Enguided – ein Textklassifikator auf Basis der DBpedia-Ontologie

David P. Gösenbauer

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2012

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 28. September 2012

David P. Gösenbauer

Inhaltsverzeichnis

Erklärung iii

Kurzfassung vii

Abstract viii

1 Einleitung 1
- 1.1 Problemstellung . . . 1
- 1.2 Motivation . . . 2
- 1.3 Zielsetzung . . . 3
- 1.4 Aufbau . . . 3

2 Grundlagen 5
- 2.1 Information Retrieval . . . 5
- 2.2 Anwendungsgebiete . . . 6
  - 2.2.1 Anfragebearbeitung . . . 6
  - 2.2.2 Browsing . . . 7
  - 2.2.3 Information Filtering . . . 7
- 2.3 Klassifikation . . . 8
  - 2.3.1 Definition . . . 8
  - 2.3.2 Klassifikationsarten . . . 9
- 2.4 Klassifikationsmethoden . . . 11
  - 2.4.1 Exkurs Machine Learning . . . 11
  - 2.4.2 Regelbasiertes Verfahren . . . 12
  - 2.4.3 Vektorraummodell . . . 14

3 Related Work 19
- 3.1 Nachteile des Vektorraummodells . . . 19
- 3.2 Hierarchische Klassifikationsmodelle . . . 20
  - 3.2.1 Big-Bang . . . 20
  - 3.2.2 Top-Down . . . 21
- 3.3 Ontologiebasierte Textklassifikation . . . 21
  - 3.3.1 Expand & Reduce . . . 22
  - 3.3.2 Hyponymie . . . 22

3.3.3 Ähnlichkeitstheorie . . . 23

4 Konzeption **24**
4.1 Spezifikation . . . 24
4.2 Extraktion inhaltsbestimmender Daten . . . 24
4.2.1 Quelltextverarbeitung . . . 24
4.2.2 Optimierung der Texte . . . 25
4.2.3 DBpedia-Ressourcen Annotation . . . 26
4.2.4 Gewichtung der Entitäten . . . 28
4.2.5 Rechenbeispiel – Importance I . . . 29
4.3 Zuweisung einer treffenden Kategorie . . . 30
4.3.1 Von der Entität zum Graphen . . . 30
4.3.2 Aufbau des Kategoriegraphen . . . 32
4.3.3 SimRank – Knotenähnlichkeitsmaß . . . 32
4.3.4 Kategoriezuweisung . . . 34
4.3.5 Rechenbeispiel – Kategoriegewichtung $w(c)$ . . . 38

5 Implementierung **39**
5.1 Überblick . . . 39
5.2 Apache UIMA . . . 39
5.2.1 UIMA Java Framework . . . 40
5.3 Systemarchitektur . . . 42
5.4 URL-Loader . . . 43
5.4.1 HTTP 405 Fehler . . . 44
5.5 Apache Tika – Markup Annotation . . . 44
5.6 DBpedia Spotlight – Entitäten Annotation . . . 45
5.6.1 Datenselektion . . . 45
5.6.2 Datenaufbereitung . . . 46
5.6.3 DBpedia Spotlight Webservice . . . 47
5.6.4 Verarbeitung der Annotationen . . . 49
5.7 Enguided – ClassificationEngine . . . 50
5.7.1 Extraktion des Kategoriegraphen . . . 50
5.8 JGraph . . . 51
5.9 *Enguided* Webservice . . . 51

6 Evaluierung **52**
6.1 Performanzmaße . . . 53
6.1.1 Precision . . . 53
6.1.2 Recall . . . 53
6.1.3 F-Maß . . . 54
6.2 Korrektheitsstandard . . . 54
6.2.1 Gold Standard „Yahoo! Directory“ . . . 55
6.3 Ablauf der Evaluierung . . . 60
6.3.1 Implementierung der Evaluierungsautomatik . . . 61

6.3.2 Auswertungsvarianten 61
6.3.3 Messungen im mehrdimensionalen Kategorieraum . . . 63
6.4 Evaluierungsergebnisse 64
6.4.1 Enguided 64
6.4.2 Vergleich mit *AlchemyAPI* 68
6.5 Diskussion 70

7 Resümee **71**
7.1 Fazit 71
7.2 Future Work 72

8 System Setup **74**
8.1 Eclipse & UIMA Setup 74
8.2 Run Configuration & Classpath 74
8.2.1 Classpath Quicksetup 75
8.2.2 Update der JDK bzw. JRE 75
8.3 Sesame Triple Store Setup 76
8.4 Inbetriebname 76

9 Inhalt der DVD **78**
9.1 Masterarbeit (PDF) 78
9.2 Online-Quellen (PDF) 78
9.3 Projekt (JAVA) 79
9.4 Ressourcen (RAR) 79
9.5 Evaluierungsdaten (JSON) 79

Quellenverzeichnis **80**
Literatur 80
Online-Quellen 82

Kurzfassung

Die Klassifikation von Dokumenten aller Art ist im Forschungsfeld des *Information Retrieval* von großer Bedeutung und ein essentieller Bestandteil vieler Anwendungen, wie zum Beispiel Suchmaschinen und Spamfilter. Traditionelle Verfahren haben den Nachteil, dass einerseits semantische Relationen durch die Art der Textrepräsentation (Termvektoren oder auch als Bag of Words bekannt) verloren gehen und andererseits das System meistens vor dem Einsatz korrekt trainiert werden muss.

Die folgende Arbeit präsentiert und wertet das System *Enguided* aus, welches ohne Training dazu in der Lage ist Dokumente und vor allem Webseiten zu klassifizieren. *DBpedia Spotlight* ermöglicht die Identifikation von Entitäten im Text, die in der *DBpedia* Ontologie definiert sind. Dies hat den Vorteil, dass jegliches Wissen ausgelagert ist und kein Training des Systems benötigt wird. Es werden traditionelle Methoden wie die Gewichtung der Entitäten durch die Termfrequenz *TF*, als auch spezifische Faktoren die deren strukturellen Kontext im Quelltext der Webseite miteinbeziehen, eingesetzt, um die wichtigsten Konzepte zu selektieren. Durch die Relation der Entitäten zum Kategoriegraphen der Ontologie ist es schlussendlich möglich eine der Hauptkategorien Wikipedias zuzuweisen und daraus eine Kategorie für das gesamte Dokument abzuleiten.

Das entwickelte Verfahren wird im Zuge dieser Arbeit evaluiert und mit bereits etablierten Techniken verglichen. In Gegenüberstellung mit dem *AlchemyAPI* Klassifikator ließ sich eine Verbesserung der F1-Metrik von 7 Prozent verzeichnen.

Abstract

Document classification plays an important role in the field of *Information Retrieval* and is concidered as an essential component of various applications such as search engines and spam filtering. Traditional techniques which make use of term vectors in order to represent texts are neglecting semantic relations between those terms and have to be trained so they can function properly.

This thesis presents the *Enguided* system and introduces an alternative approach of classifying especially web documents without the need of training. The identification of entities within texts is provided by the *DBpedia Spotlight* service, which matches those concepts with the ones that are part of the *DBpedia Ontology*. This is clearly an advantage as all knowledge is externally managed and therefore the system can function without the need of training. Traditional methods like the *term frequency TF* as well as specific factors which take the structural context, like HTML tags, of the matched entity into account, help to select the most important concepts. The relation of those to the category graph of the ontology enables the assignment of a proper category to the individual entity and furthermore deduct the category of the document by comparing the importance of those categories.

As part of this thesis the system is evaluated and compared to other well working techniques. Subsequently a gain of 7 percent in the F1-Score could be observed in comparision to the *AlchemyAPI* classification engine.

Kapitel 1

Einleitung

1.1 Problemstellung

Das Internet mit seiner Fülle an Informationen ist schon lange nicht mehr von uns Menschen selbst überblickbar. In der heutigen Zeit ist es für viele die primäre Wissensquelle und dient unter anderem zur Informationsbeschaffung. Die eigenständige Suche nach bestimmten Informationen im Internet wäre sehr aufwändig, wenn nicht sogar unmöglich. Suchmaschinen übernehmen diese Aufgabe für uns, indem sie die gewünschten Informationen binnen Sekunden suchen und ausliefern. Die automatische Katalogisierung von Webseiten ist unter anderem ein Teil eines solchen Systems, ohne die eine effiziente Suche nach Informationen, sei es als Computer oder Mensch, nicht möglich wäre. Da aufgrund der Menge an Webseiten eine manuelle Kategorisierung nicht in Frage käme, gibt es automatisierte Klassifikationsverfahren. Diese haben jedoch oft Schwierigkeiten mit der Klassifikation komplexer Texte. Mehrdeutigkeiten und im Text enthaltene, nicht offensichtliche Konzepte stellen eine Hürde für diese Verfahren dar.

Eines der großen Probleme des Webs ist die Tatsache, dass es für die menschliche Interpretation ausgerichtet ist. Dieser erkennt Konzepte und Dinge in einem Text, deren Bedeutungen ihm aus Erfahrung geläufig sind oder leitet deren Erklärungen aus dem Kontext ab. Er hat zum Beispiel keine Mühe dabei dem Text eine von zehn Kategorien zuzuweisen. Eine Maschine kann Texten jedoch keine Bedeutung zuweisen, außer der Mensch gibt diese explizit an. Das Forschungsfeld des *NLP (Natural Language Processing)* versucht dieses Problem durch Entwicklung von Techniken, wie Regelsystemen und Vergleichsmechanismen zur maschinellen Differenzierung von Textelementen, Worttypen, etc. zu lösen. Das Ziel des *Semantic Web* ist es hingegen, die einfache maschinelle Verarbeitung von Informationen im Web zu ermöglichen, was dieses Problem gar nicht erst entstehen lässt. Grundlegende Standards wie XML, RDF und OWL, sogenannte *Informations-Spezifikationssprachen*, zur Beschreibung von Informationen existieren be-

reits. RDF und OWL sind Ontologiesprachen, die zur Modellierung von Wissen einer Anwendungsdomäne, auch *Ontologie* oder *Wissensbasis*, dienen [6]. Ein sehr gutes Beispiel ist *DBpedia*, welche die komplette *Wikipedia* Online-Enzyklopädie als Ontologie abbildet.

Traditionelle Klassifikationstechniken ignorieren die semantischen Beziehungen innerhalb des Textes und haben in Folge Schwierigkeiten bei komplexeren Texten, zum Beispiel mit mehrdeutigen Wörtern. Zudem fehlt diesen Methoden der Bezug zu den semantischen Relationen im Text, aus denen wiederum mehr über diesen in Erfahrung gebracht werden könnte. Indem man Features im Text nicht als simple Terme, sondern als Konzepte einer Wissenssammlung betrachtet, lassen sich die Probleme klassischer Verfahren theoretisch lösen und die Texte exakter klassifizieren. Der Verwandtheitsgrad der Konzepte hilft festzustellen wie stimmig deren Disambiguierung[1] ist. Dabei gilt, je verwandter, umso höher ist die Wahrscheinlichkeit die richtigen Disambiguierungen gefunden zu haben. Das in der DBpedia-Ontologie enthaltene Weltwissen ist vollständig und korrekt genug, um die Konzepte die für die Umsetzung eines solchen Textklassifikators benötigt werden, zur Verfügung zu stellen. Außerdem besitzt jedes Konzept eine Relation zu einer oder mehreren Kategorien, welche wiederum Teil eines Kategoriegraphen sind. Daraus ergibt sich für diese Arbeit die Problemstellung der Identifikation und Gewichtung solcher Konzepte in Webdokumenten und der Suche nach einer geeigneten Kategorie im Graphen. Die Arbeit stellt einen möglichen Lösungsansatz vor, indem traditionelle Methoden mit semantischen Technologien, in Form der Einbindung der DBpedia-Ontologie in den Klassifikationsprozess, kombiniert werden.

1.2 Motivation

Die Motivation für die Umsetzung des Textklassifikators *Enguided* und Forschung im Zuge dieser Arbeit stützt sich zum Ersten auf das hohe Potenzial von externen Wissensbasen wie DBpedia. In Bezug auf die Textklassifikation bestehen die Vorteile darin, dass die Assoziationen von Termen zu den entsprechenden Konzepten extern abgebildet sind und demzufolge ein Training des Systems nicht nötig ist. Die laufende Erweiterung und Pflege der Wissensbasis bewirkt, dass der Klassifikator stets aktuelles Wissen einsetzt. Außerdem bildet der DBpedia-Kategoriegraph eine gute Grundlage, um die Erfüllung der Hauptaufgabe, die Klassifikation von Webdokumenten, zu ermöglichen. Weiters basiert die Motivation für diese Arbeit auch auf dem Erkenntnisinteresse in Hinsicht auf die Effektivität eines solchen Klassifikators, da es durchaus sinnvolle Anwendungszenarien gibt, die von den Vorteilen dieses Systems profitieren würden.

[1] die Auflösung sprachlicher Mehrdeutigkeit

1.3 Zielsetzung

Die Umsetzung des Klassifikators *Enguided* und dessen Evaluierung definieren die zwei Hauptziele für das im Zuge dieser Arbeit durchgeführte Forschungsprojekt. Den Klassifikator gilt es nach den genannten Kriterien umzusetzen und im Anschluss mit einem anderen Klassifikator zu vergleichen, um letztendlich die Forschungsfrage

> Ist die Kategorisierung von Webseiten mithilfe externer Wissensbasen wie DBpedia eine ebenbürtige Alternative zu herkömmlichen Verfahren hinsichtlich der Effizienz und der Qualität der Ergebnisse?

beantworten zu können. Die vorliegende Forschungsarbeit liefert einen Überblick über theoretische Grundlagen der Textklassifikation und beschreibt konzeptionelle Hintergründe, sowie konkrete Implementierungsdetails des Systems. Außerdem werden die Durchführung der Evaluierung und dessen Resultate in einem separaten Abschnitt behandelt.

1.4 Aufbau

Nach diesem einleitenden Teil folgt eine Einführung in die relevanten Themengebiete des Information Retrieval, wobei hier der Fokus auf der Textklassifikation liegt. Weiters erfolgt ein Überblick über traditionelle Methoden, sowie ein kurzer Einblick in das Feld des Machine Learning. Im Folgekapitel werden außerdem relevante Forschungsarbeiten und deren Kerninhalte kurz abgehandelt.

Das dritte Kapitel beleuchtet die Umsetzung auf konzeptioneller Ebene, beschreibt Lösungsansätze in der Theorie und wird in drei Hauptteile gegliedert. Die *Spezifikation*, die *Feature-Extraktion* und die *Kategoriezuweisung*. Es fungiert auf diese Weise als Grundlage für das folgende Implementierungskapitel. Dieses beschreibt die technischen Details der Umsetzung, das verwendete Analyseframework, die Systemarchitektur und essentielle Systemkomponenten.

Das Evaluierungskapitel stellt eingesetzte Performanzmaße vor, die Genauigkeit und Trefferquote des Systems messen und erläutert im Anschluss die Funktion eines Gold Standards und dessen systemspezifische Definition. Danach wird der Ablauf der Evaluierung erklärt und die Ergebnisse präsentiert. Diese werden mit jenen des ebenfalls evaluierten *AlchemyAPI*-Klassifikators verglichen. Am Ende dieses Kapitels wird auf die Forschungsfrage eingegangen und diese anhand der gewonnenen Erkenntnisse bestmöglich beantwortet. Inwiefern die erhaltenen Ergebnisse gültig sind und welche Kriterien nicht von der Evaluierung abgedeckt wurden, wird letztendlich in einem Diskussionsteil zusammengefasst.

Den Schluss der Arbeit bildet ein Resümee, welches die gewonnenen Erkenntnisse nochmals in einem Fazit zusammenfasst, gesetzte Ziele mit erreichten Zielen vergleicht und zukünftige Ziele zur Diskussion stellt.

Kapitel 2

Grundlagen

Dieses Kapitel behandelt wichtige Grundlagen, um die Lektüre späterer Kapitel verständlicher zu machen und umfasst Grundwissen zur Thematik des *Information Retrieval* und im speziellen der *Textklassifikation*. Es erfolgen zunächst Begriffserklärungen und Definition des Forschungsfeldes, welche gefolgt von Aufgabenstellung, Anwendungsfällen und Methoden zur Textklassifikation, eine Einführung in die Materie bieten.

2.1 Information Retrieval

Eine mögliche Definition des Begriffs *Information Retrieval* (IR) oder Informationsrückgewinnung[1], im Kontext der Informatik sei [17, siehe S. 1]:

> Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Hauptziel für IR-Verfahren ist es, einen speziellen *Informationswunsch* so gut wie möglich zu erfüllen, wobei dieser auf unterschiedliche Art und Weise geäußert werden kann. Dieser kann auch vom Benutzer selbst, zum Beispiel als Suchanfrage, in natürlicher Sprache formuliert werden oder sich auf bereits bestehende Dokumente beziehen. Die von Suchmaschinen verwendeten Methoden gehen dabei weit über einfache syntaktische Vergleiche hinaus. Stattdessen wird oft versucht auf semantischer Ebene zu agieren. Man spricht dabei von einer *inhaltsbasierten Suche* [5, S. 15].

Das Durchsuchen unstrukturierter, komplexer Inhalte zeichnet IR-Methoden aus. Strukturierte Inhalte werden hingegen beispielsweise in relationalen Datenbanken gespeichert. In der Realität gibt es kaum unstrukturierte Daten, selbst die Grammatik einer Sprache definiert eine gewisse Struktur.

[1] siehe http://de.wikipedia.org/wiki/Information_Retrieval

Außerdem handelt es sich bei einem Absatz oder einer Überschrift im Text bereits um Strukturinformationen. In dieser Arbeit ist meistens von *semistrukturierten Dokumenten*, genauer gesagt HTML, die Rede, da der *Enguided*-Klassifikator Webseiten analysiert. Generell stellt die Analyse dieser, aufgrund der engen Verknüpfung zur Suchmaschinenentwicklung, einen der größten Aufgabenbereiche im Forschungsfeld des IR dar.

Mit dem Aufkommen des Internets wurde aus dem, ursprünglich ausschließlich in diversen Berufen (z.B.: ReferenzbibliothekarIn) angewandten, IR mithilfe moderner Suchmaschinen ein Werkzeug für die Allgemeinheit. Die Einteilung kann dabei in zwei unterschiedliche Arten von Suchmaschinen geschehen. Jene die versuchen den Großteil des Webs zu durchsuchen und solche die sich bei ihrer Suche auf einen definierten Datenbestand oder Dateityp beschränken [13, S. 24 ff.]. Beide Arten haben das Ziel den Benutzer bei seiner Suche zu unterstützen, wobei dies oft so gut funktioniert, dass sie zu einem festen Bestandteil des Surfverhaltens werden. Der Benutzer springt immer seltener via Hyperlinks von Webseite zu Webseite, so wie eigentlich vorgesehen, sondern macht oft Umwege über die Suchmaschine. Diese Entwicklung ist mitunter ein Grund warum dem *Information Retrieval* immer mehr an Bedeutung zugeschrieben wird.

Da bei der Suchmaschinenentwicklung verschiedenste IR-Methoden zusammenspielen, wird diese häufig in Sachbüchern zum Thema IR behandelt. Eine grobe Einteilung aller Anwendungsgebiete wird im nächsten Abschnitt vorgestellt.

2.2 Anwendungsgebiete

Henrich kategorisiert die typischen Aufgabenstellungen für IR-Systeme in vier Klassen [5, S. 31 ff.]: *Klassifikation, Anfragebearbeitung, Browsing* und *Information Filtering*. Diese werden in diesem Abschnitt beschrieben, wobei die Klassifikation, aufgrund ihrer höheren Relevanz für diese Arbeit, in einem eigenen Abschnitt behandelt wird.

2.2.1 Anfragebearbeitung

Zu diesem Gebiet zählen jene Verfahren im IR, welche einen konkreten Informationswunsch (siehe auch Abb. 2.1) verarbeiten und bestmöglich zu erfüllen versuchen. Dabei durchsucht das System dessen Dokumentkollektion nach relevanten Daten, die dem Benutzer präsentiert werden. Typische Beispiele aus der Praxis sind Suchmaschinen wie *Google*, *Yahoo* und *Personal Search*-Anwendungen, zu denen auch die Suche und Indexierung von Daten durch das Betriebssystem zählt, siehe Apple's *Mac OS X Spotlight* oder die *Windows Instant Search*-Funktion von Microsoft.

Wie schon erwähnt setzen diese Systeme meistens auf mehrere IR-Methoden. So wäre das effiziente Durchsuchen großer Dokumentbestände ohne

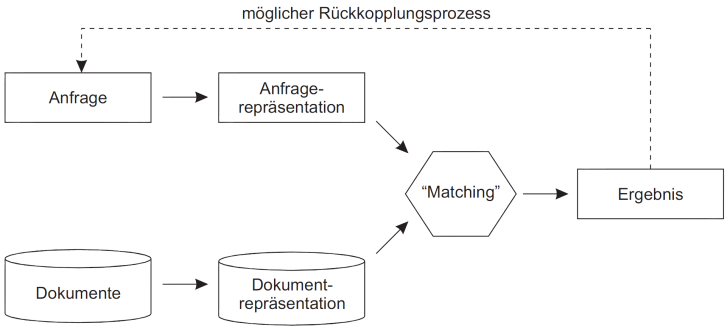

Abbildung 2.1: Verallgemeinertes Modell der Anfragebearbeitung im Information Retrieval [5, S. 40].

deren vorherige automatische Klassifikation nicht möglich. Generell lässt sich feststellen, dass sich diverse IR-Techniken eher schwach voneinander abgrenzen und kaum gesondert verwendet werden.

2.2.2 Browsing

Beim *Browsing* exploriert der Benutzer eine Kollektion von Hypertextdokumenten, die nicht immer zwingend untereinander verlinkt sein müssen. Typischerweise arbeitet sich der Benutzer dabei von allgemeineren zu spezifischeren Inhalten vor. Die inhaltsabhängige Einteilung des Korpus nennt sich *Clustering*. Dabei werden die Dokumente anhand der, für deren jeweiligen Text repräsentativsten Wörter in einen mehrdimensionalen Raum projiziert. Inhaltsähnliche Dokumente finden sich dabei näher im Raum wieder und bilden Cluster. Navigiert der Benutzer in diesem Raum, so wird er in der Nähe des aktuellen Dokuments weitere für ihn interessante Dokumente vorfinden.

2.2.3 Information Filtering

Das Ziel von *Information Filtering* besteht darin, relevante von irrelevanten Informationen zu trennen. Dieser Vorgang nennt sich *binäre Klassifikation*. Das System agiert dabei auf Basis der Benutzerinteressen, welche implizit per Fragenkatalog oder anhand von bereits betrachteten Dokumenten, in Erfahrung gebracht werden können. Jedoch existieren auch explizite Ansätze wie *Google Alerts*, ein Service, welches dem Nutzer in einem bestimmten Intervall auf Basis einer Suchanfrage die neuesten relevanten Suchergebnisse per Email zuschickt.

2.3 Klassifikation

Die Klassifikation von Dokumenten ist ein wesentlicher Bestandteil im Forschungsfeld des IR und kommt sehr oft zur Anwendung, sei es beim Filtern von Informationen oder der Suchmaschinenentwicklung.

Die Textklassifikation findet in zahlreichen Bereichen seine Anwendung [17, S. 254 f.]:

- Erkennen von **Spam**.
- **Sentiment**[2] **Detection**, wie zum Beispiel die Klassifikation von Filmreviews in *positive* und *negative*.
- **Automatische Sortierung** empfangener Emails in (benutzerdefinierte) Ordner. Bestes Beispiel ist der schon zum Standard gewordene Spamordner.
- **Vertikale Suchmaschinen** indexieren nur einen Teilbereich des Web entsprechend eines bestimmten Themengebiets.

2.3.1 Definition

Im Allgemeinen lässt sich die Klassifikation wie folgt definieren [17, S. 2]:

> Given a set of topics, standing information needs, or other categories (such as suitability of texts for different age groups), classification is the task of deciding which class(es), if any, each of a set of documents belongs to.

Nachzulesen im Wikipedia-Artikel [45] zum Thema Klassifikation, wird jener Vorgang in drei Subprozesse unterteilt: *Klassifikation, Klassifizierung* und *Klassierung*.

Klassifikation

Die *Klassifikation* wird als die „planmäßige Sammlung von abstrakten Klassen (auch Konzepten, Typen oder Kategorien), die zur Abgrenzung und Ordnung verwendet werden“ beschrieben [45].

Klassifizierung

Der Vorgang bei dem die Klassen zumeist „durch die Einteilungen von Objekten anhand bestimmter Merkmale, gewonnen und hierarchisch angeordnet“, werden nennt sich *Klassifizierung*. Das Ergebnis ist ein *kontrolliertes Vokabular* [45].

Ein **Merkmal** (auch Kriterium der Klassifikation) kennzeichnet ein Objekt als Mitglied einer Klasse und unterscheidet es von solchen anderer Klassen [40].

[2] die Stimmung des Textes

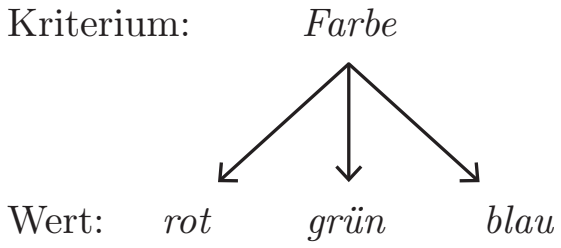

Abbildung 2.2: Eindimensionale Klassifikation als Baum dargestellt.

Klassierung

Die Klassierung ist die Zuweisung einer oder im Feld der Textklassifikation auch mehrerer, passenden Klasse(n) aus der gegebenen Klassifikation auf ein Objekt ([45]).

Die Begriffe der Klassifikation oder Kategorisierung finden im heutigen Sprachgebrauch oft als Synonyme für die Klassierung Anwendung. Auch in dieser Arbeit ist beim Einsatz der Wörter die eigentliche Klassierung gemeint. Die beschriebene Einteilung soll lediglich die Hauptschritte und Voraussetzungen für eine Klassifikation zeigen.

2.3.2 Klassifikationsarten

Bei der Klassifizierung werden die Klassen nicht nur anhand der Ableitung verschiedener Objektmerkmale beziehungsweise -kriterien gewonnen, sondern es gibt auch deren Relation implizit die Klassifikationsstruktur vor. Dabei wird zwischen zwei Grundstrukturen unterschieden: der *eindimensionalen* und der *mehrdimensionalen* Klassifikation [40].

Eindimensionale Klassifikation

Resultiert die Klassifikation aus einem einzigen Kriterium ist diese *eindimensional*. Darstellbar ist sie zum Beispiel als Matrix oder als Baum (siehe auch Beispiel Abb. 2.2).

Mehrdimensionale Klassifikation

Kommt mehr als ein Kriterium zum Einsatz, wird von einer *mehrdimensionalen* Klassifikation gesprochen. Dabei kann es sich entweder um eine *Kreuzklassifikation* oder eine *mono-* beziehungsweise *polyhierarchischen Klassifikation* handeln ([40], [45]).

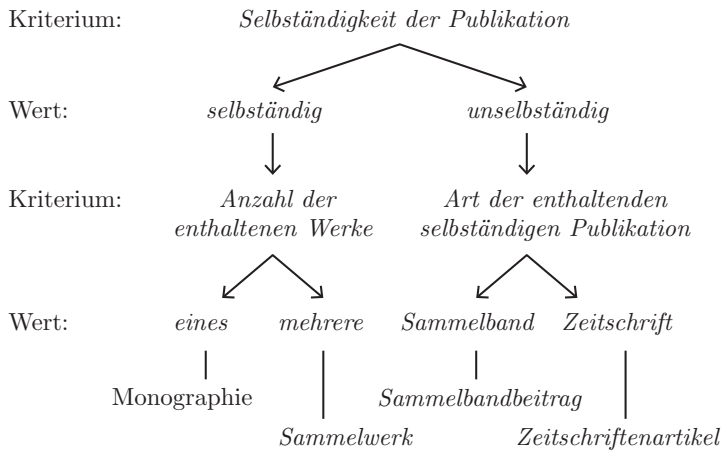

Abbildung 2.3: Beispiel für eine monohierarchische Klassifikation [40].

Kreuzklassifikation werden zwei vollständige eindimensionale Klassifikationen kombiniert, wird das Ergebnis als vollständige mehrdimensionale Kreuzklassifikation bezeichnet (siehe auch Beispieltabelle 2.1). Als *vollständig* werden jene Klassifikationen bezeichnet, deren zu klassifizierende Objekte nur genau einer Klasse angehören und jedes Objekt auch einer Klasse zugewiesen wurde [40].

Mono- beziehungsweise polyhierarchische Klassifikationen ergeben sich, wenn die Bedingungen einer Kreuzklassifikation nicht erfüllt werden. Sind zum Beispiel die Klassifikationskriterien unterschiedlich-allgemein, nur innerhalb einer Klasse anwendbar und befinden sich somit auf unterschiedlichen Ebenen der Hierarchie, ist es eine monohierarchische Klassifikation und besitzt eine Baumstruktur (siehe Beispiel Abb. 2.3). Klassifikationen mit der Struktur eines Graphen sind ebenso möglich. Sie werden auch als polyhierarchisch bezeichnet. Eine Klasse kann sich also von mehreren Oberklassen ableiten. Beispiele sind unter anderem der Wikipedia-Kategoriegraph oder das Onlineverzeichnis „Yahoo! Directory“.

Form \ Farbe	Rot	Blau
Rund	Rund & Rot	Rund & Blau
Eckig	Eckig & Rot	Eckig & Blau

Tabelle 2.1: Zweidimensionale Kreuzklassifikation.

2.4 Klassifikationsmethoden

Es gibt zahlreiche Methoden zur computergestützten Textklassifikation. Deren Ziel ist es, Dokumenten anhand bestimmter Eigenschaften eine oder mehrere Klassen eines zuvor definierten „kontrollierten Vokabulars“ zuzuweisen (siehe auch Abb. 2.4). In den folgenden Abschnitten werden zwei Methoden genauer beschrieben: der *regelbasierten Klassifikation* und der *Vektorraummethode*. Beide liefern einen ersten Einblick in die Entwicklung von Textklassifikatoren. Da sich diese Systeme meistens automatisch ein Klassifikationsmodell anhand von bereits korrekt klassifizierten Dokumenten aneignen, wird kurz auf den Bereich des *Machine-Learning* eingegangen.

2.4.1 Exkurs Machine Learning

Neben der manuellen Klassifikation wird zwischen *nicht-lernenden* und *lernenden* Systemen unterschieden. Ein Beispiel für ein nicht-lernendes System wäre die Klassifikation anhand von manuell erstellten *Regeln*, jene Methode, die unter anderem in einem späteren Abschnitt näher beschrieben wird. Da die Wartung und Erstellung dieser Regeln relativ zeitaufwändig ist, existieren Methoden, um diese automatisch aus manuell klassierten Dokumenten, einem sogenannten *Trainingsset*, abzuleiten. Dieses Trainingsset muss nach wie vor manuell erstellt werden, wobei man hier keine Experten konsultieren muss, um einem Text eine passende Kategorie aus einem vorgegebenen Vokabular zuzuweisen. Oft ist dies bereits Teil des Workflows des Systems, indem der Benutzer dessen Klassifikationsvorschläge bestätigt oder korrigiert. Das System würde in diesem Fall ständig dazulernen [17, S. 255]. Abbildung 2.5 zeigt einen typischen Ansatz, wie sich ein lernendes System ein Klassifikationsmodell aneignet. Zuerst wird das Trainingsset vom System „erlernt“. Dabei leitet es ein Klassifikationsmodell ab. Der Lernalgorithmus ist dafür zuständig, dass das Modell später auch Dokumente erkennt, die nicht direkt aus den Trainingsdaten stammen. Später wird das Klassifikationsmodell wiederum auf ein Testset angewandt und das Ergebnis im Anschluss evaluiert. Schneidet das System dabei gut ab, kann man darauf schließen, dass das

Abbildung 2.4: Grundvorgehensweise bei der Textklassifikation [26].

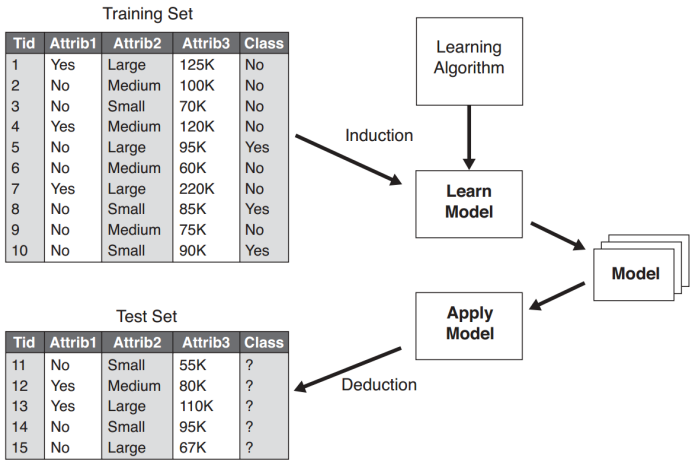

Abbildung 2.5: Anhand von korrekt klassierten Trainingsdaten wird ein Klassifikationsmodell erstellt, welches wiederum auf nicht-klassierte Dokumente angewandt werden kann [26, S. 148].

abgeleitete Klassifikationsmodell generell gut klassifiziert [26, S. 148 f.].

Der in dieser Arbeit vorgestellte Klassifikator ist ein nicht-lernendes System. *Enguided* ist ein *ad-hoc*-Klassifikator, da die Entscheidungen über die Klassenzugehörigkeit der Webseite, ohne vorheriges Training des Systems, zum Zeitpunkt der Anfrage geschieht.

2.4.2 Regelbasiertes Verfahren

Bei dieser Methode kommen, wie der Name schon sagt, Regeln zum Einsatz, um auf eine passende Klasse zu schließen. Diese sind in einer *Wenn-Dann*-Form formuliert. Sie stellen gewisse Voraussetzungen an die Dokumentrepräsentation und ziehen daraus eine vordefinierte Schlussfolgerung. Das Prinzip regelbasierter Systeme ist allgemein und findet nicht nur im IR seine Anwendung.

Die Logik des Verfahrens ist mit der des *booleschen Retrievals* (Suchmaschinenentwicklung) gleichzusetzen. Da eine Vielzahl von Recherchematerial für diesen Forschungsbereich existiert, werden gelegentlich Informationen zur Erklärung regelbasierter Klassifikatoren benutzt. Beim Booleschen Retrieval werden die Resultate anhand von Anfragen in Form von Wörtern, die untereinander mit booleschen Operatoren verknüpft sind, selektiert. Genauso lassen sich auch die Regeln in der Textklassifikation formulieren, sodass eine Bedingung für die Klasse *multicore computer chips* zum Beispiel so aussehen könnte:

$$((multicore \vee multi\text{-}core) \wedge (chip \vee processor \vee microprocessor))$$

Erfüllt der Featurevektor diese Bedingung, wird dem korrespondierenden Dokument die Klasse *multicore computer chips* zugewiesen.

Decision Trees

Solche Regelsätze können auch in Form von *Entscheidungsbäumen* oder *Decision Trees* festgelegt werden. Dieser besteht aus einem *Root*-Knoten, gefolgt von *internen* und *Blattknoten* (siehe Abb. 2.6). Sie bilden ein hierarchisches Konstrukt, welches die Regeln sequenziell als Baum darstellt und verbindet. Somit ist es dem System möglich durch eine Reihe von Folgerungen auf eine Kategorie zu schließen.

Der Root-Knoten besitzt keine eingehenden und keine oder mehrere ausgehende Kanten. Interne Knoten besitzen genau eine eingehende Kante und mindestens zwei ausgehende Kanten. Die Blattknoten repräsentieren eine Klasse, haben ebenfalls exakt eine eingehende Kante und da diese das Ende einer Folgerung bilden, keine ausgehenden Kanten.

Die Konstruktion eines *Decision Trees* basiert auf Algorithmen, deren Ziel darin besteht, einen möglichst akkuraten Baum effizient aus bestehenden Regelsätzen abzuleiten. Das Finden des akkuratesten Baumes wäre in einem exponentiell dimensionierten Suchraum in adäquater Zeit nicht möglich. Deswegen werden die von den Algorithmen gefundenen Lösungen als nicht optimal angesehen. Ein solcher Algorithmus wäre zum Beispiel der *Hunt's Algorithmus*. Auf diesen wird in dieser Arbeit jedoch nicht genauer eingegangen. Dessen genaue Funktionsweise kann in [26, S. 152 ff.] nachgelesen werden.

Ein Entscheidungsbaum wird automatisch anhand von Trainingsdaten gebildet und stellt somit eine komfortable Alternative zu manuell konstruierten Regelsätzen dar [26, S. 150 ff.]. Regelbasierte Verfahren haben ihre Vor- und Nachteile auf die im folgenden Abschnitt eingegangen wird.

Vor- und Nachteile

Eine hohe Klassifikationsqualität und Effizienz in der Anwendungsphase sprechen für regelbasierte Ansätze. Auch die Möglichkeit zur Optimierung durch manuelle Anpassung der Regeln bei nicht lernenden Verfahren dieser Kategorie kann durchaus als Vorteil gewertet werden [38]. Da sich jedoch das Ändern einer Regel meistens auf das komplette System auswirkt, sollte es nach jeder Änderung erneut auf seine Effizienz und Genauigkeit geprüft werden. Generell sind die Erstellung und Wartung solcher Systeme sehr zeitaufwändig. Da keine Gewichtung der aus dem Text extrahierten Wörter stattfindet, ist die Zuteilung zu einer ähnlichen Klasse, jene die den Text am ehesten korrekt klassifiziert, nicht möglich. Der Text erfüllt eine Regel oder nicht. Es gibt keine Möglichkeit die Ähnlichkeit zu einer Klasse zu messen oder die Struktur des Textes in den Klassifikationsprozess miteinzubeziehen.

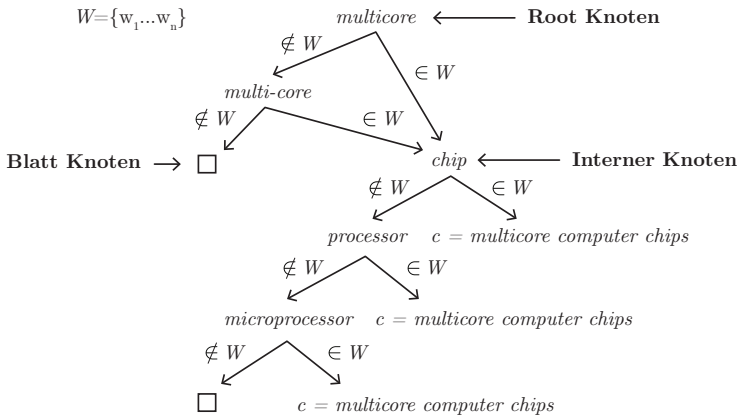

Abbildung 2.6: Exemplarischer Entscheidungsbaum der Regel $((multicore \vee multi\text{-}core) \wedge (chip \vee processor \vee microprocessor))$.

2.4.3 Vektorraummodell

Bereits 1960[3] stellte Salton [22] das *Vektorraummodell* vor. Es findet ebenso in der Suchmaschinenentwicklung als auch bei vielen Klassifikationaufgaben seine Anwendung. Verfahren, die auf diesem Modell basieren, liefern nach wie vor sehr gute Ergebnisse.

Um das Modell zu verstehen muss zuerst näher auf die Textrepräsentation in Form eines Vektors und grundlegende Gewichtungsmethoden eingegangen werden.

Textrepräsentation als Vektor

Grundsätzlich muss der Text eines Dokuments i in eine Form gebracht werden, die es dem System ermöglicht diesen überhaupt zu klassifizieren. Die Repräsentationsform wirkt sich auch auf die generelle Genauigkeit eines lernenden Systems aus. Eine gebräuchliche Variante wäre es, zuerst alle Wortstämme eines Textes zu extrahieren und zu zählen. Beispielsweise würde das Vorkommen der Wörter „computes“, „computing“ und „computer“ jeweils zum selben Wortstamm „comput“ addiert werden. Sogenannte *Stopwörter*, wie „der“, „die“, „das“, „und“ etc., werden bei der Extraktion meistens ignoriert, da diese für den Text nicht repräsentativ sind. Jeder einzigartige Wortstamm w_i wird in einer Attributliste als Feature $TF(w_i, d)$ mit der Wortfrequenz d gespeichert. Das Ergebnis nach dieser Extraktion ist ein *Featurevektor* (siehe Abb. 2.7). Die *Text Frequency* TF wurde bereits in der Arbeit von Salton [22] vorgestellt. Der Eintrag von Wörtern, welche mindestens dreimal im Text vorkommen, verhindert, dass der Featurevektor nicht

[3] zitiertes Paper datiert 1986, laut [5] bereits 1960

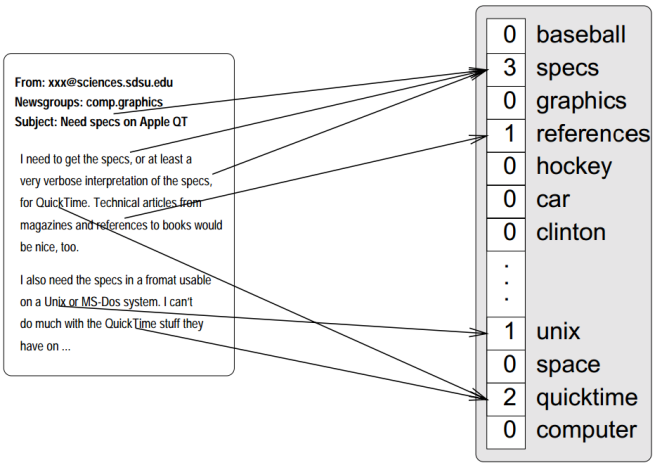

Abbildung 2.7: Illustration der Entstehung eines Featurevektors [8, S. 2].

unnötig groß wird. Im Zuge der TF stellt Salton auch die „Inverse Document Frequency (IDF)“ vor [8][22]:

$$IDF\left(w_i\right) = \log\left(\frac{n}{DF\left(w_i\right)}\right) \tag{1}$$

n steht für die Gesamtanzahl von Trainingsdokumenten und $DF\left(w_i\right)$ für die Anzahl der Dokumente, in denen w_i vorkommt. Skaliert man nun den Vektor mit den jeweiligen IDF Werten für w_i, wirkt sich dies erfahrungsgemäß positiv auf die Performanz des Systems aus. So werden Wörter, welche in mehreren Dokumenten vorkommen, niedriger gewichtet und vice versa. Letztendlich wird der Vektor normiert, um zu verhindern, dass längere Dokumente bevorzugt werden [8].

An dieser Stelle wird vermerkt, dass ein ad-hoc Textklassifikator wie *Enguided*, keine Information über die Anzahl der Vorkommnisse eines Wortes im Dokumentkorpus besitzt. Deswegen kommen auch nur die TF und alternative Gewichtungsmethoden zum Einsatz.

Ähnlichkeitskalkulation

Die Klassen des Vokabulars werden typischerweise ebenfalls durch einen Featurevektor repräsentiert. Anhand korrekt klassierter Trainingsdaten werden die Klassenvektoren automatisch abgeleitet, indem etwa die Mittelwerte der einzelnen Featuresummen jener Dokumente, die zu einer Klasse gehören, als Werte des Klassenvektors eingesetzt werden.

Nun lässt sich der Vektor des zu klassierenden Dokuments mit den Klassenvektoren vergleichen, wobei ein Ähnlichkeitswert bestimmt werden muss.

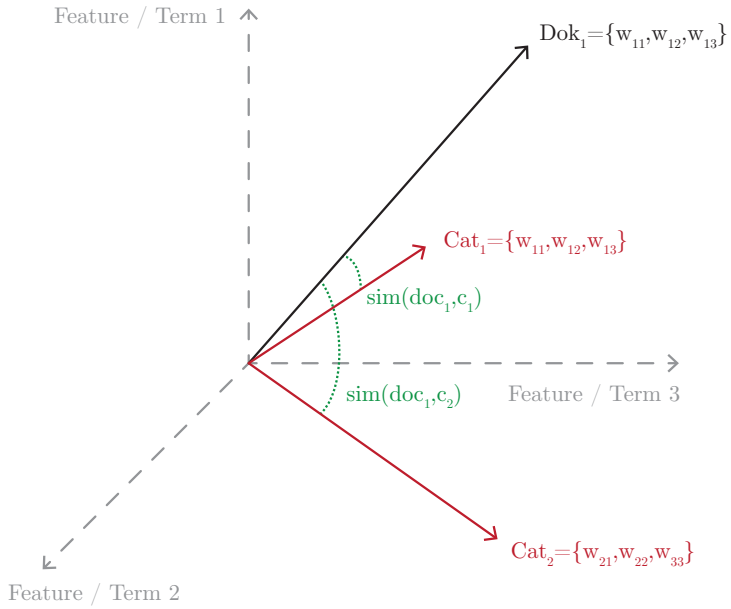

Abbildung 2.8: Die Abbildung verdeutlicht den Vergleich zweier Vektoren im Raum.

Dieser kann beispielsweise durch das *Skalarprodukt* (2) beider Vektoren oder dem *Kosinusmaß* (3) errechnet werden [22]:

$$sim\left(doc_i, c_j\right) = \sum_{k=1}^{t} w_{i,k} \cdot w_{j,k} \tag{2}$$

$$sim\left(doc_i, c_j\right) = \frac{\sum_{k=1}^{t} w_{i,k} \cdot w_{j,k}}{\sqrt{\sum_{k=1}^{t} \left(w_{i,k}\right)^2 \cdot \sum_{k=1}^{t} \left(w_{j,k}\right)^2}} \tag{3}$$

Vereinfacht könnte man sich den Vergleich mittels Kosinusmaß, welches den Winkel zwischen den beiden Vektoren berechnet, vorstellen (siehe Abb. 2.8). Dabei gilt, je kleiner der Winkel zwischen den beiden Vektoren ist, umso größer ist deren Ähnlichkeit.

SVM – Support Vector Machines

Support Vector Machines wurden von V. Vapnik [28] vorgestellt, basieren auf dem Vektorraummodell und eignen sich sehr gut für die Textklassifikation. Unter der Annahme, dass ähnliche Dokumente im Vektorraum nahe beieinander liegen, lassen sich, basierend auf dem *Structural Risk Minimization*

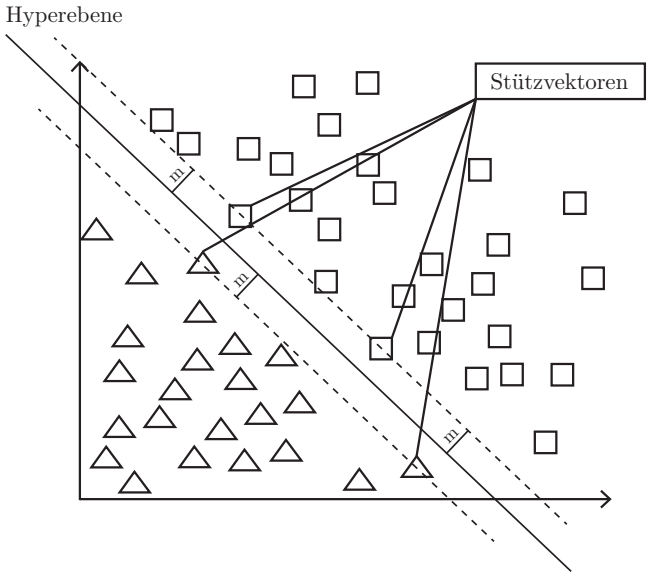

Abbildung 2.9: Die Hyperebene wird durch die Stützvektoren (Support Vectors) bestimmt.

Principle, Grenzen zwischen den Klassen berechnen. Dessen Idee ist es, eine Hyperebene h zu finden, welche Fehlklassifikationen minimiert. Im Idealfall lässt sich eine Menge von Dokumenten durch eine Hyperebene h

$$(\vec{w} \cdot \vec{x}) + b, \vec{w} \in \mathbb{R}^N, b \in \mathbb{R} \tag{4}$$

trennen. Man spricht in diesem Fall von einer linear separablen Menge. Mithilfe von Trainingsdaten lässt sich eine solche Ebene berechnen, um im Anschluss die Lage neuer Dokumente zur Ebene bestimmen zu können. Abhängig davon, auf welcher Seite der Ebene diese liegen, wird ihnen die entsprechende Klasse zugewiesen, wobei der Abstand zur Ebene m Aufschluss über die Wahrscheinlichkeit der Entscheidung gewährt. Das heißt, je größer der Abstand zur Ebene ist, umso größer ist auch die Wahrscheinlichkeit, dass die Entscheidung richtig ist. Die Bestimmung der Hyperebene geschieht unter der Bedingung, dass ihr Abstand zum nächstgelegenen Dokument der Klasse A, dem Abstand zum nächstgelegenen Dokument der Klasse B entspricht. Der Verlauf der Ebene hängt in Folge von diesen Dokumenten, den sogenannten *Support Vectors*, ab [9]. Der schematische Konstruktionsablauf wird in der Abbildung 2.9 veranschaulicht. Auf diese Weise lassen sich jedoch nicht alle Konstellationen lösen. Es gilt dann beispielsweise zu versuchen, ob solche Problemstellungen durch die Transformation in einen höherdimensionalen Vektorraum auf den linearen Fall zurückzuführt werden können [9].

Außerdem lassen sich nicht-lösbare Szenarien durch die Identifikation und dem Ausschluss jener Dokumente, welche dazu beitragen, dass sich die Datenmenge nicht trennen lässt, zu lösbaren umwandeln [8].

Für weiterführende Literatur wird an dieser Stelle auf folgende Quellen verwiesen: [22], [8], [9] und [5].

Kapitel 3

Related Work

> The Knowledge Principle: if a program is to perform a complex task well, it must know a great deal about the world in which it operates [12].

Bereits vor zwei Jahrzehnten formulierten Lenat und Feigenbaum das *Knowledge Principle* und verdeutlichten den Nutzen von externem Wissen im Forschungsfeld des Information Retrieval. Wikipedia bildet von Tag zu Tag immer mehr Wissen ab, welches man sich auch im komplexen Aufgabengebiet der Textklassifikation zu Nutze machen kann.

In einem Punkt sind sich die Autoren vieler Forschungsarbeiten immer einig, nämlich dass Wikipedia beziehungsweise die DBpedia-Ontologie, das Open Directory Project (ODP) etc. eine ausgezeichnete Wissensquelle in puncto Aktualität, Qualität und Vollständigkeit darstellen. Durch sie sind Klassifikationssysteme nicht mehr auf, durch Training erlangtes, begrenztes Wissen angewiesen und können Dokumente somit besser einordnen.

3.1 Nachteile des Vektorraummodells

Die schon länger bestehende BOW (Bag of Words) Methode, welche der Vektorenrepräsentation (siehe auch Abs. 2.4.3) entspricht, wird in vielen Arbeiten angefochten. Gabrilovitch et al. betont in [4] einerseits dessen hohe Effektivität bei leichten bis mittelschweren Klassifikationsaufgaben, andererseits auch die Nachteile. Bei der Textrepräsentation als Featurevektor wird die Reihenfolge der Wortvorkommen komplett ignoriert und somit Informationen, wie Relationen zwischen den Wörtern sowie der gesamte Kontext, verworfen. Dieses Vorgehen nimmt in Folge auch die Möglichkeit zur Disambiguierung von Wörtern. Außerdem ist aufgrund der Tatsache, dass sich das Vokabular des Modells auf die im Trainingsset vorkommenden Wörter beschränkt, keine Generalisierung von Wörtern möglich. Wörter, die nicht im Trainingsset vorkommen, werden deswegen ignoriert.

3.2 Hierarchische Klassifikationsmodelle

Sun et al. gibt in [25] einen sehr guten Überblick über hierarchische Klassifikationsmethoden und einen Einblick in deren Evaluierung. Zum Beispiel werden in der Arbeit von Sun et al. alternative Metriken zur Evaluierung von Information Retrieval Systemen vorgestellt, welche speziell auf hierarchische Klassifikationsaufgaben ausgelegt, anhand von Distanz und Ähnlichkeit, deren Effizienz und Effektivität messen. Außerdem werden vier verschiedene hierarchische Grundstrukturen vorgestellt, in welche sich die Konzepte einordnen lassen:

1. **Virtual category tree** Die Dokumente werden nur Kategorien zugewiesen, die als Blattknoten in der Baumstruktur dargestellt werden.
2. **Category tree** Basierend auf der vorangehenden Struktur, mit dem Unterschied, dass Dokumente zusätzlich internen Kategorien zugewiesen werden dürfen.
3. **Virtual directed acyclic category graph** Wiederum ähnlich dem ersten Konzept, jedoch handelt es sich hierbei um einen Graphen. Den Dokumenten dürfen nur Blattknoten-Kategorien zugewiesen werden.
4. **Directed acyclic category graph** Laut Sun et al. das populärste Strukturkonzept von Webverzeichnissen, wie *Yahoo! Directory* und dem *ODP*. Dokumente dürfen sowohl Blattknoten-Kategorien als auch internen Kategorien zugewiesen werden.

Sun et al. halten fest, dass die meisten Forschungsarbeiten von der ersten Struktur, dem *Virtual category tree* ausgehen und meistens die üblichen Metriken, wie *Precision* und *Recall* zur Evaluierung ihrer Systeme einsetzen. Diese sind laut Sun et al. für solche Systeme ungeeignet weil sie für flache Klassifikationsmethoden entwickelt wurden.

In weiterer Folge werden zwei Methoden zur hierarchischen Klassifikation von Dokumenten vorgestellt, die von den meisten Systemen übernommen wurden, die *Big-Bang*- und die *Top-Down*-Methode.

3.2.1 Big-Bang

Die Big-Bang-Methode wurde bereits in verschiedenen Varianten, etwa in [11], [29] und [23] implementiert und zeichnet sich dadurch aus, dass nur ein einziger Klassifikator eingesetzt wird. Dieser weist einem Dokument eine oder mehrere Kategorien zu, wobei es sich dabei, abhängig von der Klassifikationsstruktur, um interne oder Blattknoten-Kategorien handeln kann.

Kritisch ist, dass bei dieser Methode nur während der Trainingsphase von der Klassifikationsstruktur abgeleitete Informationen zur Verfügung stehen. Dies hat auch zur Folge, dass Änderungen der Klassifikationsstruktur

meistens ein Neutraining des Klassifikators erfordern. Da für jede Kategorie eigene Featuresets angelegt werden, obwohl zum Beispiel eine Parent-Child Beziehung zwischen zwei Kategorien besteht, können deren gemeinsame Features unterschiedliche Gewichtungen besitzen. Für Methoden, die diesen Ansatz verfolgen, ist es demnach sehr schwierig die unterschiedlichen, sich auf mehreren Ebenen befindenden Featuresets für den Klassifikationsprozess zu nutzen.

3.2.2 Top-Down

Prinzipiell bietet sich für eine hierarchische Klassifikation der *divide-and-conquer* Ansatz an, bei dem ein großes, komplexes Problem in mehrere kleinere, simplere Probleme aufgeteilt und somit leichter gelöst werden kann. In einer hierarchisch angeordneten Klassifikation kann nun ausgehend von der Root-Ebene eine Klassifikationsaufgabe über die Kindselemente jedes weiteren geöffneten Kategorieknotens definiert werden. Die Klassifikation auf einer Unterebene entspricht dann einer *flachen Klassifikation*, jedoch mit einem wesentlich kleinerem Vokabular, was den Prozess vereinfacht. Nach der Klassifikation wird, wenn vorhanden, wiederum eine Ebene tiefer, durch Öffnen des selektierten Kategorieknotens, klassiert. Sollte die selektierte Kategorie keine Kindsknoten mehr besitzen oder wird keine weitere Selektion mehr getroffen, hat das System eine geeignete Kategorie gefunden.

Die Top-Down Methode macht sich diese Idee des *divide-and-conquer* zum Vorteil und wurde bereits in mehreren Arbeiten wie [3], [2] oder [10] eingesetzt. Der *Enguided*-Klassifikator wendet diese Methode invers an, indem von einem Wikipedia-Artikel, dem spezifischsten Knoten der Ontologie, in die andere Richtung vorgegangen, also generalisiert, wird.

Aber auch diese Methode hat ihre Nachteile. Eine fehlerhafte Klassifikation führt dazu, dass korrekte Kinds- (Top-Down) beziehungsweise Elternknoten (Down-Top) als mögliche Kategorien ausgeschlossen werden, ohne dass diese überprüft werden können. Je früher eine solche Missklassifikation geschieht, umso dramatischer ist die Auswirkung auf das Ergebnis. Jedoch ist dieser Punkt nicht direkt für das *Enguided*-System zulässig, da Elemente nicht sofort ausgeschlossen werden, sondern erst wenn deren Kindselemente zu keiner Lösung geführt haben.

3.3 Ontologiebasierte Textklassifikation

Da sich *Enguided* beim Klassifikationsprozess komplett auf externes Wissen in Form der DBpedia-Ontologie verlässt, wurde vor allem in diesem Bereich intensiv recherchiert.

3.3.1 Expand & Reduce

Die Arbeit von Thomas et al. [27] stellt das System *Doozer* vor, welches für einen gegebenen Text in Form eines Dokuments, einer Anfrage oder einer Liste von Termen, ein relevantes *Domain Model* mit Wissen aus Wikipedia generiert. Anders formuliert setzt das System an einer Stelle der Online-Enzyklopädie an, dehnt sich in Form der (gierigen – greedy) Akquise relevanter Daten aus und zieht sich schlussendlich wieder zusammen, indem weniger relevante Informationen verworfen werden. Dieser Prozess namens *Expand & Reduce* liefert als Ergebnis eine für die Problemdomäne relevante Miniontologie.

Enguided benutzt eine ähnliche Strategie und verwendet auch die Artikel von Wikipedia als Einstiegspunkt in die Ontologie. Der daran anknüpfende, von Thomas et al. beschriebene Expansionsprozess, half außerdem bei der Entwicklung eines Ähnlichkeitsmaßes, indem der in [27] erwähnte *SimRank* auch bei *Enguided* verwendet wird. Hier gilt es anzumerken, dass Thomas et al. die *SimRank*-Gleichung für ihre Zwecke anpassten und *Enguided* die originale Gleichung aus [7] einsetzt.

3.3.2 Hyponymie

Mehr Einsicht in die Struktur von Wikipedia und wie sich diese für den eigenen Zweck einsetzen lässt, gewährte die Arbeit von Wang et al. In [30] erklärt Wang et al. unter anderem die Bedeutung der *Hyponymie*[1] in Zusammenhang mit Wikipedia. Da Wikipedia Artikel und Kategorien zu mehreren Überkategorien gehören können (Co-Hypernymie) wird gefolgert, dass dessen Kategorien keine Taxonomie mit Baumstruktur, sondern einen gerichteten azyklischen Graphen[2] formen und daher eher einem thematisch organisierten Thesaurus entsprechen.

In [20] werden von Ponzetto et al. verschiedene Methoden vorgeschlagen, um aus Wikipedia eine Taxonomie abzuleiten, welche ausschließlich aus *is-a* und *is-not-a* Beziehungen bestehen. Diese Methoden werden auch von Wang et al. eingesetzt, um von Konzepten Hyponyme ableiten zu können. Dies ist für *Enguided* insofern relevant, als dass die Effizienz der Klassifikation auch von der Qualität des Graphen abhängt. Die in [20] vorgestellten Methoden setzen sich unter anderem zum Ziel die Qualität des Wikipedia-Kategoriegraphen zu verbessern, indem sie beispielsweise Metakategorien, die der Administration der Enzyklopädie dienen, ausschließen. Auch *Enguided* schließt solche Kategorien aus und hält diese per *Blacklist* fest. Die in [20] erwähnten Methoden könnten in Zukunft für eine Verbesserung des *Enguided*-Systems verwendet werden.

[1] „Dackel“ ist beispielsweise ein *Hyponym*, ein Unterbegriff von „Hund“, der seinerseits wieder das Hyperonym, also den Oberbegriff, von „Dackel“ darstellt

[2] auch Directed Acyclic Graph, kurz DAG

3.3.3 Ähnlichkeitstheorie

Wang et al. definiert in [30] drei unterschiedliche Techniken zum Messen der Ähnlichkeit.

Das *content-based* Maß ermöglicht es, die Ähnlichkeit eines Artikeltupels, anhand der Summe der gemeinsamen Terme, welche die Artikel in BOW-Form repräsentieren, zu berechnen. Diese Technik könnte *Enguided* jedoch nur sehr eingeschränkt für die Kategoriebestimmung von Nutzen sein, da nicht jede Kategorie eine Beschreibung besitzt und somit das inhaltsbasierte Maß nicht immer berechenbar ist.

Der *out-link category-based* Wert bedient sich der Summe der gemeinsamen ausgehenden Kategorieverlinkungen. Ein höherer Wert spricht dabei für eine höhere Ähnlichkeit der Entitäten und entspricht dem Grundgedanken des *SimRank*, von dem *Enguided* Gebrauch macht.

Die Distanz zwischen zwei Artikeln spielt wiederum beim *distance-based* Maß eine Rolle. Dabei wird davon ausgegangen, dass deren Ähnlichkeit sich invers-proportional zur Distanz, also der kürzesten Strecke die beide Artikel verbindet, verhält. Auch diese Methode wird von *Enguided* verwendet, indem Lösungen innerhalb einer gewissen Distanz zum ursprünglichen Artikel liegen müssen, um als gültig angesehen zu werden.

Die Kombination der verschiedenen Maße ergibt schlussendlich sowohl in [30] als auch bei *Enguided*, die Ähnlichkeit zwischen zwei Entitäten.

Kapitel 4

Konzeption

Das folgende Kapitel liefert einen tiefen Einblick in die Konzeption der entwickelten Software und stellt noch ausstehendes Vorwissen für das darauf folgende Implementierungskapitel zur Verfügung.

4.1 Spezifikation

Anhand der Zielsetzung wurden verschiedene Kompetenzbereiche der Software ausgearbeitet. Die folgende Liste zeigt deren grobe Einteilung und folgende Kapitelstruktur:

1. **Extraktion** inhaltsbestimmender Daten und die
2. **Zuweisung** einer zutreffenden Kategorie mithilfe des DBpedia-Kategoriegraphen.

Der restliche Teil des Kapitels widmet sich der Beschreibung der Methodik, zur Erreichung dieser Ziele.

4.2 Extraktion inhaltsbestimmender Daten

Um eine treffende Kategorie für die Webseite zu finden, müssen relevante von irrelevanten Informationen getrennt und in deren Wichtigkeit bewertet werden. Im Grunde findet eine Selektion und Gewichtung der Daten statt. Folgende Abschnitte beschreiben diese Vorgänge in Bezug auf den, in dieser Arbeit vorgestellten, Textklassifikator *Enguided*.

4.2.1 Quelltextverarbeitung

Ausgangspunkt für die Verarbeitung der Daten war der Quelltext der zu klassifizierenden Seite. Dabei handelt es sich um semi-strukturierte Daten in Form von HTML. Der *Tag-Type* und des Tags Inhalt sind, unter der Voraussetzung, dass es sich beim Inhalt um unstrukturierten Text handelt, für die Weiterverarbeitung von besonderem Interesse. Um die relevanten Informa-

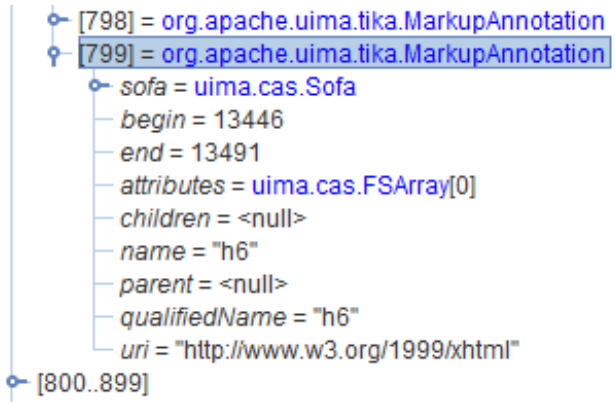

Abbildung 4.1: Relevant sind die Attribute *begin*, *end* und *name*.

tionen extrahieren zu können, wurde auf ein bestehendes, für das eingesetzte Framework entwickeltes, Modul zurückgegriffen. Übergeben wurde der zuvor geladene Quellcode, um die Antwort in Form von Annotationen[1] zu erhalten (siehe Abb. 4.1). Danach wurden diese Annotationen anhand einer *Whitelist* gefiltert. Der Taginhalt wird demnach nur weiterverarbeitet, wenn sich der Tag-Type (siehe Abb. 4.1, Attribut *name*) auf dieser Liste befindet. Bei der Erstellung der Whitelist wurden hauptsächlich Tags gewählt, die typischerweise den meisten Text beinhalten wie Absätze, Überschriften und bestimmte Meta-Informationen.

4.2.2 Optimierung der Texte

Bevor die Daten an das DBpedia Spotlight Webservice gesendet werden können, muss sichergestellt werden, dass die selektierten Texte für die Übertragung optimiert werden. Würde man alle Texte zu einem Ganzen zusammenfügen, hätte dies serverseitige Fehler zur Folge. Außerdem wurde durch zahlreiche Tests bei der Entwicklung festgestellt, dass die Annotation am Besten funktioniert, wenn genügend Text und somit auch Kontext vorhanden ist. Da die Annotation kontextabhängig funktioniert, würde sich zu viel oder zu wenig Text negativ auf die Disambiguierung[2] auswirken. Aufgrund dieser Tatsache wurden die Texte in strukturell zusammengehörige Gruppen gebündelt, welche wiederum so zusammengefasst wurden, dass die zuvor erwähnte Bedingung der idealen Textlänge erfüllt wurde.

[1] Annotationen sind Metadaten die mit bestimmten Regionen im Text assoziiert werden. Zum Beispiel wird ein gewisser Textteil mit einem Tag-Type wie <h3> assoziiert.

[2] Die Auflösung semantischer Mehrdeutigkeit

Strukturelles Gruppieren

An dieser Stelle wird näher auf das „strukturelle Gruppieren“ der Textfragmente eingegangen. Dabei werden sich deren Tag-Types zunutze gemacht, um auf die Zusammengehörigkeit zu schließen. Prinzipiell basiert die Umsetzung auf der Idee, dass Überschriften nach einem Paragraphen einen möglichen thematischen Umbruch markieren. Der Pseudocode 4.1 und 4.2 zeigt wie die Gruppierung funktioniert.

4.2.3 DBpedia-Ressourcen Annotation

Nach dem Erhalt des HTTP-Requests und des Textes als dessen Parameter erkennt und annotiert DBpedia-Spotlight im Text vorkommende DBpedia-Ressourcen und gibt die Annotationen, wie zum Beispiel hier im .json--Format, zurück:

```
...
{
 "URI": "http://dbpedia.org/resource/Politics",
 "support": "3220",
 "types": "Freebase:/comic_strips/comic_strip_genre,...",
 "surfaceForm": "Politics",
 "offset": "167",
 "similarityScore": "0.09870480746030807",
 "percentageOfSecondRank": "-1.0"
},
...
```

Relevant sind vor allem die Attribute *URI* und *similarityScore*. Die URI schlägt die Brücke zur DBpedia Ontologie und ermöglicht es später den Graphen aufzubauen (siehe Abs. 4.3). Das zweite Attribut macht die Zuversicht der korrekten Ressourcenwahl, in Bezug auf die semantische Mehrdeutigkeit des Wortes, messbar.

Anhand dieser Daten wurde nun eine Gewichtung durchgeführt. Genaueres wird im nächsten Abschnitt erläutert.

Algorithmus 4.1: Gruppieren von zusammengehörigen Texten

```
1:  GroupStructureRelatedTexts(textFragments)
    Gruppiert die sich in textFragments befindlichen Textfragmente nach
    deren strukturellen Eigenschaften.
2:      for i = 0 → sizeof(textFragments) do
3:          t1 ← textFragments[i]
4:          t2 ← textFragments[i + 1]
5:          if i + 1 ≤ sizeof(textFragments) then
6:              if tagof(t1) =<p> and tagof(t2) ≠<p> and i ≠ 0 then
7:                  group = gatherTexts(i, textFragments)
8:                  add group to groups
9:              end if
10:         else
11:             group = gatherTexts(i, textFragments)
12:             add group to groups
13:         end if
14:         i ← i + 1
15:     end for
16:     return groups.
17: end
```

Algorithmus 4.2: Sammeln der relevanten Texte

```
1:  gatherTexts(i, textFragments)
    Sammelt die relevanten Listeneinträge textFragments iterativ bis Index
    i.
2:      for j = i → 0 do
3:          t ← textFragments[j]
4:          if j = 0 then
5:              group ← group + t
6:          else if tagof(textFragments[j − 1]) =<p> then
7:              group ← group + t
8:              break
9:          else
10:             group ← group + t
11:         end if
12:         j ← j − 1
13:     end for
14:     return group.
15: end
```

4.2.4 Gewichtung der Entitäten

Die Gewichtung und Reihung der DBpedia-Ressourcen ist entscheidend, um der Webseite erfolgreich eine treffende Kategorie zuweisen zu können. Würde man einfach alle Ressourcen in den Vorgang einfließen lassen, wäre ein unperformantes System die Folge. Da davon ausgegangen wird, dass nicht alle gefundenen Entitäten inhaltsbestimmend sind, werden nur die „wichtigsten" zur Klassierung herangezogen.

Die Wichtigkeit einer *Entität* (Importance $I(e)$) wird durch eine Gleichung mit mehreren Variablen bestimmt:

$$I(e) = tf(e) \cdot tw(e) \cdot tdr(e) \cdot \max(sim(e)). \tag{4}$$

$tf(e)$ – Termfrequenz

Sie beschreibt die Anzahl der Vorkommnisse eines Terms oder einer Entität e im Text [22]:

$$tf(e) = |E|. \tag{5}$$

$tw(e)$ – Markup-Gewichtung

Jeder Tag-Type, der sich auf der Tag-Whitelist befindet, besitzt auch eine Gewichtung. Für die Berechnung bedeutet dies, dass jedes Vorkommnis der Entität e somit eine Gewichtung w besitzt, abhängig von welchen Tag-Typ t diese eingeklammert wird. Die jeweilige Gewichtung der Entitäten, die im Text öfters vorkommen, unterscheidet sich meistens. Trotzdem wurde nicht der Durchschnitt berechnet. Erstens würde sich dabei die dafür benötigte Summe der Terme $|T|$ im Nenner mit der $tf(e)$ im Zähler kürzen. Die Gleichung

$$tw(e) = \sum_{i=0}^{|T|} w(T_i) \tag{6}$$

gewichtet die Terme gewollt zu Gunsten ihrer Frequenz, welche ein stärkeres Indiz für deren Wichtigkeit darstellt.

$tdr(e)$ – TopDown-Relevanz

Hierbei handelt es sich um ein Maß, welches die Relevanz einer Entität in Abhängigkeit von der globalen Position ihrer Tagklammer $tpos(e)$ im gesamten Text des Dokuments bewertet. Somit hat sie innerhalb der ersten im Text vorkommenden Überschrift, beispielsweise vom Typ <h2>, eine höhere Relevanz als solche, die sich in späteren Überschriften derselben Tag-Types wiederfinden. Bei mehrmals vorkommenden Entitäten, wird die Position des

ersten Vorkommens für die Berechnung von $I(e)$ verwendet. $tdr(e)$ berechnet sich folgendermaßen, wobei l für die Gesamtlänge des Dokuments steht:

$$tdr(e) = 1 - \frac{tpos(e)}{l}. \tag{7}$$

$\max(sim(e))$ – **DBpedia Ähnlichkeitsmaß**

> Given the VSM representation of DBpedia resources [...] Our approach is to rank candidate resources according to the similarity score between their context vectors and the context surrounding the surface form [...] we use cosine as the similarity measure [19].

Um einem Wort im Dokument die richtige DBpedia-Ressource zuzuweisen, muss zuerst die richtige Wortbedeutung, abhängig vom textuellen Kontext, in Erfahrung gebracht werden. Unter dem textuellen Kontext versteht man in diesem Fall Wörter, deren Vorkommen im Text den Schluss auf eine bestimmte Ressource zulassen. Das *DBpedia Spotlight* System vergleicht hierbei, wie im Zitat schon beschrieben, den textuellen Kontext der Ressourcekandidaten mit jenem der zukünftigen Entität anhand des Vektorraummodells (siehe auch Kapitel 2, Abschnitt 2.4.3). Die Kosinus-Ähnlichkeit, der Winkel zwischen den beiden Häufigkeitsvektoren, gibt somit einerseits Auskunft über die kontextuelle Ähnlichkeit zwischen Dokument und Ressource und macht andererseits die Zuversicht, dass die gefundene DBpedia-Ressource auch die korrekte Wortbedeutung im Dokument erfasst hat, messbar. Aus diesem Grund fließt der höchste Ähnlichkeitswert einer Entität $\max(sim(e))$ in die Berechnung der Wichtigkeit $I(e)$ mit ein.

Wichtige Anmerkung dieser Wert ist nicht stabil. Für den gleichen Text werden jedes mal unterschiedliche Werte zurückgeliefert, was sich auf einen Fehler im System zurückführen lässt. Dieser Fehler hat auch zur Folge, dass nicht immer dieselben Entitäten im Text erkannt und annotiert werden [35]. Deshalb können die Testfälle (siehe beigelegte CD, mehr dazu im Anhang 9) nur bedingt reproduziert werden, da das Problem beim Großteil der Anfragen aufzutreten scheint.

4.2.5 Rechenbeispiel – Importance I

Diese beispielhafte Berechnung der Importance I soll dem Verständnis der vorgestellten Gleichungen dienen und wird später ebenfalls zu Erklärungszwecken weitergeführt.

Wir nehmen an, dass dem Term $e =$ `CNET` das gleichnamige Konzept $k =$ `CNET`[3] auf DBpedia zugewiesen wurde. Zusätzlich sind diese Ausgangswerte

[3] http://dbpedia.org/resource/CNET

bekannt:

$$
\begin{aligned}
&tf\,(e) = 5 && C = 0{,}8\\
&tpos\,(e) = 0 && l = 6007\\
\\
&w\,(keywords) = 0{,}5 && w\,(keywords) = 0{,}5\\
&w\,(title) = 0{,}5 && w\,(p) = 0{,}3\\
&w\,(h1) = 0{,}4\\
\\
&max\,(sim\,(e)) = 0{,}166
\end{aligned}
$$

Werden diese in die Gleichungen eingesetzt, bekommt man folgendes Ergebnis:

$$
\begin{aligned}
tdr\,(e) &= 1 - \frac{0}{6007} = 1\\
tw\,(e) &= 0{,}5 + 0{,}5 + 0{,}5 + 0{,}3 + 0{,}4 = 2{,}2\\
I\,(e) &= 5 \cdot 2{,}2 \cdot 1 \cdot 0{,}166 =\\
&= 1{,}83
\end{aligned}
$$

4.3 Zuweisung einer treffenden Kategorie

Im vorigen Abschnitt wurde die Funktionsweise der Selektion der für die Klassierung richtungsweisenden Entitäten vermittelt. Das Endprodukt, eine Auswahl der am Besten gereihten Entitäten E, bildet in diesem abschließenden Verfahren den Einstiegspunkt in die DBpedia-Ontologie. In den folgenden Abschnitten wird der tatsächliche Klassierungsprozess mithilfe des DBpedia-Kategoriegraphen erläutert.

4.3.1 Von der Entität zum Graphen

In Abschnitt 4.2.3 wurde das *URI*-Attribut bereits kurz erwähnt und findet nun an dieser Stelle seine Anwendung. Es verweist auf eine Entität in der DBpedia-Ontologie (siehe Abb. 4.2), die es wiederum ermöglicht weitere Informationen aus der Ontologie zu beschaffen. Relevant sind die Relationen der Entität zu ihren Kategorien. In der Ontologie wird dies per *DCMI Metadata Terms*-Namespace [37] abgebildet. Teil davon ist der Term *Subject*, welcher per Definition eine thematische Klasse mit der Ressource verbindet. Den Handhabungsrichtlinien zufolge sollte die Klasse nicht als Literal abgebildet werden, sondern Teil eines kontrollierten Vokabulars sein. Diese Empfehlungen werden von der DBpedia-Ontologie erfüllt, da Entitäten mit Termen des *Category*-Vokabulars verknüpft sind (siehe Abb. 4.3). Da

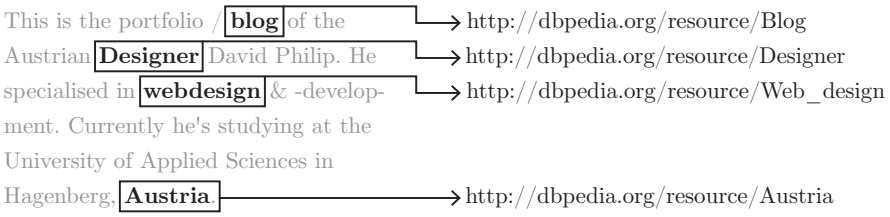

Abbildung 4.2: DBpedia Spotlight Annotation – Jede Entität besitzt eine URI, welche auf die relevante DBpedia-Ressource verweist.

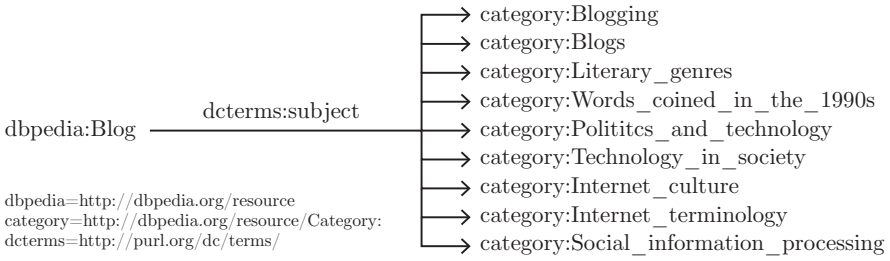

Abbildung 4.3: Mit den der Ressource zugewiesenen Kategorien lässt sich die Brücke zum Kategoriegraphen schlagen.

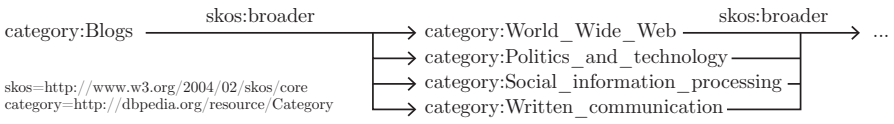

Abbildung 4.4: Die Abbildung illustriert die Relation zwischen Unter- und Überkategorien.

die einzelnen Terme dieses Vokabulars wiederum hierarchisch miteinander in Beziehung stehen, lassen sich mit den initialen, über die Entitäten gewonnenen, Termen weitere unter- beziehungsweise übergeordnete Terme des *Category*-Vokabulars abfragen (siehe Abb. 4.4). Die DBpeida-Ontologie bildet diese Termbeziehungen mit der *broader*-Klasse des *SKOS (Simple Knowledge Organization System)*-Namespace [42] ab.

4.3.2 Aufbau des Kategoriegraphen

Die zuvor beschriebenen Fakten deuten bereits an, wie die Konstruktion des Graphen funktioniert. Eine genauere Beschreibung dieses Vorgangs folgt im nächsten Abschnitt.

Eine rekursive Lösung liegt bei dieser Aufgabe nahe. Auf diese Art wird ausgehend von den ersten Kategorien, welche von den Entitäten abgeleitet wurden (siehe Abs. 4.3.1), der *Kategoriegraph* aufgebaut (siehe Algorithmus 4.3). Die maximale Rekursionstiefe von 3 erbrachte dabei ausgewogene Performance und Qualität. Im Graphen tiefer liegende Lösungen wären ohnehin nicht von Interesse, da diese sich thematisch mit großer Wahrscheinlichkeit zu weit entfernt haben. Die nächste Aufgabe stellt die Lösungsfindung dar.

Algorithmus 4.3: Konstruktion des Graphen anhand des DBpedia-Kategorievokabulars.

```
1: d ← 0
2: md ← 3
3: buildGraph(terms)
   Konstruiert rekursiv den Kategoriegraph bis zu einer best. Tiefe md
4:     if sizeof(terms) > 0 and d ≤ md then
5:         graph add terms
6:         for i = 0 → sizeof(terms) do
7:             subterms add getSubTerms(terms[i])
8:             i ← i + 1
9:         end for
10:        d ← d + 1
11:        buildGraph(subterms)
12:    end if
13: end
```

Unter anderem geht es darum, welche Heuristiken bei der Suche helfen und wie sich eine geeignete Lösung qualifiziert.

Bei der Suche im erstellten Graphen kann nicht einfach willkürlich gesucht werden. Es bedarf einer Heuristik die bestimmt, welche Knoten als nächstes geöffnet werden und einer Bedingung, die eine geeignete Lösung identifiziert. Auf diese Komponenten wird nun näher eingegangen.

4.3.3 SimRank – Knotenähnlichkeitsmaß

> We propose a complementary approach [called SimRank; David Gösenbauer] [...] that measures similarity of the structural context in which objects occur, based on their relationships with other objects [7].

SimRank ist ein Maß, welches die Ähnlichkeit zweier Knoten im Graphen anhand von Verbindungen mit anderen Knoten bestimmen lässt. Nach dem Leitsatz der Autoren „two objects are similar if they are related to similar objects“ [7] ist die Ähnlichkeit zweier Knoten proportional zu deren Schnittmenge an Nachbarknoten. Der SimRank-Score ist Teil der Entwicklung einer Heuristik, um die Suche nach einer geeigneten Kategorie zu ermöglichen.

Ein Minimalbeispiel aus [7] soll die Berechnung des SimRank-Score verständlich machen:

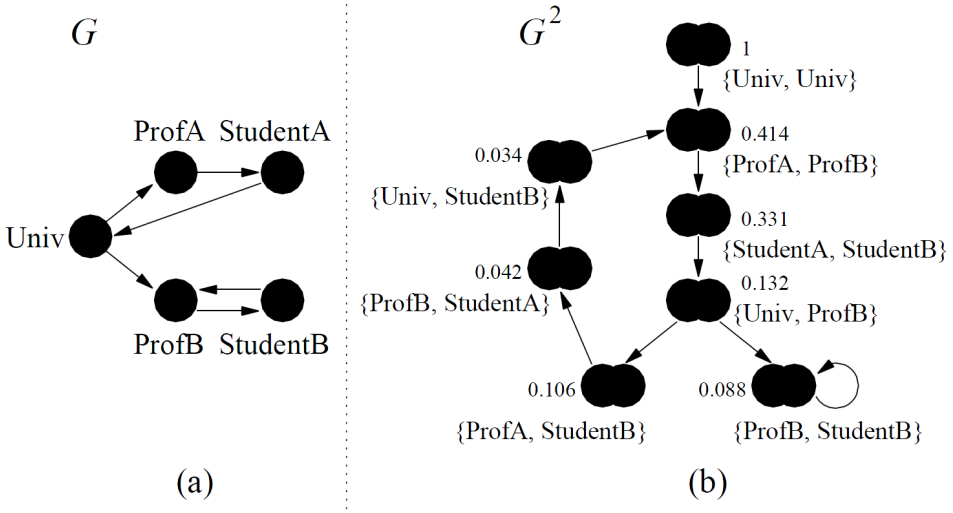

Abbildung 4.5: a) Beispielgraph G und b) G^2 in Form von Knotenpaaren

Der Graph G zeigt einen gerichteten Graphen, vergleichbar mit dem DBpedia-Kategoriegraphen. Aufgrund der strukturellen Information, dass *ProfA* und *ProfB* von *Univ* referenziert werden, nimmt SimRank deren Ähnlichkeit an. Diese Idee wird insofern verallgemeinert, als dass man aufgrund des Wissens um diesen Umstand auf die Ähnlichkeit der Knoten *StudentA* und *StudentB* schließt. Die Ähnlichkeit wird somit von Knotenpaar zu Knotenpaar propagiert. Das Beispiel könnte weitergeführt werden, indem man von *StudentA* und *StudentB* auf *Univ* und *ProfB* schließt, von dort auf *ProfA* und *StudentB*, etc.

Der soeben beschriebene Prozess lässt sich vereinfacht als Graph G^2 darstellen, wobei in G^2 ein Knotenpaar $\{a, b\}$ auf ein Knotenpaar $\{c, d\}$ zeigt, wenn in G a auf c und b auf d zeigt. Per Definition ist ein Knoten maximalähnlich mit sich selbst. Somit wird $\{Univ, Univ\}$ ein Ähnlichkeitswert von 1 zugewiesen. Auf Basis dieser initialen Zuweisungen kann nun rekursiv die Ähnlichkeit aller möglichen Knotenpaare des Graphen berechnet werden. Dies geschieht mithilfe folgender Gleichung [7, Anm.: I wird später als N

weitergeführt]:

$$s\,(a,b) = \begin{cases} \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s\,(I_i\,(a)\,, I_j\,(b)) & \text{wenn } a \neq b \\ 1 & \text{wenn } a = b \\ 0 & \text{wenn } I\,(a)\,, I\,(b) = \emptyset \end{cases} . \tag{8}$$

Die Konstante C, wobei $0 < C < 1$, stellt den Zuversichts- beziehungsweise Verfallswert dar. Anhand des Graphen G in Abbildung 4.5 lässt sich seine Existenz einfach begründen. Da $s\,(\mathit{Univ}, \mathit{Univ}) = 1$ wäre es willkürlich zu sagen, dass $s\,(\mathit{ProfA}, \mathit{ProfB}) = s\,(\mathit{Univ}, \mathit{Univ}) = 1$. Deswegen drücken wir es folgendermaßen $s\,(\mathit{ProfA}, \mathit{ProfB}) = C \cdot s\,(\mathit{Univ}, \mathit{Univ}) = 1$ aus, was unsere Zuversicht an der Ähnlichkeit des Knotenpaares verdeutlicht, die geringer ausfällt, je weiter man sich von den ursprünglichen Knotenpaaren entfernt.

Für einen Graphen mit der Größe m wäre für jedes Knotenpaar eine solche Gleichung zu lösen. Man käme demnach auf insgesamt m^2 Gleichungen. Die Anzahl der Berechnungen konnte beim *Enguided*-Klassifikator verringert werden, da die Knoten bei der Graphenkonstruktion bereits einem Objekt vom Typ *JGraph* als solche hinzugefügt wurden. Dies ermöglicht die Richtungsinformation, welche der Konstruktionsprozess bereits impliziert, in Form von gerichteten Kanten mitzuspeichern. Somit werden nur alle relevanten Knotenpaare berechnet (siehe Algorithmus 4.4), was Rechenaufwand einspart. Anders ausgedrückt werden nur Knotenpaare berechnet, die auch selbst Nachbarn sind und demnach eine Distanz von 1 aufweisen.

4.3.4 Kategoriezuweisung

Das Suchen und Finden einer validen Lösung im Graphen ist nur möglich, wenn eine solche sich aufgrund ihrer Eigenschaften von anderen Kategorien unterscheidet. Der Suchalgorithmus muss eine Lösung identifizieren, gegen andere mögliche Lösungen abwiegen und sich letztendlich für die Beste entscheiden können.

Enguided setzt auf ein Lösungsvokabular S, welches vom Kategoriegraphen der DBpedia Ontologie abgeleitet wird und dem Klassifikator dabei hilft Lösungskategorien von anderen zu unterscheiden. Die Lösungsklassen werden wie bereits erwähnt von der Ontologie abgeleitet. Dazu werden alle Kindselemente der `Main_topic_classifications`-Klasse[4] als mögliche Lösungen festgelegt (siehe Abb. 6.1).

Der Vergleich mit anderen Lösungen entfällt, da mit dem Einsatz einer Heuristik $h\,(n)$

$$h\,(n) = \frac{\sum_{i=1}^{|N(n)|} \sum_{j=1}^{|N(n)|} s\,(N_i\,(n)\,, N_j\,(n))}{|N\,(n)|} \tag{9}$$

[4] http://dbpedia.org/page/Category:Main_topic_classifications

Algorithmus 4.4: Berechnung des SimRank-Scores für alle Knotenpaare des Graphen G. Das Vorgehen ist dabei ähnlich der Graphenkonstruktion selbst (siehe 4.3).

```
1:  calculateSimRank
    Berechnung des SimRank-Scores für alle Knotenpaare des Graphen G
2:      for k = 0 → K do     ▷ Anzahl d. Berechnungsdurchläufe, laut 4.3.3
    K ≈ 5 ideal
3:          n ← getAllNodesOfGraph()
4:          for i = 0 → sizeof(n) do
5:              neighbours ← getNeighboursOf(n[i])
6:              for j = 0 → sizeof(n) do
7:                  if areNeighbours(n[i], n[j]) then
8:                      simrank ← getSimRankOfPair(n[i], n[j])   ▷ Siehe
    Gleichung 8
9:                  end if
10:                 j ← j + 1
11:             end for
12:             i ← i + 1
13:         end for
14:     end for
15: end
```

bei der Suche zuerst der optimale Weg bis zur maximalen Suchtiefe *md* gewählt wird. Sollte der Prozess während des Öffnens dieses *Pfades* auf keine Lösung stoßen, wird per iterativen rückwärtsgehen nach der nächsten besten Lösung gesucht.

Die eingesetzte Heuristik $h(n)$ entscheidet bei der Suche im Graphen G welcher Knoten als nächster geöffnet wird, wobei der *SimRank* $s(a, b)$ entscheidend ist. Genauer gesagt berechnet $h(n)$ die Summe der Ähnlichkeiten der Nachbarknoten N des Knotenkandidaten n. Somit wird gewährleistet, dass die Suche im Rahmen ähnlicher Kategorien voranschreitet und demnach die Lösung auch maximal ähnlich zur ursprünglichen Entität ist. Aus diesem Grund wird auch eine maximale Suchtiefe beziehungsweise eine Maximaldistanz, welche sich aus der Summe der geöffneten Knoten des momentanen Lösungspfades ergibt, eingesetzt.

Auf Basis dieser Ideen kam der Suchalgorithmus 4.5 zum Einsatz. Dessen Ergebnis ist eine geordnete Liste von Knoten, die den Lösungspfad im Graphen, von der Ausgangsentität zur Lösungskategorie, repräsentiert. Eine solche Liste existiert demnach für jede aus dem Text übernommene Entität und somit gibt es vorerst mehrere Lösungen. Um genau eine Kategorie zu erhalten, werden die Lösungspfade nach der ursprünglichen *Importance* $I(e)$ gewichtet. Die Länge des Pfades p beeinflusst die Endwertigkeit $w(p)$,

indem sie mit dem Verfall- beziehungsweise Zuversichtswert C (siehe auch Abs. 4.3.3) kombiniert wird:

$$w(p) = I(e) \cdot C^{|p|-1}. \tag{10}$$

Ist das letzte Element des jeweiligen Pfades p Teil des Lösungsvokabulars $S = \{c_1, ...c_n\}$, wurde eine gültige Kategorie für den Artikel gefunden. Die Wertigkeiten mehrmals vorkommender Kategorien werden summiert (siehe Gleichung 11), wobei die Lösungen nach $w(c)$

$$w(c) = \sum_{i=1}^{|P(c)|} w(P_i(c)) \tag{11}$$

gereiht werden. Sollte es zu einer Doppelplatzierung in Folge gleicher Wertigkeiten kommen, wird die öfter vorkommende Kategorie als Lösung bestimmt. Die Wahrscheinlichkeit, dass wertegleiche Kategorien gleich oft vorkommen, ist vernachlässigbar gering.

Algorithmus 4.5: Findet den besten Lösungsweg nach $h(n)$

```
1:  getBestPath(entityNode)

2:      startNode ← entityNode
3:      bestPath add entityNode
4:      d ← 0
5:      md ← 3
6:      while foundValidCategory do
7:          pathsize ← sizeof(bestPath)
8:          N ← getParentNodes(startNode)
9:          if sizeof(N) ≠ 0 and d ≤ md then  ▷ d = aktuelle Tiefe, md =
    maximale Suchtiefe
10:             if backsteps > 0 then
11:                 bestPath ← bestPath[0]...bestPath[pathsize − backsteps]
12:                 backsteps = 0
13:             end if
14:             for i = 0 → sizeof(S) do              ▷ S = Lösungskategorien
15:                 if N contains S_i then
16:                     foundValidCategory ← true, bestPath add S_i
17:                 end if
18:                 i ← i + 1
19:             end for
20:             if !foundValidCategory then
21:                 bestNode ← getBestNode(N),      ▷ Berechnet h(n) für
    alle N = {n_1...n_n}
22:                 bestPath add bestNode, openedNodes add bestNode,
23:                 startNode ← bestNode
24:             end if
25:             d ← d + 1
26:         else
27:             backsteps ← backsteps + 1, d ← d − 1
28:             if pathsize − (1 + backsteps) ≥ 0 then
29:                 startNode ← bestPath[pathsize − (1 + backsteps)]
30:             else
31:                 foundValidCategory ← true        ▷ Für entityNode gibt es
    keine Lösung, da backsteps bis zum Anfang zurückführen
32:             end if
33:         end if
34:     end while
35:     return bestPath.
36: end
```

4.3.5 Rechenbeispiel – Kategoriegewichtung $w(c)$

Dieses Beispiel führt die Rechnung von Abschnitt 4.2.5 fort. Diese Ausgangswerte sind bekannt:

$$
\begin{aligned}
P &= \{\ \texttt{CNET}\ \}, k = \texttt{CNET} \\
&\text{Schritt 1:} \\
N(k) &= \{\ \texttt{Commerce\_websites},\ \texttt{CNET\_Networks}\ \} \\
N(\texttt{Commerce\_websites}) &= \{\ \texttt{Commerce}, \\
&\quad\ \texttt{Information\_technology\_management}, \\
&\quad\ \texttt{Economics\_of\_service\_industries}\ \} \\
N(\texttt{CNET\_Networks}) &= \{\ \texttt{Supply\_chain\_management\_terms}, \\
&\quad\ \texttt{Banking\_technology}, \\
&\quad\ \texttt{Applications\_of\_cryptography}\ \}
\end{aligned}
$$

Die Berechnung von $h(n)$ laut Gleichung 9 für `Commerce_websites` und `CNET_Networks` liefert die folgenden Ergebnisse:

$$
\begin{aligned}
h(\texttt{Commerce\_websites}) &= 0{,}3 \\
h(\texttt{CNET\_Networks}) &= 0{,}1
\end{aligned}
$$

Die dafür benötigten Ähnlichkeiten $s(a, b)$ werden einer Hashmap entnommen. Nun ist es möglich den nächsten zu öffnenden Kategorieknoten auszuwählen:

$$
\begin{aligned}
h(\texttt{Commerce\_websites}) > h(\texttt{CNET\_Networks}) \\
\Rightarrow \\
P = \{\ \texttt{CNET},\ \texttt{Commerce\_websites}\ \} \\
k = \texttt{Commerce\_websites} \text{ und zurück zu Schritt 1}
\end{aligned}
$$

Auf diese Weise wird im Graphen nach einer Kindskategorie von `Main_topic_classification` gesucht. In diesem Beispiel wird der Suchalgorithmus fündig und wir erhalten dieses Ergebnis:

$$
\begin{aligned}
P &= \{\ \texttt{CNET},\ \texttt{Commerce\_websites},\ \texttt{Electronic\_commerce} \\
&\quad\ \texttt{Information\_technology\_management}, \\
&\quad\ \texttt{Information\_technology},\ \texttt{Technology}\ \}
\end{aligned}
$$

$$
w(P) = w(c) = I(e) \cdot C^{|P|-1} = 1{,}83 \cdot 0{,}8^5 = 0{,}6
$$

Kapitel 5

Implementierung

Im vorigen Kapitel wurde die Idee und Methodik erklärt, um die Textklassifikation mit Hilfe der externen Wissensbasis DBpedia zu realisieren. Das folgende Kapitel gibt einen Überblick über die technischen Implementierungsdetails des Projekts *Enguided*.

5.1 Überblick

Den Anfang macht *Apache UIMA* (siehe Abb. 5.1), welches das Fundament der entwickelten Software bildet und die Systemarchitektur zum Großteil vorgab. Dieser Teil wird detaillierter beschrieben, da Funktionsweise und Fachausdrücke für die Folgeabschnitte von Bedeutung sind.

Die eigene Komponente wird im Anschluss behandelt, wobei einerseits die *Datenaufbereitung* durch den *Tika MarkupAnnotator* eine Rolle spielt, andererseits sollen die Einzelschritte der *Datenverarbeitung* von einem technischen Standpunkt aus genauer beleuchtet werden.

Den Schluss bilden die Erklärungen des *Graphen*, sowie deren Aufbau zur Laufzeit per N-Triple-Store.

5.2 Apache UIMA

> UIMA stands for Unstructured Information Management Architecture and is a component architecture and software framework implementation for the analysis of unstructured content like text, video and audio data [44].

Wie das Zitat bereits andeutet, gibt das UIMA dem Entwickler die Möglichkeit seine eigenen modularen Analysekomponenten zu implementieren, diese optional mit anderen bereits für UIMA vorhandenen Modulen zu kombinieren und als Applikation auszuliefern. Vergleichbar mit dem GATE-Framework [1] ist das Hauptanwendungsgebiet die Textanalyse. Ursprüng-

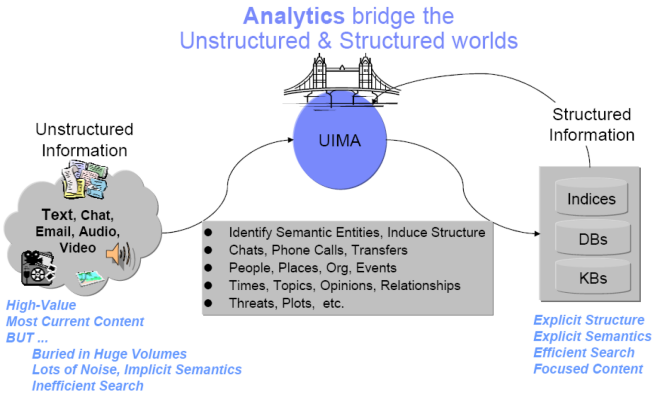

Abbildung 5.1: UIMA bildet die Basis für Applikationen die unstrukturierte in strukturierte, maschinell verarbeitbare Daten umwandeln [33].

lich von der Firma IBM entwickelt, befindet es sich nun unter der Obhut von Apache.

Ein entscheidender Vorteil gegenüber anderen Systemen ist die „Schlankheit“ der Kernsoftware und die einfache Individualisierbarkeit. Diese Punkte, eine steile Lernkurve und die Verfügbarkeit als Java-Framework, inklusive eines Eclipse-Plugins, waren der Grund für die Umsetzung des Textklassifikators *Enguided* mit UIMA.

5.2.1 UIMA Java Framework

Eine Applikation besteht aus mehreren Komponenten. Jede dieser Komponenten implementiert vom Framework vorgegebene *Interfaces* und wird per *XML-Descriptor-Datei* definiert. Das Framework ist dabei verantwortlich für das Management der Komponenten und deren Datenfluss untereinander.

Installation

Das *UIMA Java Framework* lässt sich komfortabel via Eclipse-Plugin installieren [33]. Voraussetzung für jedes UIMA-Projekt ist die Verlinkung des selbigen mit dem Hauptverzeichnis der UIMA Installation auf der Festplatte. Außerdem erfordert das Anlegen eines neuen Projekts zusätzliche Schritte, im Vergleich zur üblichen Projekterstellung in Eclipse.

Nach der Erstellung muss das Projekt per Kontextmenübefehl in die PEAR-Struktur gebracht werden [34]. Vorteil dieser Verzeichnisstruktur ist, dass das Projekt am Ende als PEAR-Datei exportiert werden kann. Ein praktisches Feature, welches die Applikation beispielsweise einem Servlet als Ressource zur Verfügung stellt.

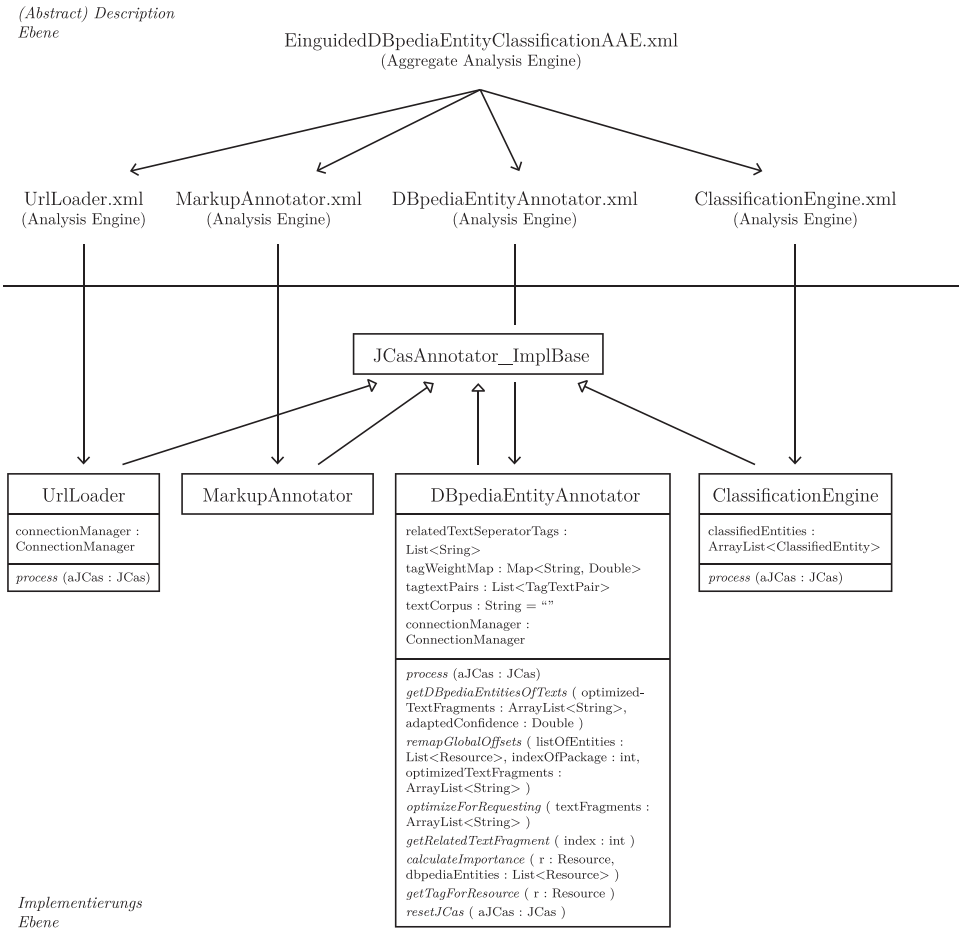

Abbildung 5.2: Die Architektur lässt sich in 2 Ebenen unterteilen. Eine Konzeptions- beziehungsweise Beschreibungsebene definiert den Aufbau des Systems und die Implementierungsebene birgt dessen Umsetzung. Die Ebenen sind über die Komponenten (Analysis Engines) miteinander verbunden.

Komponenten

Unter den vom UIMA Eclipse Plugin eingeführten Dateitypen ist auch der *Analysis Engine Descriptor*. Dabei handelt es sich um eine .xml-Datei, welche eine Komponente beschreibt und auf deren Java-Klasse verweist, welche die Programmlogik enthält (siehe Abb. 5.2). Diese Klasse ist im Großteil der Fälle eine Unterklasse von *JCasAnnotator_ ImplBase*, die bereits die Logik mitbringt, um die Komponente in das System einzubinden. Der Entwickler braucht lediglich die *process(JCas aJCas)* Methode zu überschreiben, um das eigene Programm implementieren zu können.

Type System – Annotationsschema

Das Type System beschreibt die Struktur, beziehungsweise das Schema der systemeigenen Annotationen, also das Produkt der Analyse. Der *Component Descriptor Editor*, ein mit dem Plugin mitgeliefertes GUI zur Komponentenkonfiguration, bietet die Möglichkeit ein eigenes Type System zu erstellen. Zusätzlich kann die Spezifikation über das GUI automatisch in eine Java-Klasse umgewandelt werden. Die eigenen Type-System-Klassen erben von der *Annotation*-Klasse, welche standardmäßig vom CAS eingesetzt wird, sollte keine eigene Definition existieren.

Wird ein selbst definiertes Annotationsschema von mehreren Komponenten genutzt, so muss dieses in deren jeweiligem *Component Descriptor*-File referenziert werden. Hierzu lässt sich jedes Schema im .xml-Format exportieren.

Common Analysis Structure – CAS

Die CAS hält das analysierte *Artefakt*, zum Beispiel Text oder Quellcode. Dieses kann zusätzlich in mehreren Ansichten (*Views*) abgelegt werden, um beispielsweise auf die Ergebnisse unterschiedlicher Analysemethoden zuzugreifen. Jede einzelne „View“ wird als *Sofa* bezeichnet. Außerdem werden die von den Komponenten deklarierten Type Systems in der CAS gespeichert. Zusätzlich hält es die Analyse-Metadaten selbst, deren Struktur wiederum vom jeweiligen Type System der Komponenten vorgegeben wird.

Die CAS ist umgangssprachlich formuliert nichts anderes als ein Container, der Analysedaten kapselt und dem Entwickler zur Verfügung stellt. Da jede Komponente auf dasselbe CAS-Objekt zugreift, werden auf diesem Weg Daten von einer Komponente zur nächsten überreicht.

5.3 Systemarchitektur

Abbildung 5.3 veranschaulicht die wichtigsten Teile der Systemarchitektur des Projekts. Im Zentrum stehen die vier Hauptkomponenten:

- Der **UrlLoader** ist verantwortlich für die initiale Datenbeschaffung,
- danach annotiert der **Tika MarkupAnnotator** den Text nach dessen Markup-Strukturen,
- im Anschluss markiert der **DBpediaEntityAnnotator** alle DBpedia-Entitäten im Text und
- die **EnguidedClassificationEngine** ergänzt letztendlich deren Annotationen jeweils um ein Kategorieattribut.

Die *Aggregate Analysis Engine* ist im Vergleich zu den Komponenten ein klassenloses Konstrukt, welches diese zu einer Einheit zusammenschließt,

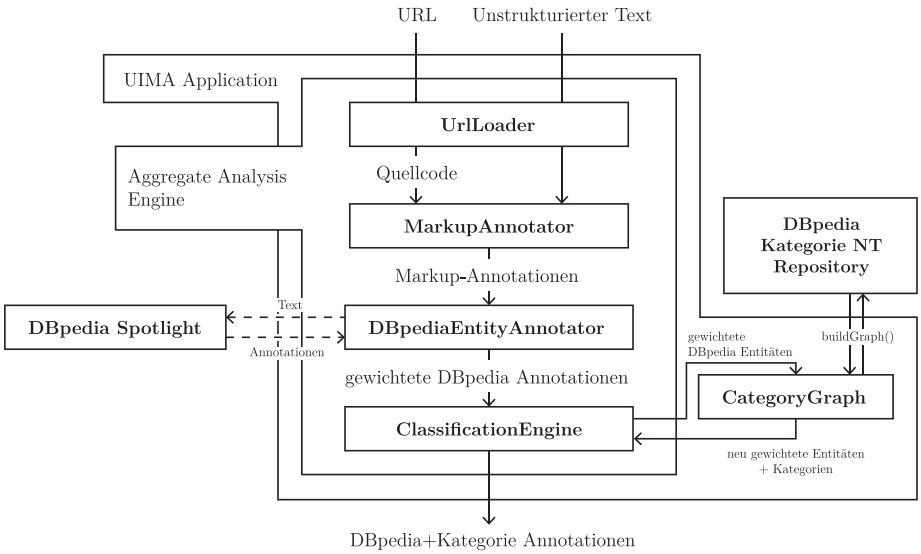

Abbildung 5.3: Die wichtigsten Bestandteile des Systems.

indem diese in Serie geschaltet werden. Das Produkt der vorangehenden Komponente ist die Ausgangslage für die nächste ihrer Art, was durch die Pfeile, welche die vier Hauptkomponenten in der Abbildung durchqueren, verdeutlicht wird.

In den nachfolgenden Sektionen wird genauer auf diese Kernkomponenten eingegangen.

5.4 URL-Loader

Der *UrlLoader* kümmert sich um die anfängliche Datenaggregation, je nachdem ob es sich beim Text um eine URL handelt oder nicht. Im Falle einer Website-URL, wird deren geladener Quellcode, ansonsten der unveränderte Text, an die nächste Komponente übergeben.

Beim Laden der Webseite kann es unter Umständen zum nachfolgenden Problem kommen.

5.4.1 HTTP 405 Fehler

Der Errorcode steht kurz für *Method not allowed*. Dieser Fehler tritt auf, sollte der Server den HTTP-Request ablehnen. Die relevanten Parameter des Requests lauten wie folgt:

```
Request URL :     (vom System erhaltene URL)
Request Method : GET
Accept :         text/html, application/xhtml+xml,
                 application/xml,*; q=.2, */*; q=.2
Accept-Charset : ISO-8859-1, utf-8; q=0.7, *; q=0.3
```

Fehlerquellen sind entweder die Servereinstellungen oder eine fehlende Spezifikation im Request-Header. Die Tatsache, dass bei den Tests der Großteil der Seiten geladen werden konnte und lediglich Seiten die von *Google Inc.* gehostet werden zu diesem Fehler führten, entlastet das eigene System. Keine Lösung liefert das Überschreiben und Ersetzen der Parameter durch die eines Browsers. Anfangs wurde vermutet, dass Google den *User-Agent* vom Typ *Java* blockiert.

Eine mögliche Lösung der Problematik wäre die Verwendung der *Google Translate API*, um einerseits nicht-englischsprachige Webseiten analysieren zu können und andererseits den Request unter dem Deckmantel des Hosts selbst abzusetzen. Dies ist ein guter Trick, um auch andere Server an der Blockade des Klassifikators zu hindern, denn die meisten Betreiber lehnen einen Request von Google nicht ab.

Stand der Dinge ist, dass das eigene System im Falle dieses Fehlers keine Klassifikationsergebnisse liefern kann.

5.5 Apache Tika – Markup Annotation

Diese Komponente ist dafür zuständig, geladenen Quelltext oder unstrukturierten Text in maschinell verarbeitbare Annotationen umzuwandeln. Basierend auf der *Apache TIKA API* [32], deren Kernfunktion die Extraktion von Metadaten und strukturierten Textinhalten unter der Verwendung existierender Parser-Bibliotheken ist, ermöglicht sie die für UIMA angebotene Komponente, unter anderem Markup-Dokumente, zu annotieren.

Dabei kommt *TagSoup* zum Einsatz [43]; ein stabiler, in Java implementierter Parser, der, unabhängig davon ob das HTML oder XML-Markup valide ist, jedes Dokument ohne Syntax-Fehlermeldungen bis zum Ende einliest. Dazu werden fehlende Tags oder Tagteile vervollständigt. So wird

```
<li>TagSoup is great!
```

zu

```
<html>
  <body>
    <ul>
      <li>TagSoup is great!</li>
    </ul>
  </body>
</html>
```

oder

```
This is <B>bold, <I>bold italic, </b>italic, </i>normal text
```

zu

```
This is <b>bold, <i>bold italic, </i></b><i>italic,</i>
normal text.
```

Betreffend der Formatierung und dem Markup ist dieses Verfahren sehr robust und tolerant. Durch die automatische Vervollständigung offener oder ausgelassener Tags stellt es sicher, dass so viele Daten wie möglich aus dem Quelltext extrahiert werden.

Das Resultat sind annotierte Markupkonstrukte des Quelltextes (siehe Abb. 4.1).

5.6 DBpedia Spotlight – Entitäten Annotation

Diese Komponente integriert das *DBpedia Spotlight Service* in die Systemarchitektur, indem es zuerst die Annotationen der vorherigen Komponente für die Übergabe als Parameter an das Service aufbereitet und dessen Antwort wiederum in das, für das System spezifizierte, Type System umwandelt. Außerdem findet die initiale Gewichtung der Entitäten nach deren *Importance I* (siehe Abs. 4.2.4) statt, sprich der im Text ausgemachten DBpedia Ressourcen.

5.6.1 Datenselektion

Im ersten Schritt werden die vom *MarkupAnnotator* generierten Annotationen für die Übertragung an das DBpedia Spotlight Service optimiert. Wie bereits im Abschnitt 4.2 beschrieben werden diese gefiltert und vereinigt. Für die Selektion der Inhalte wird Gebrauch von einem *String-Array* gemacht, in welchem die relevanten Tag-Types gespeichert werden. Danach wird über die Annotationen des `CAS` iteriert und überprüft, ob sich das aktuelle Element in dieser Liste befindet. Ist das der Fall, wird ein neues Objekt der Klasse `TagTextPair` mit dem aus der Annotation extrahierten Tag-Type, Text und Abstand zum Korpusanfang (Offset) instanziert. Die Implementierung dieser

Klasse geschah aufgrund der Voraussetzung, dass den vom Service zurückgelieferten Entitäten ein Tag-Type zugewiesen wird um diese gewichten zu können.

Außerdem wird darauf geachtet, dass keine Duplikate in die Liste aufgenommen werden, da dies die spätere Gewichtung negativ beeinflussen würde. Die Liste `List<TagTextPair> tagTextPairs` enthält nach diesem Prozess alle relevanten Daten und ermöglicht deren Weiterverarbeitung.

Extrahiert wird der Inhalt folgender Tag-Types:

`p, h1, h2, h3, h4, h5 und h6`

Nicht unterstütztes Markup

Seiten die sich zum Beispiel aus Designgründen noch auf *Tables* verlassen und Tabellendaten an sich, können nicht oder nur bedingt zur Klassifikation herangezogen werden. Da Annotationen mit dem Tag-Type *td* immer denselben Start- und Endoffset besitzen, ist die Position des darin enthaltenen Textes unbekannt und dieser somit nicht extrahierbar. Obwohl diese Problematik bei den Tag-Types *table* oder *tr* nicht besteht, liefert die Extraktion hier keine nützlichen Ergebnisse, da Verschachtelungen in Form von Markupstrukturen mitextrahiert werden. Diese stören die Annotation des DBpedia Spotlight Services und lassen die Qualität des Ergebnisses sinken. Deswegen wurde entschieden, auf die darin enthaltenen Informationen zu verzichten.

Weiters können Seiten, welche Inhalte im Hintergrund nach dem eigentlichen Ladevorgang zum Beispiel per AJAX laden und darstellen, nicht klassifiziert und annotiert werden.

5.6.2 Datenaufbereitung

Nachdem die einzelnen, von verschiedenen Tags umklammerten Textinhalte in einer Liste gespeichert wurden, werden diese nach deren vermuteter Zusammengehörigkeit gruppiert (siehe Abs. 4.2.2). Dafür werden die zu vereinenden Fragmente in einem einzigen String-Objekt zusammenhängend gespeichert. Sollte dessen Länge während des Gruppierens die als Konstante definierte Maximallänge `PACKAGE_CHAR_COUNT_MAX` einer Textgruppe überschreiten, muss es geteilt werden. Dazu wird die Zeichenfolge „|||||“ eingefügt, bevor das nächste Fragment, das die Textgruppierung zu lang werden lassen würde, an den String gehängt wird. Somit entsteht im schlechtesten Fall ein kurzer Überschuss an Text, welcher als eigene Gruppe behandelt wird. Jedoch muss unter allen Umständen dafür gesorgt werden, dass die `Request-URL` nicht zu lange wird, um zu verhindern, dass der Server den HTTP-Request ablehnt. Enthält eine Textgruppe bei der nachfolgenden Überprüfung die zuvor genannte Zeichenfolge, wird dessen Position im String

ermittelt und an dieser Stelle in weitere Gruppen unterteilt. Unabhängig davon werden die einzelnen Gruppen in der Liste `List<String> textGroups` gespeichert. Ist die Iteration über alle Objekte der Liste `List<TagTextPair> tagTextPairs` abgeschlossen, ist auch deren Gruppierung erledigt.

Es ist an dieser Stelle unvermeidlich einen Kompromiss zwischen optimaler Aufbereitung des Textes in Hinsicht auf die Disambiguierung und der Anzahl der Requests die das Service letztendlich erreichen zu schließen. Deswegen werden kleine aufeinanderfolgende Gruppen vereinigt, wobei die Textlänge mit der Konstante `PACKAGE_CHAR_COUNT_MAX` und die Maximalanzahl an vereinigten Textgruppen mit `PACKAGE_GROUPS_COUNT_MAX` begrenzt ist. Ein solches Vorgehen garantiert, dass nicht mehr Requests als nötig abgesetzt werden und hält die Beeinflussung der Disambiguierung in Grenzen.

DBpedia Spotlight Lokal Es ist auch möglich einen Spotlight-Server lokal laufen zu lassen. In diesem Fall würde der letzte Schritt entfallen, jedoch benötigt der Cache sehr lange bis er den Stand erreicht hat, dass Requests in angemessener Zeit prozessiert werden. Deswegen und aufgrund der hohen Hauptspeicheranforderungen wurde für die Projektumsetzung auf einen externen Server zurückgegriffen. In Zukunft ist die Verwendung einer lokalen Variante geplant.

5.6.3 DBpedia Spotlight Webservice

Das DBpedia Spotlight Webservice bietet drei REST-Schnittstellen [36]:

- **http://spotlight.dbpedia.org/rest/annotate** setzt Text als Eingabeparameter voraus, erkennt die darin enthaltenen zu annotierenden Entitäten / Konzepte und bestimmt deren wahrscheinlichste Disambiguierung abhängig vom Kontextes.
- **http://spotlight.dbpedia.org/rest/disambiguate** benötigt Wiki-Markup[1] womit die Suche nach Entitäten beziehungsweise Konzepten entfällt, da diese bereits durch die Markup-Struktur hervorgehoben werden und disambiguiert diese wieder anhand deren Kontext.
- **http://spotlight.dbpedia.org/rest/candidates** ist ähnlich zur *annotate*-Variante, gibt jedoch eine gereihte Listen an möglichen Disambiguierungen für jede Entität zurück.

Für die Umsetzung wurde die erste Schnittstelle gewählt und somit die Entscheidung, welche die beste Disambiguierung darstellt dem Service überlassen. Die zweite Variante kommt, aufgrund der Voraussetzung, dass der Markup-Text direkt von Wikipedia stammt, nicht in Frage.

[1] Eine Gruppe von vereinfachten Auszeichnungssprachen, die eine vereinfachte Alternative zu HTML darstellen und benutzt werden, um Beiträge in Wikis zu formatieren [46].

Die Kommunikation mit der Schnittstelle funktioniert, genauso wie das Laden der Webseiten, über die `ConnectionManager`-Klasse. Sie stellt die Verbindung her und bietet Funktionen wie das Auslesen der Antwort in einen String. Der Request-Header und dessen Parametrisierung setzt sich wie folgt zusammen:

```
Request URL:    http://spotlight.dbpedia.org/rest/annotate
Request Method: GET
Accept:         application/json
Content Type:   application/x-www-form-urlencoded
text:           (die aktuelle Textgruppe)
support:        100
confidence:     0.2
```

Die Parameter *support* und *confidence* dienen zum Ausschluss der Disambiguierungskandidaten, die unter diesen Werten liegen. Sie wurden durch Testen des Systems mit diesen Werten definiert. Ein sehr hoher *confidence*-Wert würde zwar die Qualität der zurückgelieferten Resultate bezüglich deren korrekter Disambiguierung anheben, jedoch sinkt meistens proportional dazu die Anzahl an zurückgegebenen Entitäten, sodass keine Klassierung der Webseite mehr möglich ist. Der *support* beschreibt die Etabliertheit einer DBpedia-Ressource, indem es die Anzahl ihrer eingehenden Links darstellt.

Wird ein Request mit den beschriebenen Parametern abgesetzt, erhält man je nach aktueller Auslastung des Servers relativ rasch die Antwort im .json-Format:

```
{
  "text" : "Load or edit text here.",
  "confidence" : "0.2",
  "support" : "100",
  "types" : "",
  "sparql" : "",
  "policy" : "whitelist",
  "Resources" : [
    {
      "URI": "http://dbpedia.org/resource/Editing",
      "support": "2103",
      "types": "Freebase:/book/book_subject,Freebase:/book",
      "surfaceForm": "edit",
      "offset": "8",
      "similarityScore": "0.08902490884065628",
      "percentageOfSecondRank": "0.8044725456389852"
    }, ...
  ]
}
```

Um die Entitäten gelistet als `Resources` in der .json-Datei als Java-Objekte abzubilden, wurde die *GSON*-Bibliothek [39] eingesetzt. Dabei muss eine Klasse erstellt werden, welche die Struktur des JSON-Objekts nachbildet.

5.6.4 Verarbeitung der Annotationen

Für jede Textgruppe wird das beschriebene Verfahren eingeleitet und somit die darin enthaltenen Annotationen zurückgeliefert. Das Ergebnis ist eine ungeordnete Liste an, auch mehrfach darin enthaltenen, Annotationen. Diese Liste gilt es zu Sortieren, da nicht alle, sondern nur die markantesten Annotationen für die Klassierung eingesetzt werden. Um einen solchen Sortiervorgang zu ermöglichen, ist eine Gewichtung von Nöten (siehe auch Abs. 4.2.4). Danach werden die besten Annotationen, deren Anzahl durch die Konstante `RESULT_CANDIDATES_COUNT_MAX` limitiert ist, in das definierte Type-System übertragen (siehe Erklärung in Abs. 5.2.1), um im Anschluss von der nächsten Komponente gelesen werden zu können. Das Type-System ansich wird in einer .xml-Datei definiert, wobei es auch komfortabel per GUI erstellt werden kann. Das für *Enguided* eingesetzte Type-System übernimmt den Großteil der DBpedia Spotlight Ressourcenattribute, wie zum Beispiel *URI*, *support*, *offset* etc. und fügt diese zur Speicherung der Klassifikationsergebnisse hinzu (siehe XML-Sourcecode in List. 5.1).

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <typeSystemDescription xmlns="http://uima.apache.org/resourceSpecifier">
3   ...
4   <featureDescription>
5     <name>importance</name>
6     <description>Gewichtung der DBpedia Entität.</description>
7     <rangeTypeName>uima.cas.Double</rangeTypeName>
8   </featureDescription>
9
10  <featureDescription>
11    <name>category</name>
12    <description>Berechnete Kategorie der DBpedia Entität.</description>
13    <rangeTypeName>uima.cas.String</rangeTypeName>
14  </featureDescription>
15
16  <featureDescription>
17    <name>path</name>
18    <description>Pfad der DBpedia-URIs von der Ursprungsentität bis zur
      gefundenen Kategorie.</description>
19    <rangeTypeName>uima.cas.StringArray</rangeTypeName>
20  </featureDescription>
21  ...
22 </typeSystemDescription>
```

Listing 5.1: Das eigens für *Enguided* definierte Type-System. Die dargestellten Features dienen zur Speicherung der Klassifikationsergebnisse.

5.7 Enguided – ClassificationEngine

Die `ClassificationEngine`-Klasse bestimmt in Zusammenarbeit mit der `CategoryGraph`-Klasse die Kategorien jeder einzelnen DBpedia-Entität, welche von der letzten Komponente annotiert, gefiltert und gewichtet wurden.

5.7.1 Extraktion des Kategoriegraphen

In Abschnitt 4.3.1 des vorigen Kapitels wurde bereits beschrieben wie der Aufbau des Kategoriegraphen funktioniert. Anhand des URI-Attributs der jeweiligen Entität kann der Graph aufgespannt werden. Dazu wurde der relevante Teil der Ontologie, die Relationen zwischen den Entitäten und den Kategorien (article_categories_en.nt.bz2) und die hierarchischen Relationen zwischen den Kategorien selbst (skos_categories_en.nt.bz2) heruntergeladen[2]. Es handelt sich dabei um N-Triple-Dateien, welche als solche Triple, in der S-P-O Form, mit *Sesame*[3] gespeichert wurden. Diese Datenbank läuft wie das System selbst auf einem Tomcat-Server und bietet einen SPARQL-Endpoint.

Anmerkung Mittlerweile ist bereits ein neuer Ontologie-Dump (Vers. 3.8) erschienen. Zur Entwicklung und Evaluierung wurde jedoch Version 3.7 verwendet. Diese Arbeit bezieht sich immer auf diese Version.

Um, wie in Abbildung 4.3, den Einstieg in DBpedias Kategoriegraphen zu schaffen, müssen die Kategorien, die für die Entität als *Subjects* fungieren, in Erfahrung gebracht werden. Dazu wird eine entsprechende Anfrage (Query) an den SPARQL-Endpoint der erstellten Datenbank abgesetzt. Für das Beispiel aus Abbildung 4.3 sieht der Query folgendermaßen aus:

```
SELECT ?x WHERE {
  <http://dbpedia.org/resource/Blog>
  <http://purl.org/dc/terms/subject>
  ?x
}
```

Danach erhält man die Überkategorie einer beliebigen Kategorie, zum Beispiel `Blogs`, mit dem Query:

```
SELECT ?x WHERE {
  <http://dbpedia.org/resource/Category:Blogs>
  <http://www.w3.org/2004/02/skos/core#broader>
  ?x
}
```

[2] http://http://wiki.dbpedia.org/Downloads38
[3] http://www.openrdf.org

Die Verbindungslogik zum Endpoint wird wieder vom `ConnectionManager` übernommen. Die implementierte `DBpediaStore`-Klasse bietet für die oben gezeigten unterschiedlichen Anfrageschemen jeweils eigene Methoden `ArrayList<String> getCategoriesForArticle(String article)` und `ArrayList<String> getParentCategoriesOfCategory(String category, boolean invert)`. Diese übernehmen die Parametrisierung des Requests, indem sie beispielsweise den Query anhand der URIs zusammenstellen und wandeln die Antwort des Endpoints, mit der *GSON*-Library vom .json-Format in ein Javaobjekt um.

Die weitere Vorgehensweise wurde bereits im vorigen Kapitel (Abs. 4.3.2) beschrieben und kann diesem entnommen werden. Es folgen nun zwei weitere technische Details, die bei der Implementierung des Klassifikators eine wichtige Rolle spielten.

5.8 JGraph

Um den Graphen in Java abbilden und effizient durchsuchen zu können wurde von der *JGraph* – „The most popular Java Graph Visualization Library“ – Gebrauch gemacht. Es wurde ein bestimmter Graphtyp angelegt – `DirectedWeighted- Multigraph` – ein gerichteter, gewichteter Graph der Zyklen erlaubt. Ein solcher Graph besteht aus *Nodes*, zum Beispiel `x` und `y`, und aus *Edges* die Nodes verbinden, wie `x : y` und `y : x`. Der Aufbau des Kategoriegraphen erfolgt in diesem Objekt und wird auf diese Art und Weise gespeichert. Weitere Schritte wie die Berechnung des *SimRank* oder die Suche nach der Kategorie nutzen nur mehr dieses Objekt und müssen den SPARQL-Endpoint somit nicht mehr ansprechen.

Man könnte auch direkt mit SPARQL-Queries nach einer geeigneten Kategorie suchen, jedoch müsste dafür der *SimRank*-Score für den kompletten Kategoriegraphen der Ontologie berechnet und gespeichert werden. Da angenommen wird, dass sich die beste Hauptkategorie meistens in der Nähe des Startknoten befindet, wird nur der relevante Teil des Kategoriegraphen anhand der URIs der besten Entitäten, bis zu einer gewissen Tiefe in das Graph-Objekt übertragen.

5.9 *Enguided* Webservice

Apache UIMA bietet die Möglichkeit die Applikation als *Java-Servlet* mit Tomcat über ein REST-Interface zugänglich zu machen. Dazu muss das UIMA-Projekt zuerst in ein PEAR-File umgewandelt werden. Das UIMA Eclipse-Plugin bietet dafür eine Funktion im Kontextmenü des Projekts. Die PEAR-Datei und die nötigen Bibliotheken werden vom erstellten Projekt des Typs *Dynamic Web Project* in dessen `web.xml` referenziert. Eine Schritt-für-Schritt Anleitung ist auf der Apache UIMA Webseite [41] zugänglich.

Kapitel 6

Evaluierung

Dieses Kapitel behandelt die Evaluierung des, für Webseiten entwickelten Klassifikationsverfahrens *Enguided*. Der Testumfang beläuft sich dabei auf rund 200 Webseiten, wobei sich deren korrekte kategorische Zugehörigkeit gleichmäßig auf die verschiedenen Hauptkategorien von DBpedia verteilt. Die vom System zugewiesenen Kategorien werden mit einem sogenannten „Gold Standard" verglichen. Konstruiert wurde dieser mithilfe vom Webseitenverzeichnis „Yahoo! Directory"[1].

Unter anderem wird auf die Bedingungen und Besonderheiten eingegangen, welche bei der Evaluierung und dem Vergleich der beiden Systeme *Enguided* und *AlchemyAPI* berücksichtigt wurden. Die Ergebnisse werden im Anschluss präsentiert und interpretiert. Deren zugrunde liegende Daten sind im Anhang ausführlich dargestellt. Schlussendlich werden die gewonnenen Erkenntnisse zusammengefasst und positive sowie negative Punkte hervorgehoben.

Was und wie wird gemessen? David D. Lewis veröffentlichte 1991 ein prototypisches und 1995 das finalisierte Paper „Evaluating and Optimizing Autonomous Text Classification Systems" [14, 15]. Mithilfe einiger der darin beschriebenen Vorschläge zur Evaluierung von Textklassifikatoren wurde die folgende **direkte** Evaluierung durchgeführt.

Im Prinzip werden die Entscheidungen des Klassifikators mit richtig oder falsch bewertet. Jedoch wurde der Evaluierungsprozess zusätzlich mit unterschiedlichen Schwellenwerten durchgeführt, welcher niedriger gewichtete Kategorien vom Ergebnis ausschließt. Um die besten Resultate zu erzielen, muss lediglich gemessen werden, welcher Schwellenwert die Performanz des Klassifikators maximiert [15].

Der nächste Abschnitt erläutert die verschiedenen Maße, welche eingesetzt wurden, um über die Performanz des Klassifikators zu urteilen.

[1] http://dir.yahoo.com/

6.1 Performanzmaße

Zuerst werden die vom System zugeordneten, in Frage kommenden Kategorien mit jenen verglichen, welche der Webseite im Gold Standard zugewiesen sind. Es handelt sich um eine binäre Aufgabe; entweder ist eine Kategorie C_i Teil der betrachteten Menge oder nicht. Es gibt zwei Mengen, die der korrekten Kategorien der Seite, entnommen dem Gold Standard GS_i und jene, welche das System berechnet S_i.

Über eine Wahrheitsmatrix (siehe Tabelle 6.1) werden die einzelnen Zuweisungsergebnisse zusammengefasst. Es gibt demnach vier mögliche Fälle:

1. $tp : C_i \in GS_i \wedge C_i \in S_i$
2. $fp : C_i \notin GS_i \wedge C_i \in S_i$
3. $fn : C_i \in GS_i \wedge C_i \notin S_i$
4. $tn : C_i \notin GS_i \wedge C_i \in S_i$

Von den in der Wahrheitsmatrix festgehaltenen Daten, können verschiedenste Maße abgeleitet werden, welche es ermöglichen, die Performanz des Systems zu messen. Üblicherweise kommen, im Falle eines flachen Kategorisierungssystems, *Precision* und *Recall*, die klassischen Performanzmaße im Information Retrieval (IR), zum Einsatz [25]. In den folgenden Abschnitten werden diese Maße erläutert, um später die Effizienz des entwickelten Klassifikators messen zu können.

6.1.1 Precision

Die Precision Pr_i gibt den prozentuellen Anteil der korrekten Zuweisungen aller C_i zugewiesenen Dokumente an und berechnet sich folgendermaßen:

$$Pr_i = \frac{|tp_i|}{|tp_i| + |fp_i|}. \tag{12}$$

6.1.2 Recall

Der Recall Re_i misst den prozentuellen Anteil der korrekten Zuweisungen zu C_i in Relation zu allen Dokumenten denen C_i zugewiesen werden sollte:

$$Re_i = \frac{|tp_i|}{|tp_i| + |fn_i|}. \tag{13}$$

Gefundene Kategorie C_i	Gold Standard - Wahr	Gold Standard - Falsch
System - Wahr	tp_i	fp_i
System - Falsch	fn_i	tn_i

Tabelle 6.1: Wahrheitsmatrix der Kategorie C_i.

6.1.3 F-Maß

Ein ideales System hätte einen Recall- und Precision-Wert von 1. Ein solcher Precision-Wert wird jedoch auch erreicht, wenn keinem Dokument die Kategorie C_i vom System zugewiesen wird. Respektive erreicht auch der Recall den Maximalwert, wenn das System jedes Dokument der Klasse C_i zuweist. Von den einzelnen Maßen lässt sich demnach nicht auf die generelle Effizienz eines Systems schließen [24, S. 35]. Van Rijsbergen schlägt in seinem Buch „Information Retrieval“ [21, S. 132 ff.] die Kombination beider Maße im sogenannten E-Maß

$$\begin{aligned} E_i &= 1 - \frac{1}{\alpha(\frac{1}{Pr_i}) + (1-\alpha)\frac{1}{Re_i}} \\ &= 1 - \frac{(\beta^2+1)Pr_iRe_i}{\beta^2 Pr_i + Re_i} \end{aligned} \tag{14}$$

vor, wobei $\alpha = 1/(\beta^2+1)$ und $0 \leq \beta \leq \infty$. Da geringere Werte von E_i auf eine höhere Effektivität hinweisen, kommt stattdessen oft das **F-Maß** zum Einsatz:

$$F_{\beta,i} = \frac{(\beta^2+1)Pr_iRe_i}{\beta^2 Pr_i + Re_i}. \tag{15}$$

β ermöglicht es den Einfluss der beiden Maße Recall und Precision zu steuern. Demnach gilt zum Beispiel $F_{0,i} = Pr_i$ oder $F_{\infty,i} = Re_i$. Im Bereich des Information Retrieval nimmt β häufig die Werte 0.5 (Precision ist doppelt so wichtig wie Recall), 1 (harmonisches Mittel der beiden Maße) und 2 (Recall ist doppelt so wichtig wie Precision) an. Bei der Evaluierung des Systems, welches in dieser Arbeit vorgestellt wird, wurde β mit 1 festgelegt. Da die allgemeine Qualität des Systems evaluiert wird, ist weder Recall noch Precision stärker gewichtet.

6.2 Korrektheitsstandard

Um ein System wie in dieser Arbeit beschrieben überhaupt evaluieren zu können, benötigt es einen als hundert Prozent korrekt angesehenen Testkorpus, dem sogenannten „Gold Standard“. Im Falle eines Klassifikators ist demnach eine Menge an korrekt kategorisierten Testdokumenten von Nöten. In solchen Fällen werden oft menschlich vorgenommene Kategorisierungen als Standard zugezogen.

D. Lewis verweist in seiner Publikation bereits auf die damit einhergehenden Probleme [15, S. 315], dass selbst von menschlichen Experten vorgenommene Kategorisierungen nicht hundertprozentig übereinstimmen. Um diesem Phänomen entgegen zu wirken, kann man die Kategorisierung besonders bedacht durchführen lassen oder die Evaluierung gegen mehrere Expertenindexierungen durchführen. Eine andere Möglichkeit wäre einen gewissen Grad

der Inkonsistenz zu akzeptieren und die Schnittmenge der unterschiedlichen Expertenergebnisse

$$Overlap_i = \frac{|tp_i|}{|tp_i| + |tn_i| + |fn_i|} \tag{16}$$

als maximal erreichbare Effizienz des Systems zu setzen.

Die zuvor genannten Punkte treffen beim Standard, der bei dieser Evaluierung verwendet wird, jedoch nur bedingt zu. Der Grund hierfür und das Aussehen des Standards werden im nächsten Abschnitt behandelt [15].

6.2.1 Gold Standard „Yahoo! Directory“

Der bei der Evaluierung eingesetzte Standard entstammt „Yahoo! Directory“[2], einem Webseitenverzeichnis. Dort ist es den Benutzern möglich den Betreibern das Hinzufügen einer Webseite in einer bestimmten Kategorie vorzuschlagen. Voraussetzung dafür ist, dass der Benutzer dafür die spezifischste Unterkategorie wählt, ansonsten wird der Vorschlag von den Betreibern abgelehnt. Außerdem wird jeder übermittelte Vorschlag geprüft, was der Qualität des Standards zugute kommt. Vergleichbar ist dieser Dienst mit dem „Open Directory Project (ODP)“[3]. Bei Beiden handelt es sich um „gerichtete azyklische Kategoriegraphen“ [25, S. 521], deren Struktur der des DBpedia-Kategoriegraphen entspricht. Diese Art von Gold Standard wurde bereits von B. Markines als Grundlage eines semantischen Gleicheitsmaßes eingesetzt, mit der Begründung, dass diese laut einer Studie [16] sehr akkurat seien [18, S. 648]. Mitunter ist das einer der Gründe, warum diese Art von Standard zum Einsatz kommt. Außerdem erspart man sich zusätzliche Kosten, die bei der Erstellung eines Standards von Experten entstehen würden.

Die Wahl fiel auf „Yahoo! Directory“, weil dessen Hauptkategorien jenen von DBpedia ähnlich sind und somit am wenigsten Aufwand bei der Normierung des Kategorieraums entstand. Die Abbildung 6.1 zeigt die vom System eingesetzten DBpedia-Hauptkategorien S in welche die Dokumente klassiert werden und die zugehörige Kategorie in GS, welcher die Testdokumente entnommen wurden.

Dabei stellte sich heraus, dass ein gewisser Anpassungsaufwand von Nöten ist, um eindeutige Evaluierungsergebnisse zu erhalten. Im nächsten Abschnitt wird genauer auf diesen Vorgang eingegangen.

[2] http://dir.yahoo.com/
[3] http://www.dmoz.org/

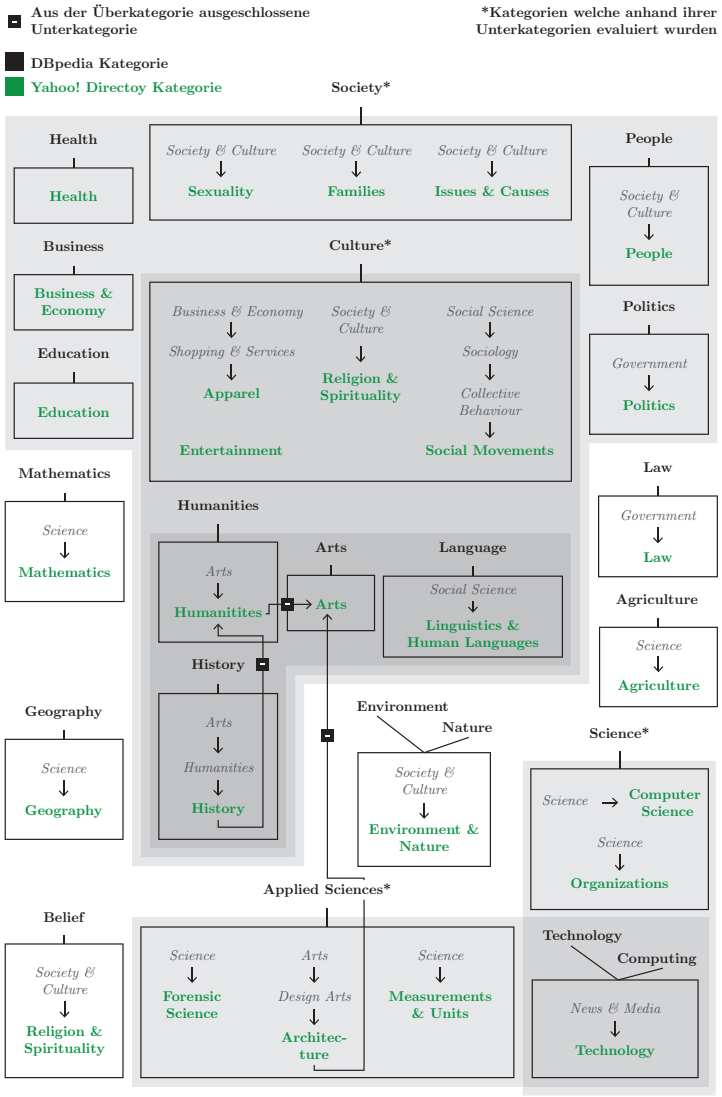

Abbildung 6.1: Die Abbildung veranschaulicht die Zuweisungen der GS-Kategorien, sowie deren jeweilige Lage im GS-Kategoriebaum in Relation zu den Systemkategorien. Die grau schattierten Bereiche in der Abbildung beschreiben die hierarchischen Beziehungen der Hauptklassen von S. Sie implizieren keine Vereinigung der Testkorpora. Diese dürfen und sollen sich nicht überlappen, um eine korrekte Evaluierung zu garantieren.

Normierung des Gold Standards

Bei der Extraktion der Testdaten zeigte sich, dass für S zuerst scheinbar passende Zuweisungen aus GS bei der Evaluierung zu Problemen führen würden. Um dies zu vermeiden mussten gewisse Anpassungen der Zuweisungen durchgeführt werden. Solche waren nötig wenn sich

1. 2 Hauptkategorien aus S im Kategoriegraphen des Gold-Standards in einer Parent-Child-Beziehung befinden oder
2. dort als eine Kategorie definiert sind.

Die erste Problematik erfordert den Ausschluss der Testdokumente des Kindelements von jenen des Elternelements. Trifft man diese Vorkehrung nicht, so kann es bei der Evaluierung der Hauptkategorie, zugewiesen dem Elternelement im GS, zu einem Ergebnis kommen, das als Fehler interpretiert wird, obwohl das System korrekt klassiert hat. In den Abbildungen 6.2 und 6.3 wird die Problematik an einem Beispiel verdeutlicht. Gegeben dem Umstand, dass alle Testdokumente der Kategorie „Humanities“ aufgrund der hierarchischen Beziehung im GS auch zu den Testdokumentkorpus der Kategorie „Arts“ zählen, wird deren korrekte Klassierung bei der Evaluierung der Klasse „Arts“ trotzdem als Fehler gewertet, da das Ergebnis nicht der Vorgabe „Arts“ des Gold-Standards entspricht.

Die zweite Problematik wirkt sich ebenfalls negativ auf die Evaluierungsergebnisse aus. Bezüglich der Symptomatik ist das Problem mit den zuvor genannten Auswirkungen ident. Hier müssen die Testdokumente der Gold Standard-Klasse manuell auf die beiden Hauptkategorien aufgeteilt werden. Wie die Abbildungen 6.4 und 6.5 illustrieren, geschieht dies anhand der Zuweisung von neuen GS-Klassen, somit Testdokumenten, zu einzelnen Unterklassen, der sich in Konflikt befindenden Hauptkategorien.

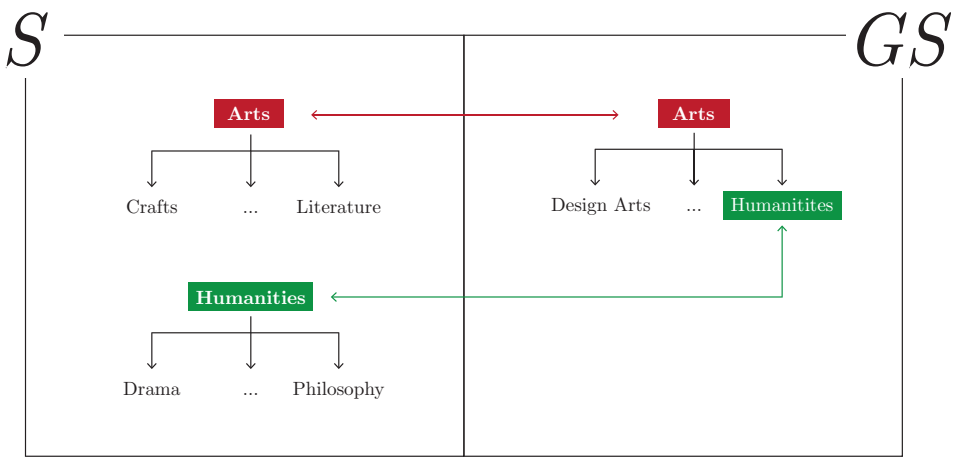

Abbildung 6.2: Die Strukturunterschiede zwischen den beiden Kategorieräumen GS und S würden, trotz korrekter Klassierung des Systems, zu einer Verfälschung des Evaluierungsergebnisses führen.

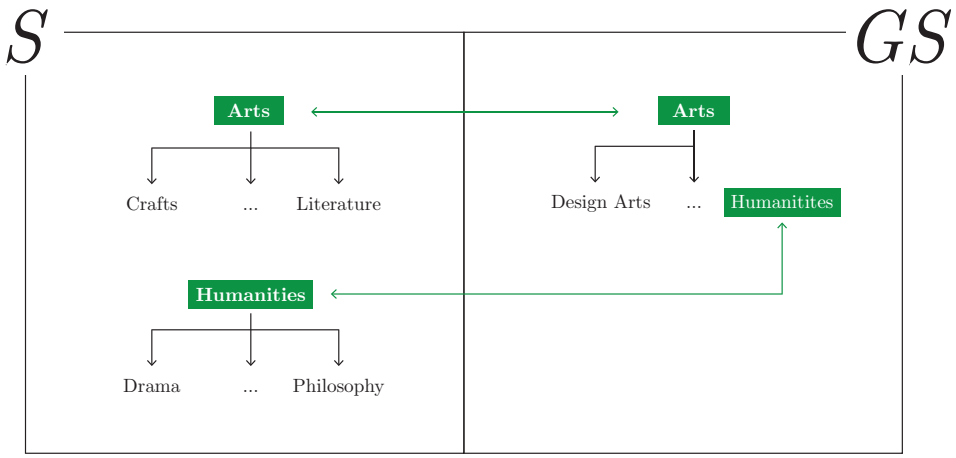

Abbildung 6.3: Durch die Trennung der beiden Elemente und dem Ausschluss der Testdokumente des Kindelements von jenen des Elternelements, lässt sich dieses Problem lösen.

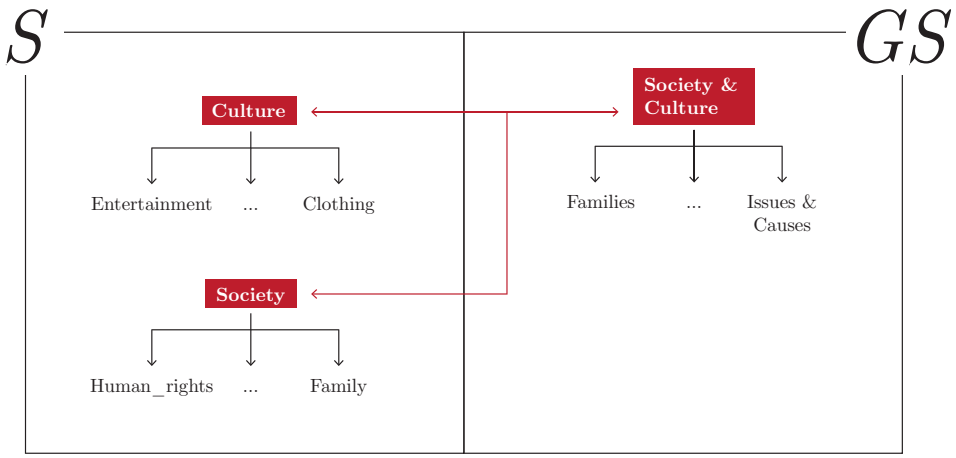

Abbildung 6.4: Die beiden Kategorien sollen nicht auf denselben Testkorpus zurückgreifen, da dies die Evaluierung negativ beeinflussen würde.

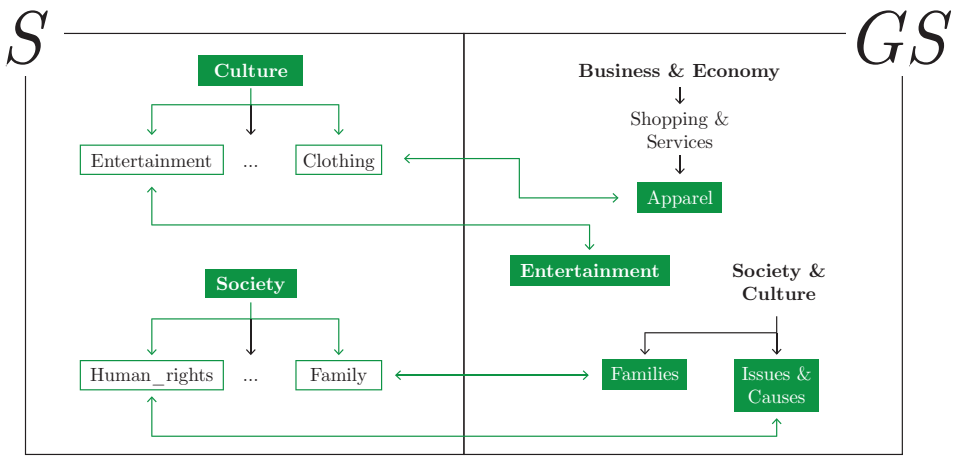

Abbildung 6.5: Den Klassen wurde anhand ihrer Unterkategorien ein neuer Testkorpus erstellt.

Generell wurde durch diese Gegebenheiten deutlich, wie wichtig die klare Abgrenzung der Testkorpora für die zu evaluierenden Klassen ist, um eine gültige Evaluierung durchzuführen. Abbildung 6.1 visualisiert mit einem Minussymbol in Kombination mit einem Pfeil, welche Testdokumente aus welchen Korpora, aufgrund der in der ersten Problemanalyse beschriebenen Umstände, ausgeschlossen werden mussten. Außerdem wird ersichtlich bei welchen, in der Grafik mit einem Stern ausgezeichneten, Hauptklassen von S eine manuelle Trennung nötig war.

6.3 Ablauf der Evaluierung

Für die Auswertung der Resultate musste zuerst der Testdatensatz von insgesamt 197 Webseiten vom System verarbeitet werden. Bis auf drei Ausnahmen mit neun Dokumenten entspricht das zehn Dokumenten pro Klasse. Der Testdatensatz wurde im ersten Schritt manuell in das .json-Format konvertiert:

```
...
{
  'url' : 'http://www.beltbucklestees.com/',
  'categorySet' :
  {
      'found_categories' : [],
      'true_categories' :
      [
          'Culture'
      ]
    }
}
...
```

In dieser Form wurde jedes Testobjekt in einem Set von insgesamt 197 Objekten in einer .json-Datei hinterlegt. Diese beinhalten bereits die zu klassierenden Webseiten und deren Gold-Standard Kategorien. Der Vorteil dabei ist, dass sich die Daten später einfach maschinell weiter verarbeiten lassen. Nach diesem Vorgang wurden die für die Performanzmaße benötigten Werte aus den verarbeiteten Daten abgeleitet und als .csv-Datei gesichert. Die darin enthaltenen Daten wurden schließlich für die Erstellung der in der folgenden Sektion dargestellten Diagramme und Tabellen hinzugezogen.

Der beschriebene Vorgang ist inspiriert von Sun und Lin's Abhandlung zu „Hierarchical Text Classification and Evaluation“ [25], wobei die von ihnen beschriebenen speziellen Metriken für die Evaluierung von hierarchischen Klassifikatoren nicht verwendet wurden. Ein Vergleich mit anderen Techniken anhand dieser Maße wäre nicht möglich gewesen. In naher Zukunft soll

eine solche Evaluierung jedoch nachgeholt werden, um das System noch besser bewerten und verbessern zu können.

6.3.1 Implementierung der Evaluierungsautomatik

Ausgehend von der zuvor erwähnten .json-Datei (siehe 9, `/Evaluierung`), wurden die enthaltenen Daten per GSON-Library[4] auf ein dafür erstelltes POJO abgebildet.
Iterativ wurde in weiterer Folge die URL des jeweilig zu klassierenden Datensatzes an das REST-Interface des entsprechenden Webservice gesendet und die Antwort im XML-Format in Empfang genommen. Diese Antwort wurde weiters mit der Java-XPath-Library[5] geparst und die relevanten Daten, also die vom System zugewiesene(n) Klasse(n), dem Datensatz hinzugefügt.

Die nun erhaltenen klassierten Dokumente dienten der Evaluierung der Klassifikatoren. Hierfür wurde Schritt für Schritt die jeweilige Systemkategorie evaluiert und deren Wahrheitsmatrix festgehalten. Das Sammeln der Evaluierungsdaten läuft im Groben wie im Pseudocode 6.1 ab. Aus den returnierten Werten wurden später die verschiedenen Performanzmaße (siehe Abs. 6.1) errechnet.

6.3.2 Auswertungsvarianten

Um das in dieser Arbeit vorgestellte Verfahren später mit anderen Methoden vergleichen zu können, war es nötig, die Ergebnisse unter Rücksichtnahme auf ein solches Vorhaben auszuwerten. Es mussten also Varianten verwendet werden, die für alle Klassifikationssysteme ihre Gültigkeit haben. Unter Rücksichtnahme dieser Bedingungen wurde zuerst das eigene System evaluiert. Dabei wurden Methoden angewandt, welche spezifische Faktoren, wie den eingesetzten Kategorieraum und dessen hierarchische Eigenschaften, berücksichtigen. Somit konnten später Vergleiche zwischen den unterschiedlichen Varianten durchgeführt werden. Für die Auswertung der Daten wurden zwei verschiedene Varianten eingesetzt:

Best Match Das Ergebnis entspricht der Klasse mit dem höchsten Importance-Wert (Begriffserklärung in Abs. 4.2.4)

Tresholded Matches Diese Methode betrachtet alle Kategorien deren Importance-Wert über einen festgelegten Schwellenwert τ liegt, wobei $0 \leq \tau \leq 1$, als Lösungen. Für zum Beispiel $\tau = 0{,}5$ sind die grünmarkierten Resultate gültige Lösungen:

[4] http://code.google.com/p/google-gson/
[5] http://www.ibm.com/developerworks/library/x-javaxpathapi/index.html

```
...
{
    'url' : 'http://www.exploratorium.edu/sports/',
    'categorySet' :
    {
        'found_categories' :
        [
            'Science', '1.00',
            'Applied_sciences', '0.64',
            'Society', '0.43',
            'People', '0.24'
        ],
        'true_categories' :
        [
            'Applied_sciences'
        ]
    }
}
...
```

Algorithmus 6.1: Evaluierung einer Systemkategorie

1: GetMeasuresOfCategory(*evaluatedCategory*, *testdata*)
Gibt die Summen der 4 möglichen Fälle tp, tn, fp, fn der Wahrheitsmatrix der Kategorie *evaluatedCategory* zurück.
2: $tp, tn, fp, fn \leftarrow 0$
3: **for all** *dataset* of *testdata* **do**
4: *trueCategory* $\leftarrow$ *trueCategory* of *dataset*
5: *assignedCategory* $\leftarrow$ *assignedCategory* of *dataset*
6: **if** *trueCategory*, *assignedCategory* $=$ *evaluatedCategory* **then**
7: $tp \leftarrow tp + 1$
8: **else if** *trueCategory*, *assignedCategory* $\neq$ *evaluatedCategory* **then**
9: $tn \leftarrow tn + 1$
10: **else if** *trueCategory* $\neq$ *evaluatedCategory* and *assignedCategory* $=$ *evaluatedCategory* **then**
11: $fp \leftarrow fp + 1$
12: **else**
13: $fn \leftarrow fn + 1$
14: **end if**
15: **end for**
16: **return** tp, fp, tn, tp.
17: **end**

Beide Verfahren können zusätzlich in zwei verschiedenen Modi ausgeführt werden:

1. **Flat** Die hierarchischen Eigenschaften der Systemklassen (siehe Abb. 6.1) werden beim Vergleich mit dem Gold-Standard nicht berücksichtigt.
2. **Hierarchical** Die hierarchischen Beziehungen werden berücksichtigt. Ist die vom System zugewiesene Klasse ein Kinds- beziehungsweise Elternelement der Gold-Standard-Klasse, wird dies nicht als Fehler gewertet.

6.3.3 Messungen im mehrdimensionalen Kategorieraum

Da es sich beim vorgestellten System um einen Binärklassifikator handelt, der über mehrere Kategorien entscheidet ob diese auf das Dokument zutreffen oder nicht, sollten die gemessenen Werte am Ende vereinigt werden, um Auskunft über die Gesamtleistung des Systems zu erhalten. Insgesamt gab es 20 Kategorien, welche zuerst einzeln evaluiert wurden. Es gibt zwei unterschiedliche Möglichkeiten, um die Gesamtperformanz des Systems zu messen.

Micro-Average und Macro-Average

Beim **Macro-Average**-Verfahren existiert für jede Kategorie eine Wahrheitsmatrix (siehe Abb. 6.1) und die daraus kalkulierbaren Maße wie Precision und Recall. Es wird über die Summe der einzelnen Werteklassen durch die Anzahl der evaluierten Kategorien, der Durchschnitt berechnet [25]:

$$\hat{Pr}^{M} = \frac{\sum_{c}^{i=1} Pr_i}{c}, \tag{17}$$

$$\hat{Re}^{M} = \frac{\sum_{c}^{i=1} Re_i}{c}. \tag{18}$$

Das **Micro-Average**-Verfahren vereinigt hingegen die einzelenen Zellenwerte tp, tn, fn, fp der Kategoriewahrheitsmatrizen in einer einzigen Matrix. Precision und Recall können nun wiederum mit diesen Summen berechnet werden [25]:

$$\hat{Pr}^{\mu} = \frac{\sum_{c}^{i=1} |tp_i|}{\sum_{c}^{i=1} (|tp_i| + |fp_i|)}, \tag{19}$$

$$\hat{Re}^{\mu} = \frac{\sum_{c}^{i=1} |tp_i|}{\sum_{c}^{i=1} (|tp_i| + |fn_i|)}. \tag{20}$$

Macro-Averaging gewichtet die Performanz jeder Kategorie gleich, egal wie oft oder selten diese im Testkorpus vorkommt. **Micro-Averaging** hingegen spricht jedem Testdokument dieselbe Gewichtung zu und begünstigt somit Kategorien, welche sich öfters im Testkorpus wiederfinden [31]. Der Vollständigkeit wegen zeigen die folgenden Evaluierungsergebnisse sowohl die Werte des **Micro-Average**- als auch des **Macro-Average**-Verfahren. Jedoch macht es keinen großen Unterschied welche Variante zum Einsatz kommt, da ein fast homogener Testkorpus zum Einsatz kam. Um auf die Gesamtperformanz des Systems schließen zu können, wurde mit den resultierenden *Precision*- und *Recall*-Werten das F-Maß ermittelt (siehe Abs. 6.1.3).

6.4 Evaluierungsergebnisse

Im folgenden Abschnitt werden die Ergebnisse der Evaluierung präsentiert. Zuerst werden die Ergebnisse des *Enguided*-Textklassifikators in den verschiedenen Modi und Varianten dargestellt und im Anschluss mit den Ergebnissen des *AlchemyAPI*-Klassifikators verglichen.

6.4.1 Enguided

Es folgen die Ergebnisse der Evaluierung des *Enguided*-Textklassifikators anhand der **Best-Match** und **Tresholded Matches** (siehe Abs. 6.3.2) Verfahren.

Best-Match

Wie die Tabelle 6.2 zeigt, grenzen sich die Werte vom **Hierarchical**-Modus deutlich von jenen des **Flat**-Modus ab. Dies lässt sich auf die Qualität im Sinne der kategorischen Abgrenzung der Testdokumente aus dem Gold-Standard zurückführen. Allgemeinere Systemkategorien wie *Society* oder *Culture*, welche unter anderem wieder Systemkategorien als Kinder in der Hierarchie aufweisen, schneiden ohne Berücksichtigung dieser Beziehungen bei der Evaluierung schlechter ab (siehe Beispiel 6.3). Trotz der Vorkehrungen die getroffen wurden (siehe auch Abs. 6.2.1 und Abb. 6.2 und 6.3) kann nicht gewährleistet werden, dass der Klassifikator sich nicht für eine Kindskategorie entscheidet, da dieser Vorgang vom Inhalt des Dokuments abhängt und nicht von der GS-Kategorie. Aus diesen Beobachtungen kann man weiters schließen, dass sich der Klassifikator eher für ein spezifischeres Ergebnis im Falle eines Parent-Child-Konstrukts entscheidet, als sich auf die allgemeine Überkategorie festzulegen.

	Flat		Hierarchical	
Enguided	Micro-Average	Macro-Average	Micro-Average	Macro-Average
F1-Score	0,442	0,468	0,804	**0,805**
Recall	0,396	0,396	**0,787**	0,764
Precision	0,500	0,573	0,822	**0,851**

Tabelle 6.2: Evaluierungsergebnisse mit **Best-Match**-Verfahren.

Tresholded-Matches

Da der *Enguided*-Klassifikator mehrere Lösungen mit unterschiedlichen *Importance*-Werten errechnet, bietet sich eine Evaluierung an, welche diese Werte berücksichtigt und mit einem Schwellenwert τ eine Menge an gültigen Lösungen definiert. Die Ergebnisse sind in Tabelle 6.4 aufgelistet.

Ein geringer Schwellenwert führt erwartungsgemäß zu einem hohen *Recall*-Wert, jedoch wirkt sich ein solcher negativ auf die *Precision* aus. Der umgekehrte Fall tritt bei der Wahl eines hohen Schwellenwerts ein, jedoch kann man bei der *Hierarchical*-Variante einen relativ stabilen *Recall*-Wert beobachten (siehe Abb. 6.6). Dies führt in Folge auch dazu, dass die **F1-Score**-Kurve in Abbildung 6.7, im Gegensatz zur *Flat*-Variante, bei der *Hierarchical*-Variante weiter steigt und nicht abflacht.

Wider der Erwartung, dass der Klassifikator mit einem durchschnittlichen Schwellenwert die beste Performanz erreichen wird, ist nach der Auswertung deutlich, dass mit der **Best-Match**-Variante die besten Ergebnisse erzielt werden. Dies deckt sich auch mit Rechercheergebnissen die besagen, dass die meisten Textklassifikatoren nur eine Lösung präsentieren. Zudem kann dieser Evaluierung entnommen werden, dass bereits der Einsatz eines Schwellenwertes von 1.0, also dem maximal zu erreichenden Wert, zu einer geringeren Performanz des Systems führt. Die wenigen Lösungsfälle, bei welchen ein solcher Schwellenwert von mehr als einer Klasse erreicht wird, vermindern den *Precision*-Wert des Klassifikators und führen somit zu einer schlechteren Gesamtleistung.

Society	Flat	Hierarchical
F1-Score	0,348	0,914
Recall	0,400	1,000
Precision	0,308	0,842

Tabelle 6.3: Gegenüberstellung der Performanz des *Enguided*-Klassifikators in der Kategorie *Society* unter und ohne Berücksichtigung der Parent-Child-Beziehungen.

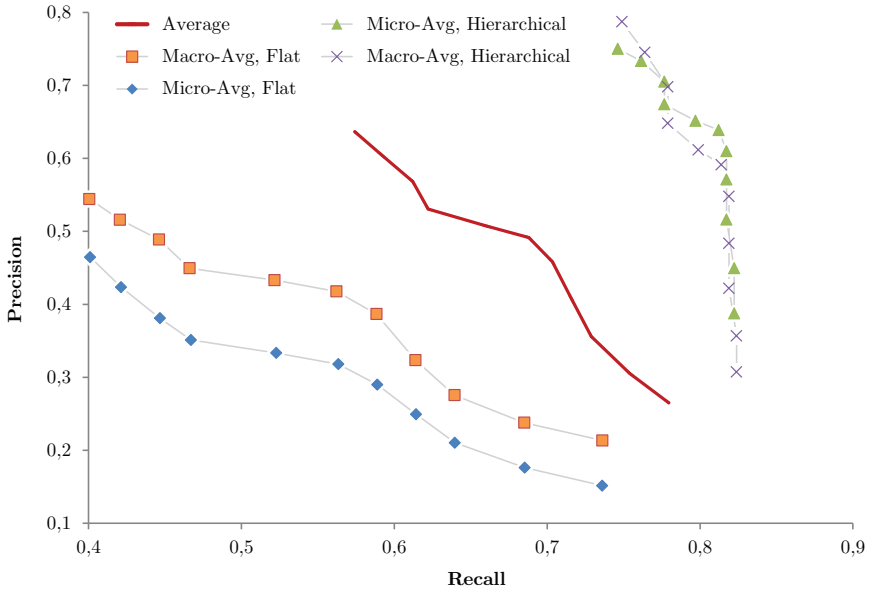

Abbildung 6.6: Precision-Recall-Performanz von T_τ ={0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}.

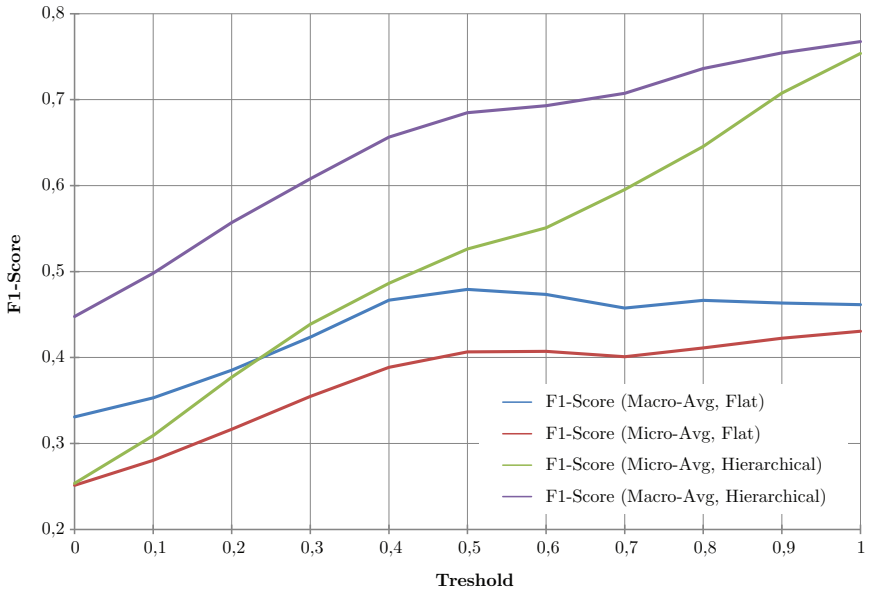

Abbildung 6.7: F1-Scores der verschiedenen Evaluierungsvarianten.

		Flat		Hierarchical	
Treshold	Metriken	Micro-Avg	Macro-Avg	Micro-Avg	Macro-Avg
0	F1-Score	0,251	0,331	0,254	0,448
	Recall	**0,736**	**0,736**	**0,822**	**0,824**
	Precision	0,152	0,214	0,388	0,307
0,1	F1-Score	0,280	0,353	0,309	0,498
	Recall	0,685	0,685	0,822	0,824
	Precision	0,176	0,238	0,449	0,357
0,2	F1-Score	0,317	0,385	0,377	0,557
	Recall	0,640	0,639	0,817	0,819
	Precision	0,210	0,276	0,516	0,422
0,3	F1-Score	0,355	0,424	0,439	0,608
	Recall	0,614	0,614	0,817	0,819
	Precision	0,249	0,323	0,571	0,483
0,4	F1-Score	0,389	0,467	0,486	0,656
	Recall	0,589	0,588	0,817	0,819
	Precision	0,290	0,387	0,610	0,548
0,5	**F1-Score**	0,407	**0,479**	0,526	0,685
	Recall	0,563	0,562	0,812	0,814
	Precision	0,318	0,418	0,639	0,591
0,6	F1-Score	0,407	0,473	0,551	0,693
	Recall	0,523	0,522	0,797	0,799
	Precision	0,333	0,433	0,651	0,612
0,7	F1-Score	0,401	0,458	0,595	0,707
	Recall	0,467	0,466	0,777	0,779
	Precision	0,351	0,449	0,674	0,648
0,8	F1-Score	0,411	0,467	0,646	0,736
	Recall	0,447	0,446	0,777	0,779
	Precision	0,381	0,489	0,705	0,698
0,9	F1-Score	0,422	0,463	0,708	0,754
	Recall	0,421	0,421	0,761	0,764
	Precision	0,423	0,516	0,733	0,745
1	**F1-Score**	**0,431**	0,462	**0,754**	**0,768**
	Recall	0,401	0,401	0,746	0,749
	Precision	**0,465**	**0,544**	**0,750**	**0,787**

Tabelle 6.4: Evaluierungsergebnisse mit dem **Tresholded-Matches**-Verfahren.

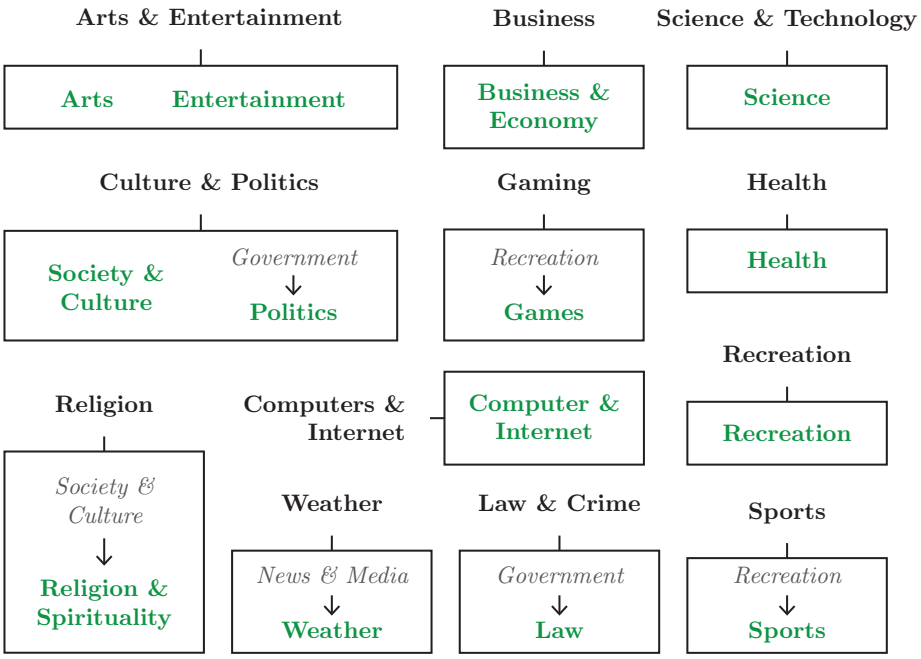

Abbildung 6.8: Zuweisungen der GS-Klassen zu den Systemkategorien des *AlchemyAPI*-Klassifikators.

6.4.2 Vergleich mit *AlchemyAPI*

Auf Basis der vorgestellten Daten, wird nun ein direkter Vergleich zwischen dem Textklassifikator der *AlchemyAPI* und dem *Enguided*-Klassifikator durchgeführt. Dazu wurde ersterer ebenfalls evaluiert. Dessen Kategorieraum umfasst 12 Klassen (siehe Abb. 6.8), wobei jeder Klasse, wie bereits bei der Evaluierung des *Enguided*-Klassifikators, eine GS-Kategorie zugewiesen wurde.

AlchemyAPI	Micro-Average	Macro-Average
F1-Score	0,705	**0,730**
Recall	0,690	**0,692**
Precision	0,721	**0,773**

Tabelle 6.5: Evaluierungsergebnisse des *AlchemyAPI*-Klassifikators.

Die Ergebnisse der Evaluierung sind in Tabelle 6.5 aufgelistet. Da der *Alchemy*-Klassifikator ein einzelnes Ergebnis zurückliefert, können diese Daten direkt mit den **Best-Match**-Ergebnissen des *Enguided*-Klassifikators verglichen werden. Bei beiden Systemen ist in diesem Fall der Auswertungsprozess

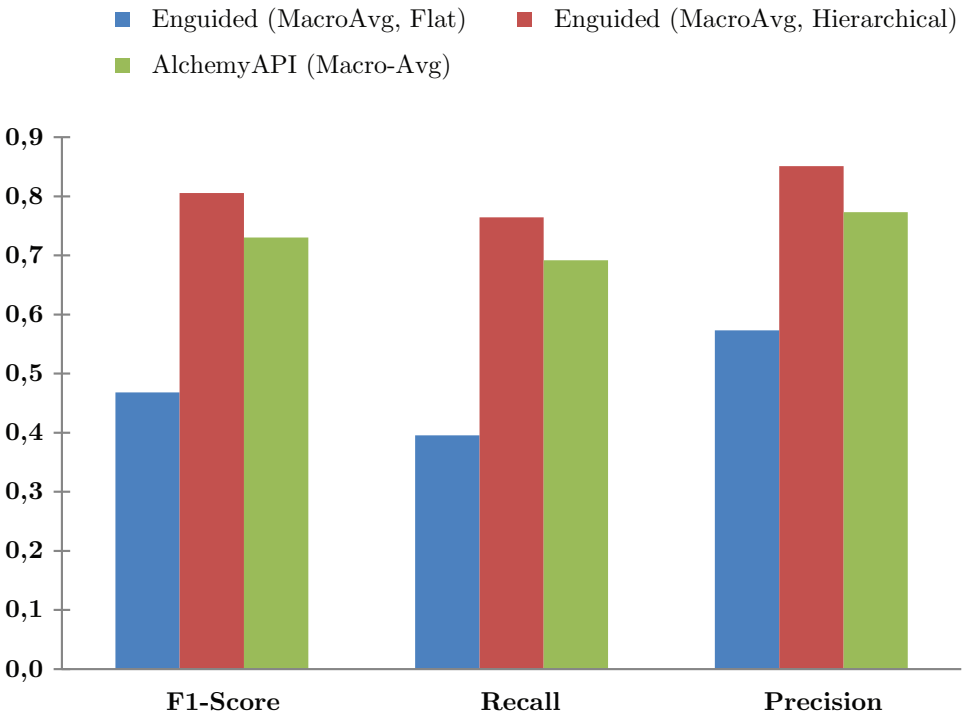

Abbildung 6.9: Vergleich des *AlchemyAPI*- mit dem *Enguided*-Klassifikator.

der Evaluierungsdaten derselbe, weshalb sich ein solcher Vergleich anbietet und bei einer Gegenüberstellung der Performanzmaße Auskunft über Stärken und Schwächen des jeweils anderen Klassifikators geben kann.

Wie der direkte Vergleich in Abbildung 6.9 sofort zeigt, ist der *Enguided*- dem *AlchemyAPI*-Klassifikator im *Flat*-Modus weit unterlegen. *Enguided* konnte hier weniger als fünfzig Prozent der Testdokumente richtig klassifizieren. Unter der Berücksichtigung der Parent-Child Beziehungen der *Enguided*-Systemklassen, also im *Hierarchical*-Modus, verdoppelt sich die Performanz des Systems jedoch nahezu. Die Ergebnisse dieses Evaluierungsmodus übertreffen schließlich auch jene des *AlchemyAPI*-Klassifikators.

Beantwortung der wissenschaftlichen Fragestellung

Mit den in diesem Abschnitt vorangehenden Resultaten und dem direkten Vergleich mit einem anderen System kann die folgende Fragestellung

> Ist die Kategorisierung von Webseiten mithilfe externer Wissensbasen wie DBpedia eine ebenbürtige Alternative zu herkömmlichen Verfahren hinsichtlich der Effizienz und der Qualität der Ergebnisse?

teilweise beantwortet werden.

Da der im Rahmen dieser Arbeit entwickelte Klassifikator nur mit einem alternativen System verglichen wurde, wäre die Behauptung, dass er eine ebenbürtige Alternative darstellt, nicht belegbar. Im Falle des Alchemy-Klassifikators ist dies, auf Basis der in diesem Kapitel dargestellten Ergebnisse, unter den dargelegten Voraussetzungen, der Fall.

6.5 Diskussion

Die in diesem Kapitel beschriebene Evaluierung hat angedeutet, dass der *Enguided*-Klassifikator durchaus konkurrenzfähig ist. Was dieser Test allerdings nicht gezeigt hat ist, wie verständlich die eingesetzten Kategorien für den Benutzer sind. Vor allem stellt sich die Frage, ob sehr allgemeine Kategorien wie „Society“ oder „Culture“, so wie in der Ontologie DBpedia.org abgebildet, überhaupt in das kontrollierte Vokabular aufgenommen werden sollten. Ein möglicher Ansatz wäre das Vokabular von insgesamt 22 Klassen durch zusammenfassen von ähnlichen Klassen zu verkleinern beziehungsweise die hierarchischen Konstrukte zu eliminieren. Wirft man einen Blick auf andere Textklassifikatoren, scheint das *Enguided*-Vokabular überdimensioniert.

Der Zeitfaktor wurde in dieser Evaluierung nicht berücksichtigt, da ein Vergleich bei diesem Stand der Entwicklung nicht sinnvoll gewesen wäre. Im Moment nützt *Enguided* das externe Webservice DBpedia Spotlight. Sobald dieses lokal installiert ist, kann mit einer Verbesserung der Klassifikationsgeschwindigkeit gerechnet werden. Ebenfalls spielt das „gierige“ Öffnen des Suchraums im Kategoriegraphen eine Rolle und bietet Verbesserungspotenzial hinsichtlich des Zeitfaktors, indem eine Heuristik bereits beim Erstellen des Graphen eingesetzt wird. Sind diese zwei Punkte in Zukunft optimiert, ist mit einem Geschwindigkeitsgewinn zu rechnen.

Im Großen und Ganzen funktioniert die Klassierung von Dokumenten per gefundener Ontologie-Entitäten gut und stellt eine pflegeleichte Alternative zu anderen Verfahren dar.

Kapitel 7

Resümee

Die im Zuge dieser Arbeit erlangten Ergebnisse werden an dieser Stelle nochmals zusammengefasst. Basierend auf diesen wird im Anschluss ein Fazit gezogen. Zusätzlich werden diverse Ideen vorgestellt, um das System in Zukunft zu verbessern und als Klassifikationskomponente einer konkreten Anwendung einzusetzen.

Die Klassifikation von Dokumenten im World Wide Web spielt nach wie vor eine wichtige Rolle und die Tatsache, dass täglich eine nicht überblickbare Menge an Dokumenten, welche wiederum neue Information in multimedialer Form mit sich bringen, hinzu kommt, verlangt nach neuen Ansätzen in diesem Feld, um diesen Umständen gerecht zu werden. Die Lösung das „Wissen“ des Klassifikators in Webenzyklopädien wie Wikipedia auszulagern, wird bereits in zahlreichen Forschungsarbeiten suggeriert und erfolgreich durchgeführt. Die Idee des Einsatzes der Taxonomie dieser Wissensbasen differenziert diese Arbeit von anderen und geht noch einen Schritt weiter. Die Suche einer geeigneten Klasse für das zu klassifizierende Dokument in dieser hierarchischen Klassifikation galt als eine der Hauptproblematiken dieser Arbeit. Die dabei gewonnenen Erkenntnisse sollen als Basis für weiterführende Arbeiten dienen.

7.1 Fazit

Der vorgestellte Lösungsansatz ist keineswegs als ideal anzusehen und bietet noch Verbesserungspotenzial. Die durchgeführte Evaluierung zeigt trotzdem, dass *Enguided* gut funktioniert und mit der *AlchemyAPI* mithalten kann.

Da das Verfahren nur vom Inhalt des Dokuments auf eine geeignete Klasse schließen kann, ist es direkt von dessen Form abhängig. Können die Daten nicht extrahiert werden, stehen zu wenige zur Verfügung oder ist der Inhalt der Seite zu vielseitig, wie bei Nachrichtenseiten etwa, liefert das System höchstwahrscheinlich keine treffende Kategorie. In dieser Hinsicht besteht sicherlich noch Verbesserungsbedarf, mit dem Ziel eines robusteren Systems.

Weiters gilt es die Verwendung der Hauptkategorien von Wikipedia zu überdenken. Die Hierarchie innerhalb der Klassifikation erschwerte die Evaluierung. Außerdem sollte die Anzahl der Hauptkategorien reduziert werden, indem diese zum Beispiel zusammengefasst werden.

Die Suche nach einer geeigneten Kategorie gestaltete sich anfangs schwierig, da eine Heuristik fehlte die vorgab wie der Graph zu durchsuchen ist. Nach erneuter Recherche in diese Richtung, konnte eine solche jedoch erstellt werden.

In externen Wissensbasen steckt noch viel mehr Potenzial. Sie werden immer verlässlicher und, trotz der vom Benutzer generierten Inhalte, vertrauenswürdiger. Dies schafft Gewissheit, dass bei jeder Iteration der Ontologie, ebenfalls der Textklassifikator davon profitiert.

Ein großer Nachteil ist, dass Fehler Dritter, wie die Instabilität der DBpedia Spotlight Similarity oder die Inkonsistenzen in der Ontologie, nur schwer korrigierbar sind. Das Verfahren ist jedoch robust genug, um trotzdem gute Ergebnisse zu liefern.

7.2 Future Work

Obwohl das System in jeder evaluierten Kategorie des Typs `Main_topic_classifications` gute Ergebnisse liefert, ist in Frage zu stellen, ob auch alle Klassen dieses Typs zum Einsatz kommen sollten. Theoretisch würde jeder Anwendungsfall andere Bedingungen an die Eigenschaften des Vokabulars stellen. Im Falle der Klassifikation von Webseiten sollte das Vokabular nicht zu groß sein, da sonst die Vorteile einer Kategorisierung verloren gehen. Die Klassen die bei *Enguided* eingesetzt werden, könnten dahingehend reduziert werden, indem generische Klassen ausgeschlossen werden, um ein flaches Vokabular ohne hierarchische Strukturen zu erhalten.

Wie bereits im Related Work Kapitel erwähnt, macht eine Bereinigung der Klassifikationsstruktur Sinn und wirkt sich positiv auf die Qualität der Ergebnisse aus. Die in [20] erwähnten Methoden würden helfen genau das zu erreichen. Da *Enguided* bereits auf eine Blacklist beim Ausschluss von Kategorien setzt, wäre zudem der Aufwand nicht allzu groß.

Ein weiterer Punkt ist die lokale Installation des DBpedia Spotlight Services, woraufhin mit einem enormen Geschwindigkeitsgewinn auf Seiten der Feature-Extraktion zu rechnen ist. Im Zuge dessen würde eine erneute detailliertere Evaluierung Sinn machen, um das System unter den hierarchischen Performanzmetriken, wie sie in [25] genannt werden, fair zu bewerten. Ein Beispiel wäre der Einsatz einer Fehlergewichtung (Error Measure). Dieses Maß könnte man mit der Distanz der falschen zur korrekten Klasse in der Hierarchie gleichsetzen. Zusätzlich wäre der Vergleich mit mehreren anderen Klassifikatoren erstrebenswert, um den Ergebnissen der Evaluierung noch mehr Bedeutung zukommen zu lassen.

Zuletzt ist der Einsatz des Klassifikators im Zuge der Umsetzung eines Google Chrome Plugins zur Kategorisierung der Browser History des Benutzers geplant. Damit lassen sich Interessensschwerpunkte identifizieren und gegebenenfalls Webseiten empfehlen, basierend auf den Daten von Benutzern die einen ähnlichen Browserverlauf besitzen.

Kapitel 8

System Setup

Alle Dokumente auf die in dieser Anleitung verwiesen wird sind in digitaler Form auf einer DVD (siehe Inhaltsverzeichnis in 9) beigelegt.

8.1 Eclipse & UIMA Setup

1. **Eclipse Installation** Siehe [33], Abs. 3.1.1.
2. **Installieren der UIMA Eclipse Plugins** Siehe [33], Abs. 3.1.2.
3. **Installieren der UIMA SDK** Siehe [33], Abs. 3.1.4.
4. **Eclipse Installation** Siehe [33], Abs. 3.1.1.

8.2 Run Configuration & Classpath

1. Abs. 3.2. und Abs. 3.5. in [33] zeigen die Grundschritte, um UIMA Projekte in Eclipse ausführen zu können. *Wichtig!* Sie müssen die in Abschnitt 3.2. beschriebenen Schritte vollständig durchführen, um die notwendigen `UIMA - Run Configurations` korrekt aufzusetzen.
2. Öffnen Sie im Anschluss in Eclipse `Run` → Run Configurations“.
3. Selektieren Sie in der Liste auf der linken Seite `Java Applications` → `UIMA CAS Visual Debugger` und wählen Sie danach den Tab `Classpath` rechts oben aus.
4. Vergewissern Sie sich das als Runtime Environment *JRE oder JDK 7* festgelegt ist. Sollte das nicht der Fall sein, folgen Sie den Anweisungen in Abschnitt 8.2.2.
5. Fügen Sie zuerst das Enguided-Projekt via Klick auf den `Add Project`-Button hinzu.
6. Nun vergleichen Sie Ihre Ansicht mit der Abbildung 8.1 und fügen Sie die verbleibenden Abhängigkeiten via `Add`-Button hinzu. Im nächsten Abschnitt werden diese in einer Liste zusammengefasst.

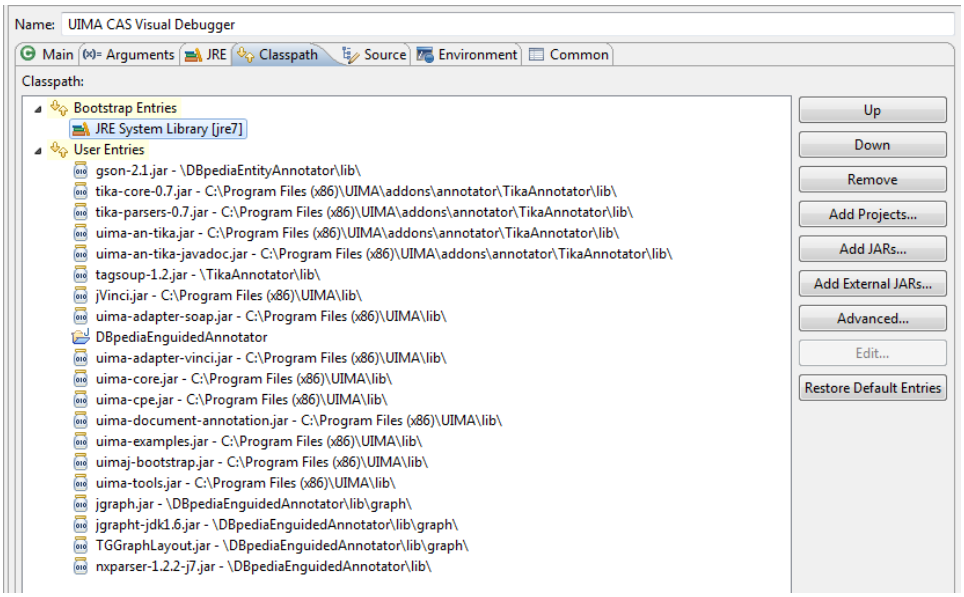

Abbildung 8.1: Das sind alle notwendigen Abhängigkeiten in den „Run Configurations“ des Debuggers

8.2.1 Classpath Quicksetup

- Alle .jar-Dateien aus dem `lib`-Ordner des importierten Enguided--Projekts.
- Das Projekt selbst via `Add Projects...`.
- Alle .jars aus `%UIMA_HOME%/addons/annotator/TikaAnnotator/lib/`.
- Alle .jars aus `%UIMA_HOME%/lib/`.

8.2.2 Update der JDK bzw. JRE

1. Downloaden und installieren Sie sich das JDK 7 oder die JRE 7[1].
2. Wählen Sie nun in Eclipse das Enguided-Projekt mit Rechtsklick aus, um das Kontextmenü zu öffnen.
3. Wählen Sie `Properties` → `Java Build Path` → `Libraries`.
4. Nun wählen Sie den Punkt `JRE System Library` in der Liste aus und klicken Sie den `Edit`-Button auf der rechten Seite.
5. Unter `Alternate JRE` wählen Sie `Add` im Popup und verweisen Sie auf das zuvor erstellte Installationsverzeichnis der JRE bzw. JDK.
6. Bestätigen Sie Ihre Angaben.

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html

8.3 Sesame Triple Store Setup

1. Downloaden und installieren Sie Tomcat 7[2]
2. Downloaden Sie das Sesame SDK[3] und ziehen Sie die Dateien aus dem `WAR`-Verzeichnis des .zip-Files in den `%TOMCAT_HOME%/webapps`-Ordner.
3. Downloaden Sie die DBpedia-NT-Dumps der **Article Categories**[4] und der **Categories (Skos)**[5] und entpackten Sie diese.
4. Starten Sie Tomcat und öffnen Sie http://localhost:8080/openrdf-workbench in einem Browser Ihrer Wahl.
5. Unter `New Repository` legen Sie einen neuen Triplestore an mit den Parametern `Type` = *Native Java Store*, `ID` = *articles_ categories_ nt*.
6. Bestätigen Sie die restlichen Formularschritte und klicken Sie im Anschluss in der Navigation auf `Add`.
7. Nun importieren Sie die zuvor heruntergeladenen .nt-Dateien, indem Sie jeweils das File im Formular mittels `RDF Data File` angeben und als Data Format `N-Triples` wählen.

8.4 Inbetriebname

1. Führen Sie die `UIMA CAS Visual Debugger Run Configuration` aus.
2. Wählen Sie `Run` → `Load AE` und anschließend im Filedialog `Projektverzeichnis` → `desc` → `EnguidedDBpediaEntityClassificationAAE`
3. Nun können Sie rechts im Textfeld normalen Text oder eine gültige Webseiten-URL einfügen und die Analyse mit `Run AE` starten.
4. Nach Abschluss präsentiert das System den analysierten Text, sowie die gefundenen Kategorien (siehe Abb. 8.2)

[2] http://tomcat.apache.org/download-70.cgi
[3] http://www.openrdf.org/download_sesame2.jsp oder DVD unter `Ressourcen`
[4] http://wiki.dbpedia.org/Downloads38#articles-categories oder s.o.
[5] http://downloads.dbpedia.org/3.8/bg/skos_categories_bg.nt.bz2 oder s.o.

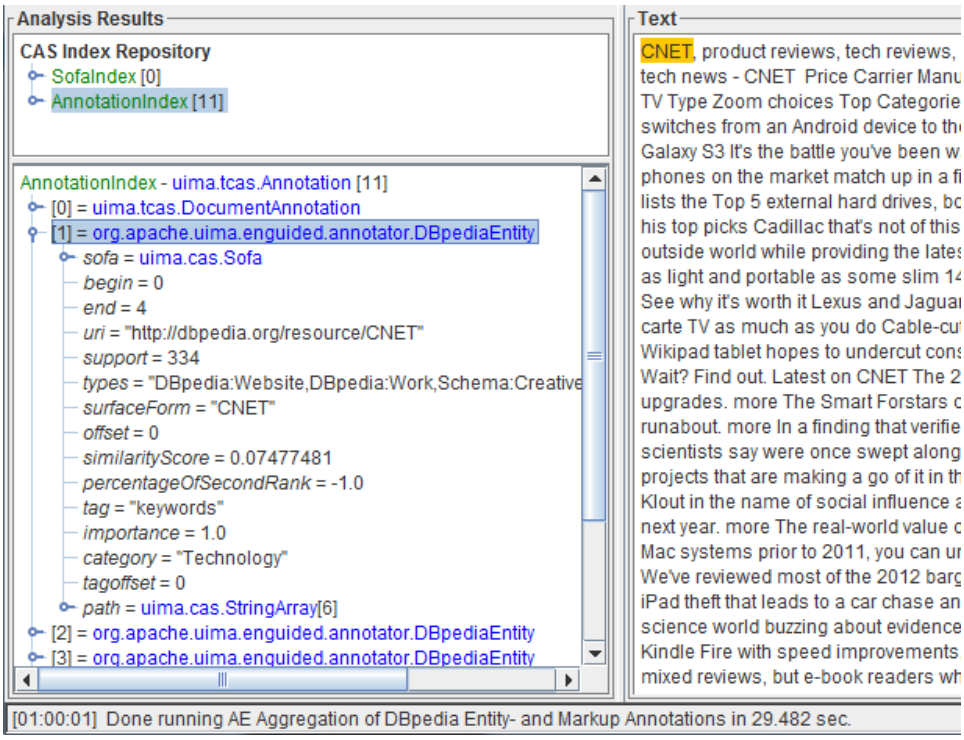

Abbildung 8.2: Der Debugger präsentiert die Resultate in einer ausklappbaren Liste auf der linken Seite.

Kapitel 9

Inhalt der DVD

Format: DVD-ROM, Single Layer, DVD-R-Format

9.1 Masterarbeit (PDF)

Pfad: /

Masterarbeit.pdf	*Enguided* - ein Webseitenklassifikator auf Basis der DBpedia-Ontologie

9.2 Online-Quellen (PDF)

Pfad: /Onlinequellen

Apache_Tika.pdf . . .	Beschreibung und Dokumentation des Tika-Annotators
TagSoup.pdf	Beschreibung und Dokumentation des TagSoup-Parsers
Apache_UIMA_Overview_Setup.pdf	Anleitung zur Installation des UIMA-Frameworks
Apache_UIMA_Pear.pdf	Anleitung für den Pear-Export von UIMA-Projekten
Apache_UIMA_Simple_Server.pdf	Anleitung für das Aufsetzen eines Webservice auf Basis eines UIMA-Projekts
Apache_UIMA_Why.pdf	Kurzbeschreibung des UIMA-Frameworks
Buecker_Referat_Textklassifikation.pdf	Referat zum Thema Textklassifikation
DCMI_Metadata_Terms.pdf	Spezifikation der DCMI Metadata Terms
SKOS_reference.pdf . .	Spezifikation des SKOS Datenmodells
Spotlight_issue39.pdf .	„Tracked Issue“ zur Similarity-Instabilität des DBpedia Spotlight Services

Spotlight_web_service.pdf Beschreibung und Dokumentation des DBpedia Spotlight Services

Lehmann_Klassifikation.pdf Erklärung und Definition des Begriffs Klassifikation

Wikipedia_Klassifikation.pdf Wikipedia-Artikel zum Thema *Klassifikation*

Wikipedia_Wikitext.pdf Wikipedia-Artikel zum Thema *Wikitext*

9.3 Projekt (JAVA)

Pfad: /Enguided

* . . . . . Importierbares Eclipseprojekt *Enguided*

9.4 Ressourcen (RAR)

Pfad: /Ressourcen

article_categories_en.nt.rar DBpedia-Dump der Artikel-Kategorie-Beziehungen

skos_categories_en.nt.rar DBpedia-Dump der Inter-Kategorie-Beziehungen

openrdf-sesame-2.6.3-sdk.tar.rar Installationsdateien der *Sesame*-Datenbank

9.5 Evaluierungsdaten (JSON)

Pfad: /Evaluierung

categorized_data_alchemyapi.json Auswertungsdaten des *AlchemyAPI*-Klassifikators

categorized_data_enguided.json Auswertungsdaten des *Enguided*-Klassifikators

Quellenverzeichnis

Literatur

[1] Hamish Cunningham u. a. *Text Processing with GATE (Version 6).* 2011.

[2] Stephen D'Alessio u. a. „The Effect of Using Hierarchical Classifiers in Text Categorization". In: *Proc. of 6th International Conference Recherche d'Information Assistee par Ordinateur (RIAO 2000).* 2000, S. 302–313.

[3] Susan Dumais und Hao Chen. „Hierarchical classification of Web content". In: *Proc. of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval.* SIGIR '00. Athens, Greece: ACM, 2000, S. 256–263.

[4] Evgeniy Gabrilovich und S Markovitch. „Harnessing the expertise of 70,000 human editors: Knowledge-based feature generation for text categorization". In: *Journal of Machine Learning Research* 8.August 2005 (2007), S. 2297–2345.

[5] Andreas Henrich. *Information Retrieval 1.* 1.2. Otto-Friedrich-Universität Bamberg, 2008.

[6] P. Hitzler u. a. *Semantic Web: Grundlagen.* Springer London, Limited, 2008.

[7] Glen Jeh und Jennifer Widom. *SimRank: A Measure of Structural-Context Similarity.* Technical Report 2001-41. Stanford InfoLab, 2001.

[8] Thorsten Joachims. „Text categorization with Support Vector Machines: Learning with many relevant features". In: *Machine Learning: ECML-98.* Hrsg. von Claire Nédellec und Céline Rouveirol. Bd. 1398. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 1998, S. 137–142.

[9] Andreas Kaster. „Automatische Dokumentklassifikation mittels linguistischer und stilistischer Features". Diplomarbeit. Universität des Saarlandes, 2005.

[10] Daphne Koller und Mehran Sahami. *Hierarchically classifying documents using very few words*. Technical Report. Stanford InfoLab, 1997.

[11] Yannis Labrou und Tim Finin. „Yahoo! as an Ontology - Using Yahoo! Categories to describe Documents“. In: *Proc. of the eighth international conference on Information and knowledge management*. CIKM '99. Kansas City, Missouri, United States: ACM, 1999, S. 180–187.

[12] Douglas B. Lenat und Edward A. Feigenbaum. „On the thresholds of knowledge“. In: *Artificial Intelligence* 47 (1991), S. 185 –250.

[13] Dirk Lewandowski. *Web Information Retrieval: Technologien zur Informationssuche im Internet*. DGI, 2005, S. 248.

[14] David D. Lewis. „Evaluating text categorization“. In: *Proc. of the workshop on Speech and Natural Language - HLT '91* (1991), S. 312–318.

[15] David D. Lewis und Murray Hill. „Evaluating and optimizing autonomous text classification systems“. In: *Proc. of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*. Seattle, Washington, United States: ACM, 1995, S. 246–254.

[16] Ana G. Maguitman u. a. „Algorithmic computation and approximation of semantic similarity“. In: *World Wide Web* 9.4 (2006), S. 431–456.

[17] Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] Benjamin Markines u. a. „Evaluating similarity measures for emergent semantics of social tagging“. In: *Proc. of the 18th international conference on World wide web - WWW '09* (2009), S. 641.

[19] Pablo N. Mendes u. a. „DBpedia spotlight: shedding light on the web of documents“. In: *Proc. of the 7th International Conference on Semantic Systems*. I-Semantics '11. Graz, Austria, 2011, S. 1–8.

[20] Simone Paolo Ponzetto und Michael Strube. „Deriving a large scale taxonomy from Wikipedia“. In: *Proc. of the 22nd Conference on the Advancement of Artificial Intelligence (AAAI)*. Vancouver, B.C., Canada: AAAI Press, 2007, S. 1440–1445.

[21] C.J. van Rijsbergen. *Information Retrieval*. 2. Aufl. Butterworths London, 1979.

[22] Gerard Salton und Michael J. McGill. *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[23] Minoru Sasaki und Kenji Kita. „Rule-Based Text Categorization Using Hierarchical Categories“. In: *Proc. of the IEEE Int.* 1998, S. 2827–2830.

[24] Fabrizio Sebastiani. „Machine learning in automated text categorization“. In: *ACM Computing Surveys* 34.1 (2002), S. 1–47.

[25] Aixin Sun und Ee-peng Lim. „Hierarchical text classification and evaluation“. In: *Proc. 2001 IEEE International Conference on Data Mining* 528 (2001), S. 521–528.

[26] P.N. Tan, M. Steinbach und V. Kumar. *Introduction to Data Mining*. Pearson Addison Wesley, 2006.

[27] Christopher Thomas u. a. „Growing Fields of Interest - Using an Expand and Reduce Strategy for Domain Model Extraction“. In: *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* 2.1 (2008), S. 496–502.

[28] V. Vapnik und S. Kotz. *Estimation of Dependences Based on Empirical Data*. Springer, 2006.

[29] Ke Wang und Ke Wang Senqiang. „Hierarchical Classification of Real Life Documents“. In: *Proc. of the 1st SIAM International Conference on Data Mining*. 2001.

[30] Pu Wang u. a. „Using Wikipedia knowledge to improve text classification“. In: *Knowledge and Information Systems* 19.3 (2008), S. 265–281.

[31] Yiming Yang. „An Evaluation of Statistical Approaches to Text Categorization“. In: *Information Retrieval* 1 (1999), S. 69–90.

Online-Quellen

[32] *Apache TIKA*. URL: http://tika.apache.org/1.2/formats.html#HyperText_Markup_Language (besucht am 03.08.2012).

[33] *Apache UIMA - Overview and Setup*. URL: http://uima.apache.org/d/uimaj-2.3.1/overview_and_setup.html (besucht am 31.07.2012).

[34] *Apache UIMA - PEAR-Packaging*. URL: http://uima.apache.org/downloads/releaseDocs/2.3.0-incubating/docs/html/tools/tools.html#ugr.tools.pear.packager.using_eclipse_plugin (besucht am 01.08.2012).

[35] *DBpedia Spotlight Similarity Issue*. URL: https://github.com/dbpedia-spotlight/dbpedia-spotlight/issues/39 (besucht am 05.08.2012).

[36] *DBpedia Spotlight Webservice*. URL: https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki/Web-service (besucht am 05.08.2012).

[37] *DCMI Metadata Terms*. URL: http://dublincore.org/documents/2012/06/14/dcmi-terms (besucht am 28.07.2012).

[38] *Einführung in die Computerlinguistik - Textklassifikation*. URL: http://user.phil-fak.uni-duesseldorf.de/~petersen/Einf_CL/Referat-Textklassifikation.pdf (besucht am 08.09.2012).

[39] *Google GSON*. URL: http://code.google.com/p/google-gson/ (besucht am 05.08.2012).

[40] *Klassifikation*. URL: http://www.christianlehmann.eu/ling/epistemology/concepts/klassifikation.php (besucht am 01.09.2012).

[41] *Simple Server Installationsanleitung*. URL: http://uima.apache.org/d/uima-addons-current/SimpleServer/simpleServerUserGuide.html (besucht am 28.07.2012).

[42] *SKOS Simple Knowledge Organization System Reference*. URL: http://www.w3.org/TR/skos-reference/#L2413 (besucht am 28.07.2012).

[43] *TagSoup Parser*. URL: http://home.ccil.org/~cowan/XML/tagsoup/ (besucht am 03.08.2012).

[44] *Why Apache UIMA*. URL: http://uima.apache.org/doc-uima-why.html (besucht am 25.07.2012).

[45] *Wikipedia Artikel: Klassifikation*. URL: http://de.wikipedia.org/wiki/Klassifikation (besucht am 01.09.2012).

[46] *Wikipedia Artikel: Wikitext / Wiki-Markup*. URL: http://de.wikipedia.org/wiki/Wikitext (besucht am 05.08.2012).