HOCHSCHULE **TRIER**
Trier University of Applied Sciences
Informatik - Computer Science

Konzeption und Implementierung eines Systems zur automatischen Analyse und Klassifikation von Berichten in Nachrichtenportalen

Conception and implementation of a system for the automatic analysis and classification of articles in newsportals

Tobias Arens

Bachelor-Abschlussarbeit

Betreuer: Karl Hans Blaesius

Trier, 13.12.2016

Kurzfassung

In dieser Arbeit wurde eine erweiterte Variante des Klassifikationsalgorithmus Naive-Bayes implementiert. Aus in Textdateien vorliegenden Konfigurationsdaten wird dynamsich ein Klassifikationskorpus erzeugt, der es ermöglicht aktuelle Nachrichtenartikel verschiedener Onlineportale zu klassifizieren. Mithilfe einer simplen graphischen Benutzeroberfläche ist es möglich diese Artikel zu durchsuchen und einen Einblick in die zugrunde liegenden Daten zu gewinnen. Benötigte Grundlagen der Textklassifikation sowie das Vorgehen bei Konzeption und Realisierung der Arbeit werden ausführlich beschrieben.

This paper shows an implementation of an advanced variant of the Naive-Bayes classification algorithm. Through available configuration files a classification corpus is dynamically generated by which it is possible to classify current news from online news portals. A simple graphical-user-interface allows it to filter the results of this search and makes the underlying data easily accessible. Required fundamentals of text-classification and the approach of design and implementation of this work are described in detail.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1

Einleitung und Problemstellung

In der heutigen Zeit gibt es viele Möglichkeiten an Informationen über Themen aller Art zu gelangen. So besteht neben einer großen Vielfalt an Zeitungen, Zeitschriften und Magazinen auch die Möglichkeit, sich über Online-Quellen, wie Informationsportale oder Webpräsenzen von Zeitungen zu informieren.

Diese Diversität hat die Folge, dass eine sehr große Bandbreite an Themen aus verschiednen Blickwinkeln beleuchtet und somit eine relativ unabhängige Informationsbeschaffung ermöglicht wird.
Doch diese Vielfalt an Informationsquellen bringt auch die Schwierigkeit mit sich, sich gezielt über ein bestimmtes Thema zu informaieren, ohne die Diversität verschiedener Meinungen und Blickpunkte aufgeben zu müssen. Es ist umständlich bei Interesse an einem bestimmten Thema passende Artikel zu finden da dafür eine Möglichkeit einer großflächigen Abdeckung verschiedener Portale fehlt.

Um Artikel thematisch zuordnen zu können, ist eine automatisierte Klassifizierung sinnvoll, die diese Artikel zielsicher mithilfe vorab definierter Klassen abbilden kann. Dies ermöglicht eine zielgerichtete, breitgefächerte Suche nach themenbezogenen Informationen.

In Rahmen dieser Arbeit wird mithilfe der funktionalen Programmiersprache Lisp eine Anwendung entwickelt, die dieses Problem lösen soll. Dafür sind allerdings mehrere Schritte notwendig: es muss eine Schnittstelle entwickelt werden, die Nachrichtenartikel verschiedener Quellen auslesen kann, um diese für die weitere Verarbeitung nutzen zu können. Die so gewonnenen Daten müssen mit entsprechenden Metriken bewertet und mithilfe eines Klassifizierungsalgorithmus korrekt zugeordnet werden. Um dies zu ermöglichen, soll ein möglichst simpler Algorithmus gefunden werden, der eine möglichst hohe Genauigkeit der Zuordnung erreicht. Damit die Anwendung bedienbar ist, wird eine graphische Oberfläche entwickelt, die eine Kategorisierung aktueller Nachrichten erlaubt und einen Überblick über die zugrunde liegenden Klassifikationsdaten ermöglicht.

Da zur Klassifikation Grunddaten benötigt werden, ist es Teil dieser Arbeit eine solche Grundlage zu schaffen, um den verwendeten Algorithmus mit diesen Daten zu trainieren. Um diese Aufgabe realistisch bewältigen zu können, wird nur eine

kleine Auswahl an Themengebieten berücksichtigt, mit denen die Anwendung arbeitet.

Die Arbeit beinhaltet Techniken aus dem Bereich des Text-Mining, welches sich mit Möglichkeiten der Gewinnung von neuen Informationen aus schwachstrukturierten Texten befasst und hat einige Überschneidungen mit dem Gebiet Information Retrieval, das sich damit befasst bestehende Informationen aufzufinden.

2

Konzeption

Damit Nachrichtenartikel korrekt klassifiziert werden können, sind mehrere Schritte notwendig, welche in diesem Kapitel behandelt und detailliert beschrieben werden.

Die erste Schwierigkeit ist das Einlesen von Artikeln per HTML-Link und die Bereitstellung dieser Daten für die weiteren Programmteile. Wenn dieser Schritt erledigt ist, muss der eingelesene Text analysiert und aufgrund dieser Analyse in eine Kategorie eingeordnet werden.

2.1 Einlesen der Nachrichtenartikel

Das Einlesen von Online-Nachrichtenartikeln besteht aus drei Schritten: Zu Beginn wird mithilfe eines sogenannten Webfetchers der HTML-Inhalt des angegebenen Links ausgelesen. Im zweiten Schritt wird dieser als String vorliegende Inhalt mithilfe eines HTML-Parsers ausgelesen. Die dadurch gewonnenen Daten liegen in einer Lisp-typischen Listen-Darstellung vor und können somit mit allen vorhandenen Mitteln traversiert werden. Um nun den gewünschten Text aus der so verarbeiteten Seite auszulesen muss die individuelle Struktur der verschiedenen verwendeten Nachrichtenportale berücksichtigt werden. Dies stellt gleichzeitig eine Einschränkung der zu entwickelnden Anwendung dar, da für jedes gewünschte Nachrichtenportal die verwendete HTML-Struktur analysiert und entsprechend ausgelesen werden muss. Um dies zu ermöglichen, wird ein einfacher Algorithmus entwickelt, der eine hinterlegte Strukturliste sowie den geparsten HTML-Inhalt der Webseite als Eingaben erhält und mithilfe dieser die gewünschten Textteile in einem einheitlichen (Listen-) Format zurückgibt. Damit wird die beschriebene Einschränkung durch eine Konfigurierbarkeit der Anwendung verlagert.

2.2 Klassifikation der Artikel

Die Klassifikation der Artikel kann als Dokumentenklassifikation realisiert werden, welche ein bekanntes Problem aus dem Bereich des Data- beziehungsweise Text-Mining, darstellt. Um dieses Problem zu lösen ist zunächst eine formale Definition des Problems nötig, um Lösungsalgorithmen verstehen und anwenden zu können.

Desweiteren ist es notwendig einen Algorithmus zu finden, der möglichst einfach funktioniert, aber dennoch eine hohe Klassifikationsgenauigkeit ermöglicht.

2.2.1 Formale Definition

In dem Werk *Introduction to Information Retrieval* der Autoren Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze, wird das Textklassifikationsproblem folgendermaßen beschrieben:
Bei der Dokumentenklassifikation soll einem Dokument $d \in X$, wobei X den sogenannten Dokumentenraum beschreibt, einer Klasse c_1 aus einer festen Menge an sogenannten Klassen $C = \{c_1, c_2, ..., c_n\}$, die typischer von Menschen definiert werden, zugerordnet werden.

Diese vordefinierten Klassen werden aus einer Trainings-Menge D gewonnen, die aus Tupeln $\langle d, c\rangle \in X \times C$ besteht. Mithilfe eines lernenden Algorithmus ist eine Klassifizierungsfunktion

$$\gamma : X \to C \tag{2.1}$$

gesucht, die jedem Dokument eine eindeutige Klasse zuordnet.[1]

Diese Methode des Lernens wird auch überwachtes Lernen genannt, da das zu erlerndende Ergebnis mithilfe der vordefinierten Trainings-Menge abgeglichen wird.

2.2.2 Bag-of-Words-Modell

Im vorherigen Unterkapitel wurde beschrieben wie die Nachrichtentexte eingelesen werden. Für die Klassifikation sind diverse Vorberechnungen sinnvoll, zum Beispiel das Zählen der Worthäufigkeit. Die Ergebnisse dieser Berechnungen können dann mit dem ausgewählten Klassifikator ausgewertet und gewichtet werden.

Zum Ermitteln der Worthäufigkeiten wird das sogenannte *Bag-of-Words-Modell* angewendet. Dieses betrachtet Texte als Mengen von Wörtern, die keinerlei Struktur oder Grammatik aufweisen. Dies ist ein gängiges Verfahren in der Textklassifikation und obwohl die grammatikalischen Informationen verloren gehen, erzeugt dieses Modell eine Vereinfachung der Texte, die die weitere Verarbeitung erheblich beeinflusst, indem nur noch eine bestimmte Menge an Wörtern gespeichert werden muss. Strukturrelle und grammatikalische Informationen werden nicht gespeichert und somit wird auch weniger Speicherplatz benötigt. So könnte beispielsweise der Satz „Ich genieße den Tag, während ich auf meinem Sessel sitze.“ als die Liste

```
1 (ich
2 genieße
3 den
4 Tag
5 während
6 auf
7 meinem
8 Sessel
9 sitze)
```

[1] [CDMS08, Vgl. Kapitel 13.1 *The text classification problem*]

dargestellt werden. Hierbei ist zu bemerken, dass alle Satzzeichen entfernt werden und doppelte Wörter nur einmalig vorkommen.

Es mag kontraintuitiv erscheinen, allerdings ist der Verlust dieser weiteren Informationen für die Textklassifikation vernachlässigbar und beeinflusst die Klassifikationsgenauigkeit kaum.

Dies wird in der Diplom-Arbeit von Andreas Kaster *Automatische Dokumentklassifikation mittels linguistischer und stilistischer Features* gezeigt:

> „Bei themenbasierten Klassifikationsproblemen (repräsentiert durch 20- Newsgroups) und der Klassifikation von Meinungen (Movie-Review-Korpus) bewegte sich der durchschnittliche Klassifikationsfehler mit alternativen Features, wie z.B. der Satzbauanalyse oder statistischen Satzlängenverteilungen, bei ca. 50%. Die Klassifikationsergebnisse waren also fast wie zufällig. Offensichtlich waren unsere alternativen Features nicht diskriminativ für diese Klassifikationsprobleme. Es besteht also keine oder nur eine geringe Korrelation zwischen stilistischen Merkmalen und den Klassifikationsthemen. Daher konnte auch die Einführung von Kombinationsstrategien die Klassifikationsergebnisse nicht wesentlich verbessern. Das Bag-of-Words-Modell ging bei unseren Experimenten als klarer Sieger hervor.“[2]

2.2.3 Algorithmen

Für die im vorherigen Abschnitt beschriebene Klassifizierungsfrage gibt es eine Reihe von Klassifikationsverfahren, von denen in diesem Kapitel drei vorgestellt werden. Dies stellt nur eine kleine Auswahl der tatsächlich vorhandenen Algorithmen, die zum Klassifizieren genutzt werden dar, allerdings sind dies Algorithmen, die eine relativ weite Verbreitung haben.

Naiver Bayes-Klassifikator

Der Naive Bayes-Klassifikator wird aus dem Satz von Bayes hergeleitet. Dieser beschreibt die Berechnung von bedingten Wahrscheinlichkeiten.

Seien A und B zwei Ereignisse wobei die Wahrscheinlichkeit $P(B) > 0$, so lässt sich die bedingte Wahrscheinlichkeit, dass A eintritt folgendermaßen berechnen:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.2}$$

Dieser Satz wird bei endlich vielen Ereignissen A_i, ..., A_n folgendermaßen angepasst:

$$P(A_i|B) = \frac{P(B|A_i) \cdot P(A_i)}{P(B)} = \frac{P(B|A_i) \cdot P(A_i)}{\sum_{j=1}^{n} P(B|A_j) \cdot P(A_j)} \tag{2.3}$$

[2] [Kas05, S. 65]

Bei der Klassifikation von Texten wird die namensgebende naive Annahme gestellt, das jedes zu einer Klasse gehörige Wort gleichermaßen aussagekräftig bezüglich der Zuordnung ist. Trotz dieser starken Vereinfachung liefert der Bayes-Klassifikator in der Regel gute Ergebnisse.
Für die Klassifikation von Texten wird dieser Satz folgendermaßen angepasst:

$$P(C|w) = \frac{P(w|C) \cdot P(C)}{P(w)} \tag{2.4}$$

Hierbei gibt $P(w|C)$ die Wahrscheinlichkeit an mit der Wörter aus Dokument w zu der Klasse C gehören, $P(C)$ die Wahrscheinlichkeit mit der die Dokumentenklasse C auftritt und $P(w)$ die Wahrscheinlichkeit mit der das Wort w auftritt. Dabei ist zu bedenken, dass die Wahrscheinlichkeit der Klasse $P(C)$ von ihrem Vorkommen in den Trainingsdaten abhängig ist.

Die Wahrscheinlichkeit, dass ein Wort w_i mit einer Klasse C assoziiert ist, berechnet sich folgendermaßen:

$$P(w_i|C) = \frac{\text{Anzahl der Dokumente aus Klasse } C \text{ mit dem Wort } w_I}{\text{Anzahl aller Dokumente ohne } C} \tag{2.5}$$

Um nun einen Text einer Klasse $\hat{y}$ zuzordnen wird folgende Gleichung verwendet:

$$\hat{y} = arg\,\max_k P(C_k) \prod_{i=1}^{n} P(w_i|C_k) \tag{2.6}$$

Dabei wird die Klassenzugehörigkeit durch das Maximum der multiplizierten Wortwahrscheinlichkeiten bezüglich aller Klassen C berechnet. Das bedeutet $\hat{y}$ ist die Klasse unter der das Produkt der einzelnen Wahrscheinlichkeiten aller Wörter w_i eines Dokumentes im Vergleich zu allen anderen Klassen am größten ist.

Support Vector Machine

Eine Support Vector Machine ist ein *unüberwachtes* Klassifikationsverfahren, welches die zu klassifizierenden Dokumente als Vektor in einem Vektorraum darstellt. Die Aufteilung in Klassen wird durch sogenannte Hyperebenen durchgeführt, die einer Trennfläche zwischen Klassen entsprechen. Dabei ist der Abstand der Trennfläche zu Objekten in den Klassen größtmöglich. Dadurch soll eine möglichst genaue Klassifizierung gewährleistet werden.

Ein Vorteil der Support Vector Machine ist es, dass nicht zwangsweise alle Trainingsdaten betrachtet werden müssen. Da diese Dokumente mit Vektoren beschrieben werden, können Vektoren, die näher an der vorhin beschriebenen Trennfläche liegen andere Vektoren „verdecken“. Die durch die Hyperebene aufgespannte Trennfläche ist somit nur von eben jenen Vektoren abhängig, deren Distanz zu ihr im Vergleich zu anderen Vektoren der Klasse am geringsten ist.

Eine solche Klassifizierung unterliegt allerdings der Einschränkung, dass nur linear trennbare Klassen darstellbar sind, Trainingsdaten in der Regel aber nicht linear trennbar sind. Abhilfe schafft der sogenannte *Kernel-Trick*. Dabei werden

die Vektoren in einen höherdimensionalen Raum überführt. Wenn diese Dimension groß genug gewählt wird, können damit die Trainingsdaten linear getrennt werden. Dies ist allerdings eine sehr rechenlastige Operation.

Nächste-Nachbarn-Klassifikation

Die Nächste-Nachbarn-Klassifikation, im Englischen auch *k-nearest neighbors algorithm* genannt, ist ein Clustering-Algorithmus, der in der Mustererkennung, worunter auch die Text-Klassifikation fällt, verwendet wird. Es ist ein lernender Algorithmus, der im Vergleich zu Naive-Bayes allerdings unüberwacht stattfindet. Das heißt, es ist nicht notwending Klassen vorzudefinieren.

Bei diesem Clustering wird versucht alle Dokumente die sich ähneln möglichst in einen Cluster einzuordnen. Diese Cluster müssen sich größtmöglich voneinander unterscheiden.

2.2.4 Auswahl

Da *Naive-Bayes* ein leicht zu implementierender Algorithmus ist, wurde er für diese Arbeit ausgewählt. Desweitern gibt es eine Arbeit die einige Probleme dieses Algorithmus beseitigt und eine recht einfache Erweiterung vorschlägt. Darauf wird im kommenden Kapitel eingegangen. *Support Vector Machines* arbeiten mit mathematisch komplexen Konzepten und Operationen und dennoch sind deren Ergebnisse nicht zwangsläufig besser als die mit *Naive-Bayes* erreichten. Die *Nächste-Nachbarn-Klassifikation* arbeitet mit unüberwachten Lernen, ist somit nicht einfach konfigurierbar und wird aus diesem Grund nicht ausgewählt.

Durch diese Auswahl ist es möglich schon frühzeitig einen Klassifikationsalgorithmus zu implementieren, der im Verlauf der Arbeit verbessert und weiterentwickelt werden kann. Dies hat den Vorteil, dass einzelne Komponenten des Programms schon in einer frühen Projektphase zusammengefügt und getestet werden können. Auch können potenzielle Schwierigkeiten im Vorhinein abgesehen werden und somit kann die Planung beziehungsweise Durchführung entsprechend angepasst werden.

2.3 Erweiterterter Naive-Bayes

In diesem Unterkapitel wird eine auf dem Naive-Bayes-Klassifikator basierende Erweiterung und Verbesserung beschrieben, die in dieser Arbeit umgesetzt wurde. Hierfür werden zu Beginn einige Berechnungen und Gleichungen vorgestellt, die in dieser Erweiterung verwendet werden.

2.3.1 Term Frequency-Inverse Document Frequency

Die sogennante *Term Frequency-Inverse Document Frequency*, kurz *tf-idf*, ist ein weiteres Bewertungsmaß aus dem Gebiet der Information Retrieval. Sie ist eine numerische Statistik, die zeigt wie wichtig ein Wort im Bezug zu einem Dokument oder einer Dokumentensammlung ist. Dies wird als Gewichtungsfaktor in Text-Mining Algorithmen verwendet und kann mit dem Naive Bayes-Klassifikator kombiniert werden.

Die *tf-idf* besteht aus den zwei namensgebenden Teilen *Term Frequency* und *Inverse Document Frequency*.

Term Frequency

Term Frequency ist eine einfache Aufzählung der Vorkommnisse d_{ij} eines Wortes w_i in einem Dokument D. Da es verschiedene Varianten dieser Gewichtung gibt, müssen einige Fälle unterschieden werden:

Art	Berechnung	Beschreibung
binär	1,0	Wertung 1, wenn das Wort im Dokument enthalten ist, sonst 0
einfache Häufigkeit	d_{ij}	Abzählen der Häufigkeit von w_i in d
logarithmisch	$1 + log(d_{ij})$	Der Logarithmus der einfachen Häufigkeit
doppelt normalisiert	$K + (1 - K)\frac{d_{ij}}{max_{\{d_{in} \in di\}} d_{in}}$, mit z.B.: $K = 0.5$	Die einfache Häufigkeit wird durch die größte Häufigkeit in d geteilt. Dies macht sehr lange und sehr kurze Dokumente besser vergleichbar

Tabelle 2.1. Verschiedene Möglichkeiten die *Term Frequency* zu berechnen.

Inverse Document Frequency

Inverse Document Frequency ist ein Maß darüber wie informativ ein Wort ist. Dies wird erreicht, indem ein Verhältnis über die Anzahl der Dokumente und der Anzahl der Dokumente, die das Wort w_i enthalten, berechnet.

Wie auch schon bei der *Term Frequency* gibt es einige Möglichkeiten die *Inverse Document Frequency* zu berechnen, von denen einige in Abbildung 2.2 aufgelistet werden.

Art	Berechnung	Beschreibung
unär	1	Keine inverse Gewichtung
einfach	$log\frac{\sum_k 1}{\sum_{i \in d}}$	Die Anzahl aller Dokumente durch die Anzahl derer in denen das Wort w_i vorkommt.
geglättet	$log(1 + \frac{\sum_k 1}{\sum_{i \in d}})$	Wie einfach, mit Glättungsparameter

Tabelle 2.2. Verschiedene Möglichkeiten die *Inverse Document Frequency* zu berechnen.

$\sum_{i \in d}$ ist hierbei die Summe aller Dokumente d in denen das Wort w_i auftaucht.

Berechnung von tf-idf

Die Berechnung von *tf-idf* ist die Multiplikation der *Term Frequency* und der *Inverse Document Frequency*.
Somit könnte ein *tf-idf*-Wert *delta* zum Beispiel mit folgender Formel berechnet werden:

$$\delta = log(d_{ij} + 1) * log(\frac{\sum_k 1}{\sum_{i \in d}} + 1) \tag{2.7}$$

2.3.2 Längen-Normalisierung

Wie schon im Kapitel 2.3.1 beschrieben, kann die Gewichtung einzelner Wörter bezüglich ihrer Klassenzugehörigkeit normalisiert werden. Das ermöglicht eine höhere Vergleichbarkeit von Texten unterschiedlicher Länge. Ermöglicht wird dies, indem Wörter eines längeren Textes verhältnismäßig geringer gewichtet werden als Wörter aus kürzeren Texten.

Sei f_i die Häufigkeit eines Wortes in einem Dokument, dann lässt sich die normalisierte Häufigkeit folgendermaßen berechnen:

$$f_i^{'} = \frac{f_i}{\sqrt{\sum_k (f_k)^2}} \tag{2.8}$$

Hierbei ist $sum_k(f_k)^2$ die Summe aller Worthäufigkeiten aller Dokumentenklassen K.

Auch wenn die Textlängenunterschiede bei verschiedenen Online-Nachrichtenartikeln eher gering ausfallen, wird die oben beschriebene Normalisierungsberechnung im verwendeten Algorithmus implementiert. Somit ist sichergestellt, dass die Textlänge für die Klassifikationsgenauigkeit keine Veränderung bewirkt.

2.3.3 Klassen-Komplement

In Kapitel 3.1 des Artikels *Tackling the Poor Assumptions of Naive Bayes Text Classifiers*[3] der Autoren Jason D. M. Rennie, Lawrence Shih, Jaime Teevan und David R. Karger aus dem Jahre 2003 wird ein weiteres Problem der Klassifizierungsgenauigkeit beschrieben. Je nach verwendeten Trainingsdaten kann es vorkommen, dass der Bayes-Klassifikator bestimmte Klassen anderen unbeabsichtigt vorzieht und die Klassifikationsgenauigkeit darunter leidet. Um dieses Problem zu beseitigen, schlagen die Autoren vor die Gewichtung einer Klasse nicht nur über die ihr zugehörigen Dokumente zu definieren, sondern über das sogenannte Komplement der Klasse.

Dies berechnet anstatt wie im unmodifizierten Algorithmus, nicht die zugehörigkeit eines Dokumentes d zu einer Klasse c, sondern die Nichtzugehörigkeit von d bezüglich aller anderen Klassen. Dies wird mit folgender Gleichung beschrieben:

$$\hat{\theta}_{\bar{c}i} = \frac{N_{\bar{c}i} + \alpha_i}{N_{\bar{c}} + \alpha} \tag{2.9}$$

Hierbei ist $N_{\bar{c}i}$ die Anzahl der Vorkommnisse des Wortes i in allen Klassen außer c und $N_{\bar{c}}$ die Gesamtzahl aller Wörter in den Klassen ohne c. α_i und α sind Glättungsparameter.

[3] [JDMRK03]

2.3.4 Gewichtungs-Normalisierung

Ähnlich wie im vorherigen Kapitel beschrieben, gibt es bei der Gewichtung das Problem einzelne Klassen anderen vorzuziehen, beispielsweise wenn die Trainingsdaten nicht ausgewogen genug sind.

Bei der Gewichtung kann es vorkommen, dass die Gewichte einer Klasse größer sind als die anderer Klassen. Dies tritt vorallem im Vergleich von kleineren Klassen auf, da die Gewichte einer Klasse immer zu 1 addiert werden können. Um dieses Problem zu beheben, können die Gewichte aller Klassen normalisiert werden.

Dies wird mit folgender Gleichung beschrieben:

$$\hat{w}_{ci} = \frac{w_{ci}}{\sum_i w_{ci}} \tag{2.10}$$

Hierbei ist $w_c i$ die normale Gewichtung eines Wortes i aus der Klasse c. Die Normalisierung $\hat{w}_{ci}$ wird durch das Teilen durch die Summe aller Gewichte der Klasse c berechnet.

2.3.5 Transformed Weight-normalized Complement Naive Bayes

In dem Artikel *Tackling the Poor Assumptions of Naive Bayes Text Classifiers* wird beschrieben, dass obwohl Naive Bayes ein häufig verwendeter Klassifikator ist, dies vorallem seiner Einfachheit und anstatt seiner Genauigkeit geschuldet ist. Doch weiter wird ausgeführt, dass weniger fehlerhafte Ergebnisse nur mit weitaus langsameren und komplexeren Algorithmen zu erzielen seien, sodass sich die Autoren diesem Problem angenommen haben und eine signifikante Verbesserung der Klassifikationsergebnisse erreichen konnten.[4]

Der von den Autoren verbesserte Algorithmus ist folgendermaßen beschrieben[5]:

- Sei $\vec{d} = (\vec{d_1}, ..., \vec{d_n})$ eine Menge von Dokumenten; d_{ij} ist die Häufigkeit von Wort i in Dokument d_j.
- Seien $\vec{y} = (\vec{y_1}, ..., \vec{y_n})$ die Dokumentenklassen.
- $TWCNB(\vec{d}\vec{y})$
  1. $d_{ij} = log(d_{ij} + 1)$ (Logarithmische Häufigkeit)
  2. $d_{ij} = d_{ij}\, log\frac{\sum_k 1}{\sum_k \delta_{ik}}$ (*tf-idf*-Maß siehe Kapitel 2.3.1
  3. $d_{ij} = \frac{d_ij}{\sqrt{\sum_k (d_k j)^2}}$ (Längen Normalisierung Kapitel 2.3.2)
  4. $\hat{\theta} = \frac{\sum_{j:y_j \neq c} d_{ij} + \alpha_i}{\sum_{j:y_j \neq c} \sum_k d_{kj} + \alpha}$ (Klassen Komplement Kapitel 2.3.3)
  5. $w_{ci} = log\hat{\theta}_{ci}$
  6. $w_{ci} = \frac{w_{ci}}{\sum_i |w_{ci}|}$ (Gewichtungs Normalisierung Kapitel 2.3.4)
  7. Sei $t = (t_1, ..., t_n)$ ein Testdokument; sei t_i die Worthäufigkeit des Wortes i.
  8. Klassifiziere das Dokument nach

$$l(t) = arg\, \min_c \sum_i t_i w_{ci} \tag{2.11}$$

In den ersten drei Schritten wird die *tf-idf* Gewichtung von d_{ij} berechnet und normalisiert. Nachdem dies durchgeführt wurde, liegt für das Dokument d eine prinzipiell nutzbare Gewichtung vor. Aufgrund der im Artikel beschriebenen Ungenauigkeiten dieser Berechnung bezüglich anderer Klassen wird aus dieser Gewichtung im vierten Schritt das Komplement der Klasse d berechnet. Dieses enthält, wie im Kapitel 2.3.3 beschrieben, die Gewichtung aller Wörter in Dokumenten ohne d. Daraufhin wird im fünften Schritt das Komplementgewicht w_{ci} mithilfe des Logarithmus bestimmt. Die daraus resultierenden Gewichte werden im 6ten Schritt noch einmal normalisiert, was den Hintergedanken hat, dass Gewichte bestimmter Wörter Tendenzen zu einer bestimmten Klasse c aufweisen und dadurch eine Bevorzugung dieser Klasse bewirken könnten.

Nach diesem Schritt ist die Berechnung der Klassengewichtung abgeschlossen.

[4] [JDMRK03, Vgl.: Kapitel 1 und Kapitel 4.4]
[5] [JDMRK03, Vgl.: Kapitel 4.4]

In den Schritten 7 und 8 wird beschrieben wie einem Dokument t eine Klasse zugeordnet wird.

Dies geschieht nicht wie im klassischen *Naive-Bayes-Klassifikator* über eine Maximum-Funktion, sondern über eine Minimum-Funktion, da die durch den erweitereten Algorithmus errechneten Gewichte negativ sind.

Damit ist die größte Zugehörigkeit zu einer Klasse durch einen signifikant kleineren Wert ersichtlich.

Die durch diesen Algorithmus gewonnenen gewichteten Klassen werden zusammengefasst auch als *Dokumenten-* oder *Klassifikations- Korpus* bezeichnet.

2.3.6 Beispielrechnung von Transformed Weight-normalized Complement Naive Bayes

Seien y_1, y_2 zwei Dokumentenklassen und $d_1 \in y_1$, $d_2 \in y_2$ zwei Dokumente mit $d_1 = (Dies\, ist\, ein\, erstes\, Beispiel)$ und $d_2 = (Dies\, ist\, ein\, zweites\, Beispiel)$.

Um aus diesen Klassen die entsprechenden Gewichtungen zu berechnen müssen die Komplemente $\hat{\theta}_1, \hat{\theta}_2$ berechnet werden. Da jedes Wort nur einmal per Dokument auftaucht ist deren logarithmische Häufigkeit $d_{ij} \approx 0,3$ ist und die inverse Häufigkeit für alle Wörter, außer „erstes“ und „zweites“, ist $idf \approx 0,3$. Für diese beiden ist die inverse Häufigkeit $idf \approx 0,22$.

Die *tf-idf*-Maße für alle Wörter sind folgende:

$$\delta_{Dies} = \delta_{ist} = \delta_{ein} = \delta_{Beispiel} \approx 0,09$$
$$\delta_{erste} = \delta_{zweites} \approx 0,066$$

Im nächsten Schritt müssen diese Werte normalisiert werden. Die Summe $\sqrt{\sum_k (d_k j)^2}$ berechnet sich dann folgendermaßen:

$$\sqrt{\sum_k (d_k j)^2} \approx \sqrt{0,09^2 + 0,09^2 + ... + 0,066^2 + 0,066^2} \approx \sqrt{0,073512} \approx 0,271$$

Die normalisierten Werte sind dann:

$$\delta_{Dies} = \delta_{ist} = \delta_{ein} = \delta_{Beispiel} \approx 0,09/0,271 \approx 0,33$$
$$\delta_{erste} = \delta_{zweites} \approx 0,066/0,271 \approx 0,244$$

Mithilfe diesen Berechnungen lassen sich die Komplementklassen $\hat{\theta}_1, \hat{\theta}_2$ mithilfe der im vierten Schritt angegebenen Gleichung berechnen:

$$\hat{\theta}_{1erstes} = \frac{\sum_{j:y_j \neq c} d_{ij} + \alpha_i}{\sum_{j:y_j \neq c} \sum_k d_{kj} + \alpha} = \frac{\alpha_i}{\delta_{Dies} + \delta_{ist} + \delta_{ein} + \delta_{Beispiel} + \delta_{zweites} + \alpha} = \frac{1}{11,56} \approx 0.087$$

Dabei muss der Komplementwert für jedes Wort der Klasse einzeln berechnet werden.

$$\hat{\theta}_{1dies} = \frac{\sum_{j:y_j \neq c} d_{ij} + \alpha_i}{\sum_{j:y_j \neq c} \sum_k d_{kj} + \alpha} = \frac{\delta_{Dies} + \alpha_i}{\delta_{Dies} + \delta_{ist} + \delta_{ein} + \delta_{Beispiel} + \delta_{zweites} + \alpha} = \frac{1,33}{11,56} \approx 0,12$$

$$\begin{aligned} \hat{\theta}_{1dies} = \hat{\theta}_{2dies} &\approx 0,12 \\ \hat{\theta}_{1ist} = \hat{\theta}_{2ist} &\approx 0,12 \\ \hat{\theta}_{1Beispiel} = \hat{\theta}_{2Beispiel} &\approx 0,12 \\ \hat{\theta}_{2zweites} &\approx 0,087 \end{aligned}$$

Hierbei wurden α_i ist hierbei 1 und α ist die Anzahl aller Wörter, in diesem Fall 10. Die errechneten $\theta_c i$ der Klassen ergeben die Gewichtungen mit denen die Klassenzugehörigkeit eines Dokumentes berechnet werden kann. In den letzten beiden

Schritten (Schritt 5 und 6), werden diese Werte logarithmiert und normalisiert. Daraus resultieren folgende Gewichte für die Klassen y_1 und y_2:

$$\begin{aligned}
w_{1dies} = w_{2dies} &\approx \frac{-0.92}{3,74} \approx -0,245 \\
w_{1ist} = w_{2ist} &\approx \frac{-0.92}{3,74} \approx -0,245 \\
w_{1ein} = w_{2ein} &\approx \frac{-0.92}{3,74} \approx -0,245 \\
w_{1Beispiel} = w_{2Beispiel} &\approx \frac{-0.92}{3,74} \approx -0,245 \\
w_{1erstes} = w_{2zweites} &\approx \frac{-0.06}{3,74} \approx -0,016
\end{aligned}$$

Dabei ist zu beachten, dass die Werte für $w_{1zweites}$ und $w_{2erstes}$ beide 0 sind, da diese nur in den Klassen y_1 beziehungsweise y_2 vorkommen.

Wenn diese errechneten Gewichte auf die Dokumente der Klassen y_1 und y_2 angewendet werden werden die Dokumente d_1 und d_2 den entsprechenden Kategorien zugewiesen. Dies wird durch die Minimum Funktion im letzten Schritt der Gleichung 2.3.5 beschrieben, in diesem Fall sind die Testdokumente $t_1 = d_1$ und $t_2 = d_2$.

$$l(t_1) = arg\min_c \sum_i t_i w_{ci}$$

Die Zuweisung von d_1 zu der Klasse y_1 berechnet sich mithilfe der Gewichte wie folgt:

$$w_{t_1 y_1} = w_{1dies} \cdot 1 + w_{1ist} \cdot 1 + w_{1ein} \cdot 1 + w_{1erstes} \cdot 1 + w_{1Beispiel} \cdot 1 \approx -0,996$$

und für die Klasse y_2 mit

$$w_{t_1 y_2} = w_{2dies} \cdot 1 + w_{2ist} \cdot 1 + w_{2ein} \cdot 1 + w_{2erstes} \cdot 1 + w_{2Beispiel} \cdot 1 \approx -0,98$$

Da $w_{t_1 y_1} < w_{t_1 y_2}$ würde Dokument d_1 korrekterweise der Klasse y_1 zugewiesen.

2.4 Auswahl der Klassen

Da eine vollständige Abdeckung der häufigsten Kategorien von Nachrichtenartikeln sehr umfangreich ist, werden in dieser Arbeit nur einige ausgewählte Kategorien betrachtet. Die Auswahl ist folgende:

- Rechtsextremismus
- Rechtspopulismus
- Islamismus
- Terrorismus
- Flüchtlingskrise
- Gewalt
- Kriminalität

Diese Auswahl wurde vorallem aufgrund der gesellschaftlichen Brisanz und Aktualität getroffen. Die Flüchtlingskrise ist politisch sehr bestimmend und sorgt für Entscheidungen, die in den Medien teilweise stark kritisiert werden. Gleichzeitig ist ein weltweites Erstarken rechtspopulistischer Positionen in politischen Diskussionen und Wahlen zu sehen, sowie das Ansteigen politisch motivierter Straftaten. Dies steht auch im Zusammenhang mit dem auch in Deutschland bestehenden Bedrohungen durch potenzielle Terroranschläge durch radikal Islamistische Gruppierungen, weshalb dies ebenso als zubetrachtete Kategorie hinzugefügt wurde.

Die Auswahl der Nachrichtenportale wurde auf *Spiegel-Online*[6] und die *Sueddeutsche-Online*[7] beschränkt.

Um eine Datengrundlage zu schaffen, ist es nötig eine möglichst große Anzahl an Nachrichtenartikeln zu lesen und zu klassifizieren. Da die Beschränkung auf zwei Nachrichtenportale und sieben Kategorien vorgenommen wurde ist die Erhebung der nötigen Artikel einfacher. Damit der Klassifikationsalgorithmus ausreichend gut funktioniert, werden mindestens 50 Artikel zusammengesucht, gelesen und entsprechend klassifiziert. Die gesammelten Artikel werden in einer Tabelle mit den Angaben *Schlagzeile*, *Datum*, *Link* und *Kategorien* gespeichert.

[6] http://www.spiegel.de
[7] http://www.sueddeutsche.de

3

Implementierung

In diesem Kapitel wird auf Implementierungsdetails der im Kapitel Konzeption beschriebenen Teilaufgaben eingegangen. Der verwendete Naive-Bayes-Algorithmus wird dabei nur grob beschrieben, da dieser im Kapitel 2.2.3 bereits ausführlich erläutert wurde.

3.1 Programmablauf

Der Programmablauf hat zwei Phasen, einmal die Startphase, in der der Klassifikationskorpus aufgebaut wird und einer zweiten Phase, in der das Programm benutzt wird und aktuelle Artikel durchsucht und klassifiziert werden.

3.1.1 Startphase

Die Startphase wird in der Datei *setup.lisp* eingeleitet. Zu Beginn werden die verschiedenen Programmteile mithilfe von *Quicklisp* in folgender Reihenfolge geladen:

1. *Quicklisp*: Damit *Quicklisp*[1] die Programmteile laden kann, muss es als erstes geladen sein. *Quicklisp* ist eine Paketverwaltung für Lisp welche Pakete aus sogenannten Repositories lädt, die dort von verschiedenen Entwicklern bereitgestellt werden.
2. *Utility-Funktionen*: Im Utility-Teil sind Hilfs-Funktionen zum Lesen und Schreiben von externen Dateien vorhanden. Dies ist nötig, um die Konfiguration zu laden.
3. *articlereader*: Dieses Paket ist notwendig, um Artikel einzulesen.
4. *Website-Crawler*: Dies ist eine von Kommolitonen entwickelte Anwendung, die den Textinhalt von Webseiten ausliest und als Zeichenkette zurückgibt. Diese ist nötig damit der *articlereader* funktioniert.
5. *indexer*: Diese Komponente berechnet Textmetriken der durch den *articlereader* eingelesenen Artikel.
6. *classificator*: Der Klassifikationsalgorithmus

[1] [Bea]

7. *data*: Eine Schnittstelle, die die vorkonfigurierten Dateien lädt und mithilfe des Klassifikators und einer Liste vorklassifizierter Artikel den Klassifikationskorpus erstellt.

Nachdem diese Programmteile geladen wurden, müssen mithilfe der *data*-Komponente die im Unterkapitel 3.5 beschriebenen Konfigurationsdateien geladen werden. Dies lädt die Kategorien, die Strukturen zum Auslesen von Artikel, sowie die vorklassifizierten Artikel.

Mithilfe der Funktion *build-classificator* wird dann der Klassifikationskorpus erstellt. Dies ist eine rechenintensive Operation, die einige Minuten in Anspruch nimmt.

Ist dies abgeschlossen kann das Programm verwendet werden.

3.1.2 Benutzungsphase

Nachdem das Programm wie im vorherigen Abschnitt beschrieben vorbereitet wurde, muss noch die GUI-Komponente geladen werden.

Dieser Komponente müssen die Kategorien sowie die mithilfe des Klassifikators erstellten Klassen-Symbole mitgeteilt werden.

Desweiteren muss eine Suchfuntkion definiert werden, die spezifiziert, was bei einer gestarteten Suche durchgeführt werden muss. Diese hat zwei Parameter, einmal einen Suchterm, der genutzt werden kann um nach Schlagworten zu filtern, sowie einer Liste von ausgewählten Kategorien nach denen gefiltert wird.

```
1  (lambda (term categories)
2     (mapcan (lambda (source)
3        (let* (teasers (read-teasers source))
4  (mapcan (lambda (teaser)
5            (let* ((tarticle (read-article teaser-link))
6                   (class  (classificator:classify-document
7                              (indexer:make-index tarticle)))
8                   (class-name (first (first class)))
9                   (class-value (second class)))
10                   (cond ((and categories (member class-name categories)
11                                          (<= class-value -0.015))
12                                          (list (list (get tarticle 'ARTICLEREADER
                                                :HEADLINE) class-name class-value (
                                                concatenate 'string prefix teaser))))
13                         ((and (not categories) (<= class-value -0.015)) (
                               list (list (get tarticle 'ARTICLEREADER:HEADLINE)
                               class-name class-value (concatenate 'string prefix
                                teaser))))
14                         ((not categories) (list (list (get tarticle '
                               ARTICLEREADER:HEADLINE) "" 0 (concatenate 'string
                               prefix teaser))))
15                         (T NIL)))) teasers)
16  )) (data:get-pagestructure-types)
```

Abb. 3.1. Vereinfacht dargestellte Suchfunktion

In Abbildung 3.1 wird die in diesem Programm verwendete Suchfunktion vereinfacht dargestellt. Zuerst werden die sogenannten *teasers* ausgelesen. Dies sind die

Links einer Übersichtsseite wie „www.spiegel.de/politik/deutschland“ die zu den konkreten Artikeln führen. Diese werden mithilfe der vorkonfigurierten Artikel- und Seitenstrukturen ausgelesen und dann klassifiziert (ab Zeile 5 3.1). Aus jedem *teaser-Link* wird der Artikel ausgelesen und die Klassifikation durchgeführt. Danach werden vier Fälle unterschieden:

1. Dem Artikel wurde eine Kategorie zugewiesen und diese wurde auch vom Benutzer ausgewählt (siehe Kapitel 3.6.1). Der Artikel wird mit der Kategorie in die Ergebnisliste geschrieben.
2. Dem Artikel wurde eine Kategorie zugewiesen und der Benutzer filtert nicht nach Kategorien: Der Artikel wird mit der Kategorie ebenso in die Ergebnisliste geschrieben.
3. Dem Artikel wurde keine Kategorie zugewiesen und es wurde nicht nach Kategorien gefiltert: der Artikel wird ohne Kategorie in die Ergebnisliste geschrieben.
4. Fehlerfall, der nicht auftreten sollte.

In den Zeilen 10 und 11 ist zu sehen, dass der Bewertungswert des Artikels mit einer Konstante verglichen wird. Diese setzt einen Schwellwert fest um zu bestimmen ob die Klassifizierung relevant genug ist. Dieser Wert muss in Abhängigkeit zu der Länge der Testdaten passend gewählt werden und kann durch ausprobieren verschiedener Werte zwischen 0 und -1 bestimmt werden.

3.2 HTML-Parser und Textextraktion

Die *articlereader* genannte Komponente liest den Text und weitere Informationen eines per Link angegebenen Nachrichtenartikels aus. Ziel ist es, alle relevanten Informationen eines Artikels in einer einheitlichen Form zur weiteren Verwendung aufzubereiten. Dabei muss bei verschiedenen Nachrichtenportalen verschieden vorgegangen werden. Um dies zu ermöglichen wurde eine Software-Komponente aus dem Teamprojekt *Entwurf und Realisierung eines Systems zur prioritätsgesteuerten Suche im Internet* der Autoren Jochen Fuchs, Jürgen Fuchs und Thomas Hormesch[2] verwendet, die den Textinhalt einer Webseite als Zeichenkette zurückliefert.

Mithilfe der Bibliothek „closure-html“ von Gilbert Bauman[3] wird aus dieser Zeichenkette eine Listenstruktur erzeugt (siehe Abbildung 3.2), mit der im weiteren Verlauf gearbeitet wird. Daraus resultiert eine Art Baumstruktur der HTML-Inhalte, die aus drei Teilen besteht dem HTML-Tag, einem Deskriptor und einer Liste der von dem HTML-Tag umschlossenen Inhalte. Dies können Zeichenketten oder weitere geparste HTML-Inhalte sein.

```
1 <body>
2    <h1>Eine Überschrift</h1>
3    <p id="eins">Ein Satz.</p>
4    <p id="zwei">Ein weiterer Satz.</p>
5 </body>
```

```
1     (:BODY NIL (
2        (:H1 NIL ("Eine Überschrift"))
3        (:P ((:ID "eins")) ("Ein Satz."))
4        (:P ((:ID "zwei")) ("Ein weiterer Satz."))))
```

Abb. 3.2. Beispiel der Umwandlung einer HTML-Struktur in die von „closure-html“ erzeugte Listenstruktur

Nachdem aus einem Artikel eine solche mit Lisp traversierbare Struktur erzeugt wurde, muss entschieden werden welche Inhalte relevant sind. Diese Inhalte müssen ausgelesen und zur weiteren Verarbeitung gespeichert werden.

Um dieses Problem zu lösen wurde eine primitive Auszeichnungssprache entwickelt, die es ermöglicht anhand aus der HTML-Struktur der Nachrichtenportale gewonnen Struktur die gewünschten Inhalte aus der HTML-Seite auszulesen. Diese wurde als Liste realisiert die angibt, an welcher Stelle des HTML-Codes gesuchte Inhalte zu finden sind. Mithilfe der zwei Bezeichner *:SEQUENCE* und *:PARALLEL*, kann angegeben werden ob Inhalte aufeinanderfolgend oder nebeneinander vorliegen. Dies entspricht einer Unterscheidung zwischen Tiefen- und Breitensuche. Abgesehen davon ist diese Auszeichnungsprache der durch „closure-html“ erzeugten Struktur sehr ähnlich, ist allerdings um Bezeichner erweitert, die es ermöglichen Inhalte als irrelevant zu deklarieren oder aber relevante Inhalte zu speichern.

[2] [JFH16]
[3] [Bau]

- *:IGNORE* Der Inhalt wird ignoriert. Bei Strukturvergleichen erzeugt dieser Ausdruck immer eine Übereinkunft und die Werte werden verworfen.
- *:TEXT* Der Inhalt wird als Artikeltext gespeichert.
- *:INTRO* Der Inhalt wird als Artikeleinleitung gespeichert.
- *:HEADLINE* Der Inhalt wird als Schlagzeile gespeichert.
- *:DATE* Der Inhalt wird als Datum gespeichert.

Es sind weitere Bezeichner vorhanden, diese sind allerdings nur zur internen Verarbeitung relevant und werden nicht aufgeführt.

```
1  (:SEQUENCE
2     (:DIV ((:CLASS "header")) (:SEQUENCE
3        (:DIV ((:DATETIME :DATE) (:CLASS "timeformat")))
4        (:H2 NIL (:SEQUENCE :IGNORE :HEADLINE))))
5    (:DIV ((:CLASS "body") (:ID "article-body")) (:PARALLEL (:P NIL :TEXT))))
```

Abb. 3.3. Auszeichnungsstruktur für Artikel der Süddeutschen Zeitung

Die so ausgelesenen Informationen über den Artikel werden in einem Symbol gespeichert. Dort wird die Schlagzeile, der Text und das Veröffentlichungsdatum sowie die Textlänge gespeichert. Die so erzeugten Symbole können im nächsten Schritt, der Indexierung, verwendet werden.

3.3 Dokumentenindexierung

Nachdem die Artikelinformationen extrahiert wurden, muss der Text des Dokuments indexiert werden. Das bedeutet, dass die Wortvorkommnisse gezählt und abgespeichert werden. Die Komponente *indexer* hat diese Aufgabe.

Die in einem Artikel-Symbol vorliegenden Textabschnitte werden zu einem vollständigen Text zusammgengefügt und Satzzeichen werden entfernt. Dies hat den Hintergrund, dass im verwendeten Klassifizierungsalgorithmus Satzzeichen, auch Stoppzeichen genannt, keine verbesserte Klassifizierung ermöglichen. Würden Satzzeichen nicht entfernt, so würden die Satzzteile „Berlin“, „Berlin.“, „Berlin!“ etc. als unterschiedliche Wörter gezählt.

Dann wird ein *Dictionary* verwendet um die vorhandenen Wörter in diesem abzuzählen. Dies wird mit einer simplen Iteration über alle Wörter des Textes vollbracht.

So wird beispielsweise der Satz „Ich genieße den Tag, während ich auf meinem Sessel sitze“ folgendermaßen indexiert:

```
1  ((ich 2)
2  (genieße 1)
3  (den 1)
4  (Tag 1)
5  (während 1)
6  (auf 1)
7  (meinem 1)
8  (Sessel 1)
9  (sitze 1))
```

```
1  ARTICLEREADER:DATE
2      "2016-11-15 08:33:00"
3  ARTICLEREADER::FULLTEXT
4      "Sie sollen Flaschen  Steine und Böller auf Polizisten geworfen haben:
5      Für ihre Beteiligung an den Krawallen in Heidenau im August 2015 müssen
           zwei Männer ins Gefängnis"
6  ARTICLEREADER:HEADLINE
7      ("Drei Männer wegen Randale vor Flüchtlingsheim verurteilt")
8  ARTICLEREADER::INTRO
9      ("Heidenau")
```

Abb. 3.4. Beispiel der *Property*-List eines Artikel-Symbols

In der Abbildung 3.4 ist ein Beispiel eines eingelesen Artikels als Lisp-Symbol dargestellt. Wird dieser Artikel nun indexiert, resultiert daraus ein neues Symbol, das eine Liste der gezählten Wörter sowie die Anzahl der Wörter beinhaltet.

3.4 Klassifikation

Der im Kapitel 2.3.5 beschriebene Algorithmus wird in der Komponente *classificator* implementiert.

3.4.1 Klassenberechnung

Die Funktion *calculate-weights* (siehe Abbildung 3.5) ist dabei eine der wichtigsten Funktionen. Sie berechnet die Gewichtungsfaktoren einer Klasse *classId*. Dafür sind fünf Schritte notwendig:

1. Das Berechnen der komplementären Klasse $\bar{c}$; hier als *complement* bezeichnet.
2. Das Errechnen der Gewichtungsmetriken *oben* der Wörter aus *classId* in allen Dokumenten *document*, die nicht in *classId* enthalten sind.
3. Die Berechnung der Summe *unten* der Gewichtungsmetrik aller Wörter, die Teil von *complement* sind.
4. Das Berechnen des logarithmischen Verhältnisses der Wörter aus *oben* zu *unten*.
5. Das Anhängen des Ergebnisses *weights* an das bestehende Symbol der Klasse *classId*.

3.4.2 Dokumenten-Klassifikation

Eine weitere wichtige Funktion ist *classify-document* (siehe Abbildung 3.6). Diese führt die Klassifizierung für ein Dokument *document* durch und liefert die Klasse mit der größten Übereinstimmung.

Dafür wird über alle vorhandenen Klassen iteriert und die Bewertung des Dokumentes im Bezug zu dieser Klasse berechnet. Die hier verwendeten Gewichte sind die aus der Funktion *calculate-weights* im Vorfeld berechneten.

Das Dokument wird mithilfe der Funktion *document-value* (siehe Abbildung 3.7) bewertet. Dabei kommt eine Bewertung $-1 >= b >= 0$ heraus die angibt wie

```
(defun calculate-weights (classId)
  (if (not *sum-of-words*) (setq *sum-of-words* (sum-of-words)))
  (let* ((complement (get-complement-class classId))
         (a 0)
         (oben (mapcan (lambda (document)
                 (mapcar (lambda (word) (setf a (+ a 1))
                   (list (first word) (+ (tf-idf document (first word))
                     1)))
                 (get classId 'INDEXER:WORD-LIST)))
               (get-documents :ignore classId)))
         (unten (+ a (reduce #'+ (mapcan (lambda (document)
                 (mapcar (lambda (word)
                   (normalize (tf-idf document (first word))))
                 (get complement 'INDEXER:WORD-LIST)))
               (get-documents :ignore classId)))))
         (weights (mapcar (lambda (o)
               (list (first o) (log (/ (second o) unten)))) oben)))

    (setf (get classId 'WEIGHTS) (normalize-weights weights))))
```

Abb. 3.5. Gewichtungsfunktion *calculate-weights*

```
(defun classify-document (document)
  (let* ((sorted (sort (mapcar (lambda (class) (list class (document-value
      document (second class)))) (get-classes)) #'< :key 'second))
       (rel (first sorted))
       (result NIL))
      (if (not (equal (second rel) 0))
        (setf result (list rel)))
     (first result)))
```

Abb. 3.6. Klassifizierungsfunktion *classify-document*

sehr das Dokument zu einer Klasse *class* gehört. Dies wird mit allen vorhandenen Klassen durchgeführt. Die daraus resultierende Liste wird sortiert und das kleinste Ergebnis als die zugehörige Klasse zurückgegeben. Die zurückgegebene Klasse muss allerdings noch geprüft werden, da bei zu kleinem b die Klassifizierung nicht genau genug ist. Dies wird allerdings von einer anderen Programmstelle erledigt und ist nicht Teil des Klassifikationsalgorithmus, da der Vergleichswert von der Größe der Trainingsmenge abhängig ist.

```
(defun document-value (document classId)
  (reduce #'+ (mapcar (lambda (x) (let ((r (find (first x) (get classId '
      WEIGHTS) :key 'first :test 'string-equal))) (if r (nth 2 r) 0.0))) (
      get document 'INDEXER:WORD-LIST))))
```

Abb. 3.7. Bewertungsfunktion eines Dokumentes im Bezug zur Klasse *classId*

Die Funktion *document-value* schälgt in der zu vergleichenden Klasse *classId* für jedes Wort nach, welche Gewichtung verwendet werden muss und addiert diese. Das Resultat ist die entsprechende Bewertung für das Dokument im Bezug zu der Klasse *classId*.

Diese Funktionen werden auf die vorklassifizierten Dokumente angewandt und daraus resultiert der Klassifikationskorpus. Dieser ist ein Lisp-Symbol, das Refe-

renzen auf alle Klassen enthält. Diese Klassen sind weitere Lisp-Symbole, die die im Kapitel 2.3.5 beschriebenen Gewichte enthalten.

3.5 Konfigurierung und vorklassifizierte Dokumente

Daten über die verwendeten Nachrichtenportale sowie die vorklassifizierten Dokumente sollen nicht fest in der Anwendung verankert sein und müssen konfigurierbar bleiben. Zu diesem Zweck wird eine Komponente namens *data* entwickelt, die diese Aufgaben übernimmt.

3.5.1 Konfiguration

Die Daten über die verwendeten Nachrichten Portale werden vergleichbar mit einem Dictionary verwaltet. Die relevanten Daten werden in verschiednen Textdateien gespeichert, die beim Programmstart eingelesen werden. Dadurch ist ein modularer Aufbau erreicht worden, der es prinzipiell erlaubt eine beliebige Anzahl an Nachrichtenportalen zu unterstützen. Für diese Aufgaben liegen die drei Konfigurationsdateien *categories.txt*, *structure.txt* und *pagestructure.txt* vor.

```
1  ((1 "Rechtsextremismus")
2  (2 "Fremdenfeindlichkeit")
3  (3 "Gewalt")
4  (4 "Rechtspopulismus")
5  (5 "Fluechtlingskrise")
6  (6 "Terrorismus")
7  (7 "Islamismus")
8  (8 "Kriminalitaet"))
```

Abb. 3.8. Kategorienkonfiguration *categories.txt*

Der Inhalt der Datei *categories.txt* ist eine einfache Liste mit Unterlisten aller Kategorien. Diese kann als *Dictionary* betrachtet werden, die alle Kategorien durchnummeriert. Dabei wird die gewählte Nummer jeder Kategorie ähnlich eines Index in einer Datenbank zur Referenzierung verwendet. Dies wird beispielsweise bei dem Hinterlegen der Trainingsdaten eingesetzt.

```
1  (...
2   ("sueddeutsche" ((:SEQUENCE (:DIV ((:CLASS "header")) (:SEQUENCE (:DIV ((:
        DATETIME :DATE) (:CLASS "timeformat"))) (:H2 NIL (:SEQUENCE :IGNORE :
        HEADLINE))))
3           (:DIV ((:CLASS "body") (:ID "article-body")) (:PARALLEL (:P NIL :
                TEXT))))
4         ((:A ((:DATA-PAGETYPE "THEME") (:CLASS "themelink")) (:SEQUENCE :TEXT)
                ))))
5   ...)
```

Abb. 3.9. Artikelkonfiguration Süddeutsche

In der Abbildung 3.9 wird beschrieben wie ein Nachrichtenartikel einer Webseite ausgelesen wird. Dabei wird zuerst ein Kürzel des Portals angegeben (in diesem Fall „sueddeutsche“) und danach eine Liste mit zwei Unterlisten. Erstere ist eine wie im Unterkapitel 3.2 beschriebene Liste zur Beschreibung der Artikelstruktur und

zweitere eine ähnlicher Liste, die automatisch zu entfernende Links spezifiziert. Das automatisierte Entfernen von Links ist in manchen Fällen nötig, da einige Links den Textfluss komplizierter gestalten und ausgelesene Texte ansonsten nachträglich verändert werden müssten.

```
1  (...
2  ("sueddeutsche" (:SEQUENCE (:DIV ((:ID "wrapper"))
3                             (:PARALLEL (:DIV ((:ID "sitecontent")(:CLASS "
                                  mainpage")(:ROLE "main")) (:PARALLEL (:DIV ((:
                                  CLASS "teaser toptop")) (:PARALLEL (:A ((:ID :
                                  TEASER) (:CLASS "entry-title") (:REL "bookmark")
                                  (:DATA-PAGETYPE "STANDARD_ARTICLE") (:DATA-ID :
                                  IGNORE)) ()))) (:DIV ((:CLASS "teaser top")) (:
                                  PARALLEL (:A ((:ID :TEASER) (:CLASS "entry-title"
                                  ) (:REL "bookmark") (:DATA-PAGETYPE "
                                  STANDARD_ARTICLE") (:DATA-ID :IGNORE)) ()))))))))
                                  )
4  ...)
```

Abb. 3.10. Seitenkonfiguration Süddeutsche

Die dritte Konfigurationsdatei *pagestructure.txt* enthält eine weitere Art Artikelstruktur, die allerdings für Übersichtsseiten, die eine Artikelsammlung darstellen verwendet wird. Mithilfe der hinterlegten Strukturen lassen sich alle Artikel-Links einer Seite auslesen. Dies wird in der Anwendung genutzt um eine Liste aktueller Artikel zu bekommen und diese zu klassifizieren und in der Ergebnisliste (siehe Abbildung 3.13) der graphischen Oberfläche zu hinterlegen.

3.5.2 Vorklassifizierte Dokumente

Um eine Datengrundlage zu schaffen wurden 58 Nachrichtenartikel in diese Themen eingeordnet. Da diese Themen teilweise inhaltlische Überschneidungen haben, wurden ihnen in diesem Fall mehrere Kategorien zugeteilt.

Damit diese Daten einfach von der Lisp-Applikation geladen werden können, sind diese in einer Textdatei in einer Lisp-typischen Listenstruktur (siehe Abbildung 3.11) gespeichert. Diese Liste enthält für jede Quelle eine weitere Liste der Struktur $(Link\,(C_1 \ldots C_n)\,Quelle)$.

```
1  (("http://www.spiegel.de/politik/deutschland/[...]-a-1121063.html" (1 3 5) "
       spiegel")
2  ("http://www.spiegel.de/politik/deutschland/[...]-a-1119137.html" (7 8) "
       spiegel")
3  ("http://www.spiegel.de/politik/deutschland/[...]-a-1121045.html" (5 6 7) "
       spiegel"))
```

Abb. 3.11. Beispiel einer Datengrundlage

Die Zahlen $C1 \ldots C$ entsprechen den in der Kategorienkonfiguration angelegten Indizes der Kategorien. Die Angabe der Quelle ist nötig, da beispielsweise Links

auf *Spiegel-Online* ohne das Präfix „www.spiegel.de“ angegeben sind und diese Information zum Auslesen von Links mit angegeben werden muss. Desweiteren wird über diese Quellenangabe die entsprechnde Artikelstruktur geladen, damit die korrekten Textinhalte extrahiert werden können.

Diese Artikel werden von der Funktion *build-classificator* eingelesen und dann mithilfe der *classificator*-Komponente zum Erstellen des Klassifikationskorpus genutzt.

3.6 Graphische-Benutzer-Oberfläche

In diesem Kapitel wird die durch die *gui* genannte Komponente erstellte Graphische-Benutzer-Oberfläche beschrieben. Im ersten Unterpunkt wird auf die Suche und Filterung eingegangen, im zweiten auf die Übersicht der verwalteten Daten.

3.6.1 Suche

Das Hauptfenster (siehe Abbildung 3.12) hat zwei Reiter. Der *Such*-Reiter beinhaltet ein Suchfeld mit dem die gesuchten Artikel per Schlagwort eingegrenzt werden können, ein Auswahlfeld aller Kategorien mit denen diese gefiltert werden können und eine Ergebnisliste. Diese listet die Schlagzeile, die Kategorie, eine Bewertungszahl und einen Link zu dem Artikel auf.

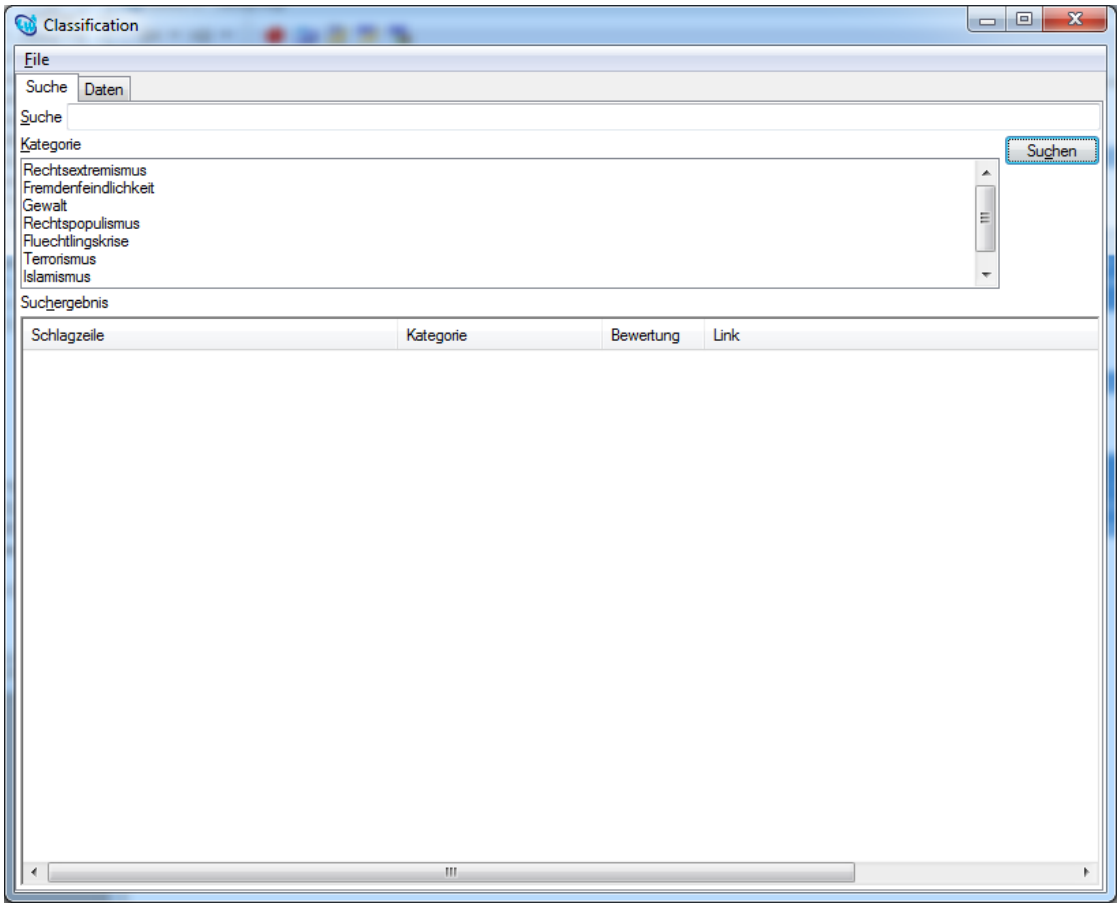

Abb. 3.12. Hauptfenster der Anwendung

Bei der Suche gibt es drei verschiedene Möglichkeiten. Die einfachste Variante ist, dass weder nach Schlagworten noch nach Kategorien gefiltert wird.

Die Suche beschränkt sich dabei auf die aktuellen Artikel der Kategorien *Politk* beziehungsweise *Deutschlandpolitik*, der Online-Nachrichtenportale *Süddeutsche* und *Spiegel*. Dies ist in der Abbildung 3.13 dargestellt.

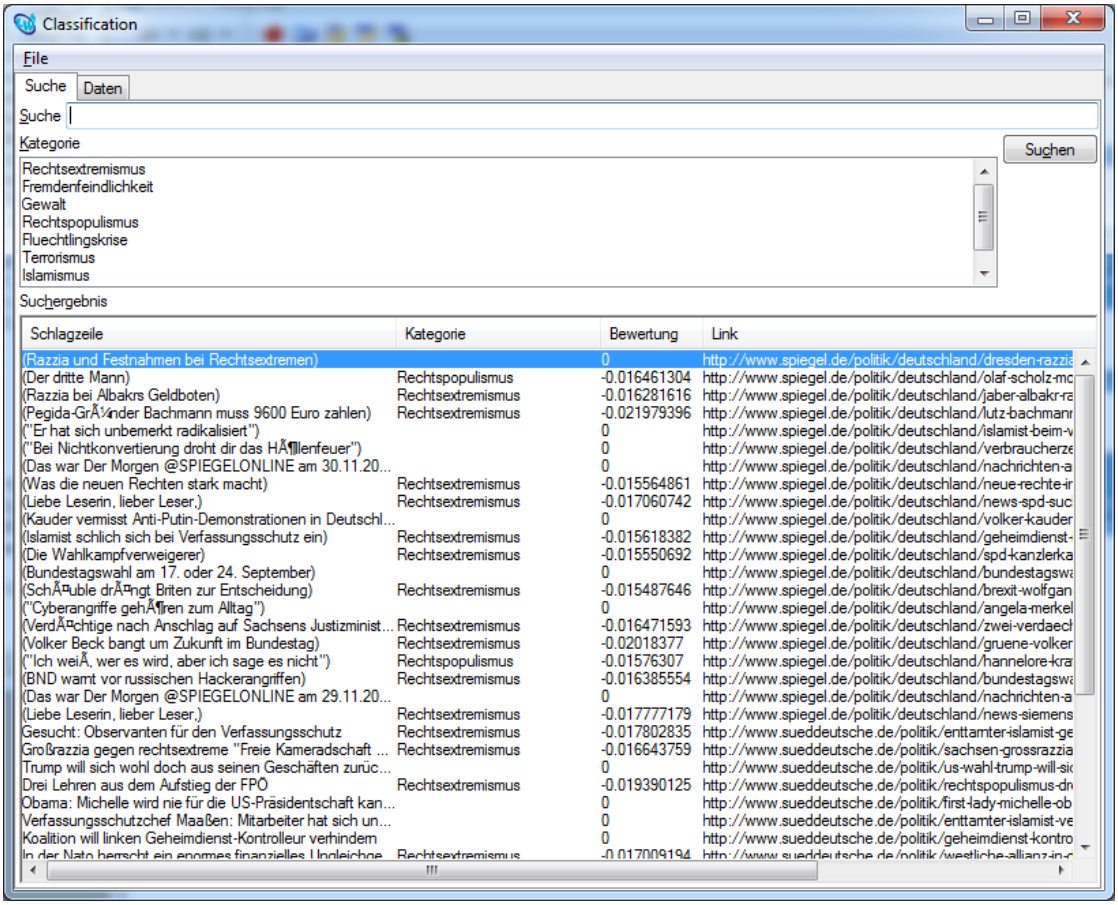

Abb. 3.13. Ergebnis einer Suche ohne Filterung

Wurden eine oder mehrere Kategorien ausgewählt, so werden nur Artikel, die einer dieser Kategorien zugewiesen werden konnten, angezeigt.

Diese Filterung ermöglicht eine effiziente Suche nach Themen, die für den Anwender von Interesse sind, ohne Artikel durchzusehen, die keinen inhaltlichen Bezug zum Interessengebiet haben.

Im letzten Anwendungsfall gibt es die Möglichkeit, zusätzlich nach Schlagworten zu filtern. Dabei wird der Artikeltext nach Vorkommnissen durchsucht. Da diese Suche allerdings keine intelligente Suche darstellt, muss diese auf grobe Schlagworte begrenzt sein und bietet nicht die Möglichkeit zusammenhängende Sätze oder Wörter gezielt zu suchen. Allerdings ermöglicht dies eine bessere Filterung nach beispielsweise Personen öffentlichen Interesses, die in den durchsuchten Artikeln erscheinen könnten.

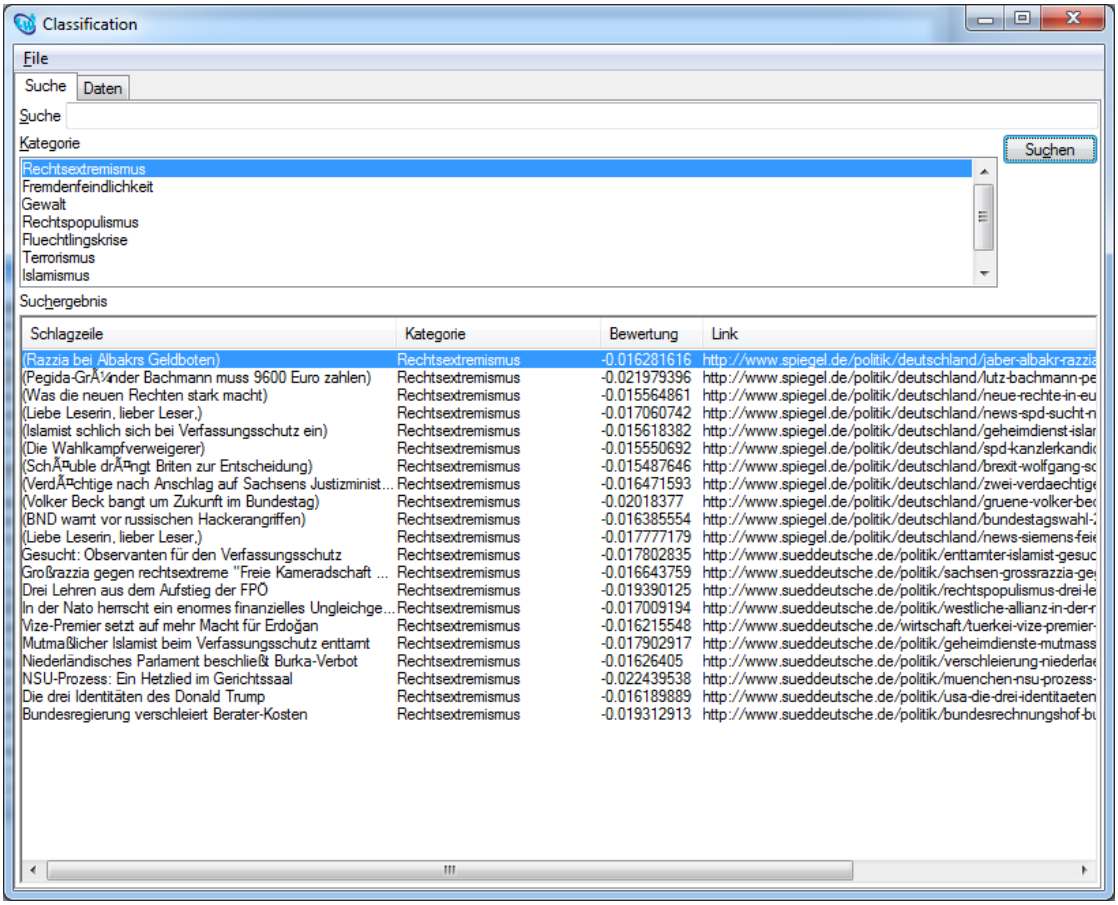

Abb. 3.14. Ergebnis einer Suche, gefiltert nach der Kategorie „Rechtsextremismus“

3.6.2 Daten

Auf dem zweiten Reiter *Daten* (siehe Abbildung 3.15) können die vordefinierten Klassen in einer Baumstruktur eingesehen werden. Diese listet alle zugehörigen Dokumente sowie die mit diesen assoziierten Artikel auf.

Wird ein Element ausgewählt, werden die Symbol-Attribute angezeigt mit denen Details ersichtlich sind.

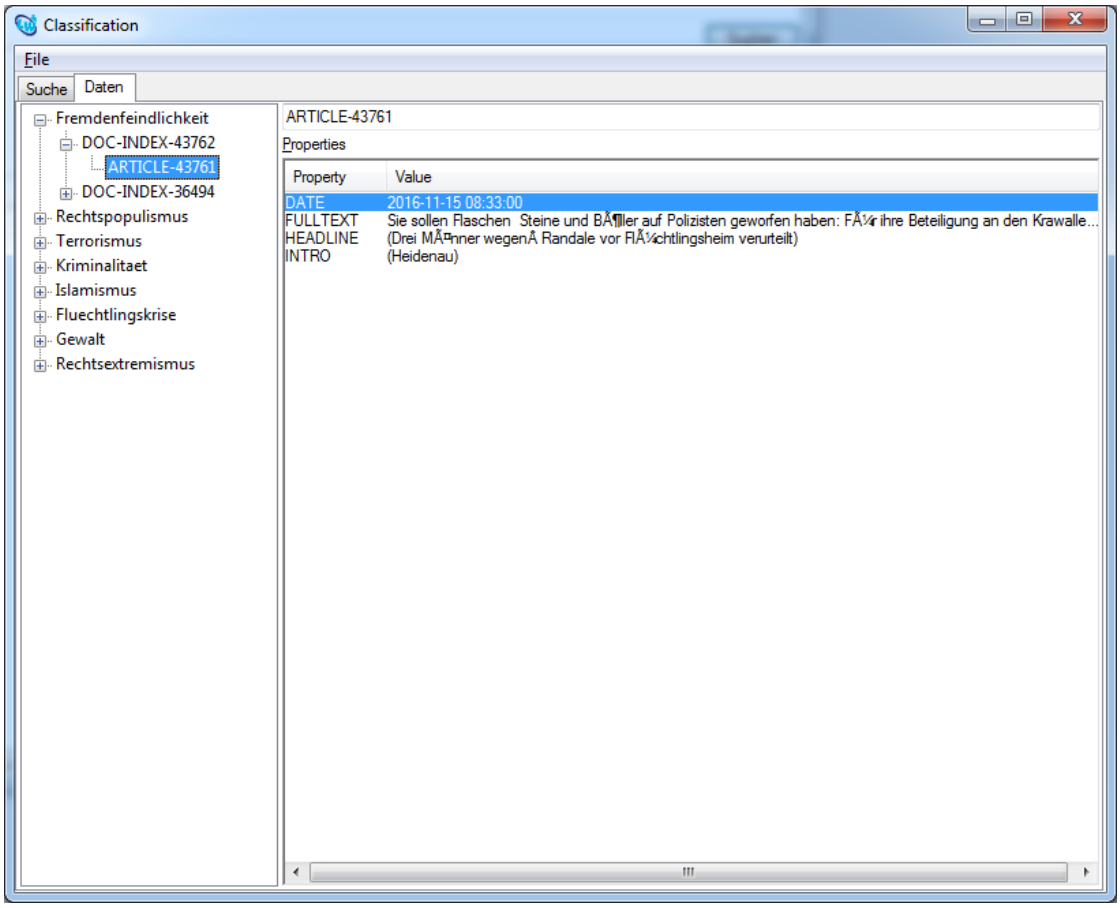

Abb. 3.15. Überblick über die vordefinierten Klassen

Diese Informationen ermöglichen es zu prüfen, ob die vordefinierten Kategorien entsprechend der Planung durch die Konfigurationsdateien erstellt wurden und liefern einen Einblick in die dahinterliegenden Datenstrukturen.

3.7 Benutzung der entwickelten Software

In diesem Kapitel wird erläutert, wie die erstellte Software zu starten ist und welche Rahmenbedingungen dafür erfüllt sein müssen.

3.7.1 Voraussetzungen

Um das erstellte Programm nutzen zu können muss die Entwicklungsumgebung *LispWorks* mit Versionsnummer 6.+ installiert sein, sowie eine Internetverbindung vorhanden sein. Mit älteren *LispWorks*-Versionen ist die Programmausführung nicht möglich und mit Versionen ab Nummer 7.0 ist die Anwendung nicht getestet und es ist nicht garantiert, dass alle Funktionen fehlerfrei arbeiten. Um das Programm in vollen Umfang nutzen zu können bietet sich die sogenannte *Professional*-Version and, da diese keine Speicher-Begrenzung hat. Dies tritt bei der kostenfreien *Personal*-Edition auf und kann zu Programmabstürzen führen, da *LispWorks* automatisch beendet wird wenn eine gewisse Speichergrenze überschritten wird, was bei einer größeren Menge an Trainingsdaten passiert. In diesem Fall ist es nötig mit einer kleineren Testmenge zu arbeiten, eine Anleitung dafür ist in der im folgenden Kapitel beschriebenen Setup-Datei hinterlegt.

3.7.2 Programmstart

Um die Anwendung zu starten muss die im *src*-Ordner befindliche Datei *setup.lisp* mit *LispWorks* geöffnet werden. Wird der Inhalt dieser Datei evaluiert öffnet sich die im Kapitel 3.6 beschriebene graphische Benutzeroberfläche. Sollte dies nicht der Fall sein, so sind in der Datei Kommentare hinterlegt, die dabei helfen das Programm erfolgreich zu starten.

Der Programmstart kann einige Minuten in Anspruch nehmen, da der gesamte Klassifikationskorpus erstellt werden muss. Die dafür verantwortliche Codestelle ist folgende:

```
1  (data:build-classificator (current-pathname "../data/categories" "txt")
2                            (current-pathname "../data/structure" "txt")
3                            (list (current-pathname "../data/new-data" "txt")))
```

Abb. 3.16. Funktion die den Klassifikationskorpus erstellt

Um einen kürzeren Programmstart zu ermöglichen kann die in Zeile 3 (Abbildung 3.16) angegebene Trainingsdatei „../data/new-data“ durch „../data/spiegel-data“ ersetzt werden. Dies kann die Startzeit halbieren, sorgt allerdings auch für eine ungenauere Klassifikation.

Wird dies getan, muss der in Kapitel 3.4.2 erwähnte Schwellwert zur Klassifikation angepasst werden, da dieser von der Länge der Trainingsdaten abhängig ist. Dies wird in der Setupdatei mithilfe der Konstante **classification-value** getan. Dieser muss ein Wert zwischen -1 und 0 zugewiesen werden. In der Setup-Datei

befinden sich mehrere mögliche Einstellungen dieser Konstante, die für die beigefügten Trainingsdaten relativ gut gewählt wurden.

4

Zusammenfassung

Das in dieser Arbeit erstellte Programm ermöglicht es, erfolgreich Artikel in passende Kategorien einzuordnen und somit eine gute Filterung von Nachrichtenartikel zu erzielen. Problematisch ist diese Kategorisierung allerdings bezüglich der Auswahl der Trainingsdaten. So ist zum einen eine sehr große Menge an Artikeln notwendig um eine genaue Klassifikation zu ermöglichen, doch ist dies auch mit einem großen Aufwand verbunden. Die in dieser Arbeit erstellten Trainingsdaten sind nicht Umfangreich genug um eine wirklich zufriedenstellende Kategorisierung zu ermöglichen, zeigen aber auch das Potenzial des Algorithmus.

Insgesamt lässt sich diese Arbeit als Erfolg werten. Die zugrunde liegenden Algorithmen sind voll funktionsfähig und die Aufteilung der Programmpakete ermöglicht eine einfache Wartung und Erweiterung. Der intensive Kontakt mit verschiedenen Verfahren zur Textklassifikation und anderen Data-Mining Algorithmen hat bei mir die Absicht geweckt sich noch weiter mit diesem Themengebiet zu beschäftigen und die genannten Verbesserungen des Programms in Zukunft zu realisieren.

5

Ausblick

Die erstellte Anwendung hat viel Potenzial für Verbesserungen, so wird beispielsweise der erstellte Klassifikationskorpus nicht gespeichert und muss bei jedem Programmstart neu erstellt werden, was mit einer mehrminütigen Wartezeit verbunden ist. Desweiteren ist die Abdeckung von nur zwei Nachrichtenportalen nicht optimal, kann aber durch die hier entworfene Bezeichnungssprache relativ einfach erweitert werden.
Eine weitere mögliche Verbesserung wäre es, die schon durchsuchten Artikel abzuspeichern und somit eine Möglichkeit zur Archivierung und langfristigen Verbesserung der Trainingsdaten zu ermöglichen. Die Suche könnte ebenso ausgeweitet werden, da sie sich zurzeit nur auf die auf der ersten Seite einer angegebenen Nachrichtenseite bezieht und ältere Artikel nicht durchsucht. Dies greift mit der erwähnten Archivierung zusammen, da so schon klassifizierte Artikel nicht erneut geprüft werden müssten.

Denkbar wäre auch eine Erweiterung die es ermöglicht auch andere Quellen zu klassifizieren, beispielsweise eingescannte Dokumente oder Magazine die mithilfe einer Texterkennungssoftware vorverarbeitet werden. So würde das Programm nicht nur von Online-Medien abhängig sein und würde Optionen bieten ein umfassendes Archiv aufzubauen. Hierfür könnte eine erweiterte Hierarchie sinnvoll sein, die es erlaubt auch Ober- beziehungsweise Unterkategorien zu definieren, mit denen die Kategorisierung und Suche verfeinert werden könnte.

Da der Klassifikationsalgorithmus mit einem simplen *Bag-of-Words*-Modell arbeitet, der lediglich einzelne Wörter verwendet, könnte auch über eine mögliche Verwendung von *N-Grammen* nachgedacht werden. Dies bedeutet statt jeweils ein Wort zu zählen, N Wörter zu zählen und dann zu klassifizieren. Es könnte ausgewertet werden ob dies eine Verbesserung erbringt oder nicht. Insgesamt können basierend auf dieser Arbeit viele solcher Optionen berücksichtigt werden. Die verwendeten Metriken können mit der im Kapitel 2.3 beschriebenen Optionen ausgetauscht werden und dann mithilfe von weiteren Testdaten die für Nachrichtenartikel beste Methode zu evaluieren.

Literaturverzeichnis

Bau. Bauman, Gilbert: *Closure-HTML Homepage*. `https://common-lisp.net/project/closure/closure-html/` (Stand: 08.12.2016).

Bea. Beane, Zach: *Quicklisp Homepage*. `https://www.quicklisp.org/beta/` (Stand: 08.12.2016).

CDMS08. Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze: *Introduction to Information Retrieval*, 2008. `http://www-nlp.stanford.edu/IR-book/` (Stand: 08.12.2016).

JDMRK03. Jason D. M. Rennie, Lawrence Shih, Jaime Teevan und David R. Karger: *Tackling the Poor Assumptions of Naive Bayes Text Classifiers*, 2003.

JFH16. Jochen Fuchs, Jürgen Fuchs und Thomas Hormesch: *Entwurf und Realisierung eines Systems zur prioritätsgesteuerten Suche im Internet*, 2016.

Kas05. Kaster, Andreas: *Automatische Dokumentklassifikation mittels linguistischer und stilistischer Features*, 2005.

A

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

Datum

Unterschrift der Kandidatin / des Kandidaten