibt, dass die Kategorien C_f und C_c jeweils viermal vorkommen. Sie sind damit die besten Kandidaten, um den Artikel $A_?$ zu kategorisieren. C_b wird dreimal gezählt, C_h und C_a jeweils zweimal. Die Liste der Kategorievorschläge setzt sich fort, bis alle Kategorien der ähnlichen Artikel in die Vorschläge aufgenommen wurden.

Abbildung 4.9: Erstellen einer Liste von Kategorievorschlägen nach der Ähnlichkeitssuche

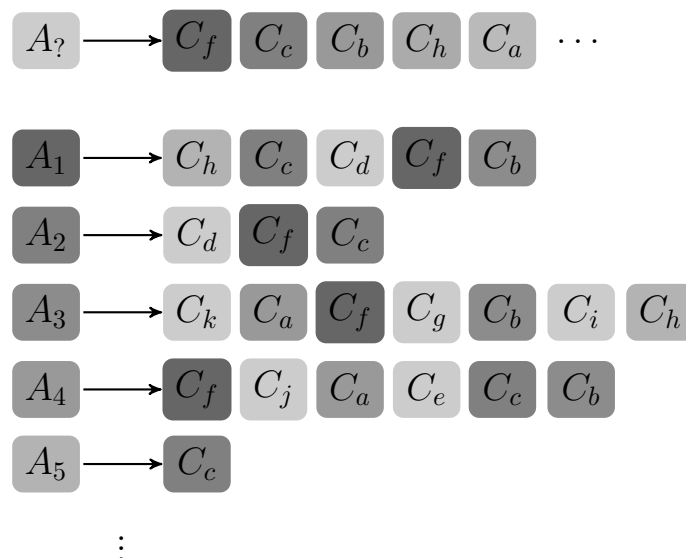


Tabelle 4.7:
Kategorievorschläge für
den Artikel *Java*
(*programming language*)

Kategorie-Vorschläge	Wert
Category:Java platform	2,7388315
Category:Java programming language	1,8314822
Category:Computing platforms	1,4403825
Category:Programming language comparisons	1,0055141
Category:Cross-platform software	0,9462694
Category:Java specification requests	0,9056765
Category:C++	0,50830543
Category:Notation	0,49797016
Category:Programming language topics	0,49797016
Category:Educational programming languages	0,48900205

Die Methode, ausschließlich auf die Häufigkeit der Kategorien zu achten, hat zwei entscheidende Nachteile. Erstens kann es wie im Fall von C_f und C_c dazu kommen, dass zwei oder mehr Kategorien gleich häufig auftreten. Welche Kategorie in den Ergebnissen zuerst genannt wird, ist nicht festgelegt. Zweitens werden die Kategorien aller ähnlichen Artikel gleich gewichtet. Tatsächlich ist die Liste der ähnlichen Artikel aber nach dem Ähnlichkeitswert geordnet (in der Abbildung durch abnehmende Sättigung der Hintergrundfarbe dargestellt). Diese Gewichtung lässt sich für die Gewichtung der Kategorievorschläge nutzen. Statt die Kategorien einfach durchzuzählen, summiert Wikitech Classify die Ähnlichkeitswerte der Artikel der Kategorien. Infolge kommt es praktisch kaum noch vor, dass zwei Kategorievorschläge das gleiche Gewicht haben. Zudem ist anzunehmen, dass die Kategorisierung dadurch insgesamt bessere Ergebnisse liefert (Abschnitt 5.3). Tabelle 4.7 listet die Kategorievorschläge, die Wikitech Classify für den Artikel *Java (programming language)* macht. Im Gegensatz zu den Werten für die Ähnlichkeitssuche sind die Werte für die Kategorievorschläge nicht normalisiert. Dadurch können die Werte nicht zwischen verschiedenen Sucheinstellungen verglichen werden. Insbesondere ergeben sich höhere Werte, je mehr ähnliche Artikel für die Kategorie-Vorschläge in Betracht gezogen werden sollen.

In den Beispielen wurden bisher nur den Artikeln direkt zugeordnete Kategorien beachtet. Diese Kategorien sind aber selbst wieder Kategorien zugeordnet. Es ergibt sich eine Hierarchie von Kategorien, wie in Abschnitt 2.4.4 vorgestellt. Die direkten Kategorien des Artikels sind Kategorien der ersten Ebene. Die Oberkategorien dieser Kategorien sind Kategorien der zweiten Ebene und so weiter. Grundsätzlich sind Kategorien höherer Ebenen allgemeinere Kategorien.

Bei der Kategorisierung ermöglicht Wikitech Classify, nur bestimmte Kategorie-Ebenen der ähnlichen Artikel in Betracht zu ziehen. Dadurch ergibt sich die Möglichkeit, allgemeinere Kategorien zu finden. Andererseits wäre es durchaus möglich, dass sich auch die Kategorisierung für die erste Ebene des Ausgangsdokumentes verbessert, wenn auch Kategorien höherer Ebenen in Betracht gezogen werden. Hierzu soll nochmals das Beispiel des Artikels *Cologne* in Erinnerung gerufen werden, bei dem die Kategorie *Cities in North Rhine-Westphalia* sowohl direkt als auch indirekt über die Kategorie *Cologne* zweimal zugewiesen wurde.

Tabelle 4.8: Vorschläge
für allgemeinere
Kategorien für den
Artikel Java
(programming language)

Kategorie-Vorschläge	Wert
Category:Programming languages	12,365349
Category:Computing	7,4234223
Category:Computing platforms	4,3794775
Category:Procedural programming languages	3,7173254
Category:Computer languages	3,1479566
Category:Programming language families	2,8287897
Category:Object-based programming languages	2,7531476
Category:Object-oriented programming	2,7531476
Category:Computer programming	2,4756446
Category:Curly bracket programming languages	2,2503195

Wikitech Classify stellt die Klasse `FindCategories` zur Verfügung, um aus einer Liste ähnlicher Artikel Kategorievorschläge abzuleiten. Um diese Klasse zu nutzen, muss zunächst ein Objekt der Klasse angelegt werden. Dafür muss dem Konstruktor (Listing 4.19) das Verzeichnis des Index mitgeteilt werden. Diesen Index benötigt `FindCategories`, um die Kategorien der ähnlichen Artikel nachzuschlagen.

Für jedes zu kategorisierende Dokument muss zuerst mithilfe der Klasse `FindSimilar` eine Liste ähnlicher Artikel (Abschnitt 4.1.3) beziehungsweise ähnlicher Abschnitte (Abschnitt 4.2.3) erstellt werden.

Listing 4.19:
Konstruktor der Klasse
`FindCategories`

```
public FindCategories(File indexDirectory)
```

Diese Liste wird jeweils an die Methode `find` (Listing 4.20) der Klasse `FindCategories` übergeben. Durch den Schalter `boolean useSimilarityWeight` kann angegeben werden, dass `find` die Ähnlichkeitswerte der gefundenen Artikel oder Abschnitte in die Berechnung der Kategorievorschläge einbezieht.

Listing 4.20: Methode
`find` der Klasse
`FindCategories`

```
public List<Result> find(
    List<Result> similarnodes, boolean useSimilarityWeight,
    boolean useDepthWeightLinear, boolean useDepthWeightHalf,
    int startDepth, int maxDepth)
```

Die restlichen vier Argumente konfigurieren, welche (Ober-)Kategorien in Betracht gezogen werden sollen. Die Argumente `int startDepth` und `int maxDepth` bestimmen die unterste beziehungsweise oberste Ebene, deren Kategorien in die Vorschlagsliste aufgenommen werden. Sollen nur die Kategorien in Betracht gezogen werden, die direkt den Artikeln zugewiesen wurden, sind beide Argumente auf 1 zu setzen. Werden beide Argumente zum Beispiel auf 3 gesetzt, liefert das System allgemeinere Kategorien. Die Tabelle 4.8 listet Kategorievorschläge, die auf Basis der dritten Ebene der ähnlichen Artikel gefunden wurden.

Kategorien verschiedener Ebenen können in die Liste der Kategorievorschläge aufgenommen werden, indem zum Beispiel `startDepth` auf 1 und `maxDepth` auf 2 gesetzt werden. Standardmäßig werden die Kategorien aller Ebenen mit gleicher Gewichtung in die Liste der Vorschläge aufgenommen. Die beiden verbleibenden Argumente stellen zwei Möglichkeiten zur Verfügung, diese Gewichtung zu ändern. `boolean useDepthWeightLinear` bestimmt, dass die Gewichtung höherer Ebenen linear abnimmt.

Tabelle 4.9:
Gewichtungsfaktoren für
die verschiedenen
Gewichtungsstrategien
(Beispiel Ebene 1-4)

Gewichtungs-Art	Ebene			
	1	2	3	4
(ohne)	1	1	1	1
linear	1	0,75	0,5	0,25
halbierend	1	0,5	0,25	0,125

`boolean useDepthWeightHalf` bestimmt, dass die Gewichtung höherer Ebenen für jede Ebene jeweils halbiert wird. Die Tabelle 4.9 gibt als Beispiel die jeweiligen Gewichtungsfaktoren für `startDepth=1` und `maxDepth=4` an. Sofern nur eine Ebene (`startDepth=maxDepth`) berücksichtigt wird, ist eine Gewichtung ohne Auswirkung für die Reihenfolge der Kategorievorschläge.

Die Wikitech-Classify-CLI stellt auch eine Aufgabe zum Finden der Kategorien bereit (`categories`). Der Schalter `-help` listet alle Einstellungsmöglichkeiten. Listing 4.21 gibt ein einfaches Beispiel für den Aufruf, um Kategorievorschläge für einen Artikel zu erhalten. Wie bei `similar` wird hier ebenfalls automatisch auf den Index zurückgegriffen, der über die Umgebungsvariable `INDEXDIR` definiert ist.

Listing 4.21: CLI-Aufruf
für Kategorievorschläge

```
classify categories -identifizier "Java_(programming_language)"
```

Die Aufgabe `categories` ist in der Klasse `Categories` (Anhang B.3) implementiert.

4.3.1 Kategorisierung auf Basis ähnlicher Abschnitte

Im letzten Abschnitt wurde eine Liste ähnlicher Artikel an die Methode `find` der Klasse `FindSimilar` übergeben. Die Kategorisierung erfolgte damit auf Basis der ähnlichen Artikel.

Um die Kategorisierung auf Basis ähnlicher Abschnitte umzustellen, müssen lediglich ähnliche Abschnitte an die `find`-Methode von `FindSimilar` übergeben werden. In Abschnitt 4.2 wurden die beiden Möglichkeiten, ähnliche Abschnitte zu finden, erläutert.

In diesem Fall erstellt `FindCategories` die Liste der Kategorievorschläge ausgehend von den ähnlichen Abschnitten. Hier ergibt sich allerdings folgendes Problem: Die Kategorien in der Wikipedia gelten immer nur für ganze Artikel und nicht für einzelne Abschnitte. Es ist trotzdem anzunehmen, dass auch die Kategorien eines Artikels, aus dem ein ähnlicher Abschnitt stammt, gute Vorschläge sind. Des Weiteren können mehrere ähnliche Abschnitte aus dem gleichen Artikel stammen. Die Kategorien dieses Artikels werden somit höher gewichtet, da sie mehrmals gezählt werden. Damit könnte es sogar zu einer Verbesserung der Kategorisierung des Ausgangsdokumentes kommen.

Die Kategorisierung ist eine Möglichkeit, die Qualität der Ähnlichkeitssuche von Wikitech Classify zu bewerten.

4.4 Einfache Volltextsuche

Mit Wikitech Classify besteht auch die Möglichkeit, eine einfache Volltextsuche über den Index durchzuführen. Für die Suchanfrage kann hier zusätzlich auf die Syntax von Lucene-Abfragen (McCandless et al., 2010, Kapitel 3) zurückgegriffen werden, die zum Beispiel die Möglichkeit bietet, Dokumente mit bestimmten Wörtern auszuschließen. Das System liefert als Ergebnis eine Liste von Artikeln oder Abschnitten, die der Suchanfrage ähnlich sind. Sie ist nach Relevanz sortiert.

Da die Volltextsuche nicht die eigentliche Aufgabe von Wikitech Classify ist, soll hier nicht auf die Details eingegangen werden. Intern ist die Volltextsuche in der Klasse `NodeSearcher` implementiert. Listing 4.22 gibt ein Beispiel für eine einfache Suchanfrage mithilfe des Kommandozeilenaufrufs.

Listing 4.22:
Volltextsuche auf der
Kommandozeile

```
classify query -query \  
"bavaria_AND_(soccer_OR_football)_NOT_munich"
```

In diesem Kapitel sollen die besten Parameter für die verschiedenen Einstellungsmöglichkeiten von Wikitech Classify gefunden und schließlich die Leistungsfähigkeit beurteilt werden.

Für die Evaluation von Textklassifikationssystemen kommen grundsätzlich zwei Möglichkeiten in Betracht. Neben automatischen Auswertungen können manuell erstellte Beurteilungen der Ergebnisse durch Benutzer herangezogen werden. Die letztere Möglichkeit hat den Vorteil, dass die Nutzer ihre Zufriedenheit mit dem System ausdrücken können. Um belastbare Werte zu erhalten, müssen die Tests für die Evaluation aber unter strengen zuvor zu erarbeitenden Richtlinien durchgeführt werden. Die Menge der Tests muss ausreichend groß sein. Die Vorbereitung, Durchführung und Auswertung solcher Versuche stellen einen erheblichen Aufwand dar (Saracevic et al., 1988).

Deswegen werden Textklassifikationssysteme meistens mithilfe automatischer Tests evaluiert. Die Leistungsfähigkeit wird hierzu mittels der etablierten Maße Recall, Precision und F-Measure (Abschnitt 2.1.5) gemessen. Durchschnittswerte über viele Tests können die Leistungsfähigkeit des Systems im Allgemeinen beschreiben.

Um Recall und Precision für einen einzelnen Versuch zu berechnen ist es notwendig zu wissen, welche Ergebnisse im Sinne der Suchanfrage relevant sind. Die tatsächlich richtigen Ergebnisse für das Ausgangsdokument müssen also bekannt sein. Für die Suche nach ähnlichen Artikeln und Abschnitten ist diese Voraussetzung nicht gegeben. In der Wikipedia ist nirgends direkt festgelegt, welche Artikel oder sogar Abschnitte ähnlich zu einem anderen Artikel sind.

Genau hier können die Kategorien der Wikipedia-Artikel herangezogen werden. Wikipedia-Kategorien haben die Hauptaufgabe, Artikel, die zu einem bestimmten Thema gehören, sich also inhaltlich ähnlich sind, zusammenzufassen (Zesch und Gurevych (2007), Wikipedia contributors (2011a)). Daneben gibt es noch spezielle Kategorien, zum Beispiel Wartungskategorien, die aber nicht innerhalb des Artikels aufgeführt und auch nicht für die Evaluation in Betracht gezogen werden.

Die Ähnlichkeitssuche kann demnach durch die Bewertung der Qualität der Kategorisierung evaluiert werden.

Umgekehrt legitimiert diese Tatsache auch das Finden der Kategorien über den Umweg der Suche nach ähnlichen Artikeln und Abschnitten. Die Ähnlichkeitssuche auf Basis von Lucene und damit auf Basis des Vektor-Raum-Modells findet inhaltlich ähnliche Dokumente (McCandless et al., 2010, Kapitel 6).

Da alle Dokumente im Index enthalten sind, liefert die Ähnlichkeitssuche in der Regel auch das Ausgangsdokument als erstes Ergebnis zurück. Für die Evaluation werden die Ausgangsdokumente allerdings nicht berücksichtigt – ihre Kategorien gehen nicht in die Berechnung der Kategorievorschläge ein¹. Die Tatsache, dass die Wörter der Ausgangsdokumente im Index enthalten sind, hat ebenfalls keinen Einfluss auf die Ergebnisse, da sich die *document frequency* (Abschnitt 2.1.3) aller Schlüsselwörter des Ausgangsdokuments jeweils genau um 1 erhöht.

Wikitech Classify stellt Klassen und eine Kommandozeilenaufgabe (*evaluate*) für die Evaluation bereit. Mit deren Hilfe können die Qualitätsmaße für einzelne Artikel oder Abschnitte berechnet werden.

Der Kommandozeilenaufruf kann zudem eine ganze Liste von Artikeln und Abschnitten verarbeiten und als CSV-Datei ausgeben. Um die Durchschnittswerte der Qualitätsmaße zu berechnen, wurde ein Python-Skript², basierend auf der csv-Bibliothek³, erstellt.

Tabelle 5.1: Verteilung der Wikipedia-Seiten auf Namensräume

Namensraum	Anzahl Artikel
Main	7.226.791
User talk	5.223.391
Main talk	3.284.931
Wikipedia	516.246
User	928.917
File	902.728
Category	551.955
Template	258.753
Category talk	184.552
Portal	82.085
Template talk	71.900
File talk	65.441
Wikipedia talk	65.078
Portal talk	9.591
MediaWiki	1.329
MediaWiki talk	759
Help	677
Book	732
Book talk	597
Help talk	333
Insgesamt	19.376.786

1. Wikitech Classify entfernt den Ausgangsartikel beziehungsweise alle Abschnitte des Ausgangsdokuments, vor der Rückgabe der Ergebnisse.

2. <http://www.python.org/>

3. <http://docs.python.org/library/csv.html>

5.1 Test-Indizes und Test-Datensätze

5.1.1 Indizes

Alle Tests im Rahmen dieser Arbeit wurden auf Basis des Wikipedia-Dumps (Abschnitt 2.4.5) der Englischen Wikipedia vom 30. Januar 2010 durchgeführt. Aus diesem Dump wurden zwei Indizes, einer für die artikelbasierte Suche und einer für die abschnittsbasierte Suche, gebildet.

Die Indizes enthalten die Inhaltsartikel (Main-Namensraum), mit Ausnahme der Weiterleitungsartikel, der Wikipedia (Abschnitt 2.4). Zudem beinhalten die Indizes alle Kategorie-Seiten. Sie sind für das Finden von Oberkategorien notwendig. Kategorie-Seiten werden aber so im Index gespeichert, dass sie von der Ähnlichkeitssuche ausgenommen sind.

Die Tabelle 5.1 auf der letzten Seite gibt einen Überblick über die Anzahl der Seiten in der Wikipedia, aufgeteilt nach den Namensräumen, in denen sie sich befinden. Insgesamt hat die Wikipedia vom 30. Januar 2010 über 19 Mio. Seiten. Davon sind aber nur 7,2 Mio. Artikel im Namensraum *Main* und 0,5 Mio. im Namensraum *Category*.

Um eine Liste der zu indizierenden Artikel zu erstellen, wurde aus der Datenbank mit dem Wikipedia-Dump zunächst eine Textdatei in Form einer Liste mit den Namen aller Wikipedia-Seiten erstellt. Mithilfe des Programmes GNU `grep`⁴ und eines regulären Ausdrucks wurden alle Seiten aus nicht relevanten Namensräumen entfernt (Listing 5.1). Die Artikelliste für das Indizieren (Abschnitt 4.1.1) beinhaltet demnach ungefähr 7,7 Mio. Wikipedia-Seiten.

Listing 5.1: Filtern der Namensräume

```
grep -v -P \  
"^(Talk|Wikipedia|User|File|Help|Book|Template|MediaWiki|\  
Portal|Thread|Summary|Category talk)( talk)?:"
```

Die 7,2 Mio. Artikel enthalten allerdings noch Weiterleitungsartikel, die nicht in den Index aufgenommen werden sollen. Diese Weiterleitungsartikel können nicht am Titel, sondern nur am Inhalt erkannt werden. Wikitech Classify stellt deswegen eine Möglichkeit zur Verfügung, Artikel vor dem Indizieren zu prüfen und gegebenenfalls auszuschließen (Abschnitt 4.1.1). Bei 21 960 Artikeln brach der Wikitech-Parser, der für das Extrahieren des Textes notwendig ist ab, weil er die Syntax nicht analysieren konnte. Diese Artikel fehlen somit im Index.

Begriffserklärungsseiten enthalten in der Regel eine kurze Beschreibung für alle Artikel, auf die sie verweisen. Werden innerhalb eines Artikels Änderungen vorgenommen, müssen auch die Begriffserklärungsseiten überprüft werden. Sie sollten deswegen nicht vom Index ausgeschlossen werden.

Wikitech Classify gibt beim Indizieren aus, ob es einen Artikel in den Index aufgenommen oder als Weiterleitungsartikel beziehungsweise wegen eines Parser-Fehlers ausgeschlossen hat. Anhand dieser Angaben kann die Statistik in Tabelle 5.2 abgeleitet werden. Von den 7,2 Mio. Artikeln im Main-Namensraum sind 3,5 Mio. (49%) Weiterleitungsartikel, somit verbleiben 3,7 Mio. Artikel. Hinzu kommen noch die 0,5 Mio.

4. <http://www.gnu.org/software/grep/>

Tabelle 5.2: Anzahl der Dokumente im Index

Namensraum	Anzahl Artikel
Main	7.226.791
Weiterleitungen	−3.513.495
Category	551.955
Parser-Fehler	−21.960
Insgesamt	4.243.291

Kategorie-Seiten. Abzüglich der 22 tausend Artikel ($<1\%$), die nicht analysiert werden konnten, ergibt sich eine Anzahl von ungefähr 4,2 Mio. Dokumenten im Index.

Zusätzlich wurde ein zweiter Index für die abschnittsbasierte Ähnlichkeitssuche auf Basis derselben Daten aufgebaut. Dafür wurden die Artikel an ihren Paragraph-Knoten aufgespalten (Abschnitt 4.2.2). Dieser Index enthält 29 Mio. Dokumente.

5.1.2 Datensätze

In der Wikipedia werden qualitativ hochwertige Artikel als sogenannte *good articles* ausgezeichnet. Sie zeichnen sich unter anderem durch Korrektheit und die umfassende Beschreibung ihres Themas aus (Wikipedia contributors, 2011c). Derzeit sind etwa 11 000 Artikel von den Nutzern der Wikipedia als *good articles* gewählt worden.

Etwa 3 000 Artikel wurden zu *featured articles* gewählt. Diese Artikel sind die besten Artikel in der englischsprachigen Wikipedia. Sie entsprechen den oben genannten Kriterien in besonderem Maße, sind zudem aktuell und stabil, besonders gut strukturiert und geschrieben (Wikipedia contributors, 2011b).

Für die Versuche in dieser Arbeit wurden drei verschiedene Artikel-Listen verwendet:

- 162 *featured articles*
- 150 *good articles*
- 200 zufällig ausgewählte Artikel

Sowohl *featured articles* als auch *good articles* sind in verschiedene Themengebiete gruppiert. Für das Erstellen der Testlisten wurden jeweils die ersten sechs Artikel aus jeder Gruppe ausgewählt. Die 200 zufällig ausgewählten Artikel wurden einmalig durch einen Zufallsgenerator bestimmt. Für die Auswahl galten die folgenden Einschränkungen: Sie durften keine *featured articles* oder *good articles* sein, mussten mindestens 200 Zeichen lang und mindestens einer Kategorie zugeordnet sein.

An dieser Stelle soll vorweggenommen werden, dass die Unterschiede der Testergebnisse für die verschiedenen Testlisten sehr stabil sind. Zur besseren Übersicht werden in den folgenden Abschnitten zunächst nur die Werte für den *good articles*-Datensatz angegeben. Die R-Precision ist am besten geeignet, um verschiedene Parameter zu vergleichen. Alle Werte sind Mittelwerte, die sich aus den einzelnen Werten für jeden getesteten Artikel beziehungsweise Abschnitt ergeben.

5.2 Schlüsselwort-Algorithmus

Zunächst soll aufgezeigt werden, welcher der in Abschnitt 4.1.2 vorgestellten Algorithmen für die Schlüsselwort-Extrahierung die besten Ergebnisse liefert und wie groß die optimale Menge der Schlüsselwörter ist. Anschließend wird die Bedeutung der Gewichtung der Schlüsselwörter dargestellt. Die Tabelle 5.3 stellt die vier Schlüsselwort-Algorithmen, jeweils für 25, 96, 150 und 256 Schlüsselwörter gegenüber. Die Gewichtung der Schlüsselwörter in der Suchanfrage war bei allen Versuchen eingeschaltet. Dabei ist jedoch zu beachten, dass bei **Simple** die Gewichtung für jedes Schlüsselwort gleich hoch ist.

Aus den Werten lässt sich ablesen, dass der Schlüsselwort-Algorithmus bei weitem nicht so relevant für die Leistungsfähigkeit des Systems ist wie vermutet – solange ausreichend viele Schlüsselwörter in die Suche einbezogen werden. Bei zu vielen Schlüsselwörtern werden die Ergebnisse allerdings wieder schlechter. Das Optimum liegt hier bei 96 Schlüsselwörtern. Insbesondere überrascht, dass **Simple** fast so gut ist wie die anderen Algorithmen. Dies könnte dadurch begründet sein, dass gerade die ersten Sätze eines Artikels in ihrer Bedeutung als Abriss oder Definition des Themas des Artikels besonders viele relevante Wörter enthalten.

GlobalTfidfFrequency weist bei allen Schlüsselwortmengen stabile Werte auf und liefert das beste Ergebnis. Er wird deswegen in allen folgenden Tests genutzt.

Tabelle 5.3: Evaluation der Schlüsselwort-Algorithmen (good articles)

Algorithmus	Anzahl (n)	R-Precision
Simple	25	0,22
Frequency		0,23
GlobalTfidf		0,28
GlobalTfidfFrequency		0,28
Simple	96	0,28
Frequency		0,29
GlobalTfidf		0,29
GlobalTfidfFrequency		0,30
Simple	150	0,29
Frequency		0,29
GlobalTfidf		0,28
GlobalTfidfFrequency		0,29
Simple	256	0,29
Frequency		0,27
GlobalTfidf		0,26
GlobalTfidfFrequency		0,27

Tabelle 5.4: Evaluation der Schlüsselwortgewichtung (good articles)

Gewichtung	Algorithmus	R-Precision
ohne	Simple	0,28
	Frequency	0,28
	GlobalTfidf	0,26
	GlobalTfidfFrequency	0,26
mit	Simple	0,28
	Frequency	0,29
	GlobalTfidf	0,29
	GlobalTfidfFrequency	0,30

Wie bedeutend die Gewichtung der Schlüsselwörter ist, wird in Tabelle 5.4 gezeigt. In der oberen Tabellenhälfte stehen die Werte für Suchanfragen, bei denen die Gewichtung nicht in die Suchanfrage einbezogen wurde. In der unteren Tabellenhälfte stehen die Werte für die Suchanfragen mit Gewichtung. Für Simple sind die Werte jeweils gleich. Da alle Schlüsselwörter dasselbe Gewicht haben, kann es nicht zu einer Verbesserung kommen. Vor allem für die TFIDF-basierten Algorithmen ist die Gewichtung aber sehr wichtig. Ohne Sie fallen die Ergebnisse sogar schlechter aus als bei Simple.

5.3 Gewichtung nach Ähnlichkeit

In Abschnitt 4.3 wurde erklärt, wie Wikitech Classify die Ähnlichkeitswerte der gefundenen Artikel in die Berechnung der Werte für die Kategorien einbeziehen kann. In diesem Versuch werden die Ergebnisse für die Kategorisierung ohne Beachtung der Ähnlichkeitswerte mit den Ergebnissen unter Beachtung der Ähnlichkeitswerte verglichen. Des Weiteren soll herausgefunden werden, wie viele ähnliche Artikel überhaupt in Betracht gezogen werden sollten. Die Ergebnisse werden in Tabelle 5.5 gelistet. In der linken Spalte steht die jeweilige Anzahl der ähnlichen Artikel, die in die Berechnung eingegangen sind. In den Spalten zwei und drei werden die R-Precision-Werte für das einfache Zählen der Kategorien der ähnlichen Artikel denen gegenübergestellt, bei denen zusätzlich die Ähnlichkeit der gefundenen Artikel einbezogen wurde.

Die Werte für den Versuch, in dem nur die Kategorien des ersten Artikels in Betracht gezogen wurden, sind identisch. Bei einem einzigen Artikel

Tabelle 5.5: Evaluation der Gewichtung nach Artikelähnlichkeit (good articles)

Anzahl	R-Precision bei	
	einfachem Zählen	Ähnlichkeitsgewichtung
1	0,25	0,25
2	0,26	0,27
3	0,25	0,28
5	0,27	0,31
10	0,30	0,31
20	0,30	0,34
40	0,28	0,33
80	0,25	0,30

kann sich die Ähnlichkeit nicht auswirken. Schon bei zwei Artikeln ist die R-Precision der dritten Spalte etwas höher. Insgesamt wirkt sich Gewichtung nach Ähnlichkeit positiv auf die Ergebnisse aus.

Die Ergebnisse werden schlechter, wenn eine zu hohe Anzahl gefundener Artikel für die Kategorienvorschläge berücksichtigt wird. Durch die niedrigere Relevanz der jeweils hinzugefügten Artikel fällt dieser Effekt in der dritten Spalte weniger stark ins Gewicht.

Dieser Versuch ergab die beste R-Precision bei 20 Artikeln unter Einbeziehung der Artikelähnlichkeit. Diese Einstellungen werden für die folgenden Tests genutzt.

5.4 Abschnitte

Das Finden ähnlicher Abschnitte ist ein Ziel dieser Arbeit. Die Qualität für die abschnittsbasierten Ergebnisse wird wie für die artikelbasierten Ergebnisse über die Kategorien der Artikel evaluiert.

Des Weiteren soll untersucht werden, inwiefern sich einzelne Abschnitte als Ausgangsdokument für die Ähnlichkeitssuche eignen. Auch hier lassen sich die Ergebnisse nur über die Kategorien der jeweiligen Artikel validieren. Dieser Test wurde nur für den Text des ersten Paragraph-Knotens (Abriss des Artikels) durchgeführt. Andere Knoten als Ausgangsdokument sind zwar eine sinnvolle Anwendung, können aber nicht automatisch validiert werden, da ein beliebiger Abschnitt aus einem Artikel nicht als repräsentativ für den gesamten Artikel und damit allen seinen Kategorien angesehen werden kann.

Wie bereits in Abschnitt 4.2 erklärt, könnte sich die Klassifizierung über den Umweg der ähnlichen Abschnitte allgemein verbessern, da auf diese die Kategorien von Artikeln mit mehreren ähnlichen Abschnitten mehrfach in das Ergebnis einbezogen werden.

Die Tabelle 5.6 listet die Ergebnisse der verschiedenen Tests. In den beiden ersten Zeilen basiert die Suche auf ähnlichen Artikeln. Dabei sind in der ersten Zeile ebenfalls Artikel das Ausgangsdokument. In der zweiten Zeile ist jeweils nur der Text des ersten Paragraph-Knotens Ausgangsdokument. Die R-Precision ist im zweiten Test etwas niedriger. Für die letzten beiden Zeilen wurde der abschnittsbasierte Index für die Ähnlichkeitssuche genutzt. In der vorletzten Zeile sind wie im ersten Test wieder Artikel die Ausgangsdokumente und in der letzten Zeile die ersten Paragraph-Knoten die Ausgangsdokumente.

Die Zeilen 2 und 4 zeigen, dass Wikitech Classify durchaus ähnliche Textstellen zu einzelnen Abschnitten finden kann. Die R-Precision ist nur wenig niedriger als in den Zeilen 1 und 3, in denen der gesamte Artikel als Ausgangsdokument dient.

*Tabelle 5.6:
Artikelklassifikation im
Vergleich mit
Abschnittsklassifikation
(good articles)*

	Quelle	Ziel	R-Precision
1	Artikel	Artikel	0,34
2	1. Paragraph	Artikel	0,31
3	Artikel	Paragraph	0,30
4	1. Paragraph	Paragraph	0,29

Die Qualität der Suche nach ähnlichen Abschnitten kann durch die Differenzen zwischen den Zeilen 1 und 3 beziehungsweise 2 und 4 beurteilt werden. Die Werte zeigen, dass das System ähnliche Abschnitte finden kann. Eine Verbesserung der Kategorisierung des Ausgangsdokumentes lässt sich allerdings entgegen der Hoffnung nicht erreichen.

5.5 Kategorie-Ebenen

In den bisherigen Auswertungen wurden nur die Kategorien betrachtet, die den Artikeln direkt zugewiesen sind. Wikitech Classify kann aber auch die Oberkategorien dieser Kategorien in Betracht ziehen (Abschnitt 4.3).

5.5.1 Finden allgemeinerer Kategorien

Wikitech Classify macht das Kategorie-System der Wikipedia auch für Texte verfügbar, die nicht im Zusammenhang mit der Wikipedia stehen. Dabei kann es aber sinnvoll sein, dass die Dokumente nicht so detailliert kategorisiert werden, wie es die Wikipedia für Artikel vorsieht. In diesem Versuch werden die Ausgangsartikel auf den Ebenen eins bis sieben klassifiziert. Die Tabelle 5.7 listet jeweils die R-Precision für die Tests auf den einzelnen Ebenen. An den Werten kann abgelesen werden, dass die Kategorisierung für allgemeinere Kategorien besser funktioniert. Mit jeder höheren Ebene steigt die R-Precision.

Wikipedia-Autoren, die sich von Wikitech Classify Kategorien für ihre neuen oder geänderten Texte vorschlagen lassen, könnten die allgemeineren Kategorien als Einstiegspunkt nutzen und sich die feinen Kategorien selbst raussuchen. Hier ist allerdings zu beachten, dass die Anzahl der Kategorien (Zeile R) mit jeder Ebene zunimmt, da jede Kategorie mehreren Oberkategorien zugeordnet werden kann.

5.5.2 Kategorisierung mithilfe der Oberkategorien

In diesem Versuch soll getestet werden, ob die Kategorisierung der Ausgangsdokumente verbessert werden kann, wenn auch die Oberkategorien der gefundenen Artikel in die Liste der Vorschläge für die Kategorisierung aufgenommen werden. Die R-Precision wird dabei ausschließlich auf Basis der Kategorien der ersten Ebene des Ausgangsartikels berechnet.

Der erste Test entspricht der bisherigen Kategorisierung, die ausschließlich auf der ersten Ebene durchgeführt wird. Danach wird jeweils die nächste Ebene hinzugenommen, bis Ebene vier erreicht ist. Die R-Precision wird dabei jeweils für die drei in Abschnitt 4.3 vorgestellten Gewichtungsmöglichkeiten der Ebenen angegeben (gleiche Gewichtung, linear abnehmende Gewichtung, halbierende Gewichtung). Die Ergebnisse werden in Tabelle 5.8 gelistet.

Tabelle 5.7:
Kategorisierung auf
verschiedenen
Kategorie-Ebenen (good
articles)

Ebene	1	2	3	4	5	6	7
R-Precision	0,35	0,42	0,45	0,50	0,57	0,63	0,67
R	3,99	8,09	17,69	37,27	74,83	136,05	216,70

Tabelle 5.8: R-Precision
für verschiedene
Kategorie-Ebenen (good
articles)

Ebenen	R-Precision bei Gewichtung			
	gleich	linear	abnehmend	halbierend
nur 1	0,34		0,34	0,34
1 bis 2	0,12		0,24	0,24
1 bis 3	0,04		0,12	0,16
1 bis 4	0,01		0,04	0,10

Die Werte zeigen, dass die Kategorisierung durch die Hinzunahme der Oberkategorien nicht verbessert werden kann. Im Gegenteil, je mehr Oberkategorien hinzukommen, desto niedriger wird die R-Precision. Die abnehmende Gewichtung der Oberkategorien lässt diesen Effekt zwar weniger drastisch ausfallen, kann ihn aber nicht verhindern.

Es ist anzunehmen, dass hier besonders die Tatsache ins Gewicht fällt, dass jeder Kategorie der ersten Ebene mehrere Kategorien der höheren Ebenen zugeordnet sein können. Dies hat zur Folge, dass die Kategorievorschläge zum Großteil aus Kategorien höherer Ebenen bestehen. Die Kategorien der höheren Ebenen verdrängen sozusagen die Kategorien der ersten Ebene, statt sie lediglich zu ergänzen. Des Weiteren können die ähnlichen Artikel, auf denen die Kategorisierung basiert, einer Oberkategorie indirekt mehrmals zugeordnet worden sein. Diese Oberkategorien werden auch entsprechend mehrmals gewichtet und verdrängen die Kategorien der ersten Ebene noch weiter von den vorderen Ergebnisplätzen.

5.6 Vergleich zwischen verschiedenen Artikel-Qualitäten

Die Ergebnisse dieses Versuchs zeigen die Unterschiede zwischen den drei gewählten Testlisten. Das obere Drittel der Tabelle 5.9 listet die Ergebnisse für den Testsatz mit *featured articles*. In der Mitte sind die Werte für den Testsatz mit *good articles* und im unteren Drittel die Ergebnisse für den Testsatz mit den zufällig ausgewählten Artikeln (random) aufgeführt. Die Spalte *R* gibt an, wie viele Kategorien den Artikeln durchschnittlich zugewiesen wurden. In der rechten Hälfte der Tabelle sind die Qualitätsmaße Precision, Recall und F-Measure jeweils für $k = 1, 2, 3, 5, 10, 20$ und *R* gelistet. Die Werte bei *R* sind dabei jeweils für das *R* des jeweiligen Testsatzes berechnet, entsprechen also der R-Precision.

Tabelle 5.9: Ergebnisse
für die verschiedenen
Testsätze

Testsatz	R	Maß	at k=						
			1	2	3	5	10	20	R
featured	6,96	Precision	0,62	0,5	0,42	0,35	0,24	0,14	0,35
		Recall	0,14	0,21	0,27	0,34	0,44	0,51	
		F-Measure	0,23	0,3	0,33	0,34	0,31	0,22	
good	7,65	Precision	0,5	0,42	0,39	0,29	0,2	0,12	0,34
		Recall	0,16	0,24	0,3	0,34	0,44	0,5	
		F-Measure	0,24	0,31	0,34	0,31	0,28	0,19	
random	3,99	Precision	0,64	0,53	0,43	0,33	0,22	0,14	0,44
		Recall	0,25	0,36	0,42	0,5	0,57	0,64	
		F-Measure	0,36	0,43	0,42	0,4	0,32	0,23	

Auffällig ist zunächst, dass die zufällig ausgewählten Artikel durchschnittlich weniger Kategorien haben als die *featured articles* und *good articles*. Der Grund hierfür ist wohl, dass für letztere ein feineres Kategoriesystem entwickelt wurde, weswegen auch mehrere detailliertere Kategorien zugewiesen wurden. Nicht zuletzt ist anzunehmen, dass den zufällig ausgewählten Artikeln einfach noch Kategorien fehlen. Des Weiteren haben *good articles* im Durchschnitt 0,69 Kategorien mehr als *featured articles*. Das könnte unter anderem daran liegen, dass beispielsweise kleine Fehler wie die doppelte Kategorisierung von *Cologne* als *Cities in North Rhine-Westphalia* einerseits direkt und andererseits indirekt über die Kategorie *Cologne* bei *featured articles* nur sehr selten vorkommen dürften.

Grundsätzlich sind die Werte für die zufällig ausgewählten Artikel höher. Es ist anzunehmen, dass für diese Artikel im Durchschnitt allgemeinere Kategorien verwendet werden und dass es gerade die sehr speziellen Kategorien (zum Beispiel *38 BC establishments*) sind, die noch fehlen.

Die Werte für die unterschiedlichen k verteilen sich wie erwartet: Die Precision wird für kleinere k höher und der Recall niedriger. Der F-Measure ist genau bei $k = R$ am höchsten.

5.7 Vergleich zur Baseline

In diesem Versuch soll die Leistungsfähigkeit des Systems mit einer sogenannten Baseline ins Verhältnis gesetzt werden. Die Baseline bildet eine Referenz beziehungsweise einen Bezugspunkt zur Bewertung der Ergebnisse eines Systems. Um Textklassifikationssysteme bewerten zu können, muss immer dann auf eine solche Referenz zurückgegriffen werden, wenn es keine Werte aus früheren Arbeiten gibt, mit denen die eigenen Werte verglichen werden können (so zum Beispiel auch in Gantner und Schmidt-Thieme (2009)).

In dieser Arbeit werden die Werte eines statischen Klassifizierers als Bezugspunkt angewandt. Dieser Klassifizierer weist jedem zu klassifizierenden Dokument immer die häufigsten Kategorien zu, unabhängig von dessen Inhalt. Die Frequenz gibt auch innerhalb der Liste der Kategorievorschläge die Reihenfolge vor.

Die Tabelle 5.10 stellt die Werte von Wikitech Classify der Baseline gegenüber. Alle Werte sind für den Testsatz mit zufällig ausgewählten Artikeln angegeben. Wikitech Classify liefert sowohl für die Precision als auch für den Recall immer deutlich bessere Werte. Für Anwender werden die Baseline-Werte nicht zufriedenstellend sein, Wikitech Classify stellt deswegen eine erhebliche Verbesserung dar.

Tabelle 5.10:
Wikitech-Classify-Werte
für zufällig ausgewählte
Artikel im Vergleich zu
einer Baseline

Methode	Maß	at						
		1	2	3	5	10	20	R
Baseline	Precision	0,14	0,08	0,05	0,04	0,02	0,01	
	Recall	0,03	0,03	0,03	0,04	0,04	0,04	0,04
	F-Measure	0,05	0,04	0,04	0,04	0,03	0,02	
Classify	Precision	0,64	0,53	0,43	0,33	0,22	0,14	
	Recall	0,25	0,36	0,42	0,5	0,57	0,64	0,44
	F-Measure	0,35	0,43	0,42	0,4	0,32	0,23	

5.8 Vergleich zu Gantner/Schmidt-Thieme (2009)

Die Arbeit Gantner und Schmidt-Thieme (2009) wurde bereits in Kapitel 3 vorgestellt. Es ist die einzige gefundene Arbeit, in der versucht wird, die Artikel der Wikipedia selbst zu klassifizieren. Tabelle 5.11 listet die Ergebnisse für die beiden Datensätze Philosoph und Eishockey. Die Baseline-Werte wurden auf dieselbe Weise ermittelt wie im letzten Abschnitt für Wikitech Classify. Allerdings wurden nur die Kategorien des jeweiligen Teil-Datensatzes verwendet.

Die Werte sehen auf den ersten Blick besser aus als die Werte von Wikitech Classify. Die Precision bei $k = 1$ ist 0,83 für den Eishockey-Datensatz. Für den Philosoph-Datensatz ist keine Precision bei $k = 1$ angegeben. Wikitech Classify erreicht bei den zufällig ausgewählten Artikeln bei $k = 1$ eine Precision von 0,64. Hierbei muss aber beachtet werden, dass die Precision-Werte schon für den Baseline-Klassifizierer bei Gantner/Schmidt-Thieme wesentlich höher sind als bei den Tests in dieser Arbeit (zum Beispiel 0,49 bei Gantner/Schmidt-Thieme gegenüber 0,14 hier). Die Precision-Werte für $k = 2$ sind dagegen fast gleich (0,53 beziehungsweise 0,52 für Philosoph und Eishockey gegenüber 0,53 bei Wikitech Classify).

Die Recall-Werte sind bei Gantner für $k = 2$ fast doppelt so hoch wie bei Wikitech Classify. Gerade beim Recall wirkt sich die unterschiedliche Anzahl der Kategorien allerdings erheblich aus. Schon bei dem Vergleich der Recall-Werte zwischen dem Philosoph-Datensatz und dem Eishockey-Datensatz ist der Recall bei dem Datensatz mit weniger Kategorien etwas besser (0,63 im Gegensatz zu 0,67).

Die beiden Datensätzen haben jeweils nur 55 und 159 Kategorien. In der gesamten englischsprachigen Wikipedia, auf der hier die Tests durchgeführt wurden, gibt es jedoch über 500 000 Kategorien. Des Weiteren haben die Artikel der beiden Datensätze von Gantner/Thieme-Schmidt im Durchschnitt weniger als zwei Kategorien. Die zufällig ausgewählten Artikel aus der gesamten Wikipedia haben dagegen durchschnittlich 4 Kategorien. Wikitech Classify hat somit im Gegensatz zu dem anderen System keine Möglichkeit, bei zwei Kategorievorschlägen alle vier tatsächlichen Kategorien zu finden.

Tabelle 5.11: Ergebnisse von Gantner/Schmidt-Thieme (2009)

	Philosoph		
	Precision	Recall	F-Measure
Baseline bei $k = 1$	0,49	0,32	0,38
Baseline bei $k = 55$	0,03	1	0,06
SVM bei $k = 2$	0,52	0,67	0,59
SVM bei $k = 3$	0,40	0,76	0,52
	Eishockey		
	Precision	Recall	F-Measure
Baseline bei $k = 2$	0,21	0,16	0,19
Baseline bei $k = 159$	0,01	1	0,02
SVM bei $k = 1$	0,83	0,63	0,72
SVM bei $k = 2$	0,53	0,78	0,63

Der große Unterschied zwischen den Recall-Werten für die Baseline-Tests ist ein weiteres Indiz, wie unterschiedlich schwer die Aufgabenstellung ist. Der Baseline-Wert für $k = 1$ bei Gantner/Schmidt-Thieme beträgt bereits 0,32. Die Werte für den Baselinetest bei Wikitech Classify für die Kategorien der gesamten Wikipedia erreichen selbst bei $k = 20$ nur einen Wert von 0,04.

Der wichtigste Unterschied zwischen den beiden Vorgehensweisen ist jedoch die Geschwindigkeit und Skalierbarkeit. Die Komplexität der von Gantner/Thieme-Schmidt verwendeten Klassifizierungsmethode steigt sowohl während des Trainings als auch während der Klassifizierung von Dokumenten in Abhängigkeit von der Anzahl der Kategorien (Yang et al., 2003). Nach deren Angaben steigt die Komplexität für die genutzte SVM zwar nur linear, der Unterschied zwischen 55 beziehungsweise 159 Kategorien zu den über 500 000 Kategorien der gesamten Wikipedia ist jedoch enorm. Für jede Kategorie muss ein eigener Klassifizierer trainiert werden. Entsprechend muss auch jedes zu klassifizierende Dokument gegen alle Klassifizierer getestet werden. Bei Wikitech Classify hat die Anzahl der Kategorien keinerlei Auswirkung auf die Komplexität.

In beiden Systemen steigt die Komplexität beim Training des Klassifizierers – bei Wikitech Classify ist es das Indizieren – linear mit der Anzahl der Dokumente und der enthaltenen Datenmenge. Änderungen am Kategoriesystem wirken sich bei Wikitech Classify nur geringfügig aus, machen bei SVM-basierten Methoden aber unter Umständen ein komplett erneutes Training notwendig.

Die Antwortzeiten von Volltextsuchmaschinen liegen im Sekundenbereich. Bei entsprechender Optimierung und geeigneter Hardware können die Antwortzeiten weit unter einer Sekunde liegen (McCandless et al., 2010, Kapitel 1).

Ein weiterer wichtiger Aspekt beim Vergleich der beiden Systeme ist, dass SVM-basierte Systeme (wie in Gantner/Schmidt-Thieme) allgemein für leistungsfähiger als VSM-basierte Systeme (wie Wikitech Classify) hinsichtlich der Qualitätsmaße gehalten werden (Manning et al., 2008). Letztendlich ist diese Leistungsfähigkeit aber nur ein – wenn auch wichtiges – Kriterium bei der Wahl eines Systems für eine bestimmte Aufgabe. Kurze Antwortzeiten, so dass die Ergebnisse möglichst schnell zur Verfügung stehen, war ein wichtiges Kriterium bei der Suche nach einem geeigneten Verfahren für die Aufgabenstellung dieser Arbeit.

Das Finden ähnlicher Artikel oder Abschnitte, das als eigenständige Aufgabe von Wikitech Classify gesehen wird, kann das SVM-basierte Modell in diesem Sinne nicht leisten.

5.9 Repräsentativität

In den bisherigen Versuchen war die Anzahl der Artikel vergleichsweise gering. Um die Ergebnisse zwischen den Versuchen vergleichbar zu machen, wurden zudem jeweils die gleichen Artikel und Abschnitte verwendet. In diesem Versuch wird gezeigt, dass die ermittelten Werte repräsentativ sind. Zu diesem Zweck wurden zehn Listen mit je 1000 zufällig ausgewählten Artikeln erstellt. Wie zuvor wurde diese Auswahl nur durch die Bedingungen eingeschränkt, dass jeder Artikel mindestens 200 Zeichen lang und mindestens einer Kategorie zugeordnet sein muss.

Die Ergebnisse werden in Tabelle 5.12 aufgelistet. Wie in den vorherigen Tests werden die Durchschnittswerte für die jeweiligen Listen angegeben. Am Ende der Tabelle werden die Durchschnittswerte für die gesamten 10 000 Artikel gelistet. Die Werte für die verschiedenen Listen unterscheiden sich nur gering. Die Evaluationsmethode liefert also stabile Werte für verschiedene Testlisten. Verglichen mit den Werten aus den vorangegangenen Abschnitten für die zufällig ausgewählten Artikel, treten ebenfalls nur geringe Unterschiede auf.

*Tabelle 5.12:
Ergebnisse für Tests mit
jeweils 1000 zufällig
ausgewählten Artikeln*

Testsatz	R	Maß	at k=						
			1	2	3	5	10	20	R
1	3,70	Recall	0,27	0,38	0,43	0,51	0,57	0,63	0,45
		Precision	0,61	0,50	0,41	0,32	0,22	0,15	
		F-Measure	0,37	0,43	0,42	0,39	0,32	0,24	
2	3,49	Recall	0,29	0,40	0,45	0,51	0,58	0,64	0,46
		Precision	0,63	0,50	0,43	0,33	0,23	0,17	
		F-Measure	0,40	0,44	0,44	0,40	0,33	0,27	
3	3,62	Recall	0,27	0,39	0,45	0,52	0,59	0,65	0,46
		Precision	0,60	0,50	0,43	0,34	0,23	0,16	
		F-Measure	0,37	0,44	0,44	0,41	0,33	0,26	
4	3,55	Recall	0,27	0,38	0,44	0,50	0,57	0,63	0,44
		Precision	0,60	0,49	0,41	0,32	0,22	0,15	
		F-Measure	0,37	0,43	0,42	0,39	0,32	0,24	
5	3,52	Recall	0,29	0,40	0,46	0,52	0,59	0,65	0,45
		Precision	0,62	0,50	0,42	0,32	0,22	0,15	
		F-Measure	0,40	0,44	0,44	0,40	0,32	0,24	
6	3,44	Recall	0,29	0,39	0,46	0,53	0,60	0,65	0,45
		Precision	0,60	0,50	0,43	0,33	0,23	0,17	
		F-Measure	0,39	0,44	0,44	0,41	0,33	0,27	
7	3,47	Recall	0,28	0,41	0,47	0,53	0,60	0,65	0,46
		Precision	0,62	0,51	0,43	0,33	0,23	0,16	
		F-Measure	0,39	0,43	0,45	0,41	0,33	0,26	
8	3,61	Recall	0,27	0,39	0,45	0,51	0,57	0,63	0,45
		Precision	0,62	0,49	0,42	0,33	0,22	0,15	
		F-Measure	0,38	0,43	0,43	0,40	0,32	0,24	
9	3,66	Recall	0,27	0,39	0,45	0,51	0,59	0,64	0,45
		Precision	0,63	0,50	0,42	0,32	0,22	0,15	
		F-Measure	0,38	0,44	0,43	0,39	0,32	0,24	
10	3,62	Recall	0,27	0,38	0,44	0,5	0,57	0,64	0,45
		Precision	0,61	0,49	0,42	0,32	0,22	0,15	
		F-Measure	0,37	0,43	0,43	0,40	0,32	0,24	
1-10	3,57	Recall	0,28	0,39	0,45	0,51	0,58	0,64	0,45
		Precision	0,61	0,50	0,42	0,33	0,22	0,16	
		F-Measure	0,38	0,44	0,43	0,40	0,32	0,26	

5.10 Zusammenfassung

In diesem Kapitel wurden zunächst die besten Parameter für das Finden ähnlicher Artikel oder Abschnitte ermittelt. Um die Qualität der Ähnlichkeitssuche zu bewerten, wurde auf die darauf aufbauende Kategorisierung zurückgegriffen.

Unter den getesteten Algorithmen für die Schlüsselwort-Extrahierung ist der TFIDF-basierte Algorithmus mit frequenzbasierter Gewichtung am besten geeignet und am wenigsten von der Anzahl der Schlüsselwörter abhängig. Etwas überraschend waren vor allem die durchaus konkurrenzfähigen Werte für den einfachen Algorithmus, der immer die ersten Wörter als Schlüsselwörter vorschlägt. Dies lässt sich wahrscheinlich auf die höhere Dichte von relevanten Wörtern im ersten Abschnitt eines Artikels, der eine Definition oder einen Abriss des Themas darstellt, zurückführen. Die Anzahl von 96 Schlüsselwörtern hat sich als beste unter den getesteten herausgestellt. Für fast alle Schlüsselwortalgorithmen waren mit ihr die Werte am höchsten.

Die Ähnlichkeit der gefundenen Artikel in die Berechnung der Gewichtung der Kategorie-Vorschläge einzubeziehen, hat sich als wichtig erwiesen. Einerseits sind die Ergebnisse besser als für das einfache Zählen der Kategorien der ähnlichen Artikel. Andererseits wurde aufgezeigt, wie bedeutend die Anzahl der ähnlichen Artikel ist, die für die Kategorievorschläge genutzt werden. Bei den Tests waren die Ergebnisse bei 20 ähnlichen Artikeln am besten. Wurden weniger oder mehr Artikel in Betracht gezogen, verschlechterte sich die Qualität der Klassifizierung.

Stichproben-Tests haben jedoch gezeigt, dass diese Zahl für einige Anfragen nicht ideal ist. Die Precision ist teilweise stark abhängig davon, wie viele Artikel in Betracht gezogen werden. Für die Kategorisierung unbekannter Texte muss jedoch im Vorfeld eine Entscheidung für eine bestimmte Anzahl getroffen werden. Desto wichtiger ist die ähnlichkeitsbasierte Aufbereitung der Kategorievorschläge. Dadurch sind die Ergebnisse auch bei einer höheren oder niedrigeren Anzahl von ähnlichen Artikeln besser.

In den Versuchen mit verschiedenen Kategorie-Ebenen hat sich gezeigt, dass Wikitech Classify allgemeinere Kategorien höherer Ebenen verlässlicher finden kann als die sehr präzisen Kategorien auf der ersten Ebene. Das Inbetrachtziehen allgemeinerer Kategorien der ähnlichen Artikel, um die Kategorisierung auf der ersten Ebene zu verbessern, hat sich als nicht erfolgversprechend herausgestellt.

Die abschnittsbasierten Tests haben ergeben, dass die Suche mit Abschnitten als Ausgangsdokumente funktioniert und darüber hinaus auch ähnliche Abschnitte gefunden werden können. Die Werte für die Tests, bei denen nur der erste Paragraph als Ausgangsdokument herangezogen wurde, sind nur wenig niedriger als die Werte bei den Tests mit dem gesamten Artikel als Ausgangsdokument. Das gleiche gilt für die Suche *nach* ähnlichen Abschnitten.

Die Klassifizierung ganzer Artikel konnte allerdings nicht über den Umweg der Suche nach ähnlichen Abschnitten verbessert werden. Um ganze Artikel zu kategorisieren, sollte deswegen auch auf einen Index mit Artikeln zurückgegriffen werden.

Die Beurteilung der Qualität der Kategorisierung und damit auch der Ähnlichkeitssuche beschränkt sich bisher strikt darauf, ob genau die richtigen Kategorien gefunden wurden. Wie in Abschnitt 2.4.4 beschrieben, sind die Kategorien aber hierarchisch geordnet. Es ist unmittelbar einzusehen, dass für einen Artikel mit der Kategorie *Deutschland* der Vorschlag *Europa* wesentlich besser ist als zum Beispiel *Bolivien* oder *Südamerika*. Statt alle drei Vorschläge als „gleich falsch“ zu bewerten, sollte die Qualitätsmessung hier toleranter sein.

Es ist somit auch anzunehmen, dass die Ähnlichkeitssuche besser funktioniert, als es die Werte für die Evaluation basierend auf den Kategorien wiedergeben. Sun und Lim (2001) setzen sich mit diesem Problem auseinander und schlagen auch Möglichkeiten für Categoriesysteme in Baumstrukturen vor. Die Struktur der Wikipedia-Kategorien (gerichteter azyklischer Graph) wird ebenfalls angesprochen, jedoch ohne Vorschläge für eine bessere Evaluation zu machen.

Die letzten beiden Versuche sollten allgemein die Leistungsfähigkeit von Wikitech Classify beurteilen. Dabei wurde deutlich, dass das System wesentlich besser ist als der „naive“ Ansatz, stets die häufigsten Kategorien zuzuweisen.

Der Vergleich mit der Arbeit von Gantner/Schmidt-Thieme stellte sich als schwierig heraus, da dort auf stark eingeschränkten Datensätzen gearbeitet wurde. Grundsätzlich ist mit dem dort verwendeten SVM-basierten System wohl eine bessere Qualität für die Klassifizierung erreichbar. Dieser Vorteil wird aber durch geringere Flexibilität und höhere Antwortzeiten erkauft. Das Finden ähnlicher Artikel oder sogar Abschnitte an sich kann dieses System nicht leisten.

Ein einem Versuch mit insgesamt 10 000 Artikeln wurde abschließend gezeigt, dass die Ergebnisse der Evaluation repräsentativ sind.

6

Ausblick und abschließende Zusammenfassung

In diesem Kapitel werden zunächst Anwendungsmöglichkeiten des in dieser Arbeit entwickelten Systems vorgestellt. Anschließend wird auf Möglichkeiten zur Verbesserung des Systems eingegangen. In Abschnitt 6.3 werden Einschränkungen und mögliche Erweiterungen der Vorgehensweise und der Software aufgezeigt. Das Kapitel schließt mit einer kurzen Zusammenfassung der Arbeit.

6.1 Anwendungsmöglichkeiten

Die Evaluation hat ergeben, dass Wikitech Classify die beiden Aufgaben dieser Arbeit – das Finden ähnlicher Abschnitte und das Kategorisieren von neuen Texten – erfüllen kann. Im Folgenden soll eine Reihe konkreterer und weiterer Anwendungsmöglichkeiten beschrieben werden.

Die ähnlichen Abschnitte können Wikipedia-Nutzern als Lesevorschläge und Autoren als Vorschläge für Links in neuen Artikel vorgelegt werden. Wenn neue Texte auf Ähnlichkeit mit bestehenden Texten überprüft werden, kann das zur Vermeidung von doppelten Inhalten beitragen. Autoren, die einzelne Abschnitte eines Artikels ändern, können von dem System eine Liste von Abschnitten erhalten, die in der Folge eventuell auch geändert werden müssen. Wikitech Classify könnte somit zur Einhaltung der Konsistenz innerhalb von Wikis beitragen.

Nachrichtenartikel in Wikipedia-Kategorien einzuordnen oder relevanten Artikeln zuzuweisen, ist eine besonders interessante Möglichkeit. Autoren, aber auch andere interessierte Nutzer, könnten sich dadurch automatisch über Nachrichten und neue Entwicklungen in ihrem Interessengebiet informieren lassen. Aus Sicht der Wikipedia könnte das zu aktuelleren Informationen in den Artikeln beitragen. Umgekehrt könnte das System eingesetzt werden, um Lesern von Internetseiten relevante Wikipedia-Einträge aufzuzeigen.

Wikis werden oft in Firmen zur internen Dokumentation eingesetzt (Barrett, 2008). Wenn zum Beispiel zwei Abteilungen zusammengelegt werden sollen, müssen oft auch deren Wikis zusammengeführt werden. Dabei könnte Wikitech Classify helfen, indem es zu den Artikeln des Wikis der einen Abteilung ähnliche Artikel im Wiki der anderen Abteilung findet. Die Mitarbeiter können dann jeweils entscheiden, ob sie einzelne Artikel oder Abschnitte verwerfen, weil sie schon in dem anderen Wiki vorhanden sind oder an welcher Stelle sie den Inhalt einfügen wollen.

Das System kann auch für die Volltextsuche in MediaWikis genutzt werden. Die Volltextsuche in der Wikipedia basiert zwar bereits auf Lucene, kann aber eventuell durch das Beachten der Struktur der Artikel und des Markups verbessert werden.

Das Finden von domänenunabhängigen Schlüsselwörtern in beliebigen Textdokumenten kann auch für andere Aufgaben eingesetzt werden. Schlüsselwörter können zum Beispiel als einfache Zusammenfassung eines Dokumentes dienen oder in anderen Information-Retrieval-Aufgaben eingesetzt werden (Hulth, 2003).

Die Klassifikation von sehr kurzen Texten wie den Projektbeschreibungen der Ohloh-Datenbank¹ stellt Klassifikationssysteme vor eine besondere Herausforderung. Erste Tests, die nicht Teil dieser Arbeit sind, haben ergeben, dass Wikitech Classify trotz der wenigen Ausgangsdaten ähnliche Wikipedia-Artikel und damit auch Kategorien zu diesen Beschreibungen finden kann.

Der erste Abschnitt eines Artikels soll, wie bereits beschrieben, eine kurze Begriffsdefinition enthalten. Zwischen dem ersten Abschnitt und dem gesamten Artikel beziehungsweise dem Rest des Artikels kann ein Ähnlichkeitswert berechnet werden. Dieser könnte sich als Maß zur Bestimmung der Artikelqualität eignen.

6.2 Verbesserungen

Aus linguistischer Sicht ist vor allem eine bessere Textanalyse während des Indizierens und der Erstellung der Suchanfrage erfolgversprechend. Jenkins und Smith (2005) schlagen einen Stemming-Algorithmus vor, der speziell für die Volltextsuche angepasst wurde. Neben der Möglichkeit des Stemmings kann auch ein Vollformenlexikon oder sogar eine echte morphologische Analyse eingesetzt werden (Hausser, 2000). Das würde wahrscheinlich vor allem die Precision, gegebenenfalls aber sogar den Recall verbessern. Diese Methoden bieten auch den Vorteil, dass sie potentiell zusammengesetzte Wörter – die vor allem im Englischen oft getrennt geschrieben werden – oder auch Eigennamen erkennen könnten.

Eine syntaktische Analyse könnte sogar die Suche nach Phrasen ermöglichen und verschiedene Bedeutungen von Wörtern auflösen. Dadurch wäre es möglich, die eigentliche Bedeutung der Wörter bei der Ähnlichkeitssuche zu beachten und folglich nur noch Dokumente zu finden, die auch wirklich Wörter mit gleichen Bedeutungen wie der des Ausgangsdokuments beinhalten. Oft können zusammengehörende Wörter sogar

1. In dieser Datenbank werden Open-Source-Projekte gesammelt (<http://ohloh.net/>).

direkt aus dem Wikitext der Artikel herausgelesen werden. So kann der Text *fourth-largest City* im Artikel *Cologne* eindeutig als Phrase erkannt werden, da er als Linktext durch entsprechendes Markup eingegrenzt ist.

Wörter, die in Überschriften oder Verweisen vorkommen, fett oder kursiv geschrieben sind, könnten sowohl beim Indizieren als auch bei der Suche – sofern das Ausgangsdokument selbst ein Wikipedia-Artikel ist – besonders gewichtet werden. Dabei kann die Liste der Kandidaten für eine besondere Gewichtung sicherlich noch weitergeführt werden. Hierfür müssen sowohl der Visitor für das Extrahieren des einfachen Textes als auch die Textanalyse erweitert werden.

Die Vergleiche zwischen den verschiedenen Algorithmen zum Extrahieren von Schlüsselwörtern haben gezeigt, dass der auf TFIDF basierende Algorithmus mit Gewichtung nach Frequenz die besten Werte liefert. Der einfachste Algorithmus, der die ersten n Wörter als Schlüsselwörter bestimmt, ist allerdings nur unwesentlich schlechter, wenn genügend viele Schlüsselwörter gewählt werden. Um das System hinsichtlich seiner Geschwindigkeit zu optimieren, könnte es interessant sein, die Antwortzeiten von Simple bei 96 Schlüsselwörtern und GobaITfidfFrequency sowohl bei 96 als auch bei 25 Schlüsselwörtern zu vergleichen. Dabei können sich durchaus Unterschiede ergeben, da Simple keine echte Berechnung für die Schlüsselwörter durchführt und damit wesentlich schneller ist. Dieser Vorteil könnte sich allerdings relativieren, da die Suche länger dauert, wenn eine größere Anzahl Schlüsselwörter genutzt wird.

Die Kategorievorschläge könnten verbessert werden, wenn die Gewichtungen normalisiert werden. Das bedeutet, dass die Gewichtungen nicht nur innerhalb einer Vorschlagsliste für eine bestimmte Suchanfrage die Ordnung angeben, sondern auch zwischen verschiedenen Suchanfragen vergleichbar sind. Der Anwender könnte dann auch anhand des Wertes der Gewichtung ablesen, wie wahrscheinlich ein einzelner Kategorievorschlag auf seine Anfrage zutrifft. Durch Tests ließe sich zudem ein Schwellwert bestimmen, der angibt, ob die Kategorievorschläge noch sinnvoll sind (*SCut*). Alternativ kann auch statistisch bestimmt werden, wie viele der besten Kategorievorschläge genutzt werden sollen, um ein Dokument zu klassifizieren (*RCut*). Diese Werte können eingesetzt werden, wenn eine große Menge von Dokumenten automatisch klassifiziert werden soll (Yang, 2001). Bei einer halbautomatischen Klassifizierung werden die Kategorien letztendlich von einem Fachexperten zugewiesen. Das System unterstützt ihn dabei lediglich mit Vorschlägen. Dieser Experte entscheidet schließlich auch, wie viele Kategorien er zuweist. Bei einer vollständig automatischen Klassifizierung muss jedoch entweder festgelegt sein, wie viele Kategorien zugewiesen werden sollen (*RCut*) oder bis zu welchem Ähnlichkeitswert Kategorien in Frage kommen (*SCut*).

Neben dem inhaltsbasierten Vorgehen für das Finden ähnlicher Dokumente gibt es speziell für Hypertext-Dokumente noch die Möglichkeit, die Links zwischen den Dokumenten in Betracht zu ziehen. Beide Methoden funktionieren und können sich gegenseitig ergänzen, um bessere Ergebnisse zu erzielen (Lu und Getoor, 2003). Dieser Ansatz kann jedoch nur für bestehende, gut verlinkte Artikel eine Verbesserung bringen. Das in Kapitel 3 vorgestellte System von Mihalcea und Csomai (2007) könnte hierbei durch seine automatischen Linkvorschläge hilfreich sein.

Angelova und Weikum (2006) beschreiben, dass sich die Textklassifikation verbessern lässt, indem auch Eigenschaften benachbarter Dokumente herangezogen werden. In der Wikipedia ist die Nachbarschaft explizit durch das Categoriesystem angegeben. In der Arbeit werden außerdem die Verweise zwischen den Dokumenten, deren Autoren und die Bibliographie-Angaben als Maße für die Nachbarschaft vorgeschlagen.

6.3 Einschränkungen und Erweiterungen

Wikitech Classify als Java-Bibliothek ist auf Java-Umgebungen beschränkt. Des Weiteren muss der Index lokal vorliegen. Apache Solr², ein Schwester-Projekt von Lucene, ist eine Client-Server-Architektur und erlaubt Abfragen durch viele Programmiersprachen. Vereinfacht gesagt stellt Solr eine Erweiterung für Lucene dar und bietet eine HTTP/XML-Schnittstelle zur Lucene-API. Der Index kann somit auch zentral zur Verfügung gestellt und gewartet werden. Solr kann des Weiteren zum Beispiel durch Caching intern optimiert und auf mehrere Server verteilt werden (Smiley und Pugh, 2009).

Katta³ und Elastic Search⁴, ebenfalls Open-Source-Projekte, stellen alternative Möglichkeiten zur Verfügung, um Lucene-Indizes im Netzwerk verfügbar zu machen.

Die einzelnen Klassen von Wikitech Classify wurden mithilfe von JUnit getestet⁵ und können als weitestgehend stabil angesehen werden. Die Behandlung von Ausnahmen (*exceptions*) muss noch verbessert werden. Die in dieser Arbeit entwickelte Vorgehensweise lässt sich im Wesentlichen auf die oben genannten Systeme übertragen, was auch vorgeschlagen wird. Der Quellcode kann ebenfalls sowohl server- als auch clientseitig in weiten Teilen genutzt werden.

Der aus Abschnitten aufgebaute Index kann nicht als Quelle für Ausgangsdokumente genutzt werden, da für die Abschnitte kein Wikitext gespeichert wird. In diesem Fall muss auf einen Index aus Artikeln zurückgegriffen werden. Dieser kann sowohl ganze Artikel als auch Abschnitte als Ausgangsdokument liefern (Anhang A.3).

Wikitech Classify ist auf englischsprachige Media-Wikis eingeschränkt. Um das System auch für andere Sprachen zu nutzen, muss die Textanalyse (Abschnitt 4.1.1) für die jeweilige Sprache angepasst werden. Die während der Evaluation erarbeiteten Einstellungen müssen für andere Sprachen eventuell überarbeitet werden.

Das System liefert standardmäßig eine entsprechende Auswahl aus allen Kategorien der Wikipedia zurück. Durch einen Filter für die Kategorievorschläge könnte das System für bestimmte Domänen interessant sein. Bei der Klassifizierung von Nachrichtenbeiträgen wäre zum Beispiel eine Einschränkung auf Kategorien wie Politik, Sport, Wirtschaft und Technik interessant.

2. <http://lucene.apache.org/solr/>

3. <http://katta.sourceforge.net/>

4. <http://www.elasticsearch.org/>

5. JUnit (<http://www.junit.org/>) ist ein Framework, um Java-Quellcode zu testen. Ein bekannter Fehler ist die falsche Berechnung des F-Measures. Die in dieser Arbeit angegebenen F-Maße wurden deswegen nachträglich berechnet.

Es können für neue Artikel und Abschnitte nur Kategorien, denen schon andere Artikel zugeordnet wurden, gefunden werden. Ist der zu kategorisierende Artikel der einzige Kandidat für eine bestimmte Kategorie, ist ein automatischer Vorschlag durch Wikitech Classify unmöglich. Die Kategorie *38 BC establishments* des Artikels Cologne ist ein Beispiel dafür (Abschnitt 2.4.4).

Die Leistungsfähigkeit ist unter anderem abhängig von der Qualität der Volltextsuche. Fang und Zhai (2007) zeigen, dass die Leistungsfähigkeit von Lucene durchaus noch verbessert werden kann.

Die Herausforderung dieser Arbeit war es, ein System zu entwickeln, das Autoren bei ihrer Arbeit helfen kann. Zu neuen und geänderten Textstellen sollten Kategorievorschläge gemacht und die Wikipedia nach ähnlichen Textstellen durchsucht werden. Die Evaluation hat gezeigt, dass das System für diese Aufgaben genutzt werden kann. Zu zeigen, dass diese Vorschläge tatsächlich die Arbeit der Autoren erleichtern und zur Einhaltung der Konsistenz der Wikipedia beitragen können, geht über den Rahmen dieser Arbeit hinaus.

6.4 Zusammenfassung

In dieser Arbeit wurde ein System entwickelt, das die Autoren der Wikipedia bei ihrer Arbeit unterstützen kann, indem es zu beliebigen Texten ähnliche Textstellen in der Wikipedia findet und Kategorien vorschlägt. Dadurch soll es zur Wahrung der Konsistenz und folglich einer höheren Qualität der Wikipedia beitragen.

Das System basiert auf einer Volltextsuchmaschine, um schnelle Antwortzeiten zu erreichen. Dies macht den Aufbau eines Volltextindexes notwendig. Hierfür werden Standardverfahren wie die Normalisierung des Textes, das Entfernen von Stopp-Wörtern und Stemming eingesetzt. Mithilfe eines speziellen Parsers kann das System die Wikipediaartikel zu Abschnitten aufspalten und dadurch eine abschnittsbasierte Suche ermöglichen. Der Parser wird ebenfalls eingesetzt, um das Wikitext-Markup, welches für die Ähnlichkeitssuche störend wäre, aus den Ausgangsdokumenten zu entfernen. In Vorbereitung auf die Ähnlichkeitssuche werden aus den Ausgangsdokumenten gewichtete Schlüsselwörter extrahiert. Aus diesen wird wiederum eine Suchanfrage erstellt. Die Volltextsuche liefert schließlich die ähnlichen Textstellen.

Die Qualität der Ähnlichkeitssuche wurde über die Kategorisierung evaluiert. Die den Artikeln zugewiesenen Kategorien dienten hierbei als Ground Truth. Für die Precision bei $k = 1$ konnte ein Wert von bis zu 0,64 und für die R-Precision ein Wert von bis zu 0,46 erreicht werden. In Anbetracht der großen Anzahl von über 500 000 Kategorien in der Wikipedia sind diese Werte erfolversprechend. Die Kategorisierung kann darüber hinaus selbst als Unterstützung für die Autoren dienen.

Es wurden verschiedene Ansätze angesprochen, um die Leistungsfähigkeit des Systems zu verbessern und einen flexibleren Einsatz zu ermöglichen.

Neben der ursprünglichen Aufgabenstellung kann das System für eine Reihe weiterer Aufgaben, wie das Extrahieren von domänenunabhängigen Schlüsselwörtern aus beliebigen Textdokumenten oder die Klassifizierung von sehr kurzen Texten eingesetzt werden.

Dieses System bringt im Vergleich zu anderen Arbeiten folgende Verbesserungen:

- Das System ist nicht auf bereits vorhandene, Wikipedia-spezifische Strukturelemente wie Links angewiesen. Diese werden vor allem bei neuen Abschnitten häufig noch fehlen. Es arbeitet ausschließlich auf dem textuellen Inhalt.
- Da es auf einer Volltextsuchmaschine basiert, liefert es innerhalb sehr kurzer Zeit Ergebnisse,
- skaliert für den gesamten Inhalt der Wikipedia und
- ermöglicht vergleichsweise einfach das Ändern und Entfernen von Inhalten.
- Neu ist die Möglichkeit, eine verfeinerte Suche auf Abschnittsebene durchzuführen.

Literaturverzeichnis

- B.T. Adler, J. Benterou, K. Chatterjee, L. De Alfaro, I. Pye, und V. Raman. Assigning trust to wikipedia content. In *WikiSym 4th Intl Symposium on Wikis*, 2008.
- R. Angelova und G. Weikum. Graph-based text classification: learn from your neighbors. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 485–492. ACM, 2006.
- D.J. Barrett. *MediaWiki (Wikipedia and Beyond)*. O'Reilly Media, 2008.
- A. Berglund, S. Boag, D. Chamberlin, M.F. Fernandez, M. Kay, J. Robie, und J. Siméon. XML path language (XPath) 2.0. *W3C working draft*, 22, 2003.
- T. Berners-Lee, R. Fielding, und L. Masinter. RFC 3986: Uniform resource identifier (uri): Generic syntax. *The Internet Society*, 2005. URL <http://www.ietf.org/rfc/rfc3986.txt>.
- J.E. Blumenstock. Size matters: word count as a measure of quality on wikipedia. In *Proceeding of the 17th international conference on World Wide Web*, pages 1095–1096. ACM, 2008.
- C. Buckley und E.M. Voorhees. Evaluating evaluation measure stability. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 33–40. ACM, 2000.
- H. Bußmann. *Lexikon der Sprachwissenschaft*. Alfred Kröner Verlag, 3. Auflage, 2002.
- C.C. Chang und C.J. Lin. LIBSVM: a library for support vector machines, 2001. *Software available at* <http://www.csie.ntu.edu.tw/~cjlin/libsvm>, 2001.
- S. Chernov, T. Iofciu, W. Nejdl, und X. Zhou. Extracting semantic relationships between wikipedia categories. In *Proc. of Workshop on Semantic Wikis (SemWiki 2006)*. Citeseer, 2006.

- W. Cunningham und B. Leuf. *The Wiki Way. Quick Collaboration on the Web*. Addison-Wesley, 2001.
- P. Denning, J. Horning, D. Parnas, und L. Weinstein. Wikipedia risks. *Communications of the ACM*, 48(12):152, 2005.
- L. Denoyer und P. Gallinari. The wikipedia xml corpus. In *ACM SIGIR Forum*, volume 40, pages 64–69. ACM, 2006.
- S. DeRose, E. Maler, und R. Daniel. XML pointer language (XPointer), 2000. URL <http://www.w3.org/TR/xptr/>.
- G. Druck, G. Miklau, und A. McCallum. Learning to predict the quality of contributions to wikipedia. *Wikipedia and Artificial Intelligence: An Evolving Synergy*, pages 7–12, 2008.
- H. Fang und C.X. Zhai. Evaluation of the Default Similarity Function in Lucene, 2007. URL http://www.ece.udel.edu/~hfang/lucene/Lucene_exp.pdf.
- Susan Feldman. The high cost of not finding information, 2004. URL <http://www.kmworld.com/articles/readarticle.aspx?articleid=9534>.
- C. Fellbaum. *WordNet: An electronic lexical database*. The MIT press, 1998.
- Jeffrey Friedl. *Mastering regular expressions*. O’Reilly Media, Inc., 3rd edition, 2006.
- Karl-Peter Fuchs, Rainer Hammwöhner, Markus Kattenbeck, und Christian Sax. Qualität der wikipedia. In *Neue Perspektiven im Kontext von Information und Wissen*, 10. Internationales Symposium für Informatikwissenschaft, pages 77–90, 2007.
- E. Gabrilovich und S. Markovitch. Overcoming the brittleness bottleneck using wikipedia: Enhancing text categorization with encyclopedic knowledge. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence*. AAAI Press, 2006.
- E. Gabrilovich und S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 6–12, 2007.
- Z. Gantner und L. Schmidt-Thieme. Automatic content-based categorization of Wikipedia articles. In *Proceedings of the 2009 Workshop on The People’s Web Meets NLP: Collaboratively Constructed Semantic Resources*, pages 32–37. Association for Computational Linguistics, 2009.
- J. Giles. Internet encyclopaedias go head to head. *Nature*, 438(7070): 900–901, 2005.
- A. Halavais und D. Lackaff. An analysis of topical coverage of Wikipedia. *Journal of Computer-Mediated Communication*, 13(2):429–440, 2008.
- R. Hammwöhner. Qualitätsaspekte der wikipedia. *Kommunikation@gesellschaft*, 2007. URL http://www.soz.uni-frankfurt.de/K.G/B3_2007_Hammwoehner.pdf.

- R. Hausser. *Grundlagen der Computerlinguistik: Mensch-Maschine-Kommunikation in natürlicher Sprache*. Springer, 2000.
- M. Hu, E. Lim, A. Sun, H. W. Lauw, und B.-Q. Vuong. Measuring article quality in wikipedia: models and evaluation. In *Conference on Information and Knowledge Management*, pages 243–252. ACM, 2007a.
- M. Hu, E.P. Lim, A. Sun, H.W. Lauw, und B.Q. Vuong. On improving wikipedia search using article quality. In *Proceedings of the 9th annual ACM international workshop on Web information and data management*, pages 145–152. ACM, 2007b.
- A. Hulth. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, volume 10, pages 216–223, 2003.
- U. Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, und T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, 1999. URL <http://www.ietf.org/rfc/rfc2616.txt>.
- M.C. Jenkins und D. Smith. Conservative stemming for search and indexing. In *Proc. ACM SIGIR*, 2005.
- D.D. Lewis, Y. Yang, T.G. Rose, und F. Li. RCV1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.
- A. Lih. Wikipedia as participatory journalism: Reliable sources? Metrics for evaluating collaborative media as a news resource. *Nature*, 2004, 2003.
- E.P. Lim, B.Q. Vuong, H.W. Lauw, und A. Sun. Measuring qualities of articles contributed by online communities. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 81–87. Citeseer, 2006.
- A. Lorenz und J. C. Türp. Die Zahnmedizin in Wikipedia: Eine quantitative und qualitative Untersuchung. *Schweizer Monatsschrift für Zahnmedizin*, 120:1117–1121, December 2010.
- M. Lorenz. Der Trend zum Wikipedia-Beleg. *Forschung & Lehre*, 3/11, 2011.
- Q. Lu und L. Getoor. Link-based classification using labeled and unlabeled data. In *ICML Workshop on The Continuum from Labeled to Unlabeled Data Mining*. Citeseer, 2003.
- C.D. Manning, P. Raghavan, und H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- M. McCandless, E. Hatcher, und O. Gospodnetić. *Lucene in Action*. In Action series. Manning, 2nd edition, July 2010.
- O. Medelyan, D. Milne, C. Legg, und I.H. Witten. Mining meaning from Wikipedia. *International Journal of Human-Computer Studies*, 2009.

- R. Mihalcea und A. Csomai. Wikify!: linking documents to encyclopedic knowledge. In *Conference on Information and Knowledge Management*, pages 233–242. ACM, 2007.
- A. Motro und I. Rakov. Estimating the quality of databases. *Flexible Query Answering Systems*, pages 94–106, 1998.
- I. Mühlhauser und F. Oser. Sind medizinische und gesundheitsinformationen auf den internetseiten von wikipedia evidenzbasiert? – eine inhaltsanalyse. *Zeitschrift für Evidenz, Fortbildung und Qualität im Gesundheitswesen*, July 2008.
- Ch. Müller und I. Gurevych. Using wikipedia and wiktionary in domain-specific information retrieval. In *Conference on Multilingual and Multimodal Information Access Evaluation*, volume 5706 of *Lecture Notes in Computer Science*, pages 219–226. Springer, 2008.
- F.Å. Nielsen et al. Scientific citations in Wikipedia. *First Monday*, 2007.
- S.P. Ponzetto und M. Strube. Deriving a large scale taxonomy from Wikipedia. In *Proceedings of the national conference on artificial intelligence*, volume 22, pages 1440–1445. MIT Press, 2007.
- M.F. Porter, C.J. van Rijsbergen, und S.E. Robertson. New models in probabilistic information retrieval, 1980.
- D. Riehle. How and why wikipedia works: an interview with Angela Beesley, Elisabeth Bauer, and Kizu Naoko. In *Proceedings of the 2006 international symposium on Wikis*, pages 3–8. ACM, 2006.
- T. Saracevic, P. Kantor, A.Y. Chamis, und D. Trivison. A study of information seeking and retrieving. I. Background and methodology. *Journal of the American Society for Information Science*, 39(3):161–176, 1988.
- P. Schönhofen. Identifying document topics using the wikipedia category network. *Web Intelligence and Agent Systems*, 7(2):195–207, 2009.
- F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.
- D. Smiley und E. Pugh. *Solr 1.4 Enterprise Search Server*. From Technologies to Solutions. Packt, August 2009.
- K. Stein und C. Hess. Does it matter who contributes: a study on featured articles in the german wikipedia. In *Proceedings of the eighteenth conference on Hypertext and hypermedia*, page 174. ACM, 2007.
- M. Strube. Wikipedia: Wissen für die künstliche intelligenz. *Spektrum der Wissenschaft*, Dezember 2010:94–102, 2010.
- M. Strube und S. P. Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence*. AAAI Press, 2006.
- B. Stvilia, M. B. Twidale, L. C. Smith, und L. Gasser. Assessing information quality of a community-based encyclopedia. In *IQ*. MIT, 2005a.

- B. Stvilia, M.B. Twidale, L. Gasser, und L. Smith. Information quality discussions in Wikipedia. In *Proceedings of the 2005 international conference on knowledge management*, pages 101–113. Citeseer, 2005b.
- B. Suh, G. Convertino, E.H. Chi, und P. Pirolli. The Singularity is Not Near: Slowing Growth of Wikipedia. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration*, pages 1–10. ACM, 2009.
- A. Sun und E.P. Lim. Hierarchical text classification and evaluation. In *icdm*, pages 521–528. Published by the IEEE Computer Society, 2001.
- J. Voss. Measuring wikipedia. In *International Conference of the International Society for Scientometrics and Informetrics*, pages 221–231, 2005.
- P. Wang, J. Hu, H.J. Zeng, L. Chen, und Z. Chen. Improving text classification by using encyclopedia knowledge. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 332–341. IEEE, 2007.
- Wikipedia contributors. Wikipedia:database download, January 2010. URL http://en.wikipedia.org/w/index.php?title=Wikipedia:Database_download&oldid=405929677. Abgerufen am 2011-01-05.
- Wikipedia contributors. Wikipedia:Categorization, January 2011a. URL <http://en.wikipedia.org/w/index.php?title=Wikipedia:Categorization&oldid=405858622>. Abgerufen am 2011-01-05.
- Wikipedia contributors. Wikipedia:Featured articles, January 2011b. URL http://en.wikipedia.org/w/index.php?title=Wikipedia:Featured_articles&oldid=412068672. Abgerufen am 2011-01-05.
- Wikipedia contributors. Wikipedia:Good articles, January 2011c. URL http://en.wikipedia.org/w/index.php?title=Wikipedia:Good_articles&oldid=406132194. Abgerufen am 2011-01-05.
- Wikipedia contributors. Wikipedia:Namespace, January 2011d. URL <http://en.wikipedia.org/w/index.php?title=Wikipedia:Namespace&oldid=405608007>. Abgerufen am 2011-01-05.
- Wikipedia contributors. Wikipedia:What is an article?, January 2011e. URL http://en.wikipedia.org/w/index.php?title=Wikipedia:What_is_an_article%3F&oldid=402529880. Abgerufen am 2011-01-05.
- W.J. Wilbur und K. Sirotkin. The automatic identification of stop words. *Journal of information science*, 18(1):45, 1992.
- Y. Yang. A study of thresholding strategies for text categorization. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 137–145. ACM, 2001.
- Y. Yang, J. Zhang, und B. Kisiel. A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, page 103. ACM, 2003.

- T. Zesch und I. Gurevych. Analysis of the wikipedia category graph for NLP applications. In *Proceedings of the TextGraphs-2 Workshop (NAACL-HLT 2007)*, pages 1–8, 2007.
- T. Zesch, C. Müller, und I. Gurevych. Extracting lexical semantic knowledge from wikipedia and wiktionary. In *Proceedings of the Conference on Language Resources and Evaluation (LREC)*, pages 08–07. Citeseer, 2008.
- X. Zhu und S. Gauch. Incorporating quality metrics in centralized/distributed information retrieval on the World Wide Web. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 288–295. ACM, 2000.

A.1 Inhalt der beigelegten CD

Da für die Übersetzung des Quellcodes ein Zugriff auf das Maven-Repository¹ der Open Source Research Group notwendig ist, lässt sich Wikitech Classify nicht außerhalb dieser Arbeitsumgebung übersetzen. Die beigelegte CD enthält deswegen neben dem Quellcode auch ein bereits vollständig lauffähiges Java-Archiv.

Ein Index der gesamten englischsprachigen Wikipedia ist mit über 30 GB zu groß, um auf dieser CD mitgeliefert zu werden. Um Wikitech Classify trotzdem testen zu können, wird ein kleinerer Beispielindex mitgeliefert. Dieser Index enthält alle Artikel, die in dieser Arbeit als Beispiele genutzt werden sowie alle Artikel aus den Testlisten. Zusätzlich wurden weitere, zufällig ausgewählte Artikel verwendet. Insgesamt enthält der Index knapp 50 000 Artikel. Er enthält keine Kategorienseiten. Deswegen können nur Kategorien der ersten Ebene gefunden werden.

Die Tabelle A.1 gibt einen Überblick über den Inhalt der CD.

Tabelle A.1: Inhalt der beigelegten CD

Pfad	Beschreibung
ma.pdf	diese Magisterarbeit im PDF-Format
readme.pdf	dieses Kapitel im PDF-Format
classify.jar	lauffähiges Java-Archiv
articleindex/	Beispielindex
articlelists/	Artikellisten für den Beispielindex
source/scripts/	Skripte
source/wikitech/classify	Wikitech-Classify-Quellcode

Im Folgenden wird zunächst eine mögliche Vorgehensweise zur Installation auf einem Linux-PC vorgestellt. Anschließend wird anhand von Beispielen die Funktionsweise der Wikitech-Classify-CLI erläutert.

A.2 Installation

Voraussetzung für die Nutzung von Wikitech Classify ist eine Java-

1. Mit Apache Maven (<http://maven.apache.org/>) werden Software-Projekte verwaltet.

Laufzeitumgebung (Java Runtime Environment, kurz JRE). Bisher wurde es nur mit der aktuellen Version 6 von Oracle² und IcedTea 6³ getestet. Das Java-Archiv kann durch den folgenden Aufruf ausgeführt werden.

Listing A.1: Aufruf des Wikitech-Classify-Archivs

```
$ java -jar wikitech-classify.jar
> usage: <program> print|index|remove|update|keywords|query|
similar|categories|evaluate|random [OPTIONS]
```

Ohne die Angabe von Argumenten lieferte es eine Liste der zur Verfügung stehenden Aufgaben. Voraussetzung ist, dass das Archiv im aktuellen Verzeichnis liegt.

Die Aufgaben entsprechen den Anwendungsmöglichkeiten des Programmes und müssen immer als erstes Argument übergeben werden. Sie werden in Abschnitt A.3 vorgestellt.

Um den Aufruf flexibler zu gestalten, bietet es sich an, ein einfaches Shell-Skript zu erstellen, das den oben genannten Aufruf ersetzt. Ein Beispiel dafür ist auf der CD in der Datei `source/scripts/classify` enthalten. Darin muss der Pfad zum Java-Archiv angepasst werden. Das Skript sollte in ein Verzeichnis des Suchpfades (zum Beispiel `/usr/share/bin`) kopiert werden und ausführbar sein.

Listing A.2: Skript für den einfachen Aufruf des Wikitech-Classify-Archivs

```
#!/bin/sh
java -jar /opt/wikitech/wikitech-classify.jar "$@"
```

Wikitech Classify kann automatisch auf den Index zurückgreifen, der durch die Umgebungsvariable `INDEXDIR` definiert ist. Der folgende Befehl setzt die Umgebungsvariable.

Listing A.3: Setzen der Umgebungsvariable für den Index

```
$ export INDEXDIR=/path/to/articleindex
```

A.3 Bedienung

Das Programm ist in Aufgaben unterteilt. Der Name der Aufgabe muss als erstes Argument an das Programm übergeben werden. Wird das Programm ohne Aufgabe aufgerufen, gibt es eine Liste der Aufgaben aus und beendet sich.

Für jede Aufgabe kann sich der Anwender eine Hilfe ausgeben lassen, indem er das Argument `-help` zum Programmaufruf hinzufügt. Die Hilfe listet alle möglichen Argumente (Schalter und Einstellungsmöglichkeiten) für die jeweilige Aufgabe. Die meisten Argumente sind mit Standardwerten belegt und können gegebenenfalls weggelassen werden.

Indizieren

Für den Aufbau eines Indexes steht die Aufgabe `index` zur Verfügung. Im folgenden Listing werden zwei Artikel direkt aus der Online-Version der Wikipedia (`-online`) geholt und in einen Index geschrieben. Der Index soll unter `/tmp/testindex/` angelegt werden. Die Namen der Artikel werden über das Argument `-identifizier` übergeben.

2. <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
3. http://icedtea.classpath.org/wiki/Main_Page

Listing A.4: CLI:
Indizieren zweier Artikel
aus der
englischsprachigen
Wikipedia

```
$ classify index -online http://en.wikipedia.org/w/ -identifizier \
    "Wallace_Neff" -targetindex /tmp/testindex/
$ classify index -online http://en.wikipedia.org/w/ -identifizier \
    "Java_(programming_language)" -targetindex /tmp/testindex/
```

Bei Änderungen können Dokumente entfernt (remove) oder überschrieben (update) werden. Im folgenden Beispiel wird in der ersten Zeile ein Artikel überschrieben und in der zweiten Zeile ein Artikel entfernt. Sowohl index als auch update und remove können mit dem Argument -idfile eine Datei mit einer Liste von Artikel- oder Abschnittsnamen übergeben werden.

Listing A.5: CLI:
Dokumente erneuern
oder aus dem Index
entfernen

```
$ classify update -identifizier "Java_(programming_language)" \
    -online http://en.wikipedia.org/w/
$ classify remove -identifizier "Wallace_Neff" \
    -online http://en.wikipedia.org/w/
```

Finden ähnlicher Artikel und Abschnitte

Die Aufgabe similar veranlasst classify, nach ähnlichen Dokumenten im Index zu suchen. Das Ausgangsdokument muss dabei selbst im Index enthalten sein. Abschnitte können als Ausgangsdokument angegeben werden, wenn deren Artikel im Index enthalten ist. Im Beispiel wird mit dem ersten Aufruf nach ähnlichen Artikeln zu *Java (programming language)* gesucht. Beim zweiten Aufruf wird nach ähnlichen Paragraph-Abschnitten und beim dritten nach ähnlichen Abschnitten gesucht, die durch Teilung der Artikel auf der zweiten Ebene der Knoten erfolgt ist. Die Abschnitte werden dabei zur Laufzeit gebildet. Die Ähnlichkeitssuche in einem Index, der bereits aus Abschnitten aufgebaut wurde, wird in den nächsten beiden Abschnitten erläutert. Der vierte Aufruf veranlasst classify, nur einen Abschnitt des Artikels als Ausgangsdokument für die Ähnlichkeitssuche zu verwenden. Der Schalter -usetermweight bewirkt, dass die Gewichtung der Schlüsselwörter in die Suchanfrage aufgenommen wird. Das Programm gibt die gebildete Suchanfrage auf dem Bildschirm aus und liefert eine Liste der ähnlichen Dokumente. Es gibt weitere Argumente, die zum Beispiel festlegen, wie viele Ergebnisse zurückgeliefert werden oder welcher Schlüsselwortalgorithmus genutzt werden soll. Standardmäßig liefert similar zehn Dokumente und nutzt den TFIDF-basierten Algorithmus mit Gewichtung nach Frequenz.

Listing A.6: CLI: Finden
ähnlicher Artikel und
Abschnitte

```
$ classify similar -identifizier "Java_(programming_language)" \
    -usetermweight
$ classify similar -identifizier "Java_(programming_language)" \
    -usetermweight -elementsplitter Paragraph
$ classify similar -identifizier "Java_(programming_language)" \
    -usetermweight -levelsplitter 2
$ classify similar -identifizier \
    "Java_(programming_language)#/Section[0]/Paragraph[2]/" \
    -usetermweight -elementsplitter Paragraph
```

Mithilfe von Tika kann classify auch andere Ausgangsdokumente wie PDF-Dateien verwenden. Statt -identifizier wird dafür das Argument -tika genutzt.

Listing A.7: CLI:
PDF-Datei als
Ausgangsdokument für
die Ähnlichkeitssuche

```
$ classify similar -tika beispiel.pdf
```

Aufbau eines Indexes mit Abschnitten

Für den Aufbau eines Indexes mit Abschnitten eignet sich ein bereits vorhandener Index mit Artikeln als Quelle (Argument `-indexdir`). Mit dem Argument `-targetdir` wird angegeben, wo der neue Index aufgebaut werden soll. Über das Argument `-elementsplitter` wird in diesem Fall festgelegt, dass der neue Index aus Paragraph-Abschnitten bestehen soll. Der Aufgabe muss eine Datei übergeben werden, die eine Liste mit Artikelnamen enthält, deren Abschnitte in den Index aufgenommen werden sollen (Argument `-idfile`).

Listing A.8: CLI: Aufbau eines Indexes mit Abschnitten

```
$ classify index -indexdir articleindex/index \
               -targetindex abschnittsindex/index/ \
               -elementsplitter Paragraph \
               -idfile articlelists/random_articles.list
```

Ähnlichkeitssuche in einem Index mit Abschnitten

Der aus Abschnitten aufgebaute Index kann nicht als Quelle für Ausgangsdokumente genutzt werden, da für die Abschnitte kein Wikitext gespeichert wird. In diesem Fall muss auf einen Index aus Artikeln zurückgegriffen werden. Dieser kann sowohl ganze Artikel als auch Abschnitte als Ausgangsdokument liefern.

Im folgenden Beispiel werden ähnliche Abschnitte zu einem einzelnen Abschnitt gesucht. Dafür muss der zu durchsuchende Index explizit durch das Argument `-searchindex` angegeben werden, da er vom Index für das Ausgangsdokument abweicht.

Listing A.9: CLI: Finden ähnlicher Abschnitte in einem Index mit Abschnitten

```
$ classify similar -indexdir articleindex/index/ \
                 -searchindexdir abschnittsindex/index -identifier \
                 "Java_(programming_language)#/Section[0]/Paragraph[2]/" \
```

Kategorisierung

Für die Kategorisierung stellt `classify` die Aufgabe `categories` zur Verfügung. Das Argument `-usesimilarityweight` legt fest, dass der Ähnlichkeitswert der gefundenen Dokumente in die Berechnung der Kategorienvorschläge eingehen soll. Beim ersten Aufruf werden die Kategorien anhand ähnlicher Artikel und beim zweiten Aufruf anhand ähnlicher Abschnitte berechnet.

Listing A.10: CLI: Kategorisierung

```
$ classify categories -identifier "Java_(programming_language)" \
                   -usetermweight -usesimilarityweight
$ classify categories -identifier "Java_(programming_language)" \
                   -usetermweight -usesimilarityweight \
                   -elementsplitter Paragraph
```

Schlüsselwörter und Volltextsuche

Wikitech Classify basiert auf einer Volltextsuche und benötigt Schlüsselwörter, um die Anfrage für die Ähnlichkeitssuche zu bilden. Beide Aufgaben lassen sich auch eigenständig nutzen. Die erste Zeile des folgenden Listings lässt `classify` die ersten 10 Schlüsselwörter ausgeben. In

der zweiten Zeile wird nach Artikeln gesucht, die die Wörter *java* und *datatypes* enthalten.

*Listing A.11: CLI:
Schlüsselwörter und
Volltextsuche*

```
$ classify keywords -identifier "Java_(programming_languages)" \  
-maxkeywords 10  
$ classify query -query "java_datatypes"
```

Evaluation

Mithilfe der Aufgabe `random` liefert `classify` eine Liste zufälliger Artikel. Dabei werden nur Artikel geliefert, die mindestens 200 Zeichen enthalten und in mindestens einer Kategorie sind. Das Argument `-numdocs` legt fest, wie viele Artikel gefunden werden sollen. Als Quelle für die Artikel wird standardmäßig der Index, der über die Umgebungsvariable festgelegt wurde, genutzt. Die Artikel werden auf *stdout* und sonstige Meldungen auf *stderr* ausgegeben. Im Beispiel werden die Artikel in die Datei `test.list` umgeleitet.

*Listing A.12: CLI:
Zufällige Artikel
ausgeben lassen*

```
$ classify random -numdocs 200 > test.list
```

Die Aufgabe `evaluate` sucht für einen Artikel oder Abschnitt nach Kategorien und berechnet Recall und Precision bei verschiedenen *k*. Die Werte werden durch Kommata getrennt auf *stdout* ausgegeben. Um eine Liste von Dokumenten zu evaluieren, muss `classify` entsprechend oft aufgerufen werden⁴.

*Listing A.13: CLI:
Evaluation*

```
$ cat test.list | xargs classify evaluate \  
-usetermweight -usesimilarityweight \  
-maxkeywords 96 -numdocs 20 \  
-identifier > results.csv
```

Für die Versuche während der Evaluation wurden Shell-Skripte verwendet, um diese Aufgabe zu automatisieren. Ein Beispiel kann unter `/source/scripts/` gefunden werden. `eval036` evaluiert eine Liste von Artikeln, die als Liste in einer Datei stehen.

4. Im Beispiel übernimmt das Programm `xargs` (<http://www.gnu.org/software/findutils/>) die Aufgabe, `classify` wiederholt aufzurufen.

B

Nutzung der Wikitech-Classify-API (Beispiele)

Im Folgenden ist der Quellcode von Klassen gelistet, die die entsprechenden Aufgaben der Wikitech-Classify-CLI übernehmen. Dieser Quellcode dient auch als Beispiel, wie die API von Wikitech Classify genutzt werden kann.

B.1 Index

Listing B.1: Index.java

```
1 package de.fau.phil.stud.spguleis.classify.cli;
2
3 import java.io.File;
4 import java.util.List;
5
6 import org.apache.commons.cli.CommandLine;
7 import org.apache.commons.cli.HelpFormatter;
8 import org.apache.commons.cli.Options;
9 import org.apache.commons.cli.ParseException;
10 import org.apache.commons.cli.Parser;
11 import org.apache.commons.cli.PosixParser;
12
13 import de.fau.phil.stud.spguleis.classify.NodeAnalyzer;
14 import de.fau.phil.stud.spguleis.classify.NodeIdentifier;
15 import de.fau.phil.stud.spguleis.classify.NodeIndexer;
16 import de.fau.phil.stud.spguleis.classify.NodeSource;
17 import de.fau.phil.stud.spguleis.classify.NodeSplitter;
18 import de.fau.phil.stud.spguleis.classify.Plaintext;
19 import de.fau.phil.stud.spguleis.classify.cli.options.
    AnalyzerOption;
20 import de.fau.phil.stud.spguleis.classify.cli.options.CacheOption
    ;
21 import de.fau.phil.stud.spguleis.classify.cli.options.HelpOption;
22 import de.fau.phil.stud.spguleis.classify.cli.options.
    IdentifierListFileOption;
23 import de.fau.phil.stud.spguleis.classify.cli.options.
    IdentifierNameOption;
```

```

24 import de.fau.phil.stud.spguleis.classify.cli.options.
    IdentifierOption;
25 import de.fau.phil.stud.spguleis.classify.cli.options.
    NodeSourceOption;
26 import de.fau.phil.stud.spguleis.classify.cli.options.
    OptimizeOption;
27 import de.fau.phil.stud.spguleis.classify.cli.options.
    RedirectOption;
28 import de.fau.phil.stud.spguleis.classify.cli.options.
    SplitterOption;
29 import de.fau.phil.stud.spguleis.classify.cli.options.
    TargetIndexOption;
30
31 public class Index
32 {
33     public static void run(String args[]) throws Exception
34     {
35         CommandLine cmd = getCommandLine(args);
36         NodeSource nodeSource = NodeSourceOption.getValue(
            cmd);
37         File target = TargetIndexOption.getValue(cmd);
38         NodeAnalyzer analyzer = AnalyzerOption.getValue(
            cmd);
39
40         NodeIndexer indexer = new NodeIndexer(nodeSource,
            target, Plaintext.propertyPlaintext, analyzer);
41
42         List<NodeIdentifier> identifiers =
            IdentifierOption.getValue(cmd);
43         boolean optimize = OptimizeOption.getValue(cmd);
44         boolean redirect = RedirectOption.getValue(cmd);
45         NodeSplitter splitter = SplitterOption.getValue(
            cmd);
46
47         indexer.index(identifiers, optimize, redirect,
            splitter);
48
49         indexer.close();
50     }
51
52     private static CommandLine getCommandLine(String[] args)
        throws ParseException
53     {
54         // dependencies
55         Options opts = new Options();
56
57         HelpOption.appendOptions(opts);
58         TargetIndexOption.appendOptions(opts);
59         AnalyzerOption.appendOptions(opts);
60         IdentifierNameOption.appendOptions(opts);
61         IdentifierListFileOption.appendOptions(opts);
62         CacheOption.appendOptions(opts);
63         OptimizeOption.appendOptions(opts);
64         RedirectOption.appendOptions(opts);
65         NodeSourceOption.appendOptions(opts);
66         SplitterOption.appendOptions(opts);
67
68         Parser parser = new PosixParser();

```

```

69         CommandLine cmd = parser.parse(opts, args);
70
71         if(HelpOption.getValue(cmd))
72         {
73             HelpFormatter formatter = new HelpFormatter
74                 ();
75             formatter.printHelp("programm_index", opts)
76                 ;
77             System.exit(0);
78         }
79         return cmd;
80     }
81 }

```

B.2 Similar

Listing B.2: Similar.java

```

1 package de.fau.phil.stud.spguleis.classify.cli;
2
3 import java.io.File;
4
5 public class Similar
6 {
7     public static void run(String args[]) throws Exception
8     {
9         CommandLine cmd = getCommandLine(args);
10
11         KeywordExtractor kwe = KeywordExtractorOption.
12             getValue(cmd);
13         NodeAnalyzer analyzer = AnalyzerOption.getValue(
14             cmd);
15         File indexedirectory = IndexdirOption.getValue(cmd)
16             ;
17         File searchindexedirectory = IndexdirSearchOption.
18             getValue(cmd);
19         NodeSplitter splitter = SplitterOption.getValue(
20             cmd);
21
22         FindSimilar similar = new FindSimilar(kwe,
23             analyzer, indexedirectory, searchindexedirectory,
24             splitter);
25
26         int maxResults = NumdocsOption.getValue(cmd);
27         boolean useTermWeight = UseTermWeightOption.
28             getValue(cmd);
29
30         List<Result> results;
31         if(TextOption.getValue(cmd) != null &&
32             IdentifierNameOption.getValue(cmd) != null)
33         {
34             throw new MissingArgumentException("You_
35                 must_specify_either_--text_or_--
36                 textfile_or_--identifier_or_--tika");
37         }
38         else if(TextOption.getValue(cmd) != null &&
39             IdentifierNameOption.getValue(cmd) != null)

```

```

28         {
29             throw new MissingArgumentException("You
                must specify either --text or --
                textfile or --identifier or -tika (not
                2 or more of them)");
30         }
31         else if(TextOption.getValue(cmd) != null)
32         {
33             String text = TextOption.getValue(cmd);
34             results = similar.find(text, maxResults,
                useTermWeight);
35         }
36         else
37         {
38             NodeIdentifier identifier =
                IdentifierNameOption.getValue(cmd);
39             boolean removeOrigin = false;
40             results = similar.find(identifier,
                maxResults, useTermWeight, removeOrigin
                );
41         }
42
43         for(Result result : results)
44         {
45             System.out.printf("%s, %s\n", result.
                getIdentifier(), result.getScore());
46         }
47     }
48
49     private static CommandLine getCommandLine(String[] args)
        throws ParseException
50     {
51         // dependencies
52         Options opts = new Options();
53
54         HelpOption.appendOptions(opts);
55         KeywordExtractorOption.appendOptions(opts);
56         AnalyzerOption.appendOptions(opts);
57         IndexdirOption.appendOptions(opts);
58         IndexdirSearchOption.appendOptions(opts);
59         SplitterOption.appendOptions(opts);
60         NumdocsOption.appendOptions(opts);
61         UseTermWeightOption.appendOptions(opts);
62         TextOption.appendOptions(opts);
63         IdentifierNameOption.appendOptions(opts);
64
65         Parser parser = new PosixParser();
66         CommandLine cmd = parser.parse(opts, args);
67
68         if(HelpOption.getValue(cmd))
69         {
70             HelpFormatter formatter = new HelpFormatter
                ();
71             formatter.printHelp("programm similar",
                opts);
72             System.exit(0);
73         }
74
75         return cmd;

```

```

76     }
77 }

```

B.3 Categories

Listing B.3:
Categories.java

```

1 package de.fau.phil.stud.spguleis.classify.cli;
2
3 import java.io.File;
4 import java.util.List;
5
6 import org.apache.commons.cli.CommandLine;
7 import org.apache.commons.cli.HelpFormatter;
8 import org.apache.commons.cli.MissingOptionException;
9 import org.apache.commons.cli.Options;
10 import org.apache.commons.cli.ParseException;
11 import org.apache.commons.cli.Parser;
12 import org.apache.commons.cli.PosixParser;
13
14 import de.fau.phil.stud.spguleis.classify.FindCategories;
15 import de.fau.phil.stud.spguleis.classify.FindSimilar;
16 import de.fau.phil.stud.spguleis.classify.KeywordExtractor;
17 import de.fau.phil.stud.spguleis.classify.NodeAnalyzer;
18 import de.fau.phil.stud.spguleis.classify.NodeIdentifier;
19 import de.fau.phil.stud.spguleis.classify.NodeSearcher;
20 import de.fau.phil.stud.spguleis.classify.NodeSplitter;
21 import de.fau.phil.stud.spguleis.classify.NodeSearcher.Result;
22 import de.fau.phil.stud.spguleis.classify.cli.options.
    AnalyzerOption;
23 import de.fau.phil.stud.spguleis.classify.cli.options.
    DontRemoveOrigin;
24 import de.fau.phil.stud.spguleis.classify.cli.options.HelpOption;
25 import de.fau.phil.stud.spguleis.classify.cli.options.
    IdentifierNameOption;
26 import de.fau.phil.stud.spguleis.classify.cli.options.
    IndexdirOption;
27 import de.fau.phil.stud.spguleis.classify.cli.options.
    IndexdirSearchOption;
28 import de.fau.phil.stud.spguleis.classify.cli.options.
    KeywordExtractorOption;
29 import de.fau.phil.stud.spguleis.classify.cli.options.
    MaxDepthOption;
30 import de.fau.phil.stud.spguleis.classify.cli.options.
    NumdocsOption;
31 import de.fau.phil.stud.spguleis.classify.cli.options.QueryOption
    ;
32 import de.fau.phil.stud.spguleis.classify.cli.options.
    SplitterOption;
33 import de.fau.phil.stud.spguleis.classify.cli.options.
    StartDepthOption;
34 import de.fau.phil.stud.spguleis.classify.cli.options.TextOption;
35 import de.fau.phil.stud.spguleis.classify.cli.options.
    UseDepthWeightHalfOption;
36 import de.fau.phil.stud.spguleis.classify.cli.options.
    UseDepthWeightLinearOption;
37 import de.fau.phil.stud.spguleis.classify.cli.options.
    UseSimilarityWeightOption;

```

```

38 import de.fau.phil.stud.spguleis.classify.cli.options.
    UseTermWeightOption;
39
40 public class Categories
41 {
42     public static void run(String[] args) throws Exception
43     {
44         CommandLine cmd = getCommandLine(args);
45
46         // get the source text
47         String queryString;
48         String similarText;
49         NodeIdentifier similarIdentifier;
50         try
51         {
52             queryString = QueryOption.getValue(cmd);
53         }
54         catch (MissingOptionException e)
55         {
56             queryString = null;
57         }
58         similarText = TextOption.getValue(cmd);
59         similarIdentifier = IdentifierNameOption.getValue(
            cmd);
60         if (queryString == null && similarText == null &&
            similarIdentifier == null)
61         {
62             throw new MissingOptionException(
63                 "You must specify either -query-
64                 queryFile-
65                 -text-textFile-identifier"
66                 );
67
68         // get similar documents
69         List<Result> similarNodes;
70         NodeAnalyzer analyzer = AnalyzerOption.getValue(
            cmd);
71         NodeSplitter splitter = SplitterOption.getValue(
            cmd);
72         File indexDirectory = IndexdirOption.getValue(cmd)
            ;
73         File indexSearchDirectory = IndexdirSearchOption.
            getValue(cmd);
74         int numDocs = NumdocsOption.getValue(cmd);
75         if (queryString != null && similarText == null &&
            similarIdentifier == null)
76         {
77             NodeSearcher nodeSearcher = new
                NodeSearcher(indexDirectory, analyzer,
                splitter);
78             similarNodes = nodeSearcher.search(
                queryString, numDocs);
79         }
80         else if (queryString == null)
81         {
            KeywordExtractor kwe =
                KeywordExtractorOption.getValue(cmd);

```

```

82         boolean useTermWeight = UseTermWeightOption.
83             getValue(cmd);
84         FindSimilar findSimilar = new FindSimilar(
85             kwe, analyzer, indexDirectory,
86             indexSearchDirectory, splitter);
87
88         if (similarText != null &&
89             similarIdentifier == null)
90         {
91             similarNodes = findSimilar.find(
92                 similarText, numDocs,
93                 useTermWeight);
94         }
95         else if (similarText == null &&
96             similarIdentifier != null)
97         {
98             boolean removeOrigin;
99             if (DontRemoveOrigin.getValue(cmd)
100                 == true)
101             {
102                 removeOrigin = false;
103             }
104             else
105             {
106                 removeOrigin = true;
107             }
108             similarNodes = findSimilar.find(
109                 similarIdentifier, numDocs,
110                 useTermWeight, removeOrigin);
111         }
112         else
113         {
114             throw new MissingOptionException(
115                 "You must specify either -"
116                 + query + "-queryFile" +
117                 "-text -textFile -"
118                 + identifier + "-tika")
119                 ;
120         }
121     }
122     else
123     {
124         throw new MissingOptionException(
125             "You must specify either -query -"
126             + queryFile +
127             "-text -textFile -identifier -"
128             + tika);
129     }
130 }
131
132 // find categories based on similar documents
133 FindCategories findCategories = new FindCategories
134     (indexDirectory);
135
136 boolean useSimilarityWeight =
137     UseSimilarityWeightOption.getValue(cmd);
138 int startDepth = StartDepthOption.getValue(cmd);
139 int maxDepth = MaxDepthOption.getValue(cmd);

```

```

123         boolean useDepthWeightLinear =
124             UseDepthWeightLinearOption.getValue(cmd);
125         boolean useDepthWeightHalf =
126             UseDepthWeightHalfOption.getValue(cmd);
127         System.err.println("Finding categories based on: "
128             + similarNodes.toString());
129         List<Result> results = findCategories.find(
130             similarNodes, useSimilarityWeight,
131             useDepthWeightLinear,
132             useDepthWeightHalf, startDepth,
133             maxDepth);
134
135         for (Result result : results)
136         {
137             System.out.printf("%s, %s\n", result.
138                 getIdentifier(), result.getScore());
139         }
140     }
141
142     private static CommandLine getCommandLine(String[] args)
143         throws ParseException
144     {
145         // dependencies
146         Options opts = new Options();
147
148         HelpOption.appendOptions(opts);
149         AnalyzerOption.appendOptions(opts);
150         IdentifierNameOption.appendOptions(opts);
151         IndexdirOption.appendOptions(opts);
152         IndexdirSearchOption.appendOptions(opts);
153         KeywordExtractorOption.appendOptions(opts);
154         MaxDepthOption.appendOptions(opts);
155         StartDepthOption.appendOptions(opts);
156         NumdocsOption.appendOptions(opts);
157         QueryOption.appendOptions(opts);
158         SplitterOption.appendOptions(opts);
159         TextOption.appendOptions(opts);
160         UseSimilarityWeightOption.appendOptions(opts);
161         UseTermWeightOption.appendOptions(opts);
162         DontRemoveOrigin.appendOptions(opts);
163         UseDepthWeightLinearOption.appendOptions(opts);
164         UseDepthWeightHalfOption.appendOptions(opts);
165
166         Parser parser = new PosixParser();
167         CommandLine cmd = parser.parse(opts, args);
168
169         if (HelpOption.getValue(cmd))
170         {
171             HelpFormatter formatter = new HelpFormatter
172                 ();
173             formatter.printHelp("programm categories",
174                 opts);
175             System.exit(0);
176         }
177
178         return cmd;
179     }
180 }

```



Persönliche Daten

- geboren am 22. August 1983 in Berlin
- wohnhaft in Forstacker 1, 91341 Röttenbach
- deutsch, ledig, eine Tochter

Ausbildung

- *1990 – 1994*: Grundschule Audenhain
- *1994 – 2002*: Johann-Walter-Gymnasium Torgau
- *Oktober 2003 – September 2005*: Ausbildung zum Industrietechnologen für Automatisierungstechnik an der Siemens Technik Akademie Erlangen
- *Oktober 2005 – Februar 2006*: Studium der Informationstechnik an der Fachhochschule Nürnberg (ohne Abschluss)
- *seit April 2006*: Studium der Linguistischen Informatik, Germanistischen Linguistik und Informatik an der Friedrich-Alexander-Universität Erlangen-Nürnberg

Berufserfahrung

- *2003*: Zivildienstleistender im Kreiskrankenhaus Wurzen
- *Oktober 2005 – Mai 2008*: Werkstudent bei Siemens IT Operations (Proxyteam)
- *seit Januar 2006*: selbständiger Programmierer
- *seit Oktober 2007*: Übungsbetreuer an der Siemens Technik Akademie
- *Mai 2008 – November 2008*: Praktikant bei Siemens Medical Solutions USA (CAD and Knowledge Solutions Group)
- *November 2008 – April 2010*: Werkstudent bei Siemens IT Operations (Mailteam)

D

Wahrheitsgemäße Erklärung

Ich erkläre hiermit, dass ich

- die eingereichte Magisterarbeit selbständig und ohne unerlaubte Hilfe angefertigt habe,
- außer den im Schrifttumsverzeichnis angegebenen Quellen und Hilfsmitteln keine weiteren benutzt und alle Stellen, die aus dem Schrifttum ganz oder annähernd entnommen sind, als solche kenntlich gemacht und einzeln nach ihrer Herkunft unter Bezeichnung des Ausgabe (Auflage und Jahr des Erscheinens), des Bandes und der Seite des benützten Werkes in der Magisterarbeit nachgewiesen habe,
- alle Stellen und Personen, welche mich bei der Vorbereitung und Anfertigung der Magisterarbeit unterstützten, genannt habe,
- die Magisterarbeit noch keiner anderen Stelle zur Prüfung vorgelegt habe.

Erlangen, den

Guido Leisker