
TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

# DIPLOMARBEIT

## Automatisiertes Verständnis von formularbasierten Webseiten durch statistische Klassifikation der Seiten und Elemente

Ausgeführt am Institut für

Informationssysteme

der Technischen Universität Wien

unter der Anleitung von

Prof. Dr. Reinhard Pichler

durch

**Andreas Mager**

Körmenderstraße 5

A-8280 Fürstenfeld

Wien, am 1. September 2008

# Danksagung

An dieser Stelle sei meinen Betreuern Mag. Dr. Robert Baumgartner und Prof. Dr. Reinhard Pichler gedankt, die mir diese Arbeit ermöglicht haben. Beide gaben mir zahlreiche wertvolle Ratschläge. Mag. Dr. Robert Baumgartner half mir bei der Auswahl der Literatur und unterstützte mich bei der Durchführung des Projektes. Spezieller Dank gebührt Max Goebel für die Hilfe beim Verständnis der SVM. Danken möchte ich auch Kathrin Scheidle und Kathrin Umrath für das geduldige Korrekturlesen. Weiters möchte ich noch meinen Eltern danken, die mir dieses Studium ermöglicht haben.

# Inhaltsverzeichnis

# 1. Einführung

Betrachtet man die Repräsentation des Wissens, spiegelt das World Wide Web, so wie es heute existiert, sehr deutlich gesellschaftliche Eigenschaften der Personen wieder, die es erschaffen haben und es warten. Es ist nicht möglich eine bestimmte Frage an "die Menschheit" zu stellen. Genausowenig ist es möglich eine bestimmte Frage an "das Internet" zu stellen. Es ist notwendig zu wissen, welche Menschen oder welche Gruppe von Menschen beziehungsweise welche Gruppe von Webseiten man fragen muss, um eine gewünschte Antwort zu erhalten. Dazu kommen noch die Sprache und die Form, in der die Fragen gestellt werden müssen, um vom Gegenüber verstanden zu werden. Auch unterscheiden sich die Antworten der Befragten in diesen Aspekten.

Suchmaschinen, wie zum Beispiel Yahoo! Search oder Google, indizieren das über Links erreichbare Web und ermöglichen damit eine Suche nach bestimmten Worten. Aktuelle Suchmaschinen sind jedoch nicht in der Lage, die Bedeutung von Webseiten zu verstehen. Weiters bleiben ihnen Inhalte verborgen, welche nur über die gezielte Eingabe in Abfrageformularen erreichbar sind.

Ein semantisches Web[BLHL01], das heißt ein Web, in dem alle Informationen strukturiert und annotiert verfügbar sind, wäre in der Lage, Fragen, wie sie im ersten Absatz besprochen wurden zu beantworten. Dieses Ziel liegt jedoch noch in weiter Ferne. Die ersten Schritte in diese Richtung sind das automatische Verstehen und Verwenden von Abfragemöglichkeiten des unstrukturierten Webs, um die daraus zugänglich gewordenen Daten in eine strukturierte Form zu bringen. Man konzentriert sich dabei anfangs auf einige wenige Domänen.

In dieser Arbeit werden die automatische Erkennung und das Verständnis von solchen domänenspezifischen Abfrageformularen behandelt. Die dabei verwendete Methode der Klassifikation wurde, wie in den folgenden Absätzen erklärt wird, durch ihre erfolgreiche Verwendung in der Spam-Bekämpfung inspiriert.

Die aktuelle Spam-Problematik hat erstaunlich effiziente Erkennungsmethoden hervorgebracht. Vor allem das Bayes-Modul des Spamassassin-Systems fällt durch seine sehr hohe Treffsicher-

heit auf. Es ist ohne weiteres möglich, ein stark frequentiertes E-Mail-Konto derart zu filtern, dass pro Jahr nur wenige unerwünschte Nachrichten den Empfänger erreichen und gleichzeitig False Positives nahezu ausgeschlossen sind.

Die Idee HTML-Seiten mittels eines Bayes'schen Klassifikators zu klassifizieren entstand während der ersten Besprechung mit Dr. rer. nat. Robert Baumgartner über den Inhalt der Arbeit.

Aus Gründen der besseren Lesbarkeit wird in der vorliegenden Arbeit an den meisten Stellen darauf verzichtet, jeweils die männliche und weibliche Form zu verwenden. Selbstverständlich werden, wenn nicht explizit anders angeführt, alle Menschen gleichermaßen angesprochen.

## 1.1. Ziele

Das Projekt "Metamorph" wird derzeit am Institut für Informationssysteme, Arbeitsgruppe für Datenbanken und Artificial Intelligence, der Technischen Universität Wien in Zusammenarbeit mit dem Spin-Off Lixto Software GmbH. entwickelt.

Mit Metamorph wird versucht Domänen semantisch zu modellieren, um dann konkrete Abfragesysteme beziehungsweise deren Bedienelemente darauf abzubilden. Die dabei verwendeten Techniken und Methoden reichen von der manuellen Annotation der Abfragesysteme bis zur vollautomatischen Erkennung. Bei der manuellen Annotation erhält der Benutzer eine Ansicht des Abfragesystems wie in einem normalen Browser und weist den Eingabeelementen eine Bedeutung zu. Es geht dabei auch um das Verständnis der Zusammenhänge der Eingabefelder und der Möglichkeiten, welche ein Benutzer eines Abfragesystems hat.

Dadurch wird es möglich Anfragen wie zum Beispiel 'Welche Flüge gibt es nächste Woche dienstags von Wien nach Dublin?' an das System zu stellen.

Ziel dieser Arbeit ist es herauszufinden, ob es möglich ist, eine automatische Annotation mit unterstützter maschineller Lernalgorithmen, wie etwa der Bayes'schen Klassifizierer, zu realisieren. Es sollen dabei unterschiedliche Klassifizierungsalgorithmen sowie Methoden, die Seiten vor dem Klassifizieren zu verändern um ein besseres Ergebnis zu erhalten, miteinander verglichen werden.

## 1.2. Motivation

Um eine Webseite automatisiert zu verstehen, sind mehrere Schritte notwendig. Erst muss eine solche Webseite gefunden und als geeignet identifiziert werden. Zum Beispiel werden mittels eines Spiders Seiten im Web durchsucht und jede einzelne wird durch einen entsprechend trainierten Klassifikator klassifiziert. Ist das Ergebnis positiv, das heißt ist die gefundene Seite in der Klasse der Seiten, die für die gestellte Aufgabe relevant sind, so kann sie weiterverarbeitet werden.

Um die gestellte Aufgabe, eine bestimmte Klasse von Informationen (zum Beispiel Flüge, bestimmte Produkte, Personen oder auch Dienstleistungen) aus beliebigen geeigneten Datenbanken im Web zu extrahieren, erfüllen zu können, muss das System die gefundene Seite automatisch verstehen. Das heißt, die für eine bestimmte Informationsklasse notwendigen und optimalen Eigenschaften müssen den Eingabeformularen der gefundenen Seite zugeordnet werden.

Dies geschieht, indem die Formularfelder aus der Seite extrahiert und danach selbst klassifiziert werden. Hierbei kann es hilfreich sein, wenn die Rohdaten (HTML-Code) vor dem Klassifizieren aufbereitet werden.

# 2. Theorie

In diesem Kapitel werden die theoretischen Grundlagen beschrieben. Es enthält einen Überblick über andere Arbeiten in den Bereichen “Webseitenklassifikation”, “Metasearch” und “Wrapper”. Weiters werden im Abschnitt 2.2 einige Klassifizierungsalgorithmen beschrieben und anhand einiger Beispiele durchgerechnet.

## 2.1. Verwandte Bereiche

Diese Arbeit ist eng verwandt mit einigen anderen, gut abgrenzbaren Forschungsbereichen der Informatik. Wie diese Bereiche mit dieser Arbeit zusammenhängen, wird in diesem Abschnitt beschrieben.

### 2.1.1. Klassifikation von Dokumenten

Es gibt bereits viele Lösungen und Methoden, Webseiten zu klassifizieren. Der am Institut für Informatik der Ludwig-Maximilians-Universität in München entwickelte Web-Crawler “Ariadne” klassifiziert zum Beispiel Hyperlinks, um das World Wide Web nach Seiten zu einem bestimmten Thema zu durchsuchen. Der Crawler findet so mehr zum Thema passende Seiten als ein generischer Crawler in derselben Zeit, indem für jeden Link entschieden wird, ob es sich lohnt, ihn zu verfolgen oder nicht.

Die Kernfunktion wird in [EG00] wie folgt erklärt:

> Die zentrale Phase des eigentlichen Crawls wiederholt für die jeweils aktuelle Webseite folgende Schritte:
>
> - Klassifikation des Textes der Webseite: Aus der geladenen Webseite werden wie in der Vorverarbeitung der reine Text extrahiert und die Häufigkeiten der

Features gezählt. Mit Hilfe eines einfachen Textklassifikators wird nun die Wahrscheinlichkeit bestimmt, dass die aktuelle Seite relevant ist.

- Klassifikation der enthaltenen Hyperlinks: Ein zweiter Klassifikator nutzt den Text der aktuellen Webseite sowie die Ankertexte und die URLs, um die in der aktuellen Seite enthaltenen Hyperlinks für die Zwecke des Crawl zu bewerten. Alle diese Hyperlinks werden mit ihrer Bewertung in eine sortierte Liste der offenen Hyperlinks eingefügt.

- Verfolgen des Links mit der besten Bewertung: Nach Abarbeiten der aktuellen Webseite wird als nächstes der beste offene Hyperlink, das heißt der Hyperlink mit der global höchsten Bewertung verfolgt. Die Webseiten werden also nur nach Prioritäten geordnet. Es werden keine Seiten explizit von der Suche ausgeschlossen.

Ein ähnliches Verfahren wird auch in [CDI98] beschrieben. Hier ist das Ziel jedoch kein Crawler, sondern nur die Klassifikation einzelner Seiten. Hierbei konnte die Genauigkeit gegenüber reinen Textklassifikatoren um bis zu 70% angehoben werden.

Da das Klassifizieren der ganzen Seiten für das verwendete Testgebiet der Flugsuche von Anfang an gut funktionierte, verzichtete ich auf die Einbeziehung der Hyperlinks. Diese kann aber noch problemlos integriert werden.

Verbindet man diese Klassifikationsmethoden mit Algorithmen, welche sich selbsttätig durch eine Menge von Dokumenten arbeiten, etwa einen Spider[1], so kann man von Information Retrieval sprechen.

### 2.1.2. Information Retrieval

Als Information Retrieval (IR) bezeichnet man das Auffinden und Herausholen von relevanten Dokumenten aus einer großen Menge von Dokumenten. Der Inhalt der Dokumente wird dabei nur zur Klassifikation analysiert Es werden noch keine Informationen im Dokument ausgewertet und auch keine Fakten daraus gewonnen, diese Tätigkeiten fallen in den Bereich der Information Extraction (Abschnitt 2.1.4).

[1] Mechanismus, welcher sich selbsttätig durch verlinkte Dokumente im Web arbeitet

Vannevar Bush [Bus45] beschrieb bereits 1945 eine Methode, welche Informationen basierend auf Assoziationen und Kontextverknüpfungen effizient finden konnte. Heutige Internetsuchmaschinen sind wohl die bekanntesten Anwendungsgebiete im Bereich des Information Retrieval. In dieser Arbeit ist der in Abschnitt 4.2.2 beschriebene Teil des Klassifizierens einer ganzen Seite ein IR-System, wenn er mit einem Spider verwendet wird. Siehe dazu Abschnitt 7.1.

### 2.1.3. Metasearch

Beim Metasearch, oder auch Metasuche, handelt es sich um ein System, welches sich anderer Such- oder Abfragesysteme bedient, um an geeignete Ergebnisse zu kommen. Eine Anfrage an ein Metasearch-System wird an ausgewählte Abfragesysteme weitergeleitet und deren Ergebnis wird gesammelt dem Benutzer übergeben. Dadurch dauern Abfragen auf Metasuchmaschinen oft länger als Abfragen auf direkten Suchmaschinen. Da solche Abfragesysteme in der Regel für die Kommunikation mit Menschen entwickelt werden, das heißt Eingabe über Maus und Tastatur und Ausgabe grafisch auf einem Bildschirm, ist es für Maschinen (Programme) nicht immer einfach, diese zu verwenden.

So muss für jedes verwendete Abfragesystem ein so genannter Wrapper, siehe Abschnitt 2.1.4, erzeugt werden. Dieser Wrapper muss auch bei Änderungen am Abfragesystem angepasst werden. Das Erstellen und Ändern eines Wrappers ist eine Tätigkeit, die für gewöhnlich von Menschen erledigt wird und in der Vergangenheit nur schwer automatisierbar war. In [ZMW+05] wird eine Methode besprochen, mit welcher Wrapper für Suchmaschinenergebnisse automatisch erstellt werden können.

Ein weiteres Problem in Verbindung mit Metasearch-Systemen ist das Auffinden geeigneter Abfragesysteme an sich. Auch hier müssen Menschen mittels Internet-Suchmaschinen nach passenden Seiten suchen und entscheiden, ob diese für eine bestimmte Aufgabe geeignet sind. Auf diese Problematik wird in Abschnitt 2.1.1 genauer eingegangen.

#### Konkrete Implementierungen und Projekte

Eines der größten Projekte in diesem Bereich ist die gemeinschaftliche Forschungsarbeit von Prof. Weiyi Meng, Prof. Clement Yu, Prof. Xiaofeng Meng und anderen ([HML+07] und [WYDM04]). Ziel dieses Forschungsprojektes ist ein integrierter Zugang zu Web-Datenbanken.

Die einzelnen Aufgaben, wie das Auffinden von geeigneten Datenbanken, sowie deren Gruppierung nach deren Anwendungsbereich, das Abbilden der Suchanfragen an das integrierte System an die passenden Web-Datenbanken und das Zusammenfassen der erhaltenen Resultate, werden dabei sehr stark automatisiert. Zwei von vielen weiteren Arbeiten zu diesem Projekt sind [MWLK03] und [HMYW04].

Weitere Projekte zu diesem Thema sind in den Folgenden Absätzen angeführt.

**MetaCrawler - Parallel Web Search Service** war das erste derartige Projekt. Es wurde 1995 von Erik Selberg und Oren Etzioni im Rahmen eines Forschungsprojektes an der University of Washington in den USA entwickelt und existiert heute noch. MetaCrawler [SE95] leitet Anfragen an andere generische Suchmaschinen wie Google, Yahoo! Search, MSN Search und andere weiter.

**MetaMorph** wird an der TU-Wien am Institut für Informationssysteme 184/2 in der Abteilung für Datenbanken und Artificial Intelligence entwickelt. MetaMorph selbst ist keine Metasuchmaschine, sondern ein System, das spezialisierte Suchmaschinen erstellt, welche auch Inhalte finden, die in Datenbanken verborgen und daher für generische Suchmaschinen nur selten zugänglich sind.

### 2.1.4. Information Extraction

Die Information Extraction (IE) befasst sich anders als IR mit der Gewinnung von Fakten aus bestimmten Dokumenten. Die Entwicklung und deren Teilgebiete, etwa die Deep-Web-Navigation [BCL05], werden in [Eik99] besprochen.

Im Zuge der IE werden Wrapper erzeugt. Wrapper sind grundsätzlich Mechanismen, welche die Funktionalität eines Systems, welches diese über eine nicht standardisierte oder verwendbare Schnittstelle zur Verfügung stellt, mittels einer anderen, von dem Anwender verwendbaren Schnittstelle anbieten.

Ein Beispiel für nicht standardisierte Datensysteme sind webbasierende Shopsysteme. Der Inhalt sind Artikelbezeichnungen und die dazugehörenden Preise, eventuell auch noch Versand- oder auch Verfügbarkeitsinformationen. Diese Systeme können sehr unterschiedlich aussehen. Selten sind alle Artikel auf eine einfache Art zugänglich.

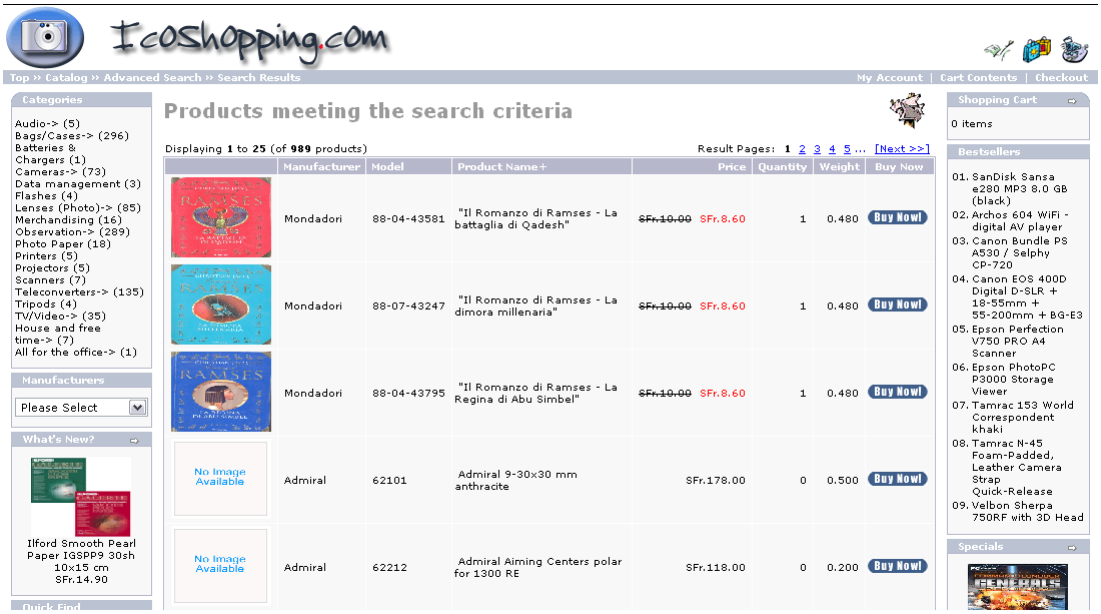


Abbildung 2.1.: Produktliste ([Fer07])

Man kann in vielen Volltextsuchen alle Einträge finden, indem man nach "%%%" sucht.

Ein Wrapper für einen Webshop würde aus diesem Shop eine von Maschinen lesbare Liste mit allen verwendbaren Informationen generieren, um diese Daten dann maschinell weiterverarbeiten zu können.

### Konkrete Implementierungen und Projekte

Anhand eines Onlineshops, siehe Abbildung 2.1, werden in diesem Abschnitt zwei Wrapper-Implementierungen verglichen, die eine sehr unterschiedliche Vorgehensweise verwenden.

**Lixto:** Lixto [BGH03] ist ein visueller Wrappergenerator [BFG01], das heißt ein Benutzer sieht die Webseite in Lixto genau so wie er sie im Browser sehen würde und kann gewünschte Elemente mit der Maus markieren. Der entsprechende Wrapper wird von Lixto automatisch generiert und kann danach, sofern notwendig, angepasst werden. Abbildung 2.2 zeigt die bereits automatisch gefundenen Produkte. Lixto ist auch für andere Dokumenttypen, wie etwa PDF[2], geeignet [HB05].

[2]Portable Document Format

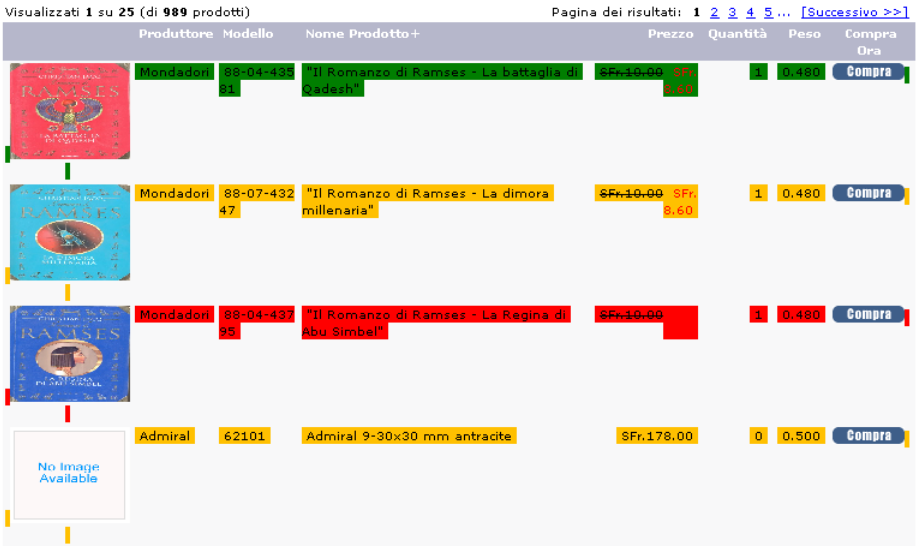


Abbildung 2.2.: Lixto findet Produkte

**Geizhals web-csv** Das von Geizhals[3] entwickelte web-csv ist ein spezialisierter Wrappergenerator. Das heißt, er wurde mit dem Ziel entwickelt, Daten aus csv-Listen[4] und einigen weit verbreiteten Webshoptypen zu extrahieren.

Die Verwendung erfolgt nicht wie bei Lixto visuell, sondern orientiert sich an den gefundenen Daten in tabellarischer Form. Bei einem Webshop wie im oben gezeigten Beispiel wird diese Tabelle durch einen Präprozessor gesteuert. Der Benutzer ordnet dann die Spaltennummern der Rohtabelle den entsprechenden gewünschten Feldern zu. Abbildung 2.3 zeigt die Konfiguration für das Beispiel.

Nach der Auswertung erhält man das fertige Ergebnis, wie in Abbildung 2.4 ersichtlich ist.

## 2.2. Klassifizierungsalgorithmen

In diesem Abschnitt werden die in der Arbeit verwendeten Klassifizierungsalgorithmen beschrieben.

Klassifizierungsalgorithmen arbeiten grundsätzlich mit Merkmalvektoren, die sich in einem Merkmalraum befinden. Beim Klassifizieren von Texten sind dies meist die relativen Häufig-

[3]Preisvergleich Internet Services AG., gegründet im Jahr 2000 von Dipl.-Ing. Marinos J. Yannikos
[4]Character Separated Values

| | |
|---|---|
| Preisfeld (Nr. oder Ausdruck): | 8<br>) |
| Produktbezeichnung (Nr. oder Ausdruck): | 7 |
| EAN-Code (Nr. oder Ausdruck): | |
| Herstellername (Nr. oder Ausdruck): | 4 |
| Herstellercode (Nr. oder Ausdruck): | 5 |
| Geizhals-ID (Vorsicht!) (Nr. oder Ausdruck): | |
| Eindeutige Händler-Artikelnr.(Nr. oder Ausdruck): | |
| Langtext / Beschreibung (Nr. oder Ausdruck): | |
| Deeplink (Nr. oder Ausdruck): | 1 |
| Lieferzeit (Nr. oder Ausdruck): | |
| Produktfoto-URL (Nr. oder Ausdruck): | |
| Exclude - Eintrag überspringen falls true (Nr. oder Ausdruck): | `$f[8] !~ /sfr/i` |

Abbildung 2.3.: Beispielkonfiguration für web-csv

| # | Interne Bezeichnung | Preis | Deeplink | Preisstand | Lieferzeit |
|---|---|---|---|---|---|
| 0 | Mondadori 88-04-43581\|\|Il Romanzo di Ramses - La battaglia di Qadesh | .8.60 | redir.cgi?h=dummy-haendler-de&loc=http:/ /www.icoshopping.com/catalog /product_info.php?products_id=81 | | |
| 1 | Mondadori 88-07-43247\|\|Il Romanzo di Ramses - La dimora millenaria | .8.60 | redir.cgi?h=dummy-haendler-de&loc=http:/ /www.icoshopping.com/catalog /product_info.php?products_id=83 | | |
| 2 | Mondadori 88-04-43795\|\|Il Romanzo di Ramses - La Regina di Abu Simbel | .8.60 | redir.cgi?h=dummy-haendler-de&loc=http:/ /www.icoshopping.com/catalog /product_info.php?products_id=82 | | |
| 3 | Admiral 62101\|\|Admiral 9-30x30 mm Anthrazit | .178.00 | redir.cgi?h=dummy-haendler-de&loc=http:/ /www.icoshopping.com/catalog /product_info.php?products_id=321 | | |

Abbildung 2.4.: Ergebnis von web-csv

keiten einzelner Wörter, wobei aber auch die Auftrittswahrscheinlichkeiten der Buchstaben oder Silben verwendet werden können.

Jedes in den Trainingsdokumenten vorkommende Wort entspricht dann einer Dimension des Merkmalraumes. Ein Dokument entspricht dann einem Merkmalvektor, dessen Koordinaten die einzelnen relativen Häufigkeiten der Wörter des Dokumentes in diesem Dokument sind. Wörter, die nicht in den Trainingsdokumenten vorkommen, sind nicht Teil des Merkmalvektors.

**Beispiel:** Betrachtet man den kurzen Satz “to be or not to be”, so lässt sich dieser in den Merkmalvektor $< 2,2,1,1 >$ überführen. Die Koordinaten entsprechen dabei: $< to, be, or, not >$. Um über mehrere Dokumente vergleichbare Werte zu erhalten, werden die Häufigkeiten relativiert. Dies kann zum Beispiel geschehen, indem man den Betrag des Vektors eines jeden Dokumentes auf einen gemeinsamen Wert, etwa 1 skaliert. Man berechnet mit dem euklidischen Abstandsbegriff den Betrag des Vektors: $\sqrt{to^2 + be^2 + or^2 + not^2}$ in Zahlen $\sqrt{2^2 + 2^2 + 1^2 + 1^2} = \sqrt{4+4+1+1} = 3,16$. Dividiert man nun alle Koordinaten des Vektors durch $\sqrt{10}$, so erhält man den normalisierten Vektor.

Ein Klassifizierungsalgorithmus ist dann eine Methode, den Merkmalraum, in dem sich diese Vektoren befinden, derart zu unterteilen, dass eine Grenze zwischen den relevanten und nicht relevanten Dokumenten beziehungsweise Punkten im Raum entsteht. Idealerweise liegen alle Trainingsdokumente auf der richtigen Seite dieser Grenze. Dies ist aber nicht bei jedem Klassifikator der Fall. Das Ziehen dieser Grenze passiert beim Trainieren des Klassifikators.

Soll ein neues Dokument klassifiziert werden, so wird es als Punkt in diesem Raum dargestellt und dann aufgrund seiner Lage bezüglich der Grenze klassifiziert.

### 2.2.1. KNN - k nächste Nachbarn

Dieser Klassifikator benutzt das so genannte “faule Lernen”. Das heißt, er speichert die Vektoren beim Training nur ab und versucht nicht schon beim Training eine Grenze zu ziehen.

Soll nun ein neues Dokument klassifiziert werden, so wird es zunächst im Merkmalraum abgebildet. Von diesem Punkt ausgehend lässt man eine Kugelumgebung wachsen bis $k$ benachbarte Dokumente in dieser Umgebung liegen. Die Entscheidung, ob das neue Dokument relevant ist oder nicht, richtet sich nach der Mehrheit der $k$ benachbarten Dokumente.

Die Wahl der Konstanten $k$ hat einen großen Einfluss auf das Ergebnis. Wird sie zu klein gewählt, so wirkt sich ein eventuell vorhandenes Rauschen in den Trainingsdaten verstärkt aus und verschlechtert das Ergebnis. Wird sie zu groß gewählt, so besteht die Gefahr, dass auch Dokumente mit einbezogen werden, die sehr weit entfernt sind.

Man kann diesem Umstand begegnen, indem man den Nachbarn in der näheren Umgebung mehr Bedeutung zumisst als jenen, welche weiter entfernt sind.

### 2.2.2. SVM - Support Vector Machine

Genau wie der oben beschriebene KNN-Algorithmus stellt auch die SVM die Daten als Vektoren dar. Sie klassifiziert neue Daten aufgrund ihrer Lage zu einer Entscheidungsebene. Im folgenden Abschnitt wird erklärt, wie diese Ebene gefunden werden kann.

#### Beschreibung einer einfachen SVM

Die mathematische Beschreibung der SVM wurde weitgehend aus [Bis06][5] übernommen und für diese Arbeit vereinfacht und angepasst.

Die Testdaten werden im Merkmalraum durch eine Ebene getrennt. Eine solche Ebene wird mit Gleichung (2.1) beschrieben, indem man $y(\mathbf{x}) = 0$ setzt. Das $\mathbf{w}$ in der Gleichung ist der Gewichtsvektor. Er gibt die Orientierung der Ebene im Raum an. Der Gewichtsvektor steht senkrecht zur Ebene. Das $\mathbf{x}$ in der Gleichung ist ein beliebiger Punkt. Liegt der Punkt in der Ebene, so ist $y(\mathbf{x}) = 0$. Liegt der Punkt in der Richtung des Gewichtsvektors über der Ebene, so ist $y(\mathbf{x}) > 0$, liegt er darunter, so ist $y(\mathbf{x}) < 0$. Das $b$ in der Gleichung ist die Basis der Ebene, es beschreibt die Position der Ebene relativ zum Nullpunkt. Der Skalar $b$ und der Vektor $\mathbf{w}$ beziehungsweise die Länge von $\mathbf{w}$: $\|\mathbf{w}\|$ sind voneinander abhängig, so dass der Abstand der Ebene vom Nullpunkt durch $\frac{-b}{\|\mathbf{w}\|}$ beschrieben wird.

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \tag{2.1}$$

In Abbildung 2.5 sind die oben geschilderten Zusammenhänge illustriert. Der Punkt $\mathbf{x}_{\perp}$ ist die Normalprojektion des Punktes $\mathbf{x}$ auf die Entscheidungsebene. Der Normalabstand $r$ des Punktes $\mathbf{x}$ zur Ebene wird mit $r = \frac{y(\mathbf{x})}{\|\mathbf{w}\|}$ ausgedrückt.

[5] Seiten 181f, 326ff und 293f

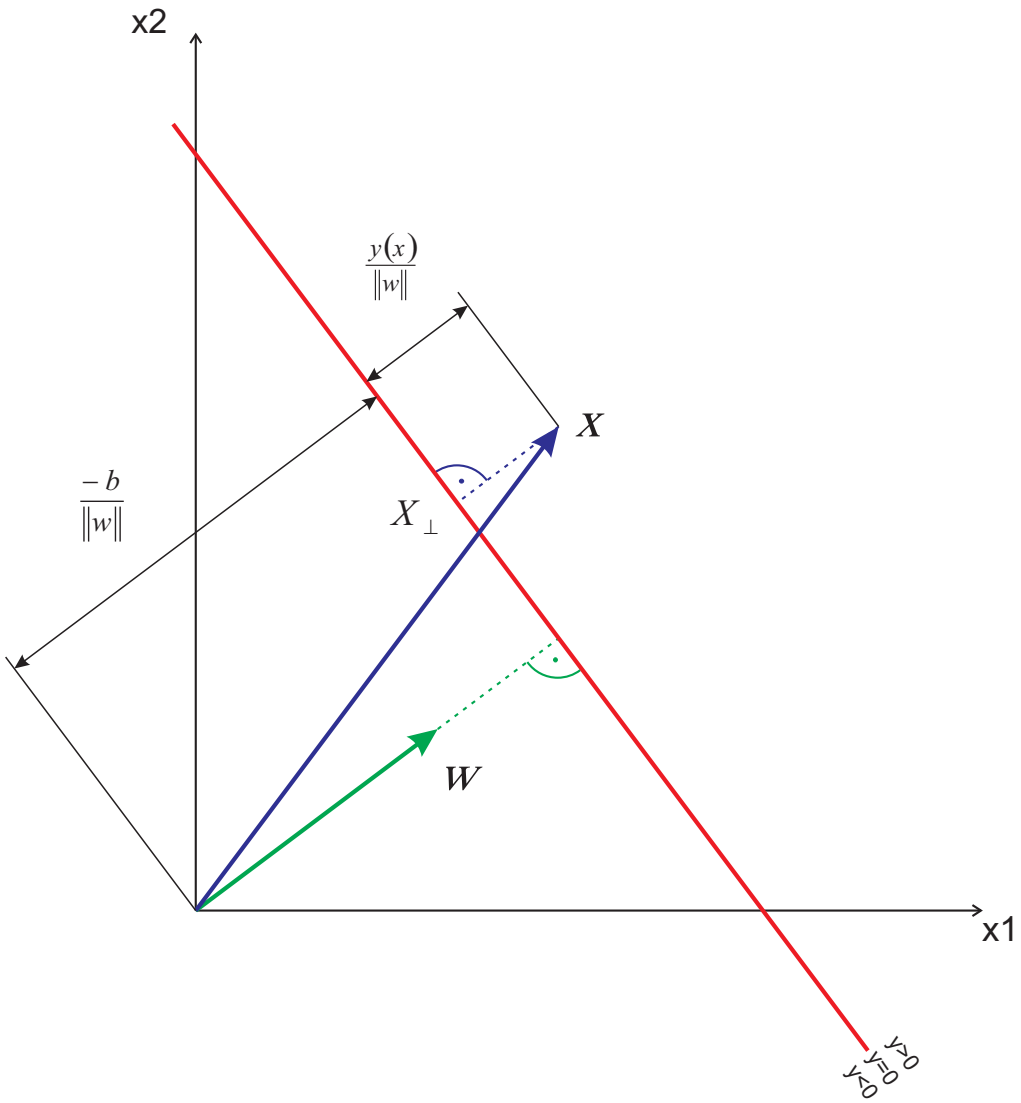


Abbildung 2.5.: Lineare Entscheidungsfunktion
Diese Abbildung zeigt eine Entscheidungsebene im zweidimensionalen Raum, bestimmt durch den Gewichtsvektor $\mathbf{w}$ und den Abstand zum Ursprung $-b/\|\mathbf{w}\|$. Weiters ist ein Punkt $x$ dargestellt und dessen Normalabstand zur Ebene $y(\mathbf{x})/\|\mathbf{w}\|$.

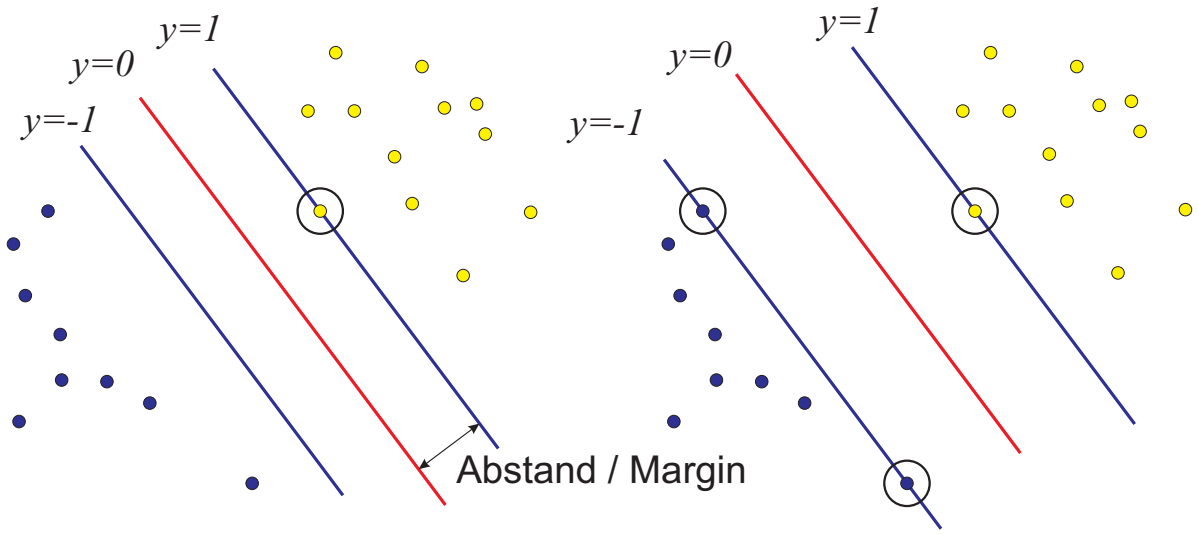


Abbildung 2.6.: Entscheidungsebene mit einem Margin
Im rechten Bild wurde der Margin der Entscheidungsebene bereits maximiert.

Die Trainingsdaten werden auch bei der SVM als Punkte im Merkmalraum interpretiert. Beim Trainieren wird versucht, eine Ebene derart zwischen den Punkten einzuziehen, dass die Punkte von dieser Ebene getrennt werden. Bei der SVM wird die Lage dieser Ebene durch den gleichen, möglichst großen Normalabstand zwischen der Ebene und möglichst wenigen Punkten, die der Ebene am nächsten liegen bestimmt. Man kann sagen, die Ebene stützt sich an diesen Support-Vektoren. Abbildung 2.6 verdeutlicht die Bestimmung der Entscheidungsebene.

Betrachtet man nun einen Satz aus $N$ Trainingsdaten oder auch Trainingsvektoren $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ mit entsprechenden Bezeichnungen oder Zielwerten $t_1, t_2, ..., t_N$ mit $t_n \in \{-1, 1\}$, so ist $y(\mathbf{x}) > 0$ für Punkte mit $t_n = +1$ und $y(\mathbf{x}) < 0$ für Punkte mit $t_n = -1$. Ein neuer Datenpunkt $\mathbf{x}$ wird anhand des Vorzeichens von $y(\mathbf{x})$ aus Gleichung (2.1) klassifiziert. Fürs Erste genügt es, wenn wir annehmen, die Daten seien linear trennbar. Wie man dem allgemeinen Fall mit nicht linear trennbaren Daten beikommt, wird in Abschnitt 2.2.2 besprochen.

Die Entscheidungsebene muss folgenden Ansprüchen genügen:

- Alle Trainingsdaten müssen durch die Ebene richtig klassifiziert werden.
- Der Normalabstand der Ebene zu den nächstgelegenen Punkten muss die größte mögliche Länge haben.

Der erste Anspruch lässt sich mathematisch mit $t_n y(\mathbf{x}) > 0$ für alle $n$ ausdrücken. In Verbindung mit Gleichung (2.1) ist auch der Normalabstand der Punkte $\mathbf{x}_n$ zur Ebene bekannt.

$$\frac{t_n y(\mathbf{x}_n)}{\|\mathbf{w}\|} = \frac{t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}+b)}{\|\mathbf{w}\|} \tag{2.2}$$

Um nun die passenden Parameter ($\mathbf{w}$ und $b$) für die Ebene zu erhalten, maximieren wir das Ergebnis der Gleichung (2.2) mittels der Veränderung der oben genannten Parameter, wobei gleichzeitig eine Minimierung der Anzahl der dabei verwendeten Punkte $\mathbf{x}_n$ angestrebt wird.

$$\arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \cdot \min_n \left[ t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \right] \right\} \tag{2.3}$$

Der Faktor $\frac{1}{\|\mathbf{w}\|}$ wurde aus der Minimierung für $n$ entfernt, da er nicht von $n$ abhängt. Da die Lösung dieses Optimierungsproblems sehr komplex werden würde, wird das Problem in ein gleichwertiges aber einfacher zu lösendes Problem umgewandelt. Die beiden Ebenenparameter $\mathbf{w}$ und $b$ sind derart voneinander abhängig, dass eine Skalierung ($\mathbf{w} \to \kappa\mathbf{w}$ und $b \to \kappa b$) keinen Einfluss auf die Gültigkeit von (2.2) für den Normalabstand eines Punktes zur Ebene hat. Durch diese Freiheit der Skalierung können wir festsetzen, dass

$$t_n \left( \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \right) = 1 \tag{2.4}$$

für den Punkt, welcher der Ebene am nächsten liegt, gilt. Dadurch erfüllen alle Datenpunkte die Bedingung

$$t_n \left( \mathbf{w}^{\mathrm{T}}\mathbf{x} + b \right) \geq 1, \qquad n = 1, \ldots, N. \tag{2.5}$$

Für jene Datenpunkte, für die der Abstand gleich eins ist, wird die Bedingung als “aktiv” bezeichnet. Ist der Abstand größer als eins, so ist diese Bedingung “passiv”. Da es stehts mindestens einen Datenpunkt mit minimalem Abstand zu Ebene gibt, ist die Anzahl der aktiven Bedingungen laut oben genannter Definition mindestens eins. Ist der Margin erst maximiert, so gibt es mindestens zwei aktive Bedingungen. Das Optimierungsproblem reduziert sich nun auf die Maximierung von $\|\mathbf{w}\|^{-1}$ oder anders ausgedrückt auf die Minimierung von $\|\mathbf{w}\|^2$. Daher müssen wir nun das vereinfachte Optimierungsproblem

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.6}$$

mit den Bedingungen aus (2.5) lösen. Der Faktor $1/2$ in (2.6) wurde eingeführt um die weiteren Schritte zu erleichtern. Der Basisparameter $b$ ist nicht, wie es scheint, aus dem Optimierungsproblem verschwunden, sondern wird implizit über die Bedingungen ermittelt.

Um dieses bedingte Optimierungsproblem zu lösen, fügen wir unbestimmte Lagrange-Multiplikatoren[6]. $a_n \geq 0$ ein — einen für jede Bedingung aus (2.5). Bei der daraus resultierenden Lagrange-Funktion

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \left\{ t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) - 1 \right\} \tag{2.7}$$

ist $\mathbf{a} = (a_1, \ldots, a_N)^{\mathrm{T}}$. Leitet man nun (2.7) nach $\mathbf{w}$ und $b$ ab und setzt diese gleich null, so erhält man die Gleichungen

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \mathbf{x}_n \tag{2.8}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{2.9}$$

Werden nun unter Zuhilfenahme dieser Gleichungen $\mathbf{w}$ und $b$ aus (2.7) entfernt, so erhält man die Dual-Representation[7] des Maximum-Margin-Problems als Maximierung von

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m \mathbf{x}_n^{\mathrm{T}} \mathbf{x}_m \tag{2.10}$$

in Bezug auf $a$ unter der Berücksichtigung der Bedingungen

$$a_n \geq 0 \qquad n = 1, \ldots, N \tag{2.11}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{2.12}$$

Das ursprüngliche Optimierungsproblem aus (2.6) ist abhängig von der Anzahl der Dimensionen des Gewichtsvektors $\mathbf{w}$ und des Merkmalraumes. In der Praxis ist die Dimensionalität deutlich höher als die Anzahl der Trainingsdaten. Der Aufwand eines solchen Optimierungsproblems liegt bei $O(N^3)$. Die Dual-Representation dieses Problems hat denselben Aufwand, ist aber nur von der Anzahl der Trainingsdaten $N$ abhängig und verringert dadurch den Aufwand für die Optimierung. Wie im nächsten Abschnitt beschrieben wird, ist es oft sinnvoll, die Datenpunkte in einen Raum höherer Dimensionalität hochzurechnen. Statt einem $\mathbf{x}$ verwendet

[6] Siehe Appendix E in [Bis06]
[7] [Bis06], Seite 293

man ein $\Phi(\mathbf{x})$. Weiters wird auch das $\mathbf{x}_n^{\mathrm{T}}\mathbf{x}_m$ aus (2.10) durch eine Kernelfunktion $k(\mathbf{x}_n, \mathbf{x}_m)$ ersetzt, welche positiv definit sein muss und ein Maß für die Ähnlichkeit zweier Vektoren ist. In dem hier beschriebenen, sehr einfachen Fall wurden $\Phi(\mathbf{x}) = \mathbf{x}$ und $k(\mathbf{x}_n, \mathbf{x}_m) = \Phi(\mathbf{x}_n)^{\mathrm{T}}\Phi(\mathbf{x}_m)$ gewählt. Die allgemeine Form von (2.10) ist dann

$$\tilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m). \tag{2.13}$$

Um nun einen neuen Datenpunkt zu klassifizieren verwendet man (2.1) und ersetzt $\mathbf{w}$ unter Zuhilfenahme von (2.8). Statt $\mathbf{x}_n$ verwendet man auch hier $\Phi(\mathbf{x}_n)$ und erhält

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \tag{2.14}$$

In [Bis06] wird gezeigt, dass bei dieser Form der Optimierung die Eigenschaften

$$a_n \geq 0 \tag{2.15}$$

$$t_n y(\mathbf{x}) - 1 \geq 0 \tag{2.16}$$

$$a_n\{t_n y(\mathbf{x}) - 1\} = 0 \tag{2.17}$$

gelten. Daher gilt für jeden Datenpunkt entweder $a_n = 0$ oder $t_n y(\mathbf{x}) = 1$. Jeder Trainingsdatenpunkt, für den $a_n = 0$ gilt, scheint nicht in der Summe aus (2.14) auf und ist für die Klassifikation nicht ausschlaggebend. Für alle übrigen Punkte gilt $t_n y(\mathbf{x}) = 1$. Diese Punkte heißen Support-Vektoren. Diese Eigenschaft der SVM bewirkt, dass man nach dem Trainieren nur mehr die Support-Vektoren benötigt. Alle übrigen Vektoren haben keinen Einfluss mehr auf die Klassifikation und können gelöscht werden.

Aus (2.14) ist noch die Basis $b$ unbekannt. Da für Support-Vektoren $t_n y(\mathbf{x}) = 1$ gilt, erhält man in Verbindung mit (2.14)

$$t_n\left(\sum_{m\in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m) + b\right) = 1, \tag{2.18}$$

wobei $S$ die Menge der Indizes der Support-Vektoren ist. Es würde genügen, wenn man diese Gleichung für einen beliebigen Support-Vektor $\mathbf{x}_n$ löst. Man erhält jedoch eine numerisch stabilere Lösung, indem man das $b$ für alle Support-Vektoren berechnet und daraus den Mittelwert bildet.

$$b = \frac{1}{N_S}\sum_{n\in S}\left(t_n - \sum_{m\in S} a_m t_m k(\mathbf{x}_n, \mathbf{x}_m)\right) \tag{2.19}$$

$N_S$ in (2.19) ist die Anzahl der Support-Vektoren.

#### SVM in der Praxis

Im vorangegangenen Abschnitt wurde angenommen, dass die Trainingsdaten linear separierbar sind. Im allgemeinen Fall ist dies jedoch selten möglich. Eine Möglichkeit, die Trainingsdaten doch noch sauber trennen zu können ist das im letzten Abschnitt besprochene Abbilden der Vektoren in einen Raum mit einer höheren Dimensionalität als der Ursprungsraum. Man überspringt jedoch meist den Schritt, ein geeignetes $\Phi(\mathbf{x})$ zu suchen, und sucht stattdessen direkt nach einer passenden Kernfunktion $k(\mathbf{x}_n, \mathbf{x}_m)$. Ein Beispiel einer solchen Kernfunktion, auch bekannt als Kernel, ist

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left(-\|\mathbf{x}_n - \mathbf{x}_m\|^2 / 2\alpha^2\right). \tag{2.20}$$

Weitere Beispiele von Kernels und Informationen, wie diese erstellt werden, finden sich in der Literatur, etwa in [Bis06] Kapitel 6.2 oder in [DHS01].

Eine weitere Möglichkeit einer nicht linear separierbaren Menge von Punkten entgegenzutreten, ist die Lockerung der Bedingung, dass alle Punkte auf der ihnen zugedachten Seite der Entscheidungsebene liegen müssen. Um das zu erreichen ersetzt man die Bedingung (2.5) durch eine entsprechende Fehlerfunktion, welche falsch zugeordnete Punkte bestraft. Diese Toleranz gegenüber einigen Fehlern beugt dem Problem der Überanpassung vor. Eine solche Fehlerfunktion ist beispielsweise der quadratische Abstand eines Punktes von der Entscheidungsebene. In der Praxis werden beide Methoden kombiniert.

### 2.2.3. Bayes

In diesem Abschnitt wird der Bayes'sche Klassifikator anhand von einfachen Beispielen beschrieben. Eines seiner Hauptanwendungsgebiete in der Informatik ist das Erkennen von Spam.

#### Theorie: Bayes'sche Formel

Die Klassifikation ist die Berechnung der Wahrscheinlichkeit eines Ereignisses A unter der Bedingung, dass ein anderes Ereignis B bereits eingetreten ist. Das Ereignis A ist hierbei die Zugehörigkeit zu einer bestimmten Kategorie. Das Ereignis B ist dann das Auftreten des zu

klassifizierenden Objektes oder einer Beobachtung. Diese bedingte Wahrscheinlichkeit wird berechnet durch die Wahrscheinlichkeit, dass beide Ereignisse auftreten, geteilt durch die Wahrscheinlichkeit des bereits eingetretenen Ereignisses.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{2.21}$$

Die Wahrscheinlichkeit, dass beide Ereignisse gemeinsam auftreten $P(A \cap B)$ ergibt sich aus $P(A) \cdot P(B)$, wenn die beiden Ereignisse stochastisch voneinander unabhängig sind. Daraus ergibt sich aus der Gleichung 2.21:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

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

$$P(A|B) = P(A)$$

Sind also die beiden Ereignisse A und B stochastisch voneinander unabhängig, so ändert das Auftreten des Ereignisses B nichts an der Wahrscheinlichkeit, dass das Ereignis A auftritt.

Für die Klassifizierung wird jedoch angenommen, dass zwischen den beiden Ereignissen, das heißt dem Auftreten eines Objektes beziehungsweise einer Beobachtung und deren Zugehörigkeit zu einer Kategorie, ein stochastischer Zusammenhang besteht. In diesem Fall ist die Wahrscheinlichkeit, dass beide Ereignisse eintreten $P(A \cap B)$ definiert durch: $P(A \cap B) = P(A|B) \cdot P(B) = P(B|A) \cdot P(A)$. Setzt man das in die Gleichung 2.21 ein, so erhält man:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

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

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

Und schließlich das Bayes'sche Theorem:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{2.22}$$

Man kann damit, wenn man $P(A)$, $P(B)$ und $P(B|A)$ kennt, $P(A|B)$ errechnen und somit Ursachenforschung betreiben. Wie diese Gleichung in der Praxis funktioniert, zeigen die folgenden Beispiele.

**Beispiel:** Aus [HKG01], Text leicht verändert. Eine 55-jährige Frau ohne einschlägige Symptome ist dem Rat ihres Arztes gefolgt, im Rahmen der Brustkrebsfrüherkennung jedes Jahr eine Mammographie durchführen zu lassen. Das Ergebnis der Untersuchung ist positiv. Schockiert von diesem Ergebnis möchte sie die genaue Wahrscheinlichkeit wissen, mit der sie tatsächlich Brustkrebs hat.

Die (a-priori) Wahrscheinlichkeit $P(K)$, dass eine symptomfreie Frau mit 55 Jahren Brustkrebs hat, liegt bei 0.6%. Hat eine Frau Brustkrebs, so ist der Test in 94% der Fälle positiv $P(D|K)$. Hat eine Frau jedoch keinen Brustkrebs, so ist der Test in 7% der Fälle positiv $P(D|\overline{K})$.

Aus diesen Daten berechnet man die Wahrscheinlichkeit, dass die Diagnose positiv ist $P(D)$, ohne das Ereignis $K$ zu betrachten.

$$P(D) = P(D|K) \cdot P(K) + P(D|\overline{K}) \cdot P(\overline{K})$$

Damit sind alle benötigten Zahlen verfügbar und man setzt in (2.22) ein.

$$P(K|D) = \frac{P(D|K) \cdot P(K)}{P(D|K) \cdot P(K) + P(D|\overline{K}) \cdot P(\overline{K})}$$

$$P(K|D) = \frac{0.94 \cdot 0.006}{0.94 \cdot 0.006 + 0.07 \cdot 0.994}$$

$$P(K|D) = 0.07498 = 7.498\%$$

Dieses Ergebnis mag verwundern oder sogar falsch erscheinen. Tatsächlich ist aber die a-priori Wahrscheinlichkeit für $K$ um eine Zehnerpotenz geringer als die False-Positive-Rate des Tests, so dass der Test nur als Entscheidungsgrund für weitere (meist aufwändigere und teurere) Tests zulässig ist, nicht aber zu der Annahme verleiten soll, dass Ereignis $K$ tatsächlich eingetreten ist.

**Beispiel:** In einem Beutel befinden sich sechs Münzen. Fünf davon tragen auf einer Seite eine Zahl und auf der anderen einen Kopf. Die sechste Münze trägt auf beiden Seiten einen Kopf.

Eine Münze wird entnommen und viermal geworfen, jedes Mal zeigt sie einen Kopf. Dieses Ereignis nennen wir Ereignis B.

Ereignis A sei definiert mit: "Die gezogene Münze ist jene mit einem Kopf an jeder Seite".

Wir wissen bereits die Wahrscheinlichkeit für das Ereignis A $P(A) = \frac{1}{6}$. Für das Ereignis B gibt es zwei Möglichkeiten. Möglichkeit eins: "Es wurde die Doppelkopfmünze gezogen ($P(B|A)$)". Wenn dies der Fall ist, so ist die Wahrscheinlichkeit für das Ergebnis (viermal Kopf) genau eins. Diese Möglichkeit hat ihrerseits die Wahrscheinlichkeit $P(A) = \frac{1}{6}$. Die Wahrscheinlichkeit des Ergebnisses bei dieser Möglichkeit ist natürlich: $P(B|A) = 1^4 = 1$. Die zweite Möglichkeit: "Es wurde eine normale Münze gezogen" hat die Wahrscheinlichkeit $P(\overline{A}) = \frac{5}{6}$. Die Wahrscheinlichkeit des oben genannten Ergebnisses bei der zweiten Möglichkeit ist $P(B|\overline{A}) = (\frac{1}{2})^4 = \frac{1}{16}$. Addiert man nun die Produkte der jeweiligen Möglichkeits- und Ergebniswahrscheinlichkeiten, so erhält man: $P(B) = P(A) \cdot P(B|A) + P(\overline{A}) \cdot P(B|\overline{A}) = \frac{1}{6} \cdot 1 + \frac{5}{6} \cdot \frac{1}{16} = \frac{1}{6} + \frac{5}{96} = \frac{7}{32}$. Nun verfügen wir über alle notwendigen Größen und setzen diese in Gleichung 2.22 ein:

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

$$P(A|B) = \frac{1 \cdot \frac{1}{6}}{\frac{7}{32}}$$

$$P(A|B) = \frac{16}{21} = 0,7619$$

Das bedeutet also, wenn wir eine Münze aus dem Sack holen, sie viermal werfen und sie jedesmal mit dem Kopf nach oben landet, dann ist die Wahrscheinlichkeit, dass wir die Münze mit zwei Köpfen gezogen haben, 0,76 oder 76%.

Diese Beispiele hatten zwar noch nichts mit dem Klassifizieren von Texten zu tun, illustrieren aber sehr effektiv die Funktionsweise der Bayes'schen Formel.

### Theorie: Das Klassifizieren von Texten

Im letzten Abschnitt wurde die Bayes'sche Formel anhand eines einfachen Beispiels erklärt. Dieser Abschnitt wird zeigen, wie sich diese Methode einsetzen lässt, um Texte zu klassifizieren.

Die Ereignisse A und B aus dem Abschnitt 2.2.3 werden beim Klassifizieren eines Textes wie folgt definiert: Ereignis A ist die Zugehörigkeit eines Textes zu einer Kategorie. Ereignis B ist das Auftreten eines bestimmten Textes.

Zu Ereignis B: Die Auftrittswahrscheinlichkeit eines ganz bestimmten Textes geht mit steigender Länge rasch gegen Null. Bei einem Naiven Bayes'schen Klassifikator definiert man das Auftreten eines Textes mit dem Auftreten der einzelnen Wörter in beliebiger Reihenfolge, wobei dann die Auftrittswahrscheinlichkeit eines Textes die Verbundwahrscheinlichkeit des Auftretens der einzelnen Wörter ist.

Die a-priori Wahrscheinlichkeiten, die wir im letzten Beispiel recht einfach berechnen konnten, werden beim Klassifizieren durch ein "Training" ermittelt.

Die a-priori Kategoriewahrscheinlichkeit $P(A)$ berechnet sich aus der Division der Gesamtzahl der trainierten Dokumente durch die Anzahl derer, welche zu der Kategorie gehören.

Wie oben erwähnt, ist die a-priori Wahrscheinlichkeit des Dokumentes $P(B)$ durch das Produkt der einzelnen Wortwahrscheinlichkeiten definiert.

$$P(B) = \prod_{i=1}^{n} P(B_i)$$

$P(B_i)$ ist die Wahrscheinlichkeit für ein bestimmtes Wort $i$ aus einer Liste von $n$ Wörtern. Diese Wahrscheinlichkeit erhält man aus der Division der Gesamtzahl der trainierten Dokumente durch die Anzahl derer, welche das Wort enthalten.

Die bedingte Wahrscheinlichkeit $P(B|A)$, dass ein bestimmtes Dokument auftritt, wenn bereits gewiss ist, dass es in der Klasse ist, berechnet man wie $P(B)$, mit dem Unterschied, dass jetzt nur die Dokumente in Betracht gezogen werden, die zu der Klasse gehören.

Am einfachsten lässt sich das anhand eines Beispiels erläutern:

**Beispiel:** 4 Dokumente. 2 in der Klasse, 2 nicht in der Klasse. Worte: word1 bis word10

In der Klasse:

```
doc1: word1 word2 word3 word4   word9 word10
doc2: word1 word2 word3 word5   word8 word10
```

Nicht in der Klasse:

```
doc3: word2 word3 word4 word5   word6 word7
doc4: word1 word2 word4 word5   word6
```

Unbekanntes Dokument:

```
docX: word3 word4 word10
```

A-priori-Wahrscheinlichkeit der Klasse: $P(klasse) = 0.5$

Wortwahrscheinlichkeiten:
$P(word1) = 0.75$
$P(word2) = 1$
$P(word3) = 0.75$
$P(word4) = 0.75$
$P(word5) = 0.75$
$P(word6) = 0.5$
$P(word7) = 0.25$
$P(word8) = 0.25$
$P(word9) = 0.25$
$P(word10) = 0.5$

Wortwahrscheinlichkeiten in der Klasse:
$P(word1|klasse) = 1$
$P(word2|klasse) = 1$
$P(word3|klasse) = 1$
$P(word4|klasse) = 0.5$
$P(word5|klasse) = 0.5$
$P(word6|klasse) = 0$
$P(word7|klasse) = 0$
$P(word8|klasse) = 0.5$
$P(word9|klasse) = 0.5$

$P(word10|klasse) = 1$

Wortwahrscheinlichkeit für das unbekannte Dokument:

$P(word3, word4, word10) = P(word3) * P(word4) * P(word10) = 0.75 * 0.75 * 0.5 = 0.28125$

Wortwahrscheinlichkeit für das unbekannte Dokument in der Klasse:

$P(word3, word4, word10|klasse) = P(word3|klasse) * P(word4|klasse) * P(word10|klasse) = 1 * 0.5 * 1 = 0.5$

Bayes'sches Theorem:

$P(klasse|word3, word4, word10) = \frac{P(word3, word4, word10|klasse) * P(klasse)}{P(word3, word4, word10)} = 0.5 * 0.5/0.28125 = 0.888..89$

Bereits bei diesen knappen Trainingsdaten erreicht der Wahrscheinlichkeitswert eine eindeutige Größe.

#### Klassifizieren in mehrere Klassen

In dem letzten Abschnitt wurde beschrieben wie man feststellt, ob ein Dokument in einer Klasse ist oder nicht. Teilt man die Dokumente in mehrere Klassen auf und möchte wissen, in welcher Klasse ein unbekanntes Dokument liegt, so berechnet man die Zugehörigkeitswahrscheinlichkeit des Dokumentes für jede einzelne Klasse. Anhand der einzelnen Wahrscheinlichkeiten, in diesem Umfeld auch Scores genannt, gibt es zwei Möglichkeiten, eine Klassifikation zu erhalten. Erlaubt man nur eine Kategorie pro Dokument, so reicht es, die Kategorie mit dem maximalen Score zu bestimmen. Definiert man einen passenden Schwellenwert, so ist es auch möglich, einem Dokument mehrere Kategorien, jene deren Scores über dem Schwellenwert liegen, zuzuordnen.

## 2.3. Qualitätsmaß der Klassifikatoren

Um einen qualitativen Vergleich der Klassifikatoren oder auch der Vorbehandlungsmaßnahmen vornehmen zu können, werden aufgrund der Ergebnisse Qualitätszahlen ermittelt, mit denen sich eine Aussage wie "Diese Methode ist besser als eine andere" nachvollziehbar herstellen lassen kann.

### 2.3.1. Dokumentklassen

Um diese Qualtätswerte berechnen zu können, werden alle klassifizierten Dokumente in vier Klassen eingeteilt.

**True Positive:** Dokumente, die einer Kategorie angehören und dieser auch zugeordnet wurden, zählen in dieser Kategorie als True Positives.

**False Positive:** Wurde ein Dokument einer Kategorie zugeordnet, der es nicht angehört, so zählt es in dieser Kategorie als False Positive.

**True Negative:** Gehört ein Dokument einer Kategorie nicht an und wurde es dieser Kategorie nicht zugeordnet, so zählt es in dieser Kategorie als True Negative.

**False Negative:** Dokumente, die zu einer Kategorie gehören, aber dieser Kategorie nicht zugeordnet wurden, zählen in dieser Kategorie als False Negative.

Die nun folgenden Qualitätszahlen werden in der Regel nur zum Beurteilen eines Klassifizierungssystems angewendet, welches nur eine Kategorie kennt und entscheidet, ob ein Testfall in dieser Kategorie liegt oder nicht. Da in dieser Arbeit Klassifikationssysteme mit mehreren Kategorien behandelt werden sollen, ist es notwendig, die Berechnung der Qualitätszahlen zur Beurteilung etwas anzupassen. Die leicht geänderte Definition der Dokumentenklassen ist der erste Schritt dazu.

### 2.3.2. Qualitätsmerkmale einzelner Kategorien

Sind die Dokumentklassen beziehungsweise die Anzahl der Dokumente in den Klassen bekannt, so erfolgt die Berechnung der Qualitätsmerkmale einzelner Kategorien analog zu einer Bewertung eines Klassifikationssystems mit nur einer Kategorie.

Die nachfolgenden Sektionen wurden bewusst sehr ähnlich formuliert, um die Unterschiede der Qualitätsmerkmale besser hervorzuheben.

### Precision

Die Precision, oder auch positiver Vorhersagewert, gibt die Wahrscheinlichkeit an, dass ein zugeordnetes Dokument auch tatsächlich der zugeordneten Kategorie angehört. Aus den in Abschnitt 2.3.1 beschriebenen Werten berechnet sich die Precision wie in Gleichung 2.23 angegeben.

$$P = \frac{TP}{TP + FP} \tag{2.23}$$

Die Precision ist das Verhältnis der Anzahl der richtig einer Kategorie zugeordneten Dokumente zur Gesamtanzahl der Dokumente, welche dieser Kategorie zugeordnet wurden.

### Recall

Der Recall, oder auch Sensitivität, gibt die Wahrscheinlichkeit an, dass ein Dokument, welches der Kategorie angehört, auch tatsächlich gefunden wird. Aus den in Abschnitt 2.3.1 beschriebenen Werten berechnet sich der Recall wie in Gleichung 2.24 angegeben.

$$R = \frac{TP}{TP + FN} \tag{2.24}$$

Der Recall ist das Verhältnis der Anzahl der richtig einer Kategorie zugeordneten Dokumente zur Gesamtanzahl der Dokumente, welche dieser Kategorie angehören.

### Spezifität

Die Spezifität, oder auch Richtignegativ-Rate, gibt die Wahrscheinlichkeit an, dass ein Dokument, welches nicht der Kategorie angehört, auch tatsächlich nicht gefunden wird. Aus den in Abschnitt 2.3.1 beschriebenen Werten berechnet sich die Spezifität wie in Gleichung 2.25 angegeben.

$$S = \frac{TN}{TN + FP} \tag{2.25}$$

Die Spezifität ist das Verhältnis der Anzahl der richtig nicht einer Kategorie zugeordneten Dokumente zur Gesamtanzahl der Dokumente, welche nicht dieser Kategorie angehören.

### Fallout

Der Fallout ist das Gegenteil der Spezifität und gibt die Wahrscheinlichkeit an, dass ein einer Kategorie zugeordnetes Dokument nicht dieser Kategorie angehört. Aus den in Abschnitt 2.3.1 beschriebenen Werten berechnet sich die Spezifität wie in Gleichung 2.25 angegeben.

$$\begin{aligned} Fa &= 1-S \\ Fa &= 1-\frac{TN}{TN+FP} \\ Fa &= \frac{TN+FP}{TN+FP}-\frac{TN}{TN+FP} \\ Fa &= \frac{FP}{TN+FP} \end{aligned} \tag{2.26}$$

Die Spezifität ist das Verhältnis der Anzahl der falsch einer Kategorie zugeordneten Dokumente zur Gesamtanzahl der Dokumente, welche nicht dieser Kategorie angehören.

### F-Maß

Die beiden Qualitätsmerkmale Precision und Recall sind die wichtigsten bei der Bewertung eines Klassifikationssystems. Jedoch ist es leicht, einen davon zu optimieren, aber der andere wird dadurch vernachlässigt.

**Beispiel:** Sind von 1000 Elementen 500 relevant und wird nur eines der 500 gefunden, so ist die Precision eins oder 100%. Der Recall beträgt jedoch nur $1/500$ oder 0.2%. Um aus diesen beiden Werten eine aussagekräftige Zahl zu machen, wird der harmonische Mittelwert der beiden Zahlen errechnet. Dieser Mittelwert wird auch als F-Maß bezeichnet.

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.27}$$

Dieser Wert bestraft Klassifikatoren, die nur auf einen der beiden Werte hin optimiert wurden und den anderen dafür vernachlässigen. Dadurch wird ein objektiver Vergleich der Klassifikationssysteme möglich. Das F-Maß ist ein gutes Maß für die Treffsicherheit eines Klassifikationsalgorithmus und dient auch als Hauptmerkmal zur Beurteilung der Methoden, welche in dieser Arbeit verwendet werden.

## 2.4. Forms

Die Hypertext Markup Language (HTML), wie sie 1989 von Tim Berners-Lee erdacht wurde, sah außer dem Abrufen von statischen Seiten noch keine weitere Interaktion zwischen Benutzer und Dienstanbieter vor. Erst mit HTML 2.0, veröffentlicht im November 1995, waren vom Benutzer ausfüllbare Formularelemente, sogenannte 'Forms', möglich. Dadurch wurden interaktive Dienste, wie zum Beispiel Datenbankabfragen, realisierbar.

Für die Eingabe von Daten sind mehrere Arten von Eingabeelementen vorgesehen. Die einfachsten sind Freitextfelder, von denen es eine einzeilige und eine mehrzeilige Version gibt. Für das Auswählen von vordefinierten Werten wurden im Standard vier Typen vorgesehen — je zwei, welche eine Mehrfachauswahl und zwei, welche genau eine Auswahlmöglichkeit zulassen.

Die beiden Felder mit Mehrfachauswahl sind das Listenfeld und die Checkbox. Drop-down Menüs und Radiobuttons lassen je nur eine Auswahl zu.

### 2.4.1. Beispiele

Um zu verdeutlichen wie solche Formulare aussehen, seien an dieser Stelle einige Beispiele angegeben.

**Leeres Formular** Formulare werden von einem "form"-Tag bestimmt. Das Attribut "action" gibt an, was mit den eingegebenen Daten passieren soll.

```
1 <form action="http://host/input.pl">
2 </form>
```

In diesem Beispiel werden die Daten über das Protokoll "http" auf dem Rechner "host" dem Programm "input.pl" übergeben.

**Textfeld** Das Textfeld ist die einfachste Form eines Eingabefeldes. Es erscheint im Browser als ein einzeiliges Texteingabefeld. Für Eingabefelder wird grundsätzlich der Tag "input" verwendet. Erst das Attribut "type", welches in diesem Beispiel den Wert "text" hat, bestimmt, dass es sich um ein Texteingabefeld handelt.

```
1 <input name="vorname" value="Hans" type="text">
```

Dieses Beispielfeld hat noch zusätzlich die Attribute “name” und “value”. Der Name des Feldes ist zugleich der Parametername bei der Übergabe an das verarbeitende System. Das Attribut “value” dient bei Textfeldern dazu, sie mit diesem Wert zu initialisieren. Das heißt, in diesem Feld steht bereits der Wert “Hans”.

Wird dieses Formular abgeschickt, so übergibt es dem System einen Parameter mit dem Namen “vorname”, welcher den Wert “Hans” hat.

**Auswahlfelder** Ein Auswahlfeld lässt nur Eingaben aus einer definierten Menge zu. Browser stellen diese Felder entweder meist einzeilig dar oder als Liste von auswählbaren Elementen. Klickt man auf das einzeilige Feld, so öffnet sich ein kleiner Bereich, in dem die Liste der möglichen Auswahlen zu sehen ist.

```
<select name="city">
    <option value="DUB">Dublin</option>
    <option value="VIE">Wien</option>
    <option value="INN" selected>Innsbruck</option>
</select>
```

Dieses Auswahlfeld hat drei mögliche Optionen. Der Benutzer sieht diese als Städtenamen dargestellt. Die Flughafenkürzel sieht der Benutzer nicht. Diese werden aber anstatt der Städtenamen an das verarbeitende System als Wert des Parameters “city” übergeben. Die Stadt “Innsbruck” ist hier voreingestellt.

# 3. Die Flugdomain

Die Domäne der Flugsuchseiten wurde gewählt, da sie bei ausreichender Komplexität noch eine wohlüberschaubare Anzahl möglicher Eingabefelder besitzt. Die Funktion dieser Seiten ist es, zu gegebenem Start- und Zielort einen Flug zu einer bestimmten Zeit zu finden.

## 3.1. Allgemeines über die Flugsuche

Man findet auf diesen Seiten grundsätzlich Formularelemente für Abflug- und Ankunftsort, welche meist als Auswahlfelder, aber selten auch als freie Textfelder ausgeführt sind. Die Eingabefelder für die Reisezeit sind vielfältiger gestaltet. Der gewöhnlichste Fall sind Auswahlfelder für Tag, Monat und Jahr. Aber auch Kombinationen wie zum Beispiel Monat und Jahr in einem Feld gemeinsam kommen genauso vor wie freie Textfelder für einzelne Teile des Datums oder auch für das gesamte Datum. Manche sind auch als anklickbare Kalender ausgeführt. In einzelnen Fällen sind die Formulare auch zweistufig ausgeführt. Das bedeutet, dass man einen Teil der Eingaben erst auf der Folgeseite tätigt.

In zur Zeit noch seltenen Fällen werden auch seitens der Fluganbieter Maßnahmen ergriffen um ein automatisches Verwenden der Webseite zu verhindern. So muss man etwa um die Seite verwenden zu können ein CAPTCHA[1] entziffern und die entzifferten Zeichen in ein eigenes Feld eingeben, ehe man die Ergebnisse erhält.

## 3.2. Beispiele

Abbildungen 3.1 und 3.2 zeigen das Eingabefeld für die Flugsuche zweier Fluglinien. Die Formulare sehen einander sehr ähnlich, denn beide bieten neben den Auswahlfeldern für Tag

[1] Steht für "Completely Automated Public Turing test to tell Computers and Humans Apart". CAPTCHAs sind Bilder von Buchstaben und Zahlen, die auf eine Art und Weise dargestellt werden, dass sie von Menschen, aber nicht von Maschinen gelesen werden können.

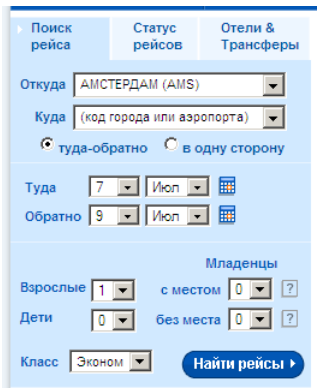


Abbildung 3.1.: Flugsuche der russischen Aeroflot ([Aer07])

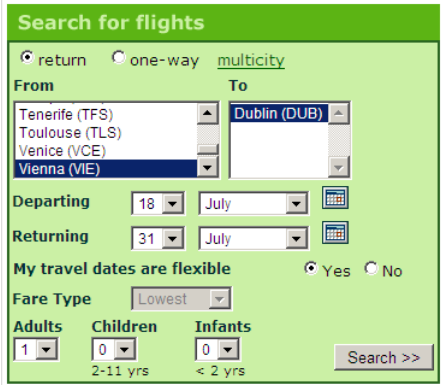


Abbildung 3.2.: Flugsuche der irischen Aer Lingus ([Lin07])

und Monat eine Möglichkeit zur Anzeige eines Kalenders an, aus welchem der gewünschte Tag auswählbar ist.

Wählt man nun eine Flugverbindung und einen Hin- sowie einen Rückflugtag, so erhält man bereits ein Ergebnis wie in Abbildung 3.3 dargestellt. Im oberen Teil der Ergebnisseite erhält man noch die Möglichkeit, den Tag der Hin- oder Rückreise zu verschieben, um eventuell einen niedrigeren Preis für den Flug zu bekommen.

Geht man einen Schritt weiter, so erhält man wie in Abbildung 3.4 auch die Abflugs- und Ankunftszeit. Dieses Ergebnis kann dann bereits für eine automatische Verarbeitung verwendet werden.


Abbildung 3.3.: Zweiter Schritt der Flugsuche bei Aer Lingus([Lin07])

Hat man sich bei Aer Lingus für ein Flugziel und einen Tag entschieden, so erhält man eine Auswahl von alternativen Flugtagen mit eventuell niedrigeren Preisen.

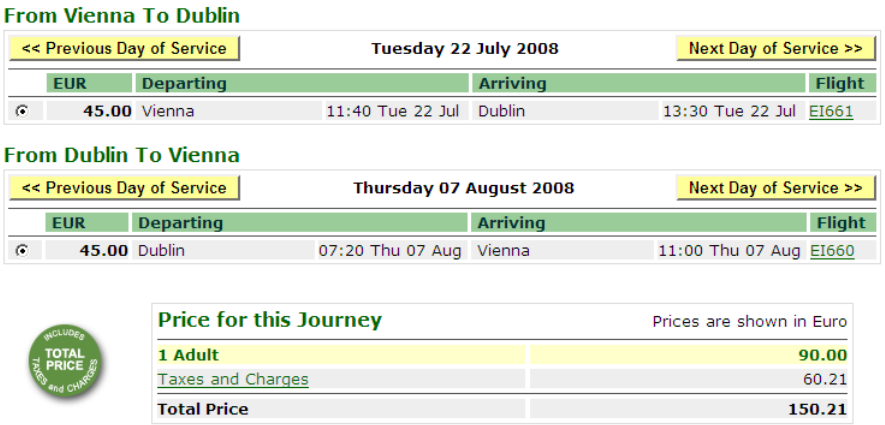


Abbildung 3.4.: Dritter Schritt der Flugsuche bei Aer Lingus([Lin07])

Nach dem nochmaligen Bestätigen oder Ändern des Flugtages erhält man eine Liste von Flügen mit der Abflugs- und Ankunftszeit.

| Bezeichnung | Bedeutung |
|---|---|
| origin_iata | Kurzbezeichnung des Abflugflughafens |
| origin_airport | Name des Abflugflughafens |
| origin_airportandiata | Name und Kurzbezeichnung des Abflugflughafens |
| origin_country | Abflugsland |
| destination_iata | Kurzbezeichnung des Zielflughafens |
| destination_airport | Name des Zielflughafens |
| destination_airportandiata | Name und Kurzbezeichnung des Zielflughafens |
| destination_country | Zielland |
| departure_date | Abflugsdatum |
| departure_day | Abflugstag |
| departure_month | Abflugsmonat |
| departure_year | Abflugsjahr |
| departure_monthofyear | Abflugsmonat und Jahr |
| departure_calendar | Kalender-Auswahlelement für den Abflug |
| return_date | Rückflugsdatum |
| return_day | Rückflugstag |
| return_month | Rückflugsmonat |
| return_year | Rückflugsjahr |
| return_monthofyear | Rückflugsmonat und Jahr |
| return_calendar | Kalender-Auswahlelement für den Rückflug |
| misc | nicht relevante Felder, wie etwa “Anzahl der Passagiere” |
| hidden | versteckte Felder |

Tabelle 3.1.: Bezeichner der Eingabefeldklassen
Diese Bezeichner entsprechen jenen, die auch im Projekt MetaMorph verwendet werden.

## 3.3. Felderbezeichnungen

Tabelle 3.1 zeigt die, im Experimentiersystem verwendeten, Kategoriebezeichnugnen der Eingabefelder. Die Flugsuche aus Abbildung 3.2 wurde im Experimentiersystem mit folgenden Bezeichnern versehen: Das Listenfeld unter dem Wort “From” ist ein “origin_airportandiata”. Das Listenfeld rechts daneben ist ein “destination_airportandiata”. Die Felder rechts neben dem Wort “Departure” sind “departure_day” und “departure_month”. Analog dazu sind die beiden Felder darunter “return_day” und “return_month”. Alle anderen Felder, wie “Adults” oder “Children” tragen die Bezeichnung “misc”.

# 4. Das Experimentiersystem: Funktionen & Verwendung

## 4.1. Verwendung

Der im Zuge dieser Arbeit entstandene Prototyp ermöglicht es dem Anwender die implementierten Klassifikationsalgorithmen und Vorfilter an einzelnen Seiten zu testen ohne diese manuell herunterladen zu müssen.

Ist das System noch leer, das heißt wenn man versucht eine Seite, wie in Abschnitt 4.3.1 beschrieben ist, zu analysieren erhält man noch kein Ergebnis. Allem voran wird man beginnen, indem man einzelne Seiten mit Kategorien versehen in das System eingibt, siehe dazu Abschnitt 4.3.2. Danach ist es bereits möglich einen Klassifikator für ganze Seiten zu erstellen (Abschnitt 4.4.3). Nun können auch neue Seiten klassifiziert werden, nicht aber deren Formulare. Dazu müssen erst jene gespeicherten Seiten, welche Formulare zur Flugsuche enthalten, bearbeitet werden. Wie in Abschnitt 4.3.3 beschrieben, werden erst die Formulare und danach die Felder mit Kategorien versehen und gespeichert.

Nun ist das System bereit um damit Experimente zu machen. In Abschnitt 4.4.1 befinden sich nähere Hinweise zu den in diesem System möglichen Experimenten. Nachdem mehrere unterschiedliche Klassifikatoren für Seiten, Formulare und Felder erstellt wurden, können diese auch ausgetauscht und beliebig kombiniert werden (Abschnitt 4.4.2). Um die Klassifikatoren vergleichen zu können, ist es möglich diese automatisch zu testen (Abschnitt 4.4.4). Die Testergebnisse werden gespeichert und können jederzeit wieder abgerufen werden (Abschnitt 4.4.5).

## 4.2. Arbeitsweise

In diesem Abschnitt wird erklärt, was bei der Bearbeitung einer Webseite passiert.

### 4.2.1. Notwendige Aufgaben zum Verstehen einer Seite

Um eine Seite zu verstehen müssen zwei Aufgaben erfüllt sein. Zuerst muss eine unbekannte Seite als eine für die gestellte Aufgabe relevante Seite identifiziert werden. Ist dies geschehen, so müssen die Dateneingabeelemente auf dieser Seite identifiziert und deren Funktion erkannt werden.

In Abbildung 4.1 wird der prinzipielle Ablauf der Verarbeitung einer Webseite verdeutlicht. Der Ablauf erfolgt in drei Schritten, die durch eine Aufteilung des Dokumentes in die nächste Klassifikationsebene getrennt sind.

### 4.2.2. Die Schritte und Ebenen im Detail

Jedem Klassifikationsschritt kommt ein Präprozessorschritt zuvor, der den Text aufbereitet. Es können unterschiedliche Präprozessoren angewendet werden, mit dem Ziel, die Genauigkeit der Klassifikatoren zu erhöhen. Dabei ist darauf zu achten, dass für die Daten derselbe Präprozessor verwendet wird, der auch für die Erstellung des Klassifikators verwendet wurde.

Die drei Klassifikationsebenen sind die ganze Seite, die Formulare und die Felder der Formulare. Der daraus resultierende Ablauf ergibt sich im Detail aus folgenden Schritten:

- Der komplette HTML-Code der Seite wird durch den Präprozessor verarbeitet.
- Der HTML-Code wird klassifiziert.
- Die Formulare werden aus dem HTML-Code extrahiert.
- Für jedes Formular:
  - Der komplette HTML-Code des Formulars wird durch den Präprozessor verarbeitet.
  - Der HTML-Code des Formulars wird klassifiziert.

- Die Felder werden aus dem HTML-Code des Formulars extrahiert.
- Für jedes Feld:
  - Der komplette HTML-Code des Feldes wird durch den Präprozessor verarbeitet.
  - HTML-Code des Feldes wird klassifiziert.

Das Aufteilen in die nächste Ebene wird in Abschnitt 5.3.4 beschrieben, beziehungsweise in Abschnitt 5.3.3 für die erste Version des Splitters. Das Modul für die Präprozessoren sowie die Präprozessoren selbst werden in Abschnitt 5.3.2 im Detail beschrieben. Das für die Klassifikation verwendete Framework "AI::Categorizer" wird in Abschnitt 5.3.5 näher erläutert.

## 4.3. Einfache Funktionen

Das Experimentiersystem verwendet ein Webinterface zur Kommunikation mit den Benutzern. Es erfüllt Funktionen um neue Klassifikationsalgorithmen und Vorfilter zu testen, es können einzelne Seiten klassifiziert werden und auch Klassifikatoren erstellt werden, welche anschließend über das in Abschnitt 4.5 beschriebene Interface verwendet werden können.

### 4.3.1. Das Klassifizieren einer beliebigen Seite

Um eine Seite klassifizieren zu lassen gibt man die URL in das Eingabefeld ein und betätigt die Schaltfläche zum Abschicken der Anfrage. Als Ergebnis erhält man eine Auswertung der Klassifikation der Seite.

Weiters werden in einer Tabelle darunter alle in der Seite gefundenen Formulare mit einer eben solchen Ergebnisdarstellung und einer Darstellung des Formulares aufgelistet. Die Formulare werden unverändert in den HTML-Code der Ergebnisseite eingebaut. Das heißt, es kommt zwangsläufig zu Darstellungsfehlern, da Bilder und CSS[1]-Elemente fehlen. Aber der Sinn des Formulars ist noch erkennbar und es kann vom Benutzer mit einer Kategorie versehen und gespeichert werden. Siehe dazu Abschnitt 4.3.3.

[1] Cascading Style Sheets

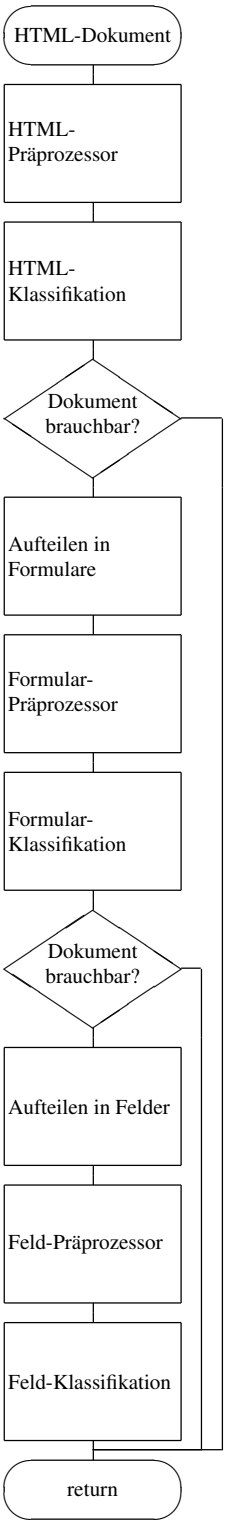


Abbildung 4.1.: Ablauf der Bearbeitung
Dieses Diagramm zeigt, welche Schritte ein HTML-Dokument im Zuge seiner Klassifikation durchläuft.

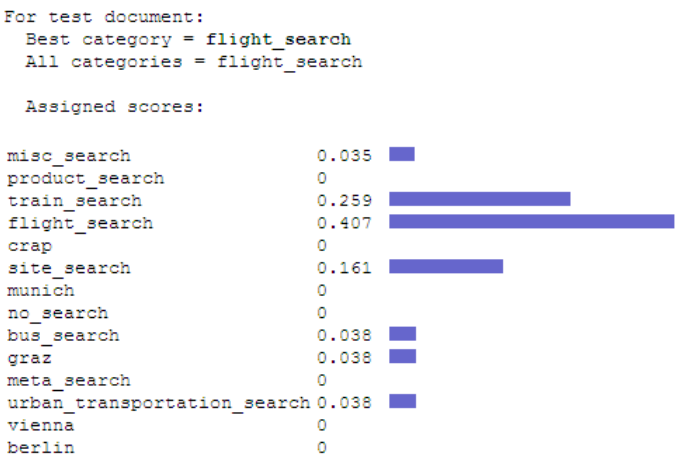


Abbildung 4.2.: Ergebnis der Seitenklassifikation

Angezeigt werden die gerundeten numerischen Werte und eine grafische Darstellung dieser Werte.

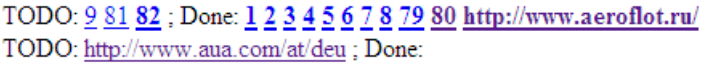


Abbildung 4.3.: TODO-Liste

Diese Listen helfen bei der Koordination der Arbeit.

### 4.3.2. Das Speichern einer Seite

Um eine Seite zu speichern genügt es die Kategorieauswahl von “don’t store” auf die gewünschte Kategorie zu ändern und danach die Schaltfläche zum Absenden zu betätigen. Die gespeicherte Seite steht danach in der Liste der Seiten im Abschnitt “TODO”. Sie wird dort als ihre URL dargestellt. Einige Webseiten werden dort als Zahlen dargestellt. Diese Einträge stammen aus einer früheren Version des Experimentiersystems.

Diese Liste existiert zweimal. Die erste Liste enthält Webseiten, welche mit dem ersten, auf Textanalyse basierenden, Splitter gespeichert wurden. Die zweite Liste enthält Seiten, welche mit der zweiten Splitterversion, die sich der Baumstruktur des HTML bedient, gespeichert wurden.

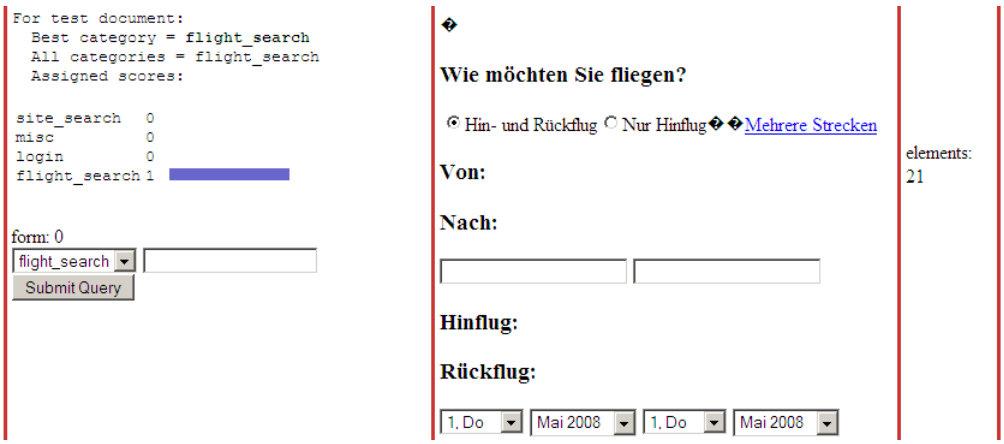


Abbildung 4.4.: Ein ausgewertetes Formular

Dieses Bild zeigt die Auswertung der Klassifikation, die Speichermöglichkeit und eine Darstellung eines Formulars.

### 4.3.3. Das Speichern der Unterelemente

Wurde eine Seite gespeichert, so ist es möglich, auch die Formulare darin mit Kategorien versehen zu speichern. Das funktioniert analog zum Speichern einer Seite. Man wählt die Kategorie und klickt auf den Knopf zum Absenden. Ein bereits gespeichertes Formular wird farbig hinterlegt.

Wird einem Formular die Kategorie “flight_search” zugewiesen, so wird dieses in seine Elemente zerlegt, die wiederum einer Kategorie zugeordnet und gespeichert werden können. Auch hier werden bereits gespeicherte Felder farbig hervorgehoben. Die Farbe für Formulare ist rot, die der Felder ist blau. Die Felder selbst werden zusätzlich zu ihrer vom Browser interpretierten Darstellung auch noch als HTML-Code dargestellt. Das erleichtert die Zuordnung insbesondere der verstecken Felder.

## 4.4. Erweiterte Funktionen

Ist eine ausreichende Datenbasis vorhanden, so können mit dem System Experimente durchgeführt und neue Klassifikatoren für den Einsatz der maschinellen Klassifikation erstellt werden. Es ist auch möglich den Klassifikator und das Experiment und damit den verwendeten Vorfilter für das Experimentieren mit einzelnen Seiten auszuwählen.

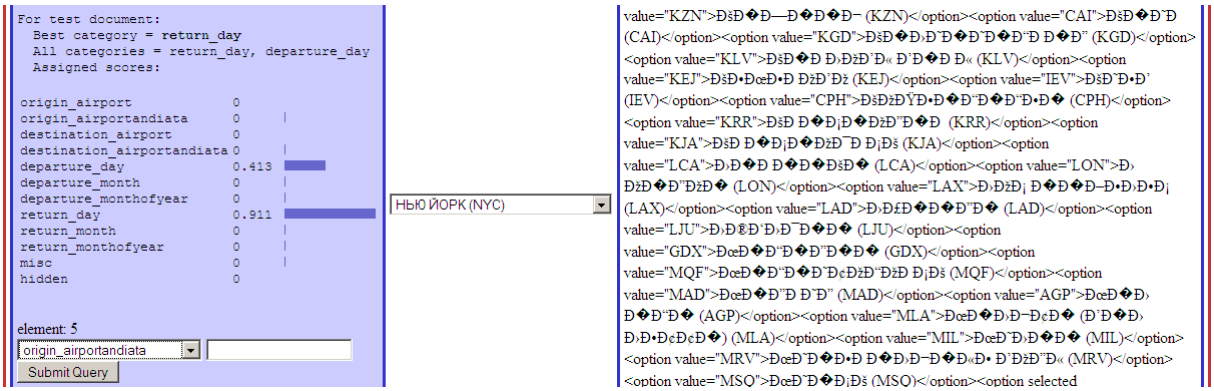


Abbildung 4.5.: Ein ausgewertetes Feld

Diese Abbildung zeigt die Auswertung der Klassifikation, die Speichermöglichkeit und eine Darstellung eines Feldes sowie dessen HTML-Code.

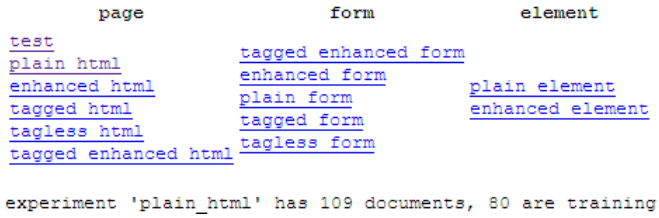


Abbildung 4.6.: Liste der Experimente

Die Tabelle der Experimente ist nach Datentypen gegliedert.

### 4.4.1. Die Experimente

Ein Experiment ist eine Relation zwischen einem Datentyp und einem Vorfilter. Die Experimente können nur direkt in der Datenbank verändert werden, siehe dazu Abschnitt 5.2.1. In der Übersicht der Experimente sind diese mit ihrem Namen angeführt. Darunter werden die gespeicherten Klassifikatoren des gewählten Experimentes angezeigt, diese können dann zum Experimentieren mit einzelnen Seiten geladen werden. Hier können auch neue Klassifikatoren erstellt und Testreihen durchgeführt werden. Es ist von hier aus auch möglich, die Ergebnisse der Testreihen eines Experimentes anzusehen.

### 4.4.2. Andere Klassifikatoren verwenden

Unter der Experimentauswahl befindet sich eine Liste der für dieses Experiment erstellten Klassifikatoren. Diese Liste enthält die Beschreibung, das Erstellungsdatum und den verwen-

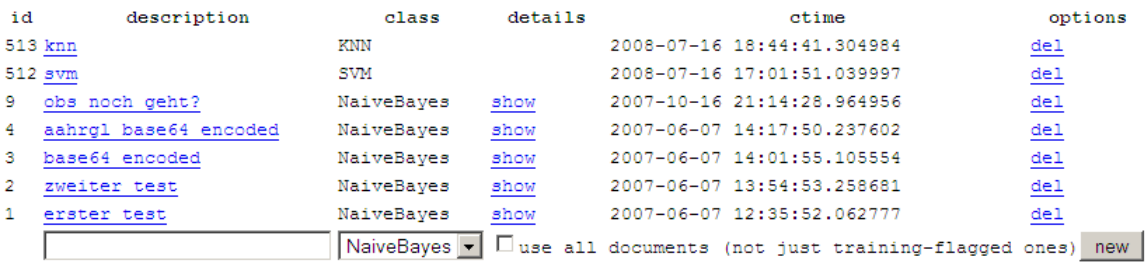


Abbildung 4.7.: Liste der Klassifikatoren

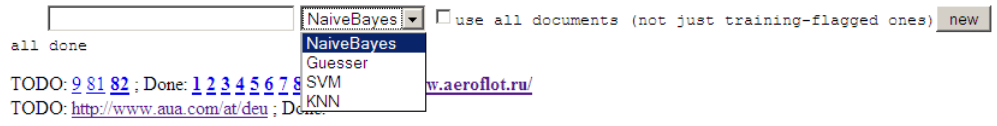


Abbildung 4.8.: Neuer Klassifikator

deten Klassifizierungsalgorithmus. Klickt man auf die Beschreibung, so wird dieser Klassifikator für kommende Experimente mit einzelnen Seiten verwendet. Für jeden Datentyp kann der Klassifikator separat gewählt werden. Da Klassifikatoren immer mit einem bestimmten Vorfilter erstellt werden, wird der Vorfilter aufgrund des Klassifikators automatisch gewählt.

Klickt man auf den Link “show”, so erhält man Informationen über die Interna des Klassifikators. Diese Funktion ist nur für den “NaiveBayes”-Algorithmus verfügbar. Mit dem Link “del” können Klassifikatoren gelöscht werden.

### 4.4.3. Das Erstellen eines Klassifikators

Um einen neuen Klassifikator zu erstellen, trägt man die Bezeichnung in das dafür vorgesehene Feld ein, wählt den gewünschten Algorithmus und entscheidet, ob man alle Daten verwenden will oder nur solche, die dafür vorgesehen sind.

Das Erstellen eines Klassifikators kann je nach Algorithmus und Datenmenge einige Sekunden in Anspruch nehmen.

| 30, NaiveBayes referenz del | |
|---|---|
| total_docs | 122 |
| total_guessed | 108 |
| recall | 0.692191544434858 |
| precision | 0.650254117467232 |
| fallout | 0.925699132111861 |
| spezifitaet | 0.0743008678881389 |
| F | 0.670567777331157 |

| 1: flight_search | |
|---|---|
| total_docs | 14 |
| total_guessed | 21 |
| good | 13 |
| false | 8 |
| missed | 1 |
| TP | 13 |
| FP | 8 |
| FN | 1 |
| TN | 94 |
| recall | 0.928571428571429 |
| precision | 0.619047619047619 |
| fallout | 0.92156862745098 |
| spezifitaet | 0.0784313725490197 |
| F | 0.742857142857143 |

Abbildung 4.9.: Ergebnistabelle

Diese Abbildung zeigt die Ergebnistabelle der Zusammenfassung und daneben eine Tabelle der Einzelergebnisse einer Kategorie.

### 4.4.4. Das Testen eines Experimentes mit einem Klassifikatortyp

Analog zum Erstellen eines Klassifikators funktioniert auch das Testen eines Experimentes mit einem bestimmten Klassifikationsalgorithmus. Man trägt eine Beschreibung ein, wählt den gewünschten Algorithmus und betätigt die Schaltfläche “test”. Dieser Vorgang ist sehr aufwändig und kann je nach Datenmenge bis zu zwei Stunden dauern.

### 4.4.5. Die Testergebnisse

Die Ergebnisse der Tests werden in der Datenbank gespeichert und können danach verglichen werden. Um zur Ergebnisansicht zu gelangen klickt man auf den Link “view stored results” in der Ansicht des gewünschten Experimentes. Man bekommt eine Zusammenstellung aller Testläufe dieses Experimentes.

Dargestellt werden die Maßzahlen, welche in Abschnitt 2.3 beschrieben wurden. Die erste Tabelle zeigt die Qualitätsmerkmale für den gesamten Testlauf. Die Tabellen darunter zeigen die Qualitätszahlen für jeweils eine Kategorie an. Abbildung 4.9 zeigt ein Beispiel solcher Tabellen.

## 4.5. Schnittstellen für das automatisierte Klassifizieren

In diesem Abschnitt werden die Schnittstellen beschrieben, über welche andere Systeme das Klassifizierungssystem verwenden können.

Beschrieben wird die aktuelle Implementierung der Schnittstelle, deren Konfigurier- und Erweiterbarkeit sowie andere mögliche Schnittstellen in genereller Form, welche bei Bedarf implementiert werden können.

### 4.5.1. Datentransport

Als Kommunikationsprotokoll wurde http[2] gewählt. Die Gründe für diese Wahl sind:

- Unabhängigkeit von der verwendeten Programmiersprache
- Unabhängigkeit vom Betriebssystem
- Gute Eignung für die Art der ausgetauschten Daten
- Das Testen des Systems ist mit einem Browser möglich

Da zu erwarten ist, dass die Datenmenge einer zu klassifizierenden Seite die Möglichkeiten einer GET-Anfrage[3] übersteigt, wird empfohlen, eine POST-Abfrage[4] dafür zu verwenden.

### 4.5.2. Eingabedaten

Zu den Eingabedaten zählen das zu klassifizierende HTML selbst und die Parameter, mit denen das Verhalten des Systems beeinflusst werden kann. Tabelle 4.1 zeigt die unterstützten Parameter. Wird ein Parameter übergeben, welcher nicht vom System unterstützt wird, so wird dieser ignoriert. Dabei wird kein Fehler erzeugt.

[2] HyperText Transfer Protocol

[3] Bei einer GET-Abfrage werden die Parameter in dem URI (Uniform Resource Identifier), in einer Zeile übergeben. Die Länge dieser Zeile ist begrenzt.

[4] Bei einer POST-Abfrage werden die Parameter nicht im URI übergeben, sondern werden vom Browser im http-Header übergeben. Diese Art der Datenübergabe eignet sich besser für große Datenmengen. Sie hat aber den Nachteil, dass die Abfragen mit den Parametern nicht so einfach als Lesezeichen gespeichert werden können.

| Parameter | notwendig | Bedeutung (default) |
|---|---|---|
| html | ja | Das zu klassifizierende HTML-Dokument (wird dieser Parameter weggelassen, so wird ein Datensatz aus der Datenbank verwendet) |
| output_mimetype | nein | Der bei der Antwort gesetzte MIME Typ (text/rdf) |
| test_forms_anyway | nein | Klassifiziere die Formulare auch, wenn es sich um keine Flugsuchseite handelt (0) |
| test_fields_anyway | nein | Klassifiziere die Felder auch, wenn es sich um kein Flugsuchformular handelt (0) |
| output_html | nein | Ergebnis der Klassifikation der ganzen Seite wird ausgegeben (0) |
| output_forms | nein | Ergebnis der Klassifikation der Forumlare wird ausgegeben (0) |
| output_fields | nein | Ergebnis der Klassifikation der Felder wird ausgegeben (1) |
| output_RDF_open_chars | nein | Das erste Zeichen eines URI (leer) |
| output_RDF_close_chars | nein | Das letzte Zeichen eines URI (leer) |
| output_RDF_separator | nein | Der Feldseparator einer Ausgabezeile (“\t”) |
| output_RDF_newline | nein | Der Zeilenseparator einer Ausgabezeile (“\n”) |
| output_RDF_verb_html | nein | Die Beziehung zwischen Xpath und zugeordneter Klasse der Seite (leer) |
| output_RDF_verb_form | nein | Die Beziehung zwischen Xpath und zugeordneter Klasse des Formulars (leer) |
| output_RDF_verb_field | nein | Die Beziehung zwischen Xpath und zugeordneter Klasse des Feldes (leer) |
| html_prefilter | nein | Der zu verwendende Vorfilter für ganze Seiten (“tag_tags”) |
| form_prefilter | nein | Der zu verwendende Vorfilter für Formulare (“tag_tags”) |
| field_prefilter | nein | Der zu verwendende Vorfilter für Felder (“element_enhance”) |
| html_classifier_id | nein | Der zu verwendende Klassifikator für ganze Seiten (4) |
| form_classifier_id | nein | Der zu verwendende Klassifikator für Formulare (8) |
| field_classifier_id | nein | Der zu verwendende Klassifikator für Felder (16) |

Tabelle 4.1.: Parameter der Schnittstelle

Die Schnittstelle zum Einsatzsystem akzeptiert diese Parameter. Alle andere Parameter werden ignoriert.

### 4.5.3. Ausgabedaten

Nach erfolgter Bearbeitung der Eingabe werden die gewünschten Daten ausgegeben. Die Ausgabe orientiert sich dabei am RDF[5] in einer vereinfachten N3[6] Syntax.

Ausgegeben wird dabei zuerst der Xpath des gefundenen Elementes und dann die zugeordnete Klasse.

Die Ausgabe erfolgt dabei exakt in dieser Reihenfolge:

- output_RDF_open_chars
- xpath
- output_RDF_close_chars
- output_RDF_separator
- output_RDF_open_chars
- output_RDF_verb_html beziehungsweise: output_RDF_verb_form oder output_RDF_verb_field
- output_RDF_close_chars
- output_RDF_separator
- die zugeordnete Klasse
- output_RDF_newline

Die aktuelle Einstellung des Exportes erinnert nicht mehr an die N3-Syntax, da nach den ersten Versuchen erkannt wurde, dass die Verwendung des RDF-Formates im vollen Umfang für die zu transportierenden Daten ungeeignet ist.

Das nun verwendete Ausgabeformat ist eine CSV-Liste[7] mit zwei Tabulatoren als Trennzeichen.

[5] Resource Description Framework
[6] Notation 3, entwickelt von Tim Berners-Lee
[7] Character seperated Values

# 5. Das Experimentiersystem: Entwurf & Implementierung

Der erste Prototyp der Experimentiersoftware wurde bereits in Perl geschrieben und diente in erster Linie der Feststellung, ob der Versuch, Webseiten mittels statistischer Analyse zu klassifizieren, überhaupt Aussicht auf Erfolg hat. Er lehnte sich stark an das Beispielprogramm des AI::Categorizer[1] Moduls aus dem CPAN[2] Archiv an. Es zeigte sich schon beim ersten Test mit wenigen Trainingsdaten, dass eine Klassifizierung von HTML-Dokumenten mittels dem Bayes'schen Klassifizierer möglich ist.

Um jedoch ein komplett annotiertes Suchformular zu erhalten, ist natürlich mehr notwendig als nur die Möglichkeit, automatisiert zu erfahren, ob eine Internetseite eine Lösung für eine gegebene Fragestellung beinhaltet. Ich habe mich dazu entschlossen, den Prozess in mehrere Teile aufzuteilen. Die ganze Seite, die einzelnen Formulare und die Formularelemente werden jeweils in einem eigenen Schritt klassifiziert. Dazwischen findet jeweils eine Aufteilung des Dokumentes in seine nächstkleineren Strukturen statt. Diese Aufteilung erfolgt in der ersten Version konventionell mit den Textanalysefunktionen von Perl, allen voran reguläre Ausdrücke. Die zweite Version verwendet ein Modul um HTML-Code in einen Baum umzuwandeln und findet die benötigten Teile in diesem Baum.

## 5.1. Komponenten

Das Experimentiersystem besteht aus einer Datenbank und einem Programm, welches über eine Web-Benutzerschnittstelle darauf zugreift.

[1] Eine Sammlung von Werkzeugen zur Textkategorisierung von Ken Williams
[2] Comprehensive PERL Archive Network, eine Sammlung von Modulen für die Programmiersprache Perl

Das Programm lässt sich grob in sechs Teile unterteilen. Der erste Teil, welcher angegebene Webseiten herunterlädt und in eine universell verwendbare Zeichencodierung[3] umwandelt, wird in Abschnitt 5.3.1 beschrieben. Ein weiterer Teil, beschrieben in den Abschnitten 5.3.3 und 5.3.4, kümmert sich um das Zerlegen der Dokumente in Unterdokumente, die im nächsten Schritt klassifiziert werden (Abschnitt 5.3.5). Je ein Teil kümmert sich um die Kommunikation mit der Datenbank und um die Darstellung der Benutzerschnittstelle. Ein weiteres Modul stellt Zugriffsfunktionen für die erstellten Klassifizierer zur Verfügung.

## 5.2. Datenbankstruktur

Die anfangs verwendete Methode der Speicherung der Daten direkt im Dateisystem stieß sehr bald an ihre Grenzen. So wurde die Struktur der Verzeichnisse und der Dateinamen bald unübersichtlich und ließ eine Speicherung der Zusammenhänge und Metainformationen nur über Umwege zu.

Als sinnvolle Alternative zum Dateisystem bot sich eine relationale Datenbank, im konkreten Fall PostgreSQL, an. Die Bedenken, dass die teilweise sehr großen Datenmengen pro Webseite die Datenbank überfordern könnten, wurden bei dem ersten Versuch bereits zerstreut. Die Tabellen sind so angelegt, dass alle Feldbezeichnungen in der ganzen Datenbank einzigartig sind. Dies verhindert Verwechslungen und erleichtert das Verknüpfen mehrerer Tabellen.

### 5.2.1. Die Tabellen im Detail

Die folgenden Abschnitte beschreiben die Felder der Tabellen und deren Verwendung. Zwischen den Tabellen gibt es folgende Zusammenhänge:

- Ein “data_type” gehört zu mehreren Einträgen[4] von “classifier”, “session”, “category”, “experiment” und “classification”. Einige der “data_type”-Referenzen sind redundant, aber helfen die Abfragen übersichtlich zu halten.
- Ein “experiment” gehört zu mehreren Einträgen von “corpus”, “classifier”, “session” und “classification”.

[3] UTF-8
[4] 1:n Relation

| Feld | Datentyp | Bedeutung |
|---|---|---|
| corpus_id | integer | Identifikationsnummer eines Textes |
| experiment_id | integer | Referenz auf ein Experiment |
| title | text | Titel, zum Beispiel die URL |
| data | text | Rohdaten |
| data_type_id | integer | Datentyp des Dokuments, siehe Abschnitt 5.2.1 |
| training | integer | Trainings oder Testdaten |
| description | text | eine Beschreibung des Datensatzes |
| corpus_ctime | timestamp | gibt den Erstellungszeitpunkt des Datensatzes an |
| id_string | text | Position im übergeordneten Dokument, siehe Abschnitt 5.2.1 |
| subelement_status | integer | Gibt es noch nichtklassifizierte Unterelemente? |
| parent_corpus | integer | Referenz zum übergeordneten Dokument. |

Tabelle 5.1.: Der corpus Table
In diesem Table werden klassifizierten Rohdaten gespeichert

- Der Table “corpus_category” stellt eine n:m Relation zwischen “corpus” und “category” her.

- Die Tabelle “classification” referenziert Einträge aus “session”, “corpus” und “classifier”.

#### Der corpus Table

Diese ist eine der umfangreichsten Tabellen. Hier werden die zu klassizierenden Rohdaten gespeichert, noch bevor sie einer Vorbehandlung (siehe Abschnitt 5.3.2) unterzogen werden. Der dafür verwendete Datentyp ist “text”, er speichert Texte ohne Beschränkung der Länge in dem von der Datenbank verwendeten Zeichensatz.

Das Feld “id_string” dient zur Orientierung. Es schlüsselt die Position des Teildokumentes im übergeordneten Dokument auf. Ein Beispiel: Ist das Teildokument ein Formular und der id_string “0”, so handelt es sich bei dem Teildokument um das erste Formular in dem übergeordneten Dokument, welches mit “parent_corpus” referenziert wird. Eine Auflistung und eine kurze Beschreibung der übrigen Felder befinden sich in Tabelle 5.1.

| Feld | Datentyp | Bedeutung |
|---|---|---|
| corpus_id | integer | Identifikationsnummer eines Textes |
| category_id | integer | Identifikationsnummer einer Kategorie |

Tabelle 5.2.: Der corpus_category Table
Dieser Table stellt eine n:m Relation zwischen den gespeicherten Rohdaten und den Kategorien her.

| Feld | Datentyp | Bedeutung |
|---|---|---|
| category_id | integer | Identifikationsnummer einer Kategorie |
| category_name | text | Kategoriebezeichnung |
| data_type_id | integer | der Datentyp der Kategorie, siehe Abschnitt 5.2.1 |

Tabelle 5.3.: Der category Table
Dieser Table speichert die Kategorien und den Datentyp, dem sie angehören.

#### Der corpus_category Table

In dieser Tabelle werden die Verknüpfungen von den Rohdaten zu den Kategorien hergestellt. Jedes Dokument kann einer oder mehreren Kategorien angehören. Die Zugehörigkeit zu mehreren Kategorien wird auch von AI::Categorizer (siehe Abschnitt 5.3.5) unterstützt, aber für das Experimentiersystem nicht verwendet. Jedoch wurde sie beim Entwurf der Datenbank berücksichtigt. Die Definition des Table ist in Tabelle 5.2 zu finden.

#### Der category Table

Hier wird einer numerischen Kategorienummer ein sprechender Name zugeordnet. Auch die Zuordnung zu einem bestimmten Datentyp (siehe Abschnitt 5.2.1) passiert hier. Tabelle 5.3 zeigt die genaue Definition des Tables im Datenbankschema.

#### Der classifier Table

Die erstellten Klassifikatoren sind instanzierte Objekte der Klasse “AI::Categorizer::Learner::NaiveBayes”. Das Feld “classifier” enthält Base64 codierte Binärdaten, welche ein solches Objekt repräsentieren. Um ein erstelltes Klassifikator-Objekt zu speichern, wird dessen Datenstruktur serialisiert und dann mit dem Algorithmus Base64 so codiert, dass daraus eine

| Feld | Datentyp | Bedeutung |
|---|---|---|
| classifier_id | integer | Identifikationsnummer eines Klassifizierers |
| classifier | text | der Klassifizierer selbst als Base64 encodierte Binärdaten |
| experiment_id | integer | Identifikationsnummer eines Experimentes, siehe Abschnitt 5.2.1 |
| classifier_description | text | eine kurze Beschreibung |
| data_type_id | integer | der Datentyp des Klassifizierers |
| classifier_ctime | timestamp | der Erstellungszeitpunkt |

Tabelle 5.4.: Der classifier Table
Dieser Table speichert die vom Experimentiersystem erstellen Klassifizierer und die dazugehörenden Informationen.

| Feld | Datentyp | Bedeutung |
|---|---|---|
| session_id | integer | Identifikationsnummer einer Testreihe |
| experiment_id | integer | das dazugehörige Experiment, siehe Abschnitt 5.2.1 |
| data_type_id | integer | der Datentyp |
| session_description | text | eine kurze Beschreibung der Testreihe |
| session_result | text | die zusammengefassten Ergebnisse |

Tabelle 5.5.: Der session Table
Dieser Table speichert allgemeine Informationen über eine Testreihe.

leicht portierbare Folge von ASCII-Zeichen[5] besteht. Die übrigen Felder sind in Tabelle 5.4 angeführt.

#### Der session Table

Für jede Testreihe werden in dieser Tabelle allgemeine Informationen gespeichert. Diese sind unter anderem eine kurze Beschreibung und die Ergebnisse. Die Tabledefinition befindet sich in Tabelle 5.5.

| Feld | Datentyp | Bedeutung |
|---|---|---|
| classification_id | integer | Identifikationsnummer einer Klassifizierung |
| experiment_id | integer | das dazugehörende Experiment, siehe Abschnitt 5.2.1 |
| session_id | integer | wird bei jedem Durchlauf neu vergeben |
| classifier_id | integer | der verwendete Klassifizierer |
| corpus_id | integer | der klassifizierte Text |
| guessed_category | integer | die geschätzte Kategorie |
| good | boolean | ist True, wenn die Kategorie erkannt wurde |
| classification_notes | text | nachträgliche Randbemerkungen |
| classification_scores | text | die erreichten Scores aller Kategorien |
| data_type_id | integer | der Datentyp |
| classification_ctime | timestamp | der Erstellungszeitpunkt |

Tabelle 5.6.: Der classification Table
Dieser Table speichert das Ergebnis einer einzigen automatischen Klassifikation.

| Feld | Datentyp | Bedeutung |
|---|---|---|
| data_type_id | integer | Identifikationsnummer des Datentyps |
| data_type_name | text | Name des Datentyps |

Tabelle 5.7.: Der data_type Table
Dieser Table speichert die Definition der unterschiedlichen Tiefen des Klassifizierens. Es gibt nur drei Einträge: das gesamte Dokument, ein einzelnes Formular und ein Formularfeld.

#### Der classification Table

Dieser Table ist vorgesehen, um eine große Zahl an automatischen Zuteilungen zu speichern, um die Qualität einzelner Klassifikatoren beurteilen zu können. Die Tabledefinition befindet sich in Tabelle 5.6.

#### Der data_type Table

Wie bereits in Kapitel 5 erwähnt, wird die Suche nach den gewünschten Eingabefeldern aufgeteilt. In den drei Einzelschritten werden unterschiedliche Datentypen klassifiziert. Im ersten Schritt sind das ganze Webseiten, sie erhalten die data_type_id "1". Die data_type_id "2" kennzeichnet einzelne Formulare und "3" schließlich ein einzelnes Element. Diese Unterscheidung

[5] American Standard Code for Information Interchange, beinhaltet Zahlen, Buchstaben ohne Umlaute und einige Sonderzeichen

| Feld | Datentyp | Bedeutung |
|---|---|---|
| experiment_id | integer | Identifikationsnummer des Experimentes |
| data_type_id | text | Identifikationsnummer des Datentyps |
| experiment_name | integer | Name des Experimentes |

Tabelle 5.8.: Der experiment Table
Dieser Table speichert Informationen über die durchführbaren Experimente

in allen maßgeblichen Tabellen ermöglicht es quasi, drei sehr ähnliche Zusammenhänge in denselben Tabellen zu speichern. Die Tabledefinition befindet sich in Tabelle 5.7.

#### Der experiment Table

Diese Tabelle dient dazu, Klassifikatoren einem Experiment zuordnen zu können. Alle Klassifikatoren eines Experimentes verwenden immer denselben Algorithmus zum Vorbereiten der Daten. Die Definition befindet sich in Tabelle 5.8.

## 5.3. Kernfunktionen des Systems

In diesem Abschnitt werden die Hauptfunktionen des Experimentiersystems beschrieben. Die dabei verwendeten, speziellen Datenstrukturen werden genauso wie das Zusammenspiel der Komponenten beschrieben.

### 5.3.1. Herunterladen der Dokumente per HTTP

Um eine einfache Möglichkeit zu haben, HTML-Dokumente von anderen Servern zu laden, wurde das Modul “easyHTTP” erstellt. Es bedient sich der CPAN Module “LWP::UserAgent” und “HTTP::Request” für die Kommunikation über http[6], sowie des Modules “Encode” um mit unterschiedlichen Zeichencodierungen zurechtzukommen.

Der Teil des Herunterladens ist sehr einfach und besteht ausschließlich darin, die beiden Module “LWP::UserAgent” und “HTTP::Request” zu verwenden. Siehe dazu Listing 5.1.

[6]HyperText Transfer Protocol

Listing 5.1: Herunterladen über HTTP

```
1 my $ua = new LWP::UserAgent;
2 my $request = HTTP::Request->new(GET => $_[0]);
3 my $response = $ua->request($request);
```

Im Vergleich dazu aufwändiger ist die richtige Behandlung der Zeichensatzcodierung. Es wird jene Zeichensatzcodierung verwendet, die im HTTP-Header angegeben ist. Werden dort mehrere Zeichencodierungen angegeben, so wird die letzte verwendet.

### Mögliche Fehlerzustände

Da diese Daten von externen Quellen kommen, muss damit gerechnet werden, dass darin Fehler und Inkonsistenzen vorliegen. Um auch bei falschen Angaben das bestmögliche Ergebnis zu erhalten, wurden folgende zwei Fehlerfälle behandelt.

**Falsche Zeichencodierung angegeben:** Bei Singlebytecodierungen wird dieser Fehler nicht erkannt und es werden die Zeichen falsch interpretiert, was zu Fehlern in der Darstellung führen kann.

Wird die Zeichencodierung der Daten als Multibytecodierung spezifiziert, so schlägt der Decodierversuch fehl, wenn die vorliegenden Daten eine Bytesequenz enthalten, welche in dieser Multibytecodierung nicht vorkommt. In diesem Fall wird die ungültige Zeichensequenz ignoriert und ausgelassen.

**Keine Zeichencodierung angegeben:** Enthält der Header keinen Hinweis auf eine Zeichencodierung, so wird versucht die Daten mit der Multibytecodierung UTF-8 zu interpretieren. Gelingt dies, so sind die Daten entweder in UTF-8 codiert oder es handelt sich um Daten im ASCII Zeichensatz, der wiederum exakt den ersten 127 Zeichen des Unicode-Zeichensatzes entspricht und in der UTF-8 Codierung keiner Konvertierung bedarf.

Misslingt die Interpretation als UTF-8, so wird eine Singlebytecodierung angenommen, die nicht ASCII ist. Die gewählte Codierung ist “Codepage 1252”. Diese Codierung wird von “Windows”-basierten Systemen häufig verwendet und enthält europäische Umlaute und Akzente sowie das Euro-Symbol.

Beide Fehler treten in der Praxis jedoch nur selten auf, denn sie machen sich auch durch eine fehlerhafte Darstellung der Webseite bemerkbar und werden in der Regel schnell gefunden und repariert.

Zur weiteren Verarbeitung der Texte wird das interne Format von Perl verwendet. Perl speichert die Zeichenketten als Unicode. Das bedeutet, jedes Zeichen wird durch eine Zahl repräsentiert. Diese Zahl kann fast beliebig groß sein. Erlaubt sind dabei nicht nur gültige Unicode-Codepunkte sondern beliebige Werte. Diese Werte abseits der im Unicode gültigen Codepunkte können aber bei der Ausgabe und bei der Kommunikation mit anderen Programmen zu Problemen führen. Bei diesem Experimentiersystem treten solche Werte allerdings nicht auf.

Die Erkennung der verwendeten Zeichencodierung erfolgt, indem bei allen angegebenen "content-type" Headern geprüft wird, ob diese auch eine "charset" Information beinhalten. Die gefundenen Zeichencodierungen werden in einer Hashmap mit der Anzahl der gefundenen Zeilen gespeichert. Diese Hashmap dient der Fehlersuche, falls es bei den Zeichencodierungen zu unvorhergesehenen Problemen kommt. Zusätzlich wird auch die jeweils letzte gefundene Zeichencodierung für die spätere Verwendung gespeichert.

Listing 5.2: Extraktion der Zeichencodierung aus dem Header

```
my @x = $response->header('content-type');
my %encodings;
for my $header_line (@x) {
  if ($header_line =~ /charset=\s*(\S+)/) {
    my $encoding = uc($1);
    $lastencoding = $encoding;
    if (exists($encodings{$encoding})) {
      $encodings{$encoding}++;
    } else {
      $encodings{$encoding} = 1;
    }
  } else {
    warn __PACKAGE__." Ignoring content-type: $header_line\n";
  }
}
```

Wurde nur eine Zeichencodierung erkannt, so ist nichts weiter zu tun als die heruntergeladenen Daten mit dieser zu interpretieren. Wurden jedoch mehrere Zeichencodierungen gefunden, so wird eine Warnung ausgegeben und die zuletzt gefundene Zeichencodierung verwendet. Beim

Fehlen jeglicher Angaben zur Zeichencodierung wird erst versucht, die Daten als UTF-8 zu interpretieren. Gelingt dies nicht[7], so werden die Daten als “Codepage 1252” interpretiert.

Listing 5.3: Verwendung der erkannten Zeichencodierung und Fehlerbehandlung

```
1  my $content = $response->content;
2  if (keys(%encodings) == 1) {
3    $content = decode($lastencoding, $content);
4  } elsif (keys(%encodings) > 1) {
5    warn __PACKAGE__." more than one encodings found, using $lastencoding\n"
         ;
6    warn __PACKAGE__.':'.__LINE__.$".
7      Data::Dumper->Dump([\%encodings], ['encodings']);
8    $content = decode($lastencoding, $content);
9  } else {
10   warn __PACKAGE__." no encoding found, trying to use utf8\n";
11   my $newcontent = '';
12   eval {$newcontent = decode_utf8($content ,Encode::FB_CROAK);};
13   if ($@) {
14     warn __PACKAGE__." failed to use utf8, using cp1252\n";
15     $newcontent = decode('cp1252',$content);
16   }
17   $content = $newcontent;
18 }
19 return $content;
```

### 5.3.2. Vor dem Klassifizieren

Die Klassifikatoren wurden für Texte in natürlichen Sprachen entwickelt und beachten daher keine Sonderzeichen. Die Beschreibungssprache HTML verwendet ihrerseits auch Wörter für die Strukturierung des Inhaltes, welche zum Teil aus natürlichen Sprachen entnommen wurden. Diese werden im Klassifikator gleich behandelt wie die Wörter aus dem Inhalt.

Das bedeutet, aus einem strukturierten HTML-Dokument

Listing 5.4: Strukturiertes HTML-Dokument

```
1 <html>
2   <head>
3     <title>Tablemanners explained</title>
```

[7] Das Argument “Encode::FB_CROAK” der Funktion “decode_utf8” bewirkt, dass die Funktion mit einem Fehler abbricht, sobald eine ungültige Bytesequenz entdeckt wird.

```
  </head>
  <body>
    <h1>Stuff on the Table</h1>
    There are a lot of things on a table.
    Here is a description of what they are for:
    <table>
        <tr><th>Thing</th><th>Application</th></tr>
        <tr><td>Spoon</td><td>Soup</td></tr>
        <tr><td>Fork</td><td>Holding things</td></tr>
        <tr><td>Knife</td><td>Cutting things</td></tr>
        <tr><td>Wineglass</td><td>Wine</td></tr>
        <tr><td>Cup</td><td>Coffee</td></tr>
    </table>
  </body>
</html>
```

wird eine Menge von Wörtern.

Listing 5.5: Text ohne Strukturinformationen

```
html
head
title Tablemanners explained title
head
body
h1 Stuff on the Table h1
There are a lot of things on a table.
Here is a description of what they are for
table
tr th Thing th th Application th tr
tr td Spoon td td Soup td tr
tr td Fork td td Holding things td tr
tr td Knife td td Cutting things td tr
tr td Wineglass td td Wine td tr
tr td Cup td td Coffee td tr
table
body
html
```

Für den Algorithmus selbst ist dann nur mehr die Anzahl der Wörter relevant, wobei Groß- und Kleinschreibung ignoriert werden. In der folgenden Zusammenstellung wurden nur einmal vorkommende Wörter bereits weggelassen:

```
20 td
```

```
12 tr
4 th
3 a
3 table
2 things
2 html
2 head
2 title
2 body
2 h1
2 on
2 are
```

Je einmal kommen die Wörter “tablemanners, explained, stuff, the, There, lot, of, things, Here, is, description, what, they, for, Thing, Application, Spoon, Soup, Fork, Holding, Knife, Cutting, Wineglass, Wine, Cup, Coffee” vor.

An diesem Beispiel erkennt man gut, dass die inhaltlich interessanten Wörter die niedrigste Häufigkeit und damit auch die niedrigste Wahrscheinlichkeit haben. Die häufigsten Wörter sind Steuerbegriffe aus dem HTML. Teilweise werden auch Wörter aus dem HTML und dem Inhalt zusammengezählt, da sie gleich lauten.

Um diesem Umstand abzuhelfen gibt es die Möglichkeit, beliebige Vorfilter zu implementieren. Für die Klassifikation ganzer Webseiten ist das noch nicht notwendig. Das funktioniert auch mit dem Rohtext gut genug.

### Implementierung

Diese Vorfilter wurden im Modul “preFilter.pm” umgesetzt. Die Implementierung ist sehr einfach. Die Methode “filter” erhält den zu filternden Text und die zu verwendenden Filter als Parameter und gibt den gefilterten Text zurück. Die Filter werden in der Reihenfolge ausgeführt, in der sie übergeben wurden.

Listing 5.6: Das Grundgerüst der “filter” Methode label

```
my ($filter_module, $text) = @_;

if (0) {
} elsif ($filter_module eq 'plain') {
  return $text;
...
```

```
7 } else {
8   return $text;
9 }
```

Die einzelnen Filter und deren Verwendung werden mit Beispielen im nächsten Abschnitt beschrieben.

#### Die Filter im Detail

**plain:** Dieser Filter gibt die Daten unverändert wieder. Er wurde implementiert, um die Effektivität der anderen Filter einfacher beurteilen zu können.

**tag_tags:** Um den Tags, den Steuerelementen der HTML-Sprache, gegenüber dem Text eine höhere Bedeutung zukommen zu lassen und um diese von gleichnamigen Wörtern unterscheiden zu können, wurde dieser Filter implementiert. Er stellt den Tagnamen die Zeichenkette "tag_" voran. Aus "<table>" wird dann "<tag_table>" beziehungsweise "tag_table" in der weiteren Verarbeitung.

**no_tags:** Es bestand die Vermutung, dass die Tags das Ergebnis negativ beeinflussen und bessere Ergebnisse zu erzielen wären, wenn man sie vor dem Klassifizieren aus dem Text entfernen würde. Dieser Filter wurde implementiert, um dieser Vermutung nachzugehen. Er entfernt alle Tags aus dem HTML-Code.

**tla_enhance:** Flughafenbezeichnungen werden oft als Kombinationen aus drei Buchstaben angegeben. Solche IATA-Kürzel[8] sind zum Beispiel: "GRZ" für Graz, "VIE" für Wien oder "DUB" für Dublin. Dieser Filter setzt neben eine Kombination aus drei Buchstaben das Wort "threeLetterMark". Die Motivation für diesen Filter ergibt sich aus der Beobachtung, dass ein Eingabefeld mit fixen Werten und Gruppen aus drei Buchstaben auf einer Flugsuchseite generell dem Abflug- oder Zielflughafen zugeordnet ist. Textklassifikatoren erkennen den gemeinsamen Sinn dieser Kürzel nicht und profitieren daher von der Unterstützung durch diesen Filter.

Die Funktionsweise sei am Beispiel eines Auszuges der Flugsuchseite der Fluglinie Aer Lingus verdeutlicht.

[8] International Air Transport Association

Aus:

```
<option value="DUB">Dublin (DUB)
</option>
<option value="DBV">Dubrovnik (DBV)
</option>
<option value="DUS">Dusseldorf (DUS)
</option>
```

wird:

```
<option value="DUB threeLetterMark ">Dublin (DUB threeLetterMark )
</option>
<option value="DBV threeLetterMark ">Dubrovnik (DBV threeLetterMark )
</option>
<option value="DUS threeLetterMark ">Dusseldorf (DUS threeLetterMark )
</option>
```

**month_enhance:** Monate werden oft in den unterschiedlichsten Arten dargestellt. Für einen Menschen ist es leicht, diese zu erkennen und den Zusammenhang herzustellen. Die Klassifikatoren jedoch können dies nicht, sie behandeln die Wörter “Juli”, “July” und “jul” sowie alle anderen Bezeichnungen und Kürzel für Monate als eigenständige Wörter, die miteinander in keinem Zusammenhang stehen. Um diesem Umstand beizukommen, ergänzt dieser Filter alle erkannten Bezeichnungen und Abkürzungen für Monate durch das Wort “monthMark”.

Als Beispiel für die Funktionsweise dient wie oben ein Auszug aus der Seite der Aer Lingus.

Aus:

```
<option value="6">July</option>
<option value="7">August</option>
<option value="8">September</option>
```

wird:

```
<option value="6"> monthMark July</option>
<option value="7"> monthMark August</option>
<option value="8"> monthMark September</option>
```

Der reguläre Ausdruck zum Finden der Monatsbezeichnungen lautet:

```
our $month = qr/\b(?:jan|feb|mar|apr|may|jun|jul|aug|sep|oct|nov|dec)/i;
```

Damit werden alle englischsprachigen Monate gefunden und auch einige Monatsnamen in anderen Sprachen. Für die Effektivität dieses Filters ist es nicht notwendig, dass alle Monatsbezeichnungen gefunden werden. Es genügt, wenn einige gefunden werden, um die Häufigkeit des Wortes "monthMark" in diesem Teil des Textes signifikant zu erhöhen.

**element_enhance:** Dieser Filter kombiniert die beiden oben genannten Filter. Er dient als Sammelfilter für alle elementspezifischen Filter.

### 5.3.3. Das Aufteilen

Die Webseite wird in drei Schritten klassifiziert. Zwischen den Schritten findet eine Aufteilung der Daten statt. Erst wird die ganze Seite in ihre Forms unterteilt, und diese werden danach in die Eingabeelemente zerlegt. Bei diesen Vorgängen kommen hauptsächlich reguläre Ausdrücke von Perl zum Einsatz.

#### Splitten in Forms

Das Aufteilen der ganzen Seite in einzelne Forms ist sehr einfach. Die Seite wird nach öffnenden und schließenden <form> Tags durchsucht und an diesen Tags geteilt. Das Ergebnis ist eine Liste von einer ungeraden Anzahl von Teilen, in der die Teile an einer geraden Stelle jeweils ein komplettes Formular enthalten. Die Textteile an ungeraden Stellen sind HTML-Code außerhalb der Formulare. Ein einfaches Dokument mit nur einem Formular ergibt nach dieser Aufteilung eine Liste aus drei Elementen. Das erste Element ist der Teil des HTML-Codes vor dem Formular, das zweite das Formular selbst und der dritte Teil beinhaltet den Rest des Dokumentes.

Die Aufteilung erfolgt mit dieser Zeile:

```
1 my @html_chunks = split /(<form.*?<\/form>)/is, $html;
```

Das Kommando "split" teilt einen Text an den in dem regulären Ausdruck vorkommenden Stellen. Wird der reguläre Ausdruck in Klammern gesetzt, so ist auch der Text, auf den der Ausdruck passt, ein eigener Teil der Ergebnisliste.

Um die Formularteile von den Restteilen zu trennen, wird die resultierende Liste in zwei Listen aufgeteilt, wobei je ein Element abwechselnd in die Liste der Restelemente und in die Liste der Formularelemente kopiert wird.

Der Code dafür ist sehr einfach:

```
my $flipflop = 1;
for my $chunk (@chunks) {
    if ($flipflop ^= 1) { # x xor 1 = !x
        push @form_chunks, $chunk;
    } else {
        push @rest_chunks, $chunk;
    }
}
```

Zur weiteren Verarbeitung werden nur die Formularelemente herangezogen. Es wäre auch möglich, die Restelemente mit in die Klassifizierung einzubeziehen, zum Beispiel das einem Formularelement jeweils direkt vorangestellte Restelement. Es stellte sich aber heraus, dass diese Maßnahme bei der Klassifizierung der Forms nicht notwendig ist.

### Splitten in Elemente

Das Splitten der Forms in die einzelnen Elemente ist etwas komplexer als das Aufteilen in Formulare, da die Elemente durch unterschiedlichste Tags gebildet werden. Zudem gibt es Elemente, die aus einem einzelnen Tag bestehen, und auch solche, die andere Inhalte einklammern. Bei letzteren wird auch der gesamte Inhalt verwendet. Ein besonderer Typ sind "Radiobuttons", welche aus mehreren einzelnen Tags bestehen, die nur durch ihr "name"-Attribut als zusammengehörend erkannt werden.

Der Algorithmus arbeitet rekursiv. Das ist notwendig, da Tags fallweise gruppiert und nachgefasst werden müssen. Die Funktionsweise kann im Groben wie folgt umrissen werden:

- Der Text wird nach Feldern durchsucht.
- Wenn notwendig, werden Teile nachgefasst.
- Der Vorgang wird mit dem Rest wiederholt.

Der reguläre Ausdruck für das Auffinden der Tags wird in mehreren Stufen zusammengesetzt. Die Namen der Tags werden in Arrays gespeichert und dann mittels des Alternative-Zeichens "|" zu einem regulären Ausdruck zusammengesetzt. Dies passiert für Einzeltags und klammernde Tags getrennt. Anschließend werden diese noch von entsprechenden regulären Ausdrücken umgeben, so dass auch die Spitzklammern beziehungsweise der schließende Tag mit einbezogen werden. Der Ausdruck für klammernde Tags ist anfällig für fehlerhaftes HTML,

da ein geöffneter Tag von jedem beliebigen schließendem Tag aus der Liste geschlossen werden kann. Dieser Fehler kann jedoch erst mit dem Regex[9] Engine von Perl 5.10 behoben werden. Ab dieser Version gibt es “Named Captures”. Bei den numerischen Captures von Perl 5.8.8 würden die Indizes der erkannten Textstellen verschoben werden, würde man Back-References einsetzen, um einen gleichnamigen geschlossenen Tag zu erkennen.

Der Code für das Erstellen der Suchmuster ist nicht sehr kompliziert:

```
1    # single tags
2    my $single_tags = join '|', qw(input button);
3    # spanning tags
4    my $spanning_tags = join '|', qw(button textarea select);
5
6    my $single_tags_regex = "<(?:$single_tags).*?>";
7    my $spanning_tags_regex = "<(?:$spanning_tags).*?".
8                              "<\/(?:$spanning_tags)>";
9
10   my $regex = "^(.*?)($single_tags_regex|".
11                 "$spanning_tags_regex)(.*)";
```

Zur Anwendung kommt diese Regex beim rekursiven Aufruf der Splitter-Routine. Akzeptiert die Regex den bei dem Aufruf übergebenen Text, so teilt sie ihn in drei Teile auf:

- Der Text vor dem Element
- Das Element selbst
- Der Text nach dem Element inklusive aller eventuell noch folgenden Elemente

Die beiden ersten Teile werden in einem Array gespeichert. Der dritte Teil mit dem restlichen Text wird derselben Methode wieder als Argument übergeben. Das Array, welches der rekursive Aufruf zurückliefert, wird an das eigene angehängt.

Akzeptiert jedoch die Regex den Text nicht, dann bedeutet das, es gibt keine weiteren Elemente mehr. In diesem Fall wird der gesamte Text in das eigene Array gespeichert. Auch findet kein weiterer rekursiver Aufruf statt, dieser Fall stellt die Abbruchbedingung der Rekursion dar.

Das eigene Array wird am Ende der Methode an das aufrufende Programm zurückgegeben. Das Programmgerüst der rekursiven Methode lässt sich auf folgende Zeilen zusammenkürzen:

[9] Mit den Mustererkennungsmethoden von Perl ist mittlerweile schon mehr möglich als mit normalen regulären Ausdrücken, deswegen nennt man die entsprechende Engine auch “Regex-Engine” und meint damit nicht mehr “Regular Expressions Engine”.

```
    if ($html =~ /$regex/is) {
        my ($no_element, $form_element, $rest) = ($1, $2, $3);
        push @html_chunks, $no_element;
        # follow up detection
        push @html_chunks, $form_element,
            $self->split_elements($rest);
    } else {
        push @html_chunks, $html;
    }
    return @html_chunks;
}
```

In der Zeile mit dem Kommentar “follow up detection” passiert die oben angesprochene Behandlung von mehrteiligen Eingabefeldern. Der größte Teil der Methode dient dieser Nachbehandlung. Zuerst wird mit Hilfe einer Regex abgefragt, ob es sich bei dem gefundenen Tag um einen der Tags handelt, welche aus mehreren Teilen bestehen. Ist dies der Fall, so wird das “name”-Attribut des Tags ermittelt. Es wird dazu benötigt, zusammengehörende, also gleichnamige Tags im weiteren Text zu finden. Wird das Attribut nicht gefunden, bricht die Suche ab. Dies sollte in der Praxis nicht vorkommen, da dieses Attribut zum Funktionieren des Formulars notwendig ist, aber trotzdem sind solche HTML-Seiten nicht auszuschließen.

Die nächsten Schritte sind das Holen des nächsten Tags, eine Überprüfung, ob etwas gefunden wurde, und die Überprüfung, ob das Gefundene zu dem ersten gefundenen Tag passt. Dazu werden die Attribute “name” und “type” verglichen. Alles das passiert in der Durchlaufbedingung der “while”-Schleife. In der Schleife selbst wird vom Resttext das Stück vom Anfang bis zum Ende des gefundenen Tags abgeschnitten und an den Text des ersten gefundenen Tags angehängt.

Dieser Vorgang wird nun so oft wiederholt, bis ein Eingabefeld eines anderen Typs oder mit einem anderen Namen gefunden wird. Es werden sozusagen so lange Textteile vom Rest in den Text des aktuellen Formularelementes übernommen, bis es keine passenden Textteile mehr gibt.

Der Programmcode für diesen Teilbereich ist etwas komplexer:

```
if ($form_element =~
    /^<input[^>]*type=\W?($multiple_input_types)/is) {
# we have to fetch more
    my $type = $1;
    if ($form_element =~
        /^<input[^>]*name=(?:(\W)(.+?)\1|([^\s>]+))/is) {
```

```
7     # does it have a name (it should have, but who knows).
8         # $1 quotation string, used as a backreference.
9         # $2 the actual name.
10        # we match either on a quoted name
11        # (any non_word char may be used as a quotation)
12        # and a name which may consist of
13        # any chars but the quotation char
14        # or an unquoted name which may consist
15        # of any char but '>' and whitespaces.
16        my $name = $2;
17        my $next_input_tag;
18        while ($rest =~ /(<input.*?>)/is
19                # fetch the next input tag
20                && defined($next_input_tag = $1)
21                # 0 and '' are interpreted as 'false'.
22                # we don't want this to be false.
23                # and, yes, this is quite perlish. :)
24                && $next_input_tag =~ /type=\W?$type/is
25                && $next_input_tag =~ /name=\W?$name/is) {
26            $rest =~ s/^(.*?\Q$next_input_tag\E)(.*)$/$2/s;
27            # get all up to the first additional group element
28            # and remove it from the rest
29            $form_element .= $1;
30            # add the element to the current element.
31        }
32
33    } else {    # input tag does not have a name.
34                # this should not happen.
35    }
36 }
```

Dieser Teil des Systems ist empfindlich gegenüber fehlerhaftem HTML. Es ist zwar nicht zu erwarten, dass es zu Programmfehlern kommt oder gar Sicherheitslöcher entstehen, aber das Ergebnis kann speziell bei ineinander verflochtenen Tags unerwartet ausfallen. Jedoch können solche Strukturen auch von Browsern nicht mehr richtig verarbeitet werden.

### 5.3.4. Das Aufteilen Version 2

Im Zuge der weiteren Entwicklung wurde deutlich, dass die im letzten Abschnitt beschriebene Methode des Aufteilens für die Aufgabe, die ermittelten Ergebnisse mit anderen Systemen

zu teilen, nicht geeignet war. Ein gefundenes Element muss nicht nur eine Kategoriezuordnung besitzen, sondern muss auch von einem anderen System im Dokument wiedergefunden werden können. Dazu dient der XPath.

Der XPath orientiert sich an der Baumstruktur eines HTML-Dokumentes. Es ist daher nicht mit vertretbarem Aufwand möglich, den XPath für einen gegebenen Teilbereich des HTML-Dokumentes zu ermitteln.

Es ist sinnvoller, den XPath schon beim Aufteilen zu generieren und danach mit dem entsprechenden Teil zu speichern. Aus diesem Grund wurde der Splitter neu entwickelt. Um mit den bisherigen Daten kompatibel zu bleiben, wurde der Splitter beibehalten und der neue Algorithmus als Splitter2 implementiert.

#### Grundsätzliche Funktionsweise

Aus dem HTML-Dokument wird mittels des Moduls “HTML::TreeBuilder” eine Baumstruktur erstellt. Diese wird rekursiv verarbeitet und dabei der XPath eines jeden Knotens erstellt.

Entspricht ein Knoten einem gewünschten Suchmuster, das heißt ist er ein Formular oder ein Formularelement, so wird er mit allen Unterknoten in Textform als einzelne Zeichenkette und mit seinem XPath versehen separat gespeichert.

#### Implementierung

Bei der Implementierung waren die bei dem Modul “Splitter” gewonnen Erkenntnisse sehr hilfreich und beschleunigten die Fertigstellung.

**Konstruktor: “new”** Dem Konstruktor wird das HTML-Dokument in Textform übergeben. Daraus wird mittels “HTML::TreeBuilder” der HTML-Baum erstellt.

**Initialisierung der Rekursion: “generate_element_lists”** Diese Methode hat keine Parameter. Sie dient lediglich dem ersten Aufruf der Rekursionsmethode mit den Initialwerten sowie der Erstellung der für die Rekursion notwendigen Variablen.

Die Initialisierungen im Detail:

```
my $in_group = 0;
my $group_name = '';
my $group_element_container_ref = undef;
$self->generate_recursive_element_lists($self->{root}, '', undef, \
    $in_group, \$group_name, \$group_element_container_ref);
```

**Die Rekursion selbst: "generate_recursive_element_lists"** In dieser Methode wird der Baum rekursiv abgearbeitet, wobei jeder Ast erst bis zu seinem tiefsten Element durchsucht wird, ehe zum nächsten Ast übergegangen wird.

Der erste Teil der Methode hängt an den bisherigen XPath die Identifikation des aktuellen Elementes. Dazu wird der Name des Elementes herangezogen und soweit vorhanden auch die Attribute "name", "id" und "type". Es wird auch stets eine zweite Variante des XPath aus der Elementadresse mit Hilfe der Methode "addr2path" erstellt.

Beide Varianten des XPath werden gespeichert. Die erste Variante benennt die Elemente auf dem Weg zum aktuellen Element, soweit vorhanden auch mit gewissen Attributen. Hier ein Beispiel:

```
/html/body/table/tr/td/table/tr/td/table/tr/td/table/tr/td/table/tr/td/
    table/tr/td/form[@id='RequestAirForm'][@name='RequestAirForm']/input[
    @type='hidden'][@name='page']
```

Die zweite Variante wird aus der numerischen Adresse erstellt und richtet sich nach der Reihung der Unterelemente eines Elementes. Zum Beispiel bedeutet die Adresse "4.0.1": "Das fünfte Unterelement des Wurzelelementes, von diesem das erste Unterelement und von diesem wiederum das zweite.". Die Zählung der Adresse beginnt mit null, die des XPaths bei eins. Ein Beispiel eines solchen XPath:

```
/*[1]/*[2]/*[1]/*[1]/*[2]/*[1]/*[4]/*[10]/*[1]/*[1]/*[1]/*[6]/*[1]/*[2]/*[3]
```

Danach folgen die Einzelteile des Algorithmus zur Extraktion der gewünschten Elemente. Das beginnt sehr einfach mit dem Beenden einer möglicherweise begonnenen Gruppe von zusammengehörenden Elementen wie zum Beispiel einer Radiobutton-Gruppe beim Verlassen eines "form"-Elementes.

```
if (!$current_form) {
    $$in_group = 0;
}
```

Die Variable “$current_form” ist im Baum nur unterhalb eines “form”-Elementes gesetzt, sie wird als Wert an weitere rekursive Aufrufe übergeben und kann auf der momentanen Ebene nicht von darunterliegenden Abläufen verändert werden. Die Variable “$in_group” jedoch ist eine Referenz auf eine Variable, diese behält ihren Wert für alle nachfolgenden Verwendungen im Baum bei und muss daher zurückgesetzt werden, sobald die Verarbeitung aus einem “form”-Element auftaucht. Der Einfachheit halber wird sie bei jedem Verarbeitungsschritt zurückgesetzt, der nicht in einem Formular stattfindet.

Darauf folgt die Behandlung eines gefundenen “form”-Elementes. Dieses wird mit beiden Varianten seines XPath in ein Element_Container-Objekt gespeichert. Dieser Container wird mit einer Referenz auf eine noch leere Liste der Eingabefelder in einem Hash gespeichert. Die “$current_form” enthält dann eine Referenz auf diesen Hash. Als letzter Schritt wird diese Referenz noch der Liste der gefundenen Formulare hinzugefügt.

```
1 if ($tagname eq 'form' && !$current_form) {
2     $element_container->set_element($element, $path, $num_path);
3     $current_form = {
4         element_container => $element_container,
5         subelements => [],
6     };
7     push @{$self->{forms}}, $current_form;
8 }
```

Der letzte Teil dieser Methode verarbeitet einzelne Formularfelder. Entspricht der Tagname dem eines Formularfeldes und befindet sich der Algorithmus innerhalb eines Formulares, so werden die Informationen des Feldes ausgewertet.

Das bedeutet im Detail:

- Befindet sich der Algorithmus in einer Gruppe, hat das gefundene Feld ein “name”-Attribut und ist dieses ungleich des Namens der momentanen Gruppe, so endet die Gruppe und die entsprechenden Variablen werden zurückgesetzt.
- Ist das gefundene Element eines, welches einer Gruppe angehört und es ist momentan keine Gruppe aktiv, so werden die globalen Variablen für die Gruppen mit den passenden Werten initialisiert.
- Am Ende dieses Teils wird das gefundene Feld in die entsprechende Liste gespeichert. Dabei wird unterschieden, ob es sich um ein Feld in einer Gruppe handelt oder ob das Feld für sich alleine steht.

```
if ($tagname =~ /$element_regex/i && $current_form) {
    if ($$in_group
            && defined($name)
            && ($name ne $$group_name)) { # ende einer gruppe.

        $element_container = new Element_Container;
        $$group_element_container_ref = undef;
        $$in_group = 0;
        $$group_name = '';
    }
    if ($tagname eq 'input'
        && $element->attr('type') =~ /$multiple_input_types/i
        && !$$in_group) { # start einer gruppe

        $$group_element_container_ref = $element_container;
        $$in_group = 1;
        $$group_name = $name;
    }
    if ($$in_group) {
        $element_container->add_group_element($element,1, $path, $num_path
            );
    } else {
        $element_container->set_element($element, $path, $num_path);
        push @{$current_form->{subelements}}, $element_container;
    }
} elsif ($$in_group) {
    $element_container->add_group_element($element,0, $path, $num_path);
}
```

Ist das momentane Element kein Feld, aber befindet sich der Algorithmus in einer Gruppe, so wird das Element der Gruppenliste hinzugefügt.

```
for my $subelement ($element->content_list()) {
    $self->generate_recursive_element_lists($subelement, $path,
        $current_form, $in_group, $group_name,
        $group_element_container_ref)
        if (ref $subelement eq 'HTML::Element');
}
```

**Getter für die Liste der Formulare: "get_forms"** Diese Methode liefert den Inhalt der Objektvariablen "forms" zurück, eine Liste der gefundenen Formulare.

**Umwandlung einer Adresse in einen XPath: “addr2path”** Für die Umwandlung der numerischen Adressen in einen entsprechenden XPath wurde eine eigene kurze Methode implementiert.

Listing 5.7: Splitter2::addr2path

```
1 sub addr2path { # 0.2.10.0 -> /*[1]/*[3]/*[11]/*[1]
2     my $self = shift;
3     my $addr = shift;
4     $addr =~ s/\.*(\d+)/'\/*['.($1+1).']'/ge;
5     return $addr;
6 }
```

Die eigentliche Arbeit passiert nur in der vierten Zeile. Hier wird jede Folge aus beliebig vielen Punkten gefolgt von einer Zahl ersetzt durch einen Slash, einen Stern und der vorher gefundenen Zahl plus eins in eckigen Klammern. Die Adressen beginnen bei Null mit der Zählung, der XPath aber mit Eins.

## 5.3.5. Das Klassifizieren

Im Experimentiersystem werden die Daten mit dem AI::Categorizer Framework klassifiziert. Um die Verarbeitung mehrerer, mit unterschiedlichen Daten trainierten Klassifikatoren zu erleichtern, habe ich mich dazu entschlossen, diese in einer Datenbank zu speichern. Weiters habe ich ein Perl-Modul entwickelt, welches mehrere Klassifikatoren im Speicher behält und diese auch je nach Bedarf nachlädt.

### Das Modul Classifier_Container

In diesem Modul werden Klassifizierer trainiert, in die Datenbank gespeichert und wieder von dieser in den Speicher geladen. Dieses Modul kümmert sich auch um die Verarbeitung der Trainingssets.

Ein Classifier_Container beinhaltet jeweils einen Klassifizierer. Der Konstruktor erfordert zumindest die Angabe einer “data_type_id”. Durch diese wird festgelegt, welchen Typ von Daten dieser Klassifizierer behandelt. Auch die Angabe einer “experiment_id” ist notwendig. Sie ordnet den neuen Klassifikator einem bestimmten Experiment zu.

Wird statt der “experiment_id” eine “classifier_id” angegeben, so wird der Klassifikator mit dieser ID aus der Datenbank geladen.

Die Methoden dieses Moduls sind:

**fetch_trainingset:** Diese Methode holt die Trainingsdaten aus der Datenbank und speichert sie in einer Objektvariable. Es können alle relevanten Daten geladen werden oder nur solche, die explizit als Trainingsdaten markiert sind.

Listing 5.8: Classifier_Container::fetch_trainingset

```
sub fetch_trainingset {
    my $self = shift;
    my $all = $_[0];
    my $training_filter = '';
    my $dbh = $self->{dbh};
    unless ($all) {
        $training_filter = ' and training = 1';
    }
    my @datarows = $dbh->qrh("select * ".
        "from corpus natural join corpus_category natural join category ".
        "where data_type_id = $self->{data_type_id} ".
#       "and experiment_id = $self->{experiment_id} ".
        "$training_filter;");
    $self->{fetched} = 1;
# warn __PACKAGE__." got ".@datarows." datarows\n";
    $self->{corpus} = \@datarows;
}
```

**build_knowledge_set:** Diese Methode benötigt grundsätzlich keine Parameter. Ihr Zweck ist es, aus einem geholten Trainingsset ein sogenanntes KnowledgeSet zu erstellen. Aus den Dokumenten und Kategorien werden entsprechende Objekte des "AI::Categorizer"-Frameworks erstellt und diese dann dem Konstruktor des KnowledgeSets übergeben. Dieses wird wiederum im Classifier_Container Objekt gespeichert. Wurde zuvor kein Trainingsset geladen, so wird ein Fehler generiert.

Wurde eine Dokumentnummer übergeben, so wird dieses Dokument nicht in das KnowledgeSet aufgenommen sondern stattdessen als Document-Objekt zurückgegeben. Diese Funktion wird beim Ermitteln der Fehlerraten verwendet. Wurden Vorfilter gesetzt, so werden diese auch in dieser Methode auf die Daten angewendet. Das fertige KnowledgeSet wird wieder in einer Objektvariable gespeichert.

Listing 5.9: Classifier_Container::build_knowledge_set

```
sub build_knowledge_set {
    my $self = shift;
    my $exclude = shift;
    my $excluded_doc_obj;
    my $filter = new preFilter;
    if ($self->{fetched}) {
        my %categories;
        my %documents;
        for my $datarow (@{$self->{corpus}}) {
            $categories{$datarow->{category_name}} =
                AI::Categorizer::Category->by_name(name => $datarow->{
                    category_name})
                unless (exists($categories{$datarow->{category_name}})
                    );
            my $data = $filter->filter(
                        $datarow->{data},
                        split (/ /, $self->{preprocessor})
                    );
            if (!defined($exclude) || ($exclude != $datarow->{corpus_id}))
                 {
                $documents{$datarow->{corpus_id}} =
                    AI::Categorizer::Document->new(content => $data)
                        unless (exists($documents{$datarow->{corpus_id}}))
                            ;
                $categories{$datarow->{category_name}}->add_document(
                    $documents{$datarow->{corpus_i
d}});
                $documents{$datarow->{corpus_id}}->add_category(
                    $categories{$datarow->{category_nam
e}});
            } else {
                $excluded_doc_obj =
                    AI::Categorizer::Document->new(content => $data)
                        unless (defined($excluded_doc_obj));
                $excluded_doc_obj->add_category($categories{$datarow->{
                    category_name}});
            }
        }
        my $knowledge_set = AI::Categorizer::KnowledgeSet->new( verbose =>
             0,
                categories => [values %categories],
```

```
                documents => [values %documents]);
        $self->{knowledge_set_built} = 1;
        $self->{knowledge_set} = $knowledge_set;
        return $excluded_doc_obj || 1;
    } else {
        warn "trainingset not fetched";
    }
}
```

**train_classifier:** Die Methode "train_classifier" erstellt mit dem KnowledgeSet einen funktionierenden Klassifikator. Um auch aus mehreren Klassifikationsalgorithmen wählen zu können, wurde ein Teil des Programmcodes als Zeichenkette zusammengestellt und wird erst zur Laufzeit ausgewertet. Diese Art der Programmierung birgt das potentielle Risiko, dass unbekannter Code ausgeführt wird. Diesem Umstand wird begegnet, indem der veränderbare Teil des Codestückes unmittelbar vor der Ausführung auf seine Plausibilität überprüft wird.

Listing 5.10: Classifier_Container::train_classifier

```
sub train_classifier {
    my $self = shift;
    if ($self->{knowledge_set_built} && $self->{classifier_class}) {
        my $classifier;
        my $class = $self->{classifier_class};
        if ($allowed_classes{$class}) {
            $classifier = eval "AI::Categorizer::Learner::${class}->new(
                verbose => 0 )";
            warn "error: $@" if $@;
        } else {
            warn "invalid class: $class";
            return 0;
        }
        $classifier->train( knowledge_set => $self->{knowledge_set} );
        $self->{classifier_trained} = 1;
        $self->{classifier} = $classifier;
    } else {
        warn "knowledge_set not built";
    }
    return 1;
}
```

**get_classifier:** Diese Methode gibt den im Classifier_Container enthaltenen Klassifikator als Objekt zurück.

**get_classifier_id:** Diese Methode gibt die ID des Klassifikators als Zahl zurück.

**store_classifier:** Diese Methode speichert den Klassifikator in der Datenbank. Dazu wird das Klassifikator-Objekt serialisiert. Das bedeutet, die interne Datenstruktur mit allen Datentypen wird derart auf einer einzelnen Skalarvariable abgebildet, dass daraus zu einem späteren Zeitpunkt wieder ein funktionsfähiger Klassifikator mit genau denselben Eigenschaften erstellt werden kann. Der Inhalt dieser Skalarvariable sind Binärdaten. Diese werden mit einer Base64-Codierung[10] in eine Reihe von Textzeichen umgewandelt und anschließend in der Datenbank gespeichert.

**load_classifier:** Die "load_classifier"-Methode ist das Gegenstück zur oben erklären Methode zum Speichern der Klassifikatoren in der Datenbank. Eine ID wird als Parameter übergeben und der dazu passende Datensatz aus der Datenbank geholt. Der Text wird mit Base64 in Binärdaten umgewandelt, aus denen das Klassifikatorobjekt erstellt wird. Zudem werden alle Parameter des Klassifizierers wie zum Beispiel seine ID, der Datentyp und das Experiment, zu dem er gehört, im Classifier_Container gespeichert.

**get_internals:** Diese Methode extrahiert interne Daten wie a-priori Wahrscheinlichkeiten aus dem Klassifikator und gibt sie in einer vereinfachten Datenstruktur zurück.

### AI::Categorizer

Das AI::Categorizer Framework von Ken Williams liegt momentan in der Version 0.09 vor. Diese Version wurde auch für diese Arbeit verwendet. In diesem Abschnitt werden die wichtigsten Teile des Frameworks und grundlegende Abläufe beschrieben.

Klassifiziert werden Dokumente, die durch die Klasse "Document" repräsentiert werden. Gespeichert wird darin nicht der tatsächliche Inhalt des Dokumentes[11], sondern nur der "Feature-Vector", eine Liste aller Wörter mit ihren Häufigkeiten. Ein Document-Objekt kann auch eine

[10] Binärdaten werden auf die 64 Zeichen A–Z, a–z, 0–9, + und / abgebildet und damit für textbasierte Systeme verwendbar

[11] Dieser wird nur dem Konstruktor übergeben und dort verarbeitet.

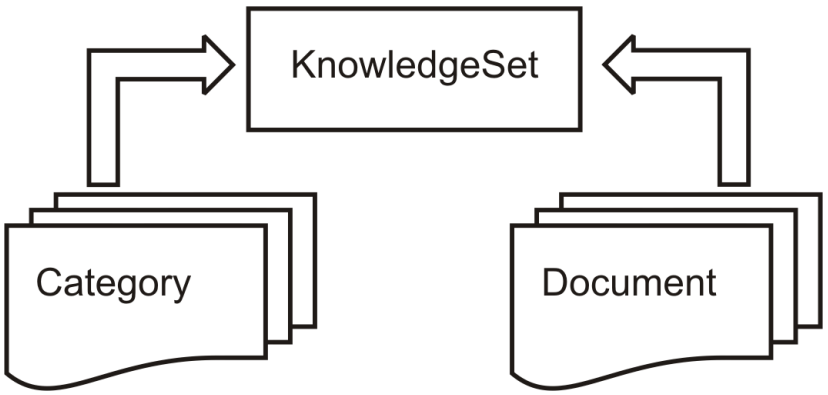


Abbildung 5.1.: KnowledgeSet
Ein KnowledgeSet-Objekt wird aus einer Reihe von Document- und Category-Objekten erzeugt.

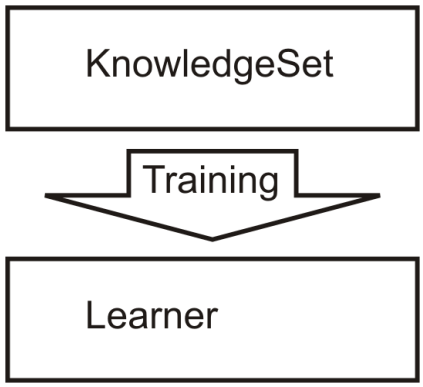


Abbildung 5.2.: Learner
Ein Learner-Objekt wird trainiert, indem ihm ein KnowledgeSet-Objekt übergeben wird.

oder mehrere Kategorien beinhalten. Ein solches Objekt kann zum Training eines neuen Klassifikators verwendet werden.

Eine Reihe von bereits vorklassifizierten Dokumenten kann zu einem “KnowledgeSet”-Objekt zusammengefasst werden, siehe Abbildung 5.1. An einem solchen KnowledgeSet-Objekt kann dann ein Klassifikator mit dessen “train”-Methode trainiert werden (siehe Abbildung 5.2).

Der Klassifizierer ist nun fertig trainiert und kann eingesetzt werden. Zum Klassifizieren werden die Texte als Document-Objekte der “categorize”-Methode übergeben. Als Rückgabe erhält man ein Hypothesis-Objekt, in welchem die Ergebnisse der Klassifikation gespeichert sind (siehe Abbildung 5.3).

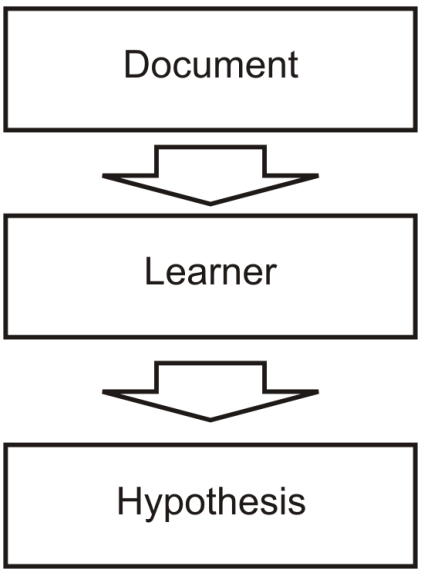


Abbildung 5.3.: Hypothesis

Übergibt man einem Learner-Objekt ein Dokument, so erhält man ein Hypothesis-Objekt, welches die Klassenzugehörigkeit mittels Wahrscheinlichkeiten beschreibt.

Das Hypothesis-Objekt enthält die einzelnen Kategoriewahrscheinlichkeiten, die zugeteilten Kategorien[12] sowie die Kategorie mit der höchsten Wahrscheinlichkeit.

#### Klassifizieren der Daten

Um diesen Vorgang etwas zu vereinfachen enthält das Modul Classifier_Container eine Methode "classify", welche die oben genannten Vorgänge zusammenfasst. Zusätzlich dazu werden auch die eingestellten Vorfilter auf die Daten angewendet. Die Methode liefert ein "Hypothesis"-Objekt zurück.

Listing 5.11: Classifier_Container::classify

```
1 sub classify {
2     my $self = shift;
3     if ($self->{classifier_trained}) {
4         my $data = shift;
5         my $filter = new preFilter;
6         $data = $filter->filter($data, split (/ /, $self->{preprocessor}))
              ;
7         my $doc = AI::Categorizer::Document->new( content => $data );
```

[12] Alle Kategorien, deren Wahrscheinlichkeiten über einem gewissen Schwellenwert liegen.

```
8           return $self->{classifier}->categorize($doc);
9       }
10  }
```

Das Klassifizieren der ganzen Seiten und der Formulare erfolgt im Experimentiersystem im Hauptprogramm. Das Klassifizieren der Felder erfolgt in der Methode “html_formlist” des Moduls “Output”. Die Felder eines Formulars werden nur dann klassifiziert, wenn dieses Formular in der Datenbank gespeichert ist und die Kategoriebezeichnung “flight_search” trägt.

Beim Einsatzsystem erfolgen alle Klassifikationen im Hauptprogramm. Hier werden die Felder nur dann klassifiziert, wenn das Formular, dem sie angehören als “flight_search” klassifiziert wurde. Genauso werden die Formulare auch nur dann klassifiziert, wenn die Seite als “flight_search” erkannt wurde. Dieses Verhalten ist jedoch einstellbar.

### 5.3.6. Qualitätsanalyse

Zur Qualitätsanalyse eines Experimentes wird von allen daran beteiligten Testdaten eines ausgewählt. Mit den übrigen wird dann ein Klassifikator trainiert und anschließend das ausgewählte Testdokument klassifiziert. Das Ergebnis der Klassifikation wird in der Datenbank im Table “classification”[13] gespeichert. Danach können die gespeicherten Ergebnisse für statistische Berechnungen verwendet werden.

#### Das Erstellen eines Klassifikators für ein Testdokument

Der Methode build_knowledge_set der Klasse Classifier_Container kann ein Argument übergeben werden. Dieses Argument beinhaltet die Identifikationsnummer eines Testdokumentes, welches bei der Erstellung des Klassifikators ausgelassen werden soll.

Ist das Argument “exclude” nicht definiert oder entspricht sein Wert nicht der Identifikationsnummer des aktuellen Dokumentes, so wird das aktuelle Dokument in das KnowledgeSet aufgenommen.

Anderenfalls wird das entsprechende Dokument in einer Variablen gespeichert, mit der entsprechenden Kategorie versehen und am Ende der Methode zurückgegeben. Dies erspart einen weiteren Datenbankzugriff für das Abholen des Testdokumentes.

[13] siehe Tabelle 5.6

Der Quellcode für den entsprechenden Teil der Methode build_knowledge_set:

```
if (!defined($exclude)
    || ($exclude != $datarow->{corpus_id})) {

    $documents{$datarow->{corpus_id}} =
        AI::Categorizer::Document->
            new(content => $datarow->{data})
        unless (exists($documents{$datarow->{corpus_id}}));
    $categories{$datarow->{category_name}}->
        add_document($documents{$datarow->{corpus_id}});
    $documents{$datarow->{corpus_id}}->
        add_category($categories{$datarow->{category_name}});
} else {
    $excluded_doc_obj =
        AI::Categorizer::Document->new(
            content => $datarow->{data})
        unless (defined($excluded_doc_obj));
    $excluded_doc_obj->
        add_category($categories{
            $datarow->{category_name}});
}
```

Das KnowledgeSet selbst wird wie schon beim normalen Erstellen eines Klassifikators in dem Classifier_Container Objekt abgespeichert. Nachdem mit dem KnowledgeSet ein Klassifikator trainiert wurde, wird das ausgewählte Testdokument mit diesem klassifiziert. Das Ergebnis dieser Klassifikation wird in der Datenbank in der Tabelle classification (siehe Tabelle 5.6) gespeichert.

Die Implementierung der oben beschriebenen Funktion ist sehr einfach. Der erste Teil kümmert sich um das Erstellen des Klassifikators sowie um das Klassifizieren des Testdokumentes. Am Ende wird noch überprüft, ob die Klassifikation korrekt war. Der zweite Teil ist im Grunde genommen nur eine einzige Zeile, in der das Ergebnis in der Datenbank gespeichert wird.

Listing 5.12: Classifier_Container::test_one_document

```
sub test_one_document {
    my $self = shift;
    my $test_doc_id = shift;
    my $test_doc_obj =
        $self->build_knowledge_set($test_doc_id);
    my @categories = $test_doc_obj->categories();
    $self->train_classifier();
```

```
8     my $classifier = $self->get_classifier();
9     my $hypothesis =
10        $classifier->categorize($test_doc_obj);
11    my $correct = 0;
12    my $false = 0;
13    for my $category (@categories) {
14        if ($hypothesis->best_category()
15            eq $category->name) {
16            $correct += 1;
17        }
18    }
19
20    $self->{dbh}->store_classification(
21                data_type_id => $self->{data_type_id},
22                experiment_id => $self->{experiment_id},
23                session_id => $self->{session_id},
24 # not needed for throw away classifiers:
25 #               classifier_id => $self->{classifier_id},
26                corpus_id => $test_doc_id,
27                good => $correct,
28                classification_scores =>
29                    encode_base64(nfreeze($hypothesis)),
30                guessed_category_id =>
31                    $self->{dbh}->get_category_by_name(
32                        $hypothesis->best_category(),
33                        $self->{data_type_id})
34            );
35    return $correct;
36 }
```

#### Das Durchtesten eines ganzen Experimentes

Die in Abschnitt 5.3.6 beschriebene Methode wird nur klassenintern von der Methode “test_experiment” aufgerufen. In dieser Methode werden zuerst die Informationen zur Session in der Datenbank gespeichert (siehe Tabelle 5.5).

Danach werden alle Document Ids des Experimentes der Reihe nach der in Abschnitt 5.3.6 beschriebenen Methode übergeben. Am Ende wird noch eine kurze Zusammenfassung der gefundenen Ergebnisse generiert. Diese wird aber nicht weiterverwendet. Sie dient ausschließlich dazu, den Anwender nach der langen Wartezeit etwas zu entschädigen.

Der Aufbau der Methode ist wieder sehr einfach. Nach den Initialisierungen und dem Abspeichern der Informationen über den Test beginnt der eigentliche Testlauf in einer Schleife.

Listing 5.13: Classifier_Container::test_experiment

```
sub test_experiment {
    my $self = shift;
    my %result = (
        total => 0,
        good => 0,
        bad => 0,
        );
    $self->{session_id} = $self->{dbh}->store_session(
            session_description => $_[0],
            data_type_id => $self->{data_type_id},
            experiment_id => $self->{experiment_id},
        );

    for my $doc_id ($self->get_corpus_ids()) {
        print "testing $doc_id";
        $result{total}++;
        my $good = $self->test_one_document($doc_id);
        if ($good) {
            $result{good}++;
            print " good\n";
        } else {
            $result{bad}++;
            print " bad\n";
        }
    }
    return %result;
}
```

Nachdem der Testlauf beendet wurde, können die gesammelten Ergebnisse analysiert werden.

### Die Auswertung

Von den ermittelten Ergebnissen werden Qualitätsmerkmale in Form von Precision und Recall ermittelt. Diese Qualitätskriterien von Klassifikatoren sind generell für das Klassifizieren in einer Kategorie (gehört dazu/gehört nicht dazu) definiert, werden aber hier auch für das Klassifizieren in mehreren Kategorien verwendet.

**Precision:** Die Precision ist die Wahrscheinlichkeit, dass eine getätigte Klassifikation eine passende Kategorie gefunden hat.

**Recall:** Der Recall ist die Wahrscheinlichkeit, dass ein zu klassifizierendes Dokument bei der Klassifikation seiner Kategorie zugeordnet wird.

Diese Werte werden für jede Kategorie extra berechnet. Am Ende wird von den einzelnen Kategoriewerten ein gewichtetes[14] Mittel für den gesamten Testlauf ermittelt. Die Berechnung dieser Werte wird in Abschnitt 2.3 beschrieben.

Die Implementierung der Funktionalität ist etwas umfangreicher. Die Ergebnisse werden in der Datenbank durch einen Join aus drei Tabellen gebildet.

Die Berechnungen werden in zwei Durchläufen durchgeführt. Der erste Durchlauf passiert schon beim Auswerten des Resultates aus der Datenbank. Der Table “classification” enthält die, durch die in Abschnitt 5.3.6 beschriebene Methode, gespeicherten Zuordnungen. Der Table “corpus_category” enthält die tatsächlichen Kategorien der Dokumente. Verknüpft werden die beiden Tabellen über die Eigenschaft “corpus_id”. Die mehrfachen Kategorien eines einzelnen Dokumentes werden, von Tabulatoren getrennt, in einem Feld aus der Datenbank geholt.

Listing 5.14: guessDBH::get_sessions SQL Ergebnisse werden aus der Datenbank geholt

```
1  sub get_sessions {
2      my $self = shift;
3      my $experiment_id = $self->int2(shift);
4      my $sql = "select tabjoin(category_id) as category_ids, ".
5              'guessed_category_id, '.
6              'session_id '.
7          "from corpus_category ".
8              "natural join classification ".
9          "where experiment_id = $experiment_id ".
10         "group by corpus_id, guessed_category_id, ".
11             "session_id, classification_id";
12     my @table = $self->qrh($sql);
```

Im nächsten Schritt werden die Initialisierungen vorgenommen, sobald diese notwendig werden. Das Ergebnis wird in einem Hash-Baum, der in Abbildung 5.4 grafisch dargestellt wurde, gespeichert.

[14] Aufgrund der Anzahl der Dokumente in den Kategorien

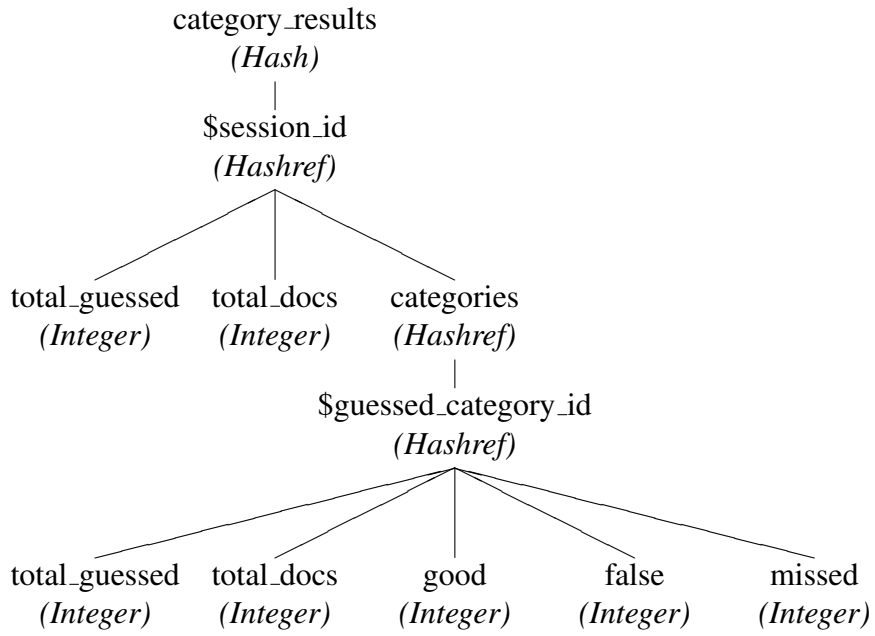


Abbildung 5.4.: category_results
Dieser Baum entspricht dem category_results-Hash nach der ersten Phase der Datenaufbereitung.

Der Wurzel-Hash enthält als Schlüssel die Identifikationsnummern der Testläufe “session_id”. Die Werte dieses Hashes sind Hash-Referenzen auf Hashes, die jeweils einen Testdurchlauf abbilden. Der Testdurchlaufhash hat drei Schlüssel: “total_guessed”, “total_docs” und “categories”. Die Werte von “total_guessed” und “total_docs” enthalten die Gesamtzahl der zugeordneten (guessed) und der tatsächlichen (docs) Dokumente. Die beiden Werte sind in diesem Kontext gleich. Der Wert von “categories” ist eine Hash-Referenz, die auf den Ergebnis-Hash der Kategorien verweist. Die Schlüssel dieses Hashes sind die Identifikationsnummern der Kategorien. Die Werte sind wiederum Hash-Referenzen auf den Ergebnis-Hash der jeweiligen Kategorie. Dieser Ergebnis-Hash besteht aus fünf Schlüsseln: “total_guessed”, “total_docs”, “good”, “false” und “missed”. Der Wert “total_guessed” beinhaltet ähnlich wie oben die Anzahl der Dokumente, die vom Klassifizierer dieser Kategorie zugeordnet wurden. Die Anzahl der Dokumente, die dieser Kategorie angehören, wird im Wert des Schlüssels “total_docs” gespeichert. Gehört ein Dokument mehr als einer Kategorie an, so wird es bei jeder Kategorie mitgezählt. Die Summe der “total_docs”-Werte aller Kategorien entspricht daher nicht notwendigerweise dem “total_docs”-Wert des Testlaufes! Wird ein Dokument richtig klassifiziert, so wird es durch den “good”-Wert der Kategorie gezählt. Stimmt die Zuordnung nicht, so wird das Dokument bei der Kategorie, in die es zugeordnet wurde, als “false” gezählt sowie

bei allen Kategorien, denen es angehört, als “missed”.

Listing 5.15: guessDBH::get_sessions Initialisierung

```
    my %category_results; # $category_results{session}{category}
    my %category_names;
    for my $row (@table) {
        unless (exists $category_results{$row->{session_id}}) {
            $category_results{$row->{session_id}}{total_guessed} = 0;
            $category_results{$row->{session_id}}{total_docs} = 0;
        }
        unless (exists $category_results{$row->{session_id}}
                    {categories}{$row->{guessed_category_id}}) {
            $category_results{$row->{session_id}}{categories}
                {$row->{guessed_category_id}}{total_guessed} = 0;
            $category_results{$row->{session_id}}{categories}
                {$row->{guessed_category_id}}{total_docs} = 0;
            $category_results{$row->{session_id}}{categories}
                {$row->{guessed_category_id}}{good} = 0;
            $category_results{$row->{session_id}}{categories}
                {$row->{guessed_category_id}}{false} = 0;
            $category_results{$row->{session_id}}{categories}
                {$row->{guessed_category_id}}{missed} = 0;
        }
```

Nachdem der Baum für die zugewiesene Kategorie initialisiert wurde (sofern dies nötig war), werden nun die Kategorien abgearbeitet, zu denen das gerade behandelte Dokument gehört. In dieser Schleife werden auch die “total_docs”-Zähler der entsprechenden Kategorien um Eins erhöht.

Listing 5.16: guessDBH::get_sessions Kategoriezähler

```
        my $regex = $row->{category_ids};
        $regex =~ s/^\t//;
        $regex =~ s/\t/|/g;
#       $regex = "^$regex\$";
        for my $category_id (split /\|/, $regex) {
            unless (exists $category_results{$row->{session_id}}
                    {categories}{$category_id}) {
                $category_results{$row->{session_id}}{categories}
                    {$category_id}{total_guessed} = 0;
                $category_results{$row->{session_id}}{categories}
                    {$category_id}{total_docs} = 0;
                $category_results{$row->{session_id}}{categories}
```

```
                    {$category_id}{good} = 0;
                $category_results{$row->{session_id}}{categories}
                    {$category_id}{false} = 0;
                $category_results{$row->{session_id}}{categories}
                    {$category_id}{missed} = 0;
            }
            $category_results{$row->{session_id}}{categories}
                {$category_id}{total_docs}++;
            $category_results{$row->{session_id}}{total_docs}++;
        }
```

Im folgenden Teil werden die übrigen Zähler inkrementiert. Wie oben werden auch hier alle Kategorien behandelt, denen ein Dokument angehört.

Listing 5.17: guessDBH::get_sessions Testlaufzähler

```
        $category_results{$row->{session_id}}{categories}
            {$row->{guessed_category_id}}{total_guessed}++;
        $category_results{$row->{session_id}}{total_guessed}++;
        if ($row->{guessed_category_id} =~ /^$regex$/) {
            $category_results{$row->{session_id}}{categories}
                {$row->{guessed_category_id}}{good}++;
        } else {
            $category_results{$row->{session_id}}{categories}
                {$row->{guessed_category_id}}{false}++;
            for my $category_id (split /\|/, $regex) {
                $category_results{$row->{session_id}}{categories}
                    {$category_id}{missed}++;
            }
        }
    }
    return %category_results;
}
```

Am Ende der Methode wird der vorbereitete Resultat-Hash zurückgegeben. Die weiteren Berechnungen finden in der Methode “render_sessions” der Klasse “Output” statt. Sie benötigen die in der oben beschriebenen Methode getätigten Vorberechnungen.

Der erste Teil der “render_sessions” geht die in Abbildung 5.4 dargestellte Datenstruktur mittels zweier geschachtelter Schleifen durch. Die äußere Schleife durchläuft die Testläufe, die innere behandelt die Kategorien.

Listing 5.18: Outout::render_sessions Grundstruktur

```
sub render_sessions {
    my $self = shift;
    my $data_type_id = shift;
    my $sessions = shift;
    my $sessions_html = '<div>';
    my %categories = $dbh->get_categories($data_type_id);
    # pre
    for my $session_id (sort keys %$sessions) {
        for my $category_id
            (sort keys %{$sessions->{$session_id}{categories}}) {
        }
    }
    # html output
}
```

Die Programmzeilen zur Berechnung wurden aus Gründen der Übersichtlichkeit aus der Darstellung der Methode entfernt und werden im folgenden Abschnitt genauer beschrieben.

Abbildung 5.5 zeigt die neu hinzugekommenen Schlüssel des Kategoriehashes. Die Abkürzung TP steht hier für True Positive, also für richtig dieser Kategorie zugeordnete Dokumente. Für die fälschlicherweise zugeordneten Dokumente steht FP (False Positives). Die Dokumente, die dieser Kategorie angehören, aber nicht zugeteilt wurden[15], werden in FN (False Negatives) gezählt. Die bis jetzt beschriebenen Werte sind einfach andere Namen für bereits gefundene Größen: TP entspricht dem Wert "good", FP ist ident mit "false" und FN gleicht "missed". Die True Negatives, also alle Dokumente, die weder dieser Kategorie angehören, noch ihr zugeteilt wurden, werden in TN gezählt. Eine Ausnahme bilden Dokumente, die mehreren Kategorien angehören. Werden diese einer ihrer Kategorien zugeteilt, so werden sie bei allen anderen Kategorien, also auch bei jenen, denen sie eigentlich angehören, als True Negative gezählt. Diese Abweichung von der Definition der TN entsteht durch die Erweiterung der Analyseverfahren auf die Möglichkeit eines Dokumentes, mehreren Kategorien anzugehören[16]. Die TN berechnet sich aus der Summe aller Dokumente abzüglich aller anderer Parameter der Kategorie (TP, FP und FN).

Die Berechnung dieser Werte passiert am Anfang der Methode "render_sessions" der Klasse Output.

[15] Wird ein Dokument nicht einer seiner Kategorien zugeordnet, so wird bei allen Kategorien, denen das Dokument angehört, ein FN eingetragen. Stimmt die Zuteilung jedoch, so wird kein FN eingetragen.

[16] In Abschnitt 6.2 wird dieser Umstand noch eingehender diskutiert.

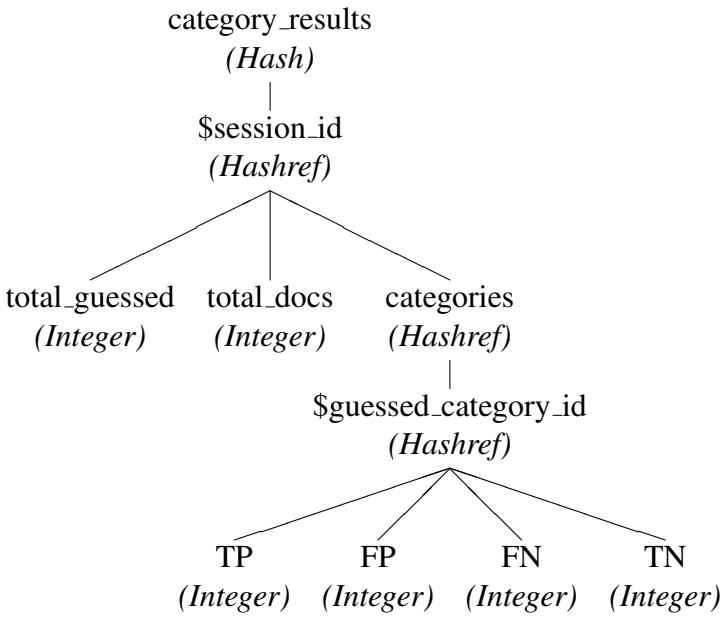


Abbildung 5.5.: category_results mit Ergänzungen

Dieser Baum entspricht dem “category_results”-Hash nach der ersten Phase der Datenaufbereitung. Aus Gründen der Übersichtlichkeit werden in dieser Grafik nur die neu hinzugekommenen Schlüssel des Kategorie-Hashes dargestellt.

Listing 5.19: Outout::render_sessions Ausgangswerte

```
$sessions->{$session_id}{categories}{$category_id}{TP} =
    $sessions->{$session_id}{categories}{$category_id}{good};
$sessions->{$session_id}{categories}{$category_id}{FP} =
    $sessions->{$session_id}{categories}{$category_id}{false};
$sessions->{$session_id}{categories}{$category_id}{FN} =
    $sessions->{$session_id}{categories}{$category_id}{missed};
$sessions->{$session_id}{categories}{$category_id}{TN} =
    $sessions->{$session_id}{total_guessed}-
    $sessions->{$session_id}{categories}{$category_id}{good}-
    $sessions->{$session_id}{categories}{$category_id}{missed};
```

Die folgenden Texte zu Recall und Precision stammen aus [Wik08]. Auf die Beschreibungen der Qualitätsmerkmale folgt jeweils der dazugehörende Quellcode.

### Recall

Der Recall eines Verfahrens ergibt sich aus dem Verhältnis richtig positiver zu allen relevanten Dokumenten TP / (TP + FN). Er entspricht der statistischen Sensitivität (auch True Positive Rate, TPR).

Listing 5.20: Outout::render_sessions Recall

```
$sessions->{$session_id}{categories}{$category_id}{recall} = 0;
$sessions->{$session_id}{categories}{$category_id}{recall} =
    $sessions->{$session_id}{categories}{$category_id}{TP} /
    ($sessions->{$session_id}{categories}{$category_id}{TP}+
    $sessions->{$session_id}{categories}{$category_id}{FN})
        if
            ($sessions->{$session_id}{categories}{$category_id}{TP}+
            $sessions->{$session_id}{categories}{$category_id}{FN})
        ;
```

### Precision

Die Precision ergibt sich aus dem Verhältnis richtig positiver zu allen gefundenen Dokumenten TP / (TP + FP). Sie entspricht der statistischen Relevanz (auch positiver Vorhersagewert).

Listing 5.21: Outout::render_sessions Precision

```
$sessions->{$session_id}{categories}{$category_id}{precision} = 0;
$sessions->{$session_id}{categories}{$category_id}{precision} =
    $sessions->{$session_id}{categories}{$category_id}{TP} /
    ($sessions->{$session_id}{categories}{$category_id}{TP}+
    $sessions->{$session_id}{categories}{$category_id}{FP})
        if
            ($sessions->{$session_id}{categories}{$category_id}{TP}+
            $sessions->{$session_id}{categories}{$category_id}{FP})
        ;
```

**Spezifität** Die Spezifität ergibt sich aus dem Verhältnis nicht gefundener, irrelevanter Dokumente (richtig negativ) zu allen nicht relevanten Dokumenten TN / (FP + TN).

Listing 5.22: Outout::render_sessions Spezifität

```
$sessions->{$session_id}{categories}{$category_id}{spezifitaet} = 0;
$sessions->{$session_id}{categories}{$category_id}{spezifitaet} = 1 -
    $sessions->{$session_id}{categories}{$category_id}{TN} /
    ($sessions->{$session_id}{categories}{$category_id}{FP}+
    $sessions->{$session_id}{categories}{$category_id}{TN})
        if
            ($sessions->{$session_id}{categories}{$category_id}{FP}+
            $sessions->{$session_id}{categories}{$category_id}{TN})
        ;
```

**Fallout** Der Fallout ist das Gegenteil der Spezifität und ergibt mit ihr zusammen 100% beziehungsweise eins. Fallout = 1 - Spezifität.

Listing 5.23: Outout::render_sessions Fallout

```
$sessions->{$session_id}{categories}{$category_id}{fallout} = 0;
$sessions->{$session_id}{categories}{$category_id}{fallout} =
    ($sessions->{$session_id}{categories}{$category_id}{TN} /
    ($sessions->{$session_id}{categories}{$category_id}{FP}+
    $sessions->{$session_id}{categories}{$category_id}{TN}))
        if
            ($sessions->{$session_id}{categories}{$category_id}{FP}+
            $sessions->{$session_id}{categories}{$category_id}{TN})
        ;
```

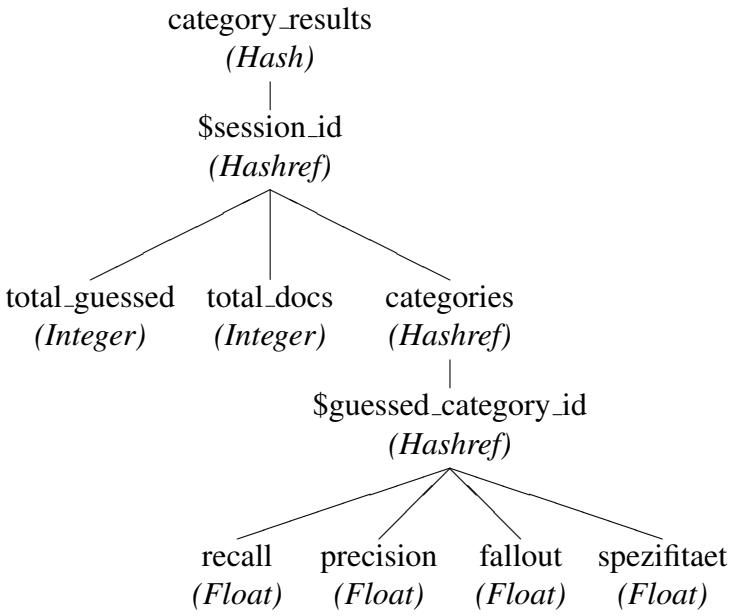


Abbildung 5.6.: category_results mit Kategoriequalitäten

Dieser Baum entspricht dem "category_results"-Hash nach der zweiten Phase der Datenaufbereitung. Aus Gründen der Übersichtlichkeit werden in dieser Grafik wieder nur die neu hinzugekommenen Schlüssel des Kategorie-Hashes dargestellt. Diese sind die in Abschnitt 2.3 beschriebenen Qualitätsmerkmale.

**Fehlerbehandlung** Die Berechnungen beinhalten eine Division, bei welcher der Divisor null sein kann. Deswegen werden die Werte nur dann berechnet, wenn dies möglich ist. Ist dies nicht der Fall[17], so wird der Wert auf null gesetzt.

**Summenbildung** Um die Qualitätsmerkmale für den gesamten Testlauf zu erhalten, wird der nach der Anzahl der Dokumente gewichtete Mittelwert über die Qualitätsmerkmale der Kategorien gebildet. Dazu werden die einzelnen Qualitäten mit der Anzahl der Dokumente multipliziert und beim Testlauf aufsummiert.

Listing 5.24: Outout::render_sessions Summenbildung für die Gesamtqualitäten

```
1 $sessions->{$session_id}{categories}{$category_id}{total_recall} =
2     $sessions->{$session_id}{categories}{$category_id}{recall} *
3     $sessions->{$session_id}{categories}{$category_id}{total_docs};
4 $sessions->{$session_id}{categories}{$category_id}{total_precision} =
5     $sessions->{$session_id}{categories}{$category_id}{precision} *
```

[17] Zum Beispiel, wenn nicht genügend Daten vorliegen.

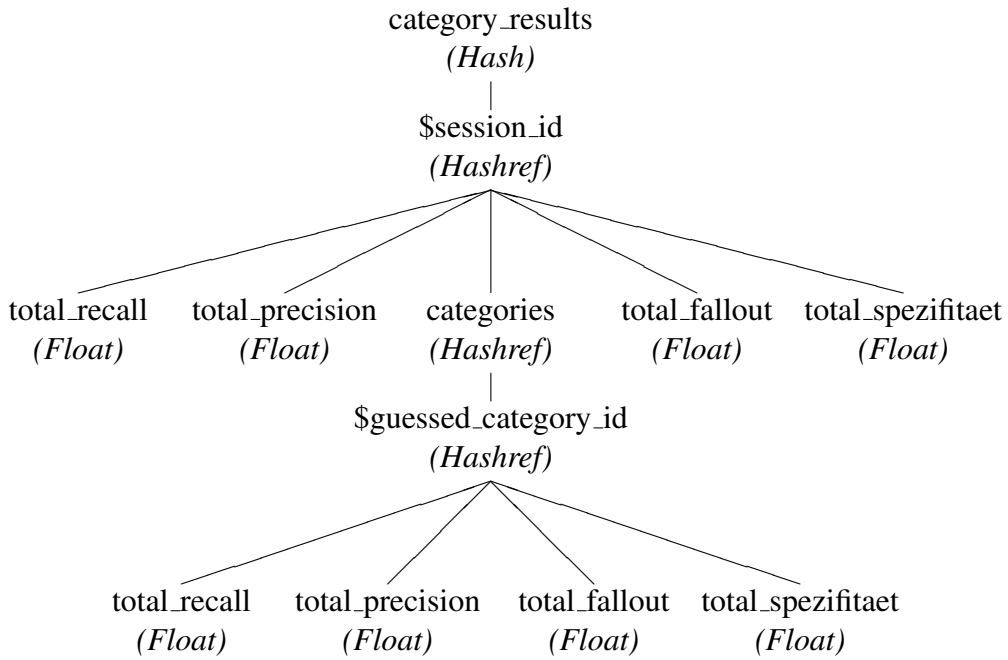


Abbildung 5.7.: category_results mit Qualitätssummen

Dieser Baum entspricht dem "category_results"-Hash nach der dritten Phase der Datenaufbereitung. Aus Gründen der Übersichtlichkeit werden in dieser Grafik wieder nur die neu hinzugekommenen Schlüssel des Kategorie- und des Testlauf-Hashes dargestellt.

```
    $sessions->{$session_id}{categories}{$category_id}{total_docs};
$sessions->{$session_id}{categories}{$category_id}{total_fallout} =
    $sessions->{$session_id}{categories}{$category_id}{fallout} *
    $sessions->{$session_id}{categories}{$category_id}{total_docs};
$sessions->{$session_id}{categories}{$category_id}{total_spezifitaet} =
    $sessions->{$session_id}{categories}{$category_id}{spezifitaet} *
    $sessions->{$session_id}{categories}{$category_id}{total_docs};

$sessions->{$session_id}{total_recall} +=
    $sessions->{$session_id}{categories}{$category_id}{total_recall};
$sessions->{$session_id}{total_precision} +=
    $sessions->{$session_id}{categories}{$category_id}{total_precision};
$sessions->{$session_id}{total_fallout} +=
    $sessions->{$session_id}{categories}{$category_id}{total_fallout};
$sessions->{$session_id}{total_spezifitaet} +=
    $sessions->{$session_id}{categories}{$category_id}{total_spezifitaet};
```

Dieser Schritt ergibt dann die in Abbildung 5.7 dargestellten neuen Hash-Einträge.

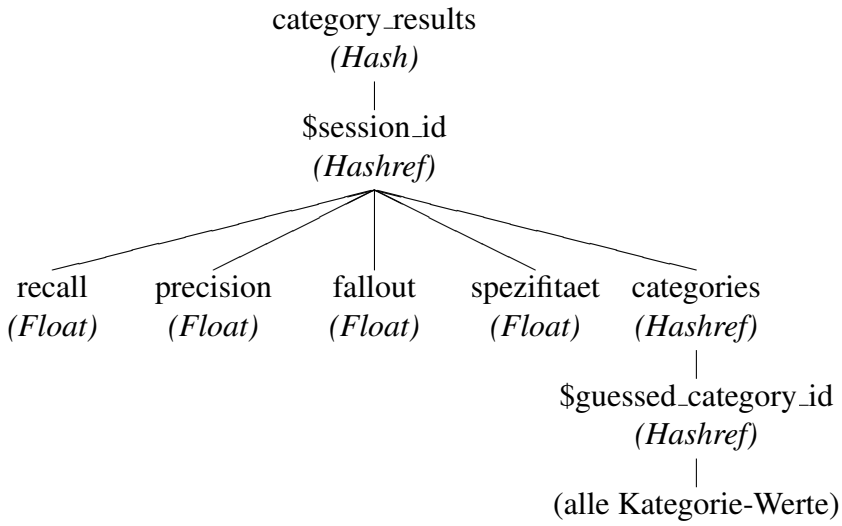


Abbildung 5.8.: category_results mit Qualitätssummen

Dieser Baum entspricht dem "category_results"-Hash nach der vierten Phase der Datenaufbereitung. Aus Gründen der Übersichtlichkeit werden in dieser Grafik wieder nur die neu hinzugekommenen Schlüssel des Testlauf-Hashes dargestellt.

Um die endgültigen Qualitäten zu erhalten, werden die Summen noch durch die Summe der Anzahl der Dokumente in den einzelnen Kategorien dividiert. Diese Summe kann größer sein als die Gesamtsumme der Dokumente, da Dokumente zu mehr als einer Kategorie gehören können.

Listing 5.25: Outout::render_sessions Gesamtqualitäten

```
1  $sessions->{$session_id}{recall} =
2      $sessions->{$session_id}{total_recall} /
3      $sessions->{$session_id}{total_docs};
4  $sessions->{$session_id}{precision} =
5      $sessions->{$session_id}{total_precision} /
6      $sessions->{$session_id}{total_docs};
7  $sessions->{$session_id}{fallout} =
8      $sessions->{$session_id}{total_fallout} /
9      $sessions->{$session_id}{total_docs};
10 $sessions->{$session_id}{spezifitaet} =
11     $sessions->{$session_id}{total_spezifitaet} /
12     $sessions->{$session_id}{total_docs};
```

Danach ist der Ergebnisbaum komplett. Abbildung 5.8 zeigt die neuen Werte.

Der zweite Teil der render_sessions Methode dient nun dazu, die eben berechneten Werte in HTML zu verpacken, um sie über einen Browser für den Benutzer lesbar zu machen.

Listing 5.26: Outout::render_sessions HTML-Erzeugung

```
for my $session_id (sort keys %$sessions) {
    $sessions_html .= '<div style="float: left;">';
    $sessions_html .= '<table border=1>';
    $sessions_html .= "<tr><th colspan=2>$session_id</td></tr>";
    for my $field (qw(total_docs total_guessed recall precision fallout
        spezifitaet)) {
        $sessions_html .= "<tr><td>$field</td>".
            "<td>$sessions->{$session_id}{$field}</td></tr>\n";
    }
    $sessions_html .= '</table>';
    for my $category_id (sort keys %{$sessions->{$session_id}{categories
        }}) {
        $sessions_html .= '<table border=1>';
        $sessions_html .= "<tr><th colspan=2>$category_id: $categories{
            $category_id}</th></tr>";
        for my $field (qw(total_docs total_guessed good false missed TP FP
             FN TN recall precision fallout spezifitaet)) {
            $sessions_html .= "<tr><td>$field</td>".
                "<td>$sessions->{$session_id}{categories}{$category_id}{
                    $field}</td></tr>\n";
        }
        $sessions_html .= '</table>';
    }
    $sessions_html .= '</div>';
}
return "$sessions_html</div><br><br clear=\"all\">";
```

Das Ergebnis präsentiert sich als senkrechte Tabellengruppen. Die erste Tabelle beinhaltet jeweils die Zusammenfassung des Testlaufes. Die Tabellen darunter beinhalten die Informationen über die einzelnen Kategorien. Einzelne Testläufe werden nebeneinander dargestellt, so können unterschiedliche Werte der einzelnen Testläufe direkt verglichen werden.

# 6. Ergebnisse

In diesem Kapitel werden die in dieser Arbeit durch das Experimentiersystem gewonnen Ergebnisse beschrieben.

## 6.1. Erfahrungen, Fallstricke und Probleme

Die im Zuge der Diplomarbeit implementierten Algorithmen und Methoden wurden im Hinblick auf einfache und standardkonforme Webseiten entwickelt. Da jedoch Standards in Bezug auf Webseiten im Allgemeinen nicht als bindend angesehen werden, ist es nicht auszuschließen, dass diese Methoden bei manchen Seiten nicht wie erhofft funktionieren. Dieser Abschnitt behandelt mögliche Probleme und deren Auswirkungen.

### 6.1.1. Unsauberes HTML

Die HyperText Markup Language beschreibt eine baumartige Datenstruktur, welche das Dokument abbildet. In einer solchen Struktur sind Verflechtungen wie zum Beispiel: “Fettdruck <u>Fett und unterstrichen nur unterstrichen</u>” zwar notierbar, aber grundsätzlich nicht erlaubt. Dennoch werden solche Notierungen von den meisten Browsern akzeptiert und sinngerecht dargestellt, was dazu führt, dass solche Konstrukte bei der Erstellung von Webseiten nicht bemerkt werden und daher oft Verwendung finden.

Die in dieser Arbeit verwendeten Klassifizierungsalgorithmen berücksichtigen die Baumstruktur des HTML nicht und sind daher grundsätzlich nicht anfällig für diese Art von Problemen.

Gleiches gilt auch für die in Abschnitt 5.3.3 beschriebene erste Implementierung des Splitters, welche ebenfalls die Baumstruktur außer Acht lässt.

Die zweite Implementation des Splitters (Splitter2, siehe Abschnitt 5.3.4) nutzt jedoch ein Modul zur Erstellung eines Baumes aus einem gegebenen HTML-Dokument und ist daher mit diesem Problem konfrontiert. Selbstverständlich sind auch Browser damit konfrontiert. Diese verwenden, um an einen validen Baum zu kommen, implizit erstellte Tags beziehungsweise Elemente. So werden zum Beispiel nicht geschlossene Tags geschlossen und an manchen Stellen zusätzliche Elemente eingefügt.

Auch das verwendete Modul "HTML::TreeBuilder" bedient sich einer solchen Methode. Es gibt jedoch keine generelle Regel, wie ein nicht valides HTML-Dokument auf diese Art repariert wird. Deshalb sind auch die von den Browsern und dem "HTML::TreeBuilder" eingesetzten Methoden nicht notwendigerweise ident und produzieren ein unterschiedliches Ergebnis, das jedoch dieselben Zielvorgaben erfüllt.

Die Probleme, die dadurch entstehen können, betreffen den XPath der erkannten Elemente. Dieser XPath hat das vom "HTML::TreeBuilder" aufbereitete HTML-Dokument als Grundlage. Wenn nun ein anderes System dasselbe Dokument verwendet, so besteht die Gefahr, dass die vom Klassifikationsystem angegebenen XPaths im anderen System nicht verwendbar sind.

### 6.1.2. Javascript

Javascript ist an sich nicht Teil von HTML, kann aber in HTML-Dokumente integriert werden oder von diesen aus separaten Dateien nachgeladen werden. Die Möglichkeiten von Javascript bestehen unter anderem darin, das HTML-Dokument nach dem Laden, zum Beispiel als Reaktion auf Aktionen des Benutzers, noch zu verändern.

Es ist möglich, große Teile des HTML-Codes in einem Javascript-Programm in unkenntlicher Form zu übertragen und erst vom Browser des Benutzers in eine für diesen lesbare Form zu bringen und in den HTML-Baum des Browsers einzufügen.

Diesem Problem ist nur mit einem Javascript-Interpreter beizukommen. Dieser würde dann das Javascript-Programm ausführen und auf einen HTML-Baum anwenden. Dies ist potentiell gefährlich, da fremder Programmcode auf eigenen Systemen ausgeführt wird. Aus diesem Grund und auch weil ein solches Vorhaben den Rahmen dieser Arbeit sprengen würde, wurde auf die Implementierung dieser Lösung verzichtet.

Die Möglichkeit, mittels eines Javascript-Programmes nachträglich Auswahlfelder in Formularen mit Werten zu befüllen, ist ein Teilbereich des oben genanntes Problems. Hier wird sie

aber extra erwähnt, da diese Methode sehr oft verwendet wird, etwa um die Auswahl der Zielflughäfen aufgrund der getroffenen Auswahl des Abflug-Flughafens anzupassen. Sind diese Daten nicht von Anfang an in dem Element enthalten, so werden sie bei der Klassifikation nicht berücksichtigt. Dabei gehen oft wertvolle Informationen verloren, die für eine korrekte Klassifikation wichtig sind.

Eine weitere Möglichkeit von Javascript ist es, Daten aus mehreren Formularen zu verwenden. Das bedeutet, die Eingabefelder für eine bestimmte Abfrage müssen nicht notwendigerweise im selben Formular sein. Dadurch kann die Klassifikation der Formulare gestört werden.

### 6.1.3. Lösungen

Eine Lösung für einen Teil der in den letzten beiden Abschnitten genannten Probleme stellt die Verwendung von vorverarbeitetem HTML-Code dar. Ein Client System bedient sich dabei des HTML-Interpreters eines Browsers und verwendet den dabei entstehenden HTML-Code, welcher valid ist und alle initialen Transformationen durch Javascript schon eingearbeitet hat. Dieser HTML-Code wird vom "HTML::TreeBuilder" nicht mehr verändert. Das hat zu Folge, dass der angegebene XPath auf alle Fälle stimmt. Weiters sind auch die Formularfelder weitgehend mit verwertbaren Daten befüllt.

## 6.2. Leistungsdaten der Klassifizierer und Vorfilter

Die verwendeten Klassifizier- und Vorfilteralgorithmen wurden mit der in Abschnitt 5.3.6 beschriebenen Methode getestet. Die dabei ermittelten Ergebnisse werden in diesem Abschnitt ausgewertet. Eine komplette Liste der Ergebnisse befindet sich im Anhang. Die Analyse wird für die drei Klassifizierungsaufgaben HTML, Formulare und Felder getrennt beschrieben.

### 6.2.1. Das Klassifizieren der ganzen Seite

Die ersten Versuche in dem Projekt wurden mit der Klassifikation von ganzen HTML-Seiten ausgeführt. Diese Versuche zeigten bereits, dass es grundsätzlich machbar ist, Webseiten wie Texte zu klassifizieren.

Das Diagramm in Abbildung 6.1 zeigt einen Vergleich der verwendeten Klassifizierungsmethoden und Vorfilter.

**Ausgangsbedingungen:** Das Trainingsset umfasste 109 Dokumente, davon 15 mit der Kategorie “flight_search”.

#### Bewertung der Klassifikationsalgorithmen

Die Balken des Diagramms in Abbildung 6.1 sind nach Klassifikationsalgorithmen gruppiert. Die drei Klassifikatoren zeigen eine sehr ähnliche Leistung, wobei der Bayes-Klassifikator etwas besser arbeitet als die beiden anderen. Interessant ist noch das Verhalten der Support Vector Machine, diese wird nur sehr gering durch das Weglassen von Informationen durch den Filter “no_tags” beeinflusst.

#### Bewertung der Vorfilter

Der Filter “no_tags” zeigt, dass in den Tags für die Algorithmen Bayes und k-nächste Nachbarn wichtige Informationen enthalten sind, deren Fehlen das Ergebnis stark beeinträchtigt. Dies ist auch der einzige Filter, welcher beim Bayes-Algorithmus eine Veränderung bewirkt. Alle anderen Vorfilter zeigen hier keine Veränderung des Ergebnisses. Die Kombination aus den Filtern “tag_tags” und “enhance_element” liefert bei den Algorithmen KNN und SVM das beste Ergebnis, wobei sich nur der KNN gemäß der Erwartung verhält. Das heißt, beide Einzelfilter bringen eine Verbesserung gegenüber dem ungefilterten Text und die Kombination aus beiden Filtern liefert ein besseres Ergebnis als die jeweiligen Einzelfilter.

Bei der SVM hingegen verhalten sich die Filter anders. Einerseits wird, wie bereits vorher erwähnt wurde, dieser Algorithmus nicht durch das Weglassen der Tags beeinträchtigt. Andererseits brachte hier das Kennzeichnen der Tags mittels “tag_tags” eine Verschlechterung gegenüber dem Rohtext, dieser Filter zeigt sogar das schlechteste Ergebnis bei diesem Algorithmus. Trotzdem kann dieser Filter aber das Ergebnis des “enhance_element”-Filters weiter verbessern, wenn er gemeinsam mit diesem angewendet wird. Hier wird sogar das beste Ergebnis in Verbindung mit der SVM erreicht.

### 6.2.2. Das Klassifizieren der Formulare

Am besten funktioniert das Klassifizieren der Formulare. Die Ursache dafür ist die geringere Anzahl der Kategorien. Man erkennt dies an der vergleichsweise guten Leistung des “Guesser”.

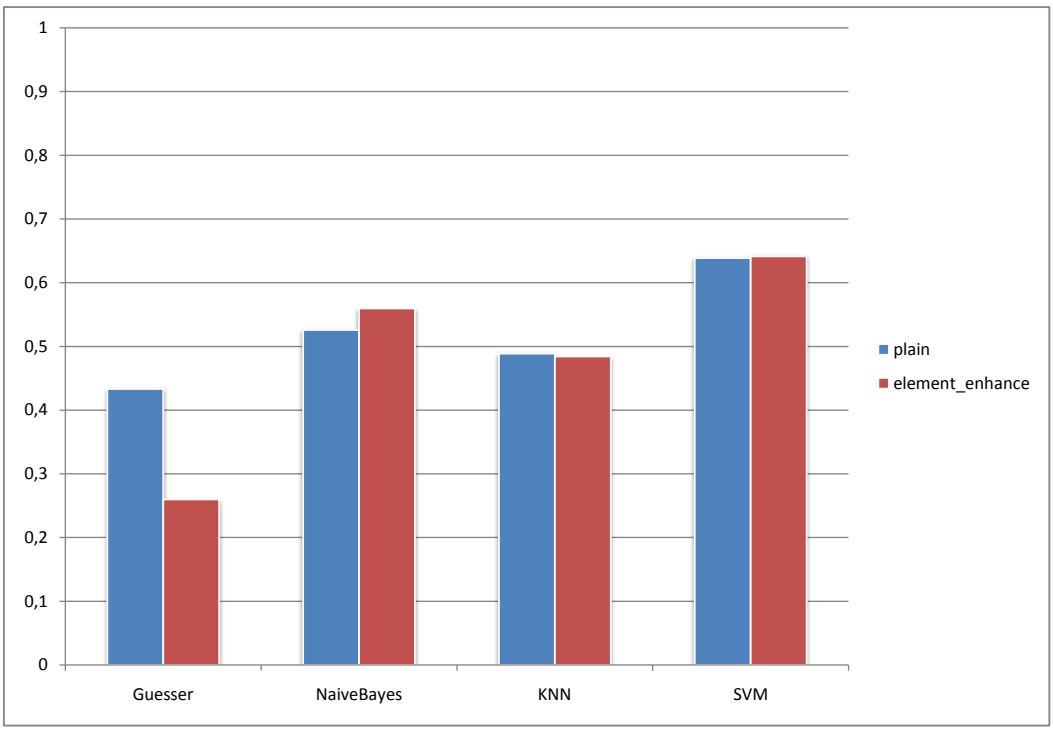


Abbildung 6.1.: F-Wert ganze Seiten
Ergebnisse der Klassifikation einer ganzen Seite als Darstellung der F-Werte

Das Diagramm in Abbildung 6.2 zeigt einen Vergleich der verwendeten Klassifizierungsmethoden und Vorfilter.

**Ausgangsbedingungen:** Das Trainingsset umfasste 43 Dokumente, davon 15 mit der Kategorie “flight_search” und 26 mit “misc”.

#### Bewertung der Klassifikationsalgorithmen

Die Klassifikationsalgorithmen zeigen bei diesem Test kaum Auffälligkeiten. Alle drei liefern ein ähnlich gutes Ergebnis, wobei der SVM-Algorithmus im Gegensatz zum Klassifizieren der ganzen Seite besser als der Bayes-Algorithmus funktioniert.

#### Bewertung der Vorfilter

Keiner der verwendeten Vorfilter brachte eine Verbesserung des Ergebnisses. Wie auch im letzten Test verursachte das Weglassen der Tags eine Verschlechterung des Ergebnisses. Hier sogar noch deutlicher. Auch die SVM ist bei diesem Test im Gegensatz zum vorher Beschriebenen durch das Weglassen der Tags im selben Maße beeinträchtigt wie der Bayes-Klassifikator. Das Anfügen domänenspezifischer Information, wie es im “enhance_element” Filter passiert, wirkt sich bei diesem Experiment negativ auf das Ergebnis aus. Das Kennzeichnen der Tags hatte keinen Einfluss auf das Ergebnis.

### 6.2.3. Das Klassifizieren der Felder

Am schwierigsten gestaltete sich die Klassifikation der einzelnen Felder. Die Gründe dafür sind die wenigen Daten, die pro Feld verfügbar sind und auch die Kreativität der vergebenen Feldnamen, so dass es kaum zwei Felder mit derselben Bedeutung gibt, die auch den gleichen Namen haben. Da Klassifikatoren Wörter nur exakt vergleichen, sind auch ähnliche Namen ein Unterschied.

Ein weiteres Problem stellt die Kategorieverteilung der Testdaten dar. Die dominanten Kategorien sind “hidden” mit 43% und “misc” mit 24%. Die Kategorien, welche für das Verständnis der Seite ausschlaggebend sind, sind deutlich seltener vertreten.

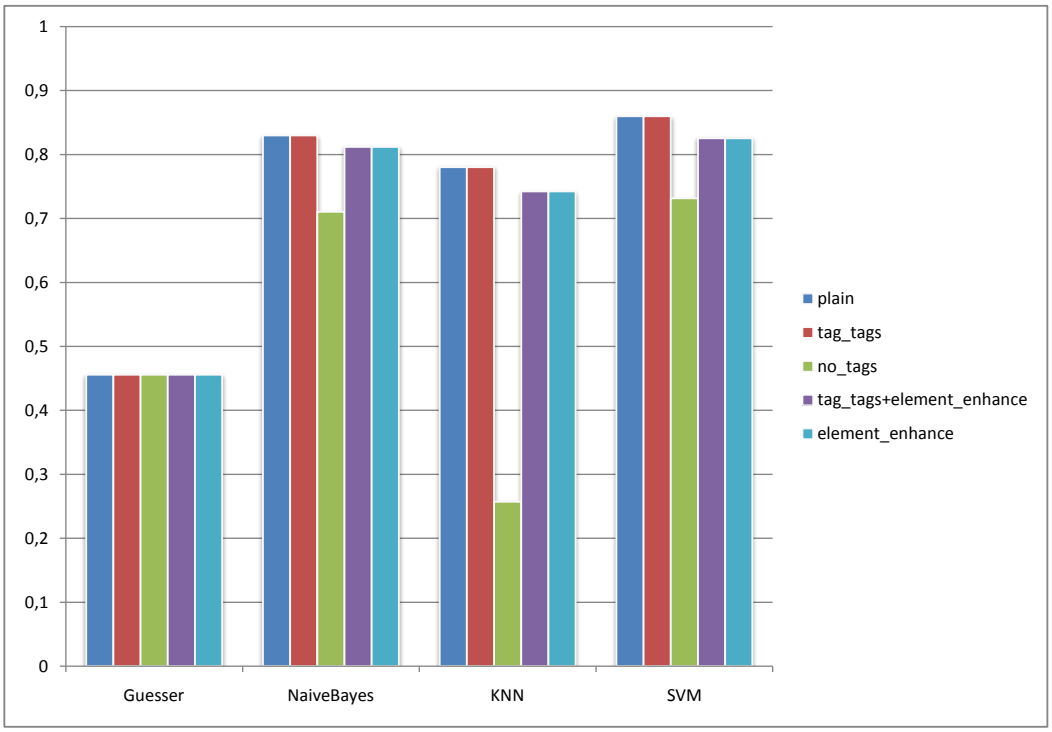


Abbildung 6.2.: F-Wert Formulare
Ergebnisse der Klassifikation der Formulare als Darstellung der F-Werte

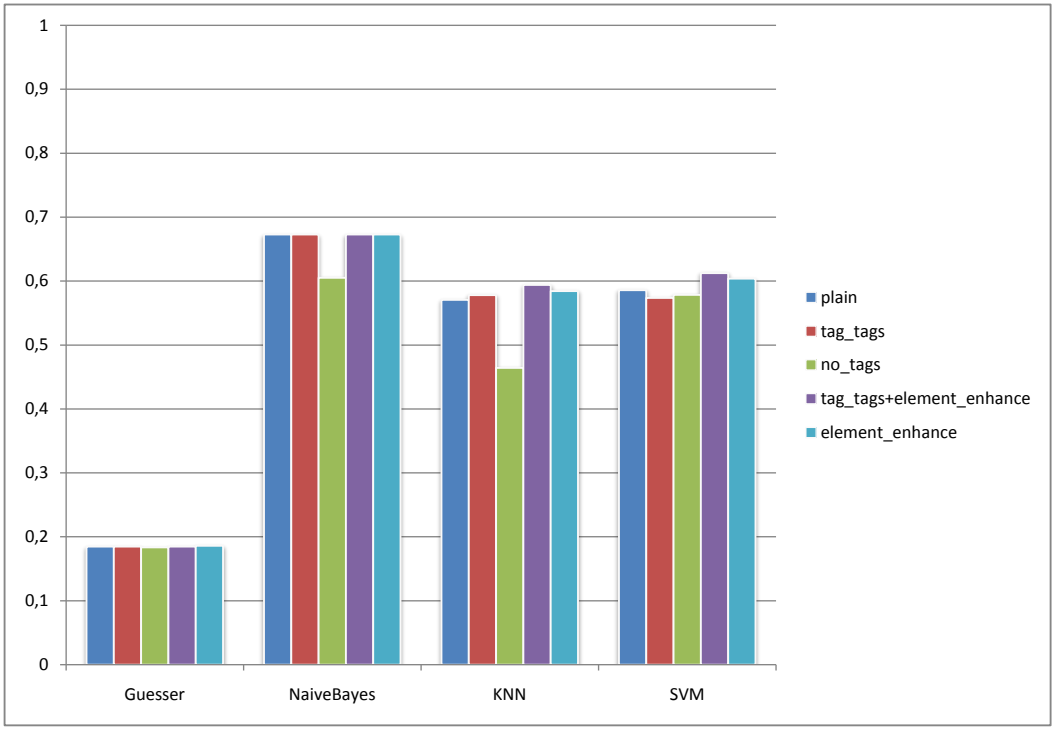


Abbildung 6.3.: F-Wert Felder
Ergebnisse der Klassifikation der Eingabefelder als Darstellung der F-Werte

**Ausgangsbedingungen:** Das Trainingsset umfasste 109 Dokumente, die häufigsten Kategorien waren “hidden” mit 47 Dokumenten und “misc” mit 26 Dokumenten.

### Bewertung der Klassifikationsalgorithmen

Bei diesem Test zeigt sich die SVM als den anderen deutlich überlegen. Die Algorithmen Bayes und KNN erreichen kaum Werte, die sich von dem Guesser abheben.

### Bewertung der Vorfilter

Der eigens für die Verbesserung der Feldklassifikation entwickelte Filter “element_enhance” zeigte nur beim Bayes-Algorithmus Wirkung. Bei den anderen beiden Klassifikatoren zeigte er kaum eine Verbesserung des Ergebnisses.

### 6.2.4. Zusammenfassung

Die Frage, ob Webseiten mittels Textklassifikatoren verstanden werden können, kann mit “ja” beantwortet werden, wobei jedoch einiges beachtet werden muss. Diese Einschränkungen werden im nächsten Abschnitt genauer beschrieben. Die gesamten Ergebnisse in Zahlen sind in Tabellenform in Appendix A aufgeführt.

#### Erfahrungen

Die Festlegung der Kategorien ist entscheidend für den Erfolg. In der Anfangsphase der Arbeit war noch nicht klar, welche Arten von Webseiten betrachtet werden sollen und wie diese aufzuteilen sind. Es wurde ein System gewählt, mit dem Webseiten auch mehreren Kategorien zugeordnet werden können. Etwa wurden der Seite der Wiener Linien die Kategorien “urban transportation” und “vienna” zugeordnet.

Betrachtet man beispielsweise nur die Domäne der Flugsuche, so ist es zielführend, nur zwei Kategorien zu verwenden: “flight search” und “nothing”.

Das Weglassen von Informationen durch einen Vorfilter brachte nie eine Verbesserung der Ergebnisse. Aber auch das Hinzufügen brachte nicht immer einen Erfolg.

# 7. Ausblick und Zusammenfassung

In diesem Kapitel werden mögliche Anwendungen und Folgeprojekte dieser Arbeit besprochen, sowie die daraus gewonnen Erkenntnisse zusammengefasst.

## 7.1. Ausblick

Diese Arbeit stellt eine Basis für eine Reihe weiterer Projekte dar. Zwei davon werden in den folgenden Abschnitten beschrieben. Bis das Internet, viel mehr das World Wide Web, zu einem Semantic-Web geworden ist, wird noch sehr viel Arbeit notwendig sein. Annotierungssysteme, wie das in dieser Arbeit beschriebene, sind eine Möglichkeit um Eigenschaften eines Semantic-Web schon jetzt zu erhalten.

### 7.1.1. Metamorph

Eine direkte Folge dieser Arbeit ist sicher die Integration in MetaMorph als automatisches Annotierungsmodul für Webseiten. Die Eingabefelder werden auf dem semantischen Modell der jeweiligen Domäne abgebildet. Weiters ist die Koppelung mit dem Spider-Mechanismus einer Suchmaschine denkbar, um relevante Seiten selbsttätig aufzufinden. Diese Suchmaschine verfügt dann über die notwendigen Informationen um auch komplexe Anfragen[1] verstehen und beantworten zu können. Zudem kann diese Suchmaschine ihren Index damit auch auf Inhalte ausdehnen, welche nur über Abfragen zugänglich sind. Dieses Query-Probing [IGS00] kann durch das automatische Verständnis der abgefragten Seiten zielgerichteter durchgeführt werden.

[1] Beispiel: ”Welche Flüge gehen morgen von Wien nach Dublin?”

### 7.1.2. Firefox Plugin

Bereits jetzt gibt es Funktionen in Firefox und in einigen Plugins, welche Formulare automatisch ausfüllen können. Diese Funktionen greifen jedoch auf Heuristiken zurück, um die benötigten Felder zu erkennen. Mit einer statistischen Klassifikation könnte die Erfolgsrate solcher Funktionen erhöht werden.

Geht man noch weiter, so ist ein System denkbar, welches im Stil des "Web 2.0" die Benutzer dazu verwendet, um neue Seiten zu finden und zu annotieren. So wird dem Benutzer bei einer Webseite angezeigt, ob diese bereits von Menschen annotiert wurde oder nicht. Ist dies nicht der Fall, so wird die Seite automatisch annotiert und die Ergebnisse dem Benutzer angezeigt. Dieser kann dann Korrekturen vornehmen oder das Ergebnis bestätigen. Das System wird so von den Benutzern immer weiter verbessert, wie etwa das unter `http://www.20q.net/` zu findende Ratespiel.

## 7.2. Zusammenfassung

In dieser Arbeit habe ich untersucht, inwieweit man statistische Textklassifikationsalgorithmen auf semistrukturierte Daten, in diesem Fall HTML, einsetzen kann, um diese Daten zu verstehen. Diese Arbeit ist eng mit den Bereichen Information Retrieval[2] und Information Extraction[3] verbunden. Ich habe dabei unterschiedliche Klassifikationsalgorithmen und Vorfilter verglichen. Ein weiterer Punkt ist die Beschreibung der Klassifikationsalgorithmen in der Theorie. Ich legte großes Augenmerk auf die leichte Verständlichkeit der Beschreibungen für fachfremde Personen.

Im Zuge der Arbeit entstand ein Experimentiersystem, mit welchen auch die Ergebnisse aus Kapitel 6 ermittelt wurden. Für die Anbindung an andere Systeme wurde eine Schnittstelle vorgesehen, mit welcher Webseiten annotiert werden können. Die Ergebnisse der Tests sind gut, aber die Auswahl und die Kategorisierung der Trainingsdaten hatte sich als noch nicht optimal herausgestellt. Für eine praktische Anwendung ist ein größerer Trainingsdatensatz notwendig.

[2] Das Klassifizieren der ganzen Seiten.
[3] Das Klassifizieren der Elemente einer Seite.

# Appendices

# A. Ergebnistabellen

In den folgenden Tabellen ist N die Anzahl der klassifizierten Dokumente. Die Spalte Filter beschreibt den verwendeten Vorfilter und Algorithmus den Klassifikationsalgorithmus.

| Filter | Algorithmus | N | recall | precision | fallout | spezifitaet | F |
|---|---|---|---|---|---|---|---|
| plain | Guesser | 123 | 0.333 | 0.128 | 0.171 | 0.829 | 0.185 |
| plain | NaiveBayes | 123 | 0.695 | 0.652 | 0.074 | 0.926 | 0.673 |
| plain | KNN | 123 | 0.589 | 0.553 | 0.112 | 0.888 | 0.571 |
| plain | SVM | 123 | 0.561 | 0.613 | 0.072 | 0.928 | 0.586 |
| tag_tags | Guesser | 123 | 0.333 | 0.128 | 0.171 | 0.829 | 0.185 |
| tag_tags | NaiveBayes | 123 | 0.695 | 0.652 | 0.074 | 0.926 | 0.673 |
| tag_tags | KNN | 123 | 0.597 | 0.560 | 0.109 | 0.891 | 0.578 |
| tag_tags | SVM | 123 | 0.544 | 0.607 | 0.073 | 0.927 | 0.574 |
| no_tags | Guesser | 123 | 0.333 | 0.127 | 0.169 | 0.831 | 0.183 |
| no_tags | NaiveBayes | 123 | 0.612 | 0.598 | 0.102 | 0.898 | 0.605 |
| no_tags | KNN | 123 | 0.477 | 0.452 | 0.143 | 0.857 | 0.464 |
| no_tags | SVM | 123 | 0.509 | 0.670 | 0.058 | 0.942 | 0.578 |
| tag_tags,element_enhance | Guesser | 123 | 0.333 | 0.128 | 0.171 | 0.829 | 0.185 |
| tag_tags,element_enhance | NaiveBayes | 123 | 0.695 | 0.652 | 0.074 | 0.926 | 0.673 |
| tag_tags,element_enhance | KNN | 123 | 0.613 | 0.576 | 0.098 | 0.902 | 0.594 |
| tag_tags,element_enhance | SVM | 123 | 0.586 | 0.641 | 0.066 | 0.934 | 0.612 |
| element_enhance | Guesser | 123 | 0.333 | 0.129 | 0.170 | 0.830 | 0.186 |
| element_enhance | NaiveBayes | 123 | 0.695 | 0.652 | 0.074 | 0.926 | 0.673 |
| element_enhance | KNN | 123 | 0.597 | 0.572 | 0.094 | 0.906 | 0.584 |
| element_enhance | SVM | 123 | 0.578 | 0.632 | 0.068 | 0.932 | 0.604 |

Tabelle A.1.: Ergebnisse HTML

| Filter | Algorithmus | N | recall | precision | fallout | spezifitaet | F |
|---|---|---|---|---|---|---|---|
| plain | Guesser | 43 | 0.605 | 0.366 | 0.302 | 0.698 | 0.456 |
| plain | NaiveBayes | 43 | 0.837 | 0.823 | 0.095 | 0.905 | 0.830 |
| plain | KNN | 43 | 0.767 | 0.793 | 0.126 | 0.874 | 0.780 |
| plain | SVM | 43 | 0.837 | 0.884 | 0.093 | 0.907 | 0.860 |
| tag_tags | Guesser | 43 | 0.605 | 0.366 | 0.302 | 0.698 | 0.456 |
| tag_tags | NaiveBayes | 43 | 0.837 | 0.823 | 0.095 | 0.905 | 0.830 |
| tag_tags | KNN | 43 | 0.767 | 0.793 | 0.126 | 0.874 | 0.780 |
| tag_tags | SVM | 43 | 0.837 | 0.884 | 0.093 | 0.907 | 0.860 |
| no_tags | Guesser | 43 | 0.605 | 0.366 | 0.302 | 0.698 | 0.456 |
| no_tags | NaiveBayes | 43 | 0.721 | 0.700 | 0.221 | 0.779 | 0.710 |
| no_tags | KNN | 43 | 0.326 | 0.212 | 0.126 | 0.874 | 0.257 |
| no_tags | SVM | 43 | 0.698 | 0.769 | 0.191 | 0.809 | 0.732 |
| tag_tags,element_enhance | Guesser | 43 | 0.605 | 0.366 | 0.302 | 0.698 | 0.456 |
| tag_tags,element_enhance | NaiveBayes | 43 | 0.814 | 0.810 | 0.103 | 0.897 | 0.812 |
| tag_tags,element_enhance | KNN | 43 | 0.721 | 0.765 | 0.142 | 0.858 | 0.742 |
| tag_tags,element_enhance | SVM | 43 | 0.814 | 0.837 | 0.129 | 0.871 | 0.825 |
| element_enhance | Guesser | 43 | 0.605 | 0.366 | 0.302 | 0.698 | 0.456 |
| element_enhance | NaiveBayes | 43 | 0.814 | 0.810 | 0.103 | 0.897 | 0.812 |
| element_enhance | KNN | 43 | 0.721 | 0.765 | 0.142 | 0.858 | 0.742 |
| element_enhance | SVM | 43 | 0.814 | 0.837 | 0.129 | 0.871 | 0.825 |

Tabelle A.2.: Ergebnisse Form

| Filter | Algorithmus | N | recall | precision | fallout | spezifitaet | F |
|---|---|---|---|---|---|---|---|
| plain | Guesser | 109 | 0.440 | 0.426 | 0.214 | 0.786 | 0.433 |
| plain | NaiveBayes | 109 | 0.532 | 0.520 | 0.090 | 0.910 | 0.526 |
| plain | KNN | 109 | 0.505 | 0.474 | 0.111 | 0.889 | 0.489 |
| plain | SVM | 109 | 0.633 | 0.644 | 0.030 | 0.970 | 0.639 |
| element_enhance | Guesser | 109 | 0.431 | 0.186 | 0.216 | 0.784 | 0.260 |
| element_enhance | NaiveBayes | 109 | 0.560 | 0.559 | 0.076 | 0.924 | 0.560 |
| element_enhance | KNN | 109 | 0.569 | 0.421 | 0.140 | 0.860 | 0.484 |
| element_enhance | SVM | 109 | 0.633 | 0.650 | 0.031 | 0.969 | 0.641 |

Tabelle A.3.: Ergebnisse Field

# B. Literaturverzeichnis


[Aer07] Aeroflot. Flugsuche. Internet, 2007. http://www.aeroflot.ru/main.aspx.

[BCL05] R. Baumgartner, M. Ceresna, and G. Ledermüller. Deep web navigation in web data extraction. In *Proc. of IAWTIC*, 2005.

[BFG01] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Declarative information extraction, web crawling, and recursive wrapping with lixto, 2001.

[BGH03] Robert Baumgartner, Georg Gottlob, and Marcus Herzog. Lixto - Halfway to the Semantic Web. *OEGAI-Journal 1*, pages 19–24, 2003.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. 1st edition, 2006.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. In *Scientific American*, May 2001.

[Bus45] Vannevar Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.

[CDI98] Soumen Chakrabarti, Bryon Dom, and Piotr Indyk. Enhanced hypertext categorisation using hyperlinks. Technical report, IBM Almaden, 1998.

[DHS01] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. 2nd edition, 2001.

[EG00] Martin Ester and Matthias Groß. Adriane: ein fokussierter web-crawler mit adaptiver klassifikation der hyperlinks. Technical report, Institut für Informatik, Ludwig-Maximilians-Universität München, 2000.

[Eik99] Line Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.

[Fer07] Fernglasshop. Produktliste. Internet, 2007. http://www.icoshopping.com/.

[HB05] Tamir Hassan and Robert Baumgartner. Intelligent wrapping from pdf documents with lixto. In *Proc. of IAWTIC*, 2005.

[HKG01] Ulrich Hoffrage, Stephanie Kurzenhäuser, and Gerd Gigerenzer. Positive mammographie = brustkrebs? von den schwierigkeiten im umgang mit statistischen informationen. *Schweizer Zeitschrift für Managed Care and Care Management*, 3:22–25, 2001.

[HML+07] Hai He, Weiyi Meng, Yiyao Lu, Clement Yu, Zonghuan Wu, and Prof Weiyi Meng. Towards deeper understanding of the search interfaces of the deep web. *World Wide Web*, 10:133–155, 2007.

[HMYW04] Hai He, Weiyi Meng, Clement Yu, and Zonghuan Wu. Automatic integration of web search interfaces with WISE-integrator. *The VLDB Journal*, 13(3):256–273, 2004.

[IGS00] Panagiotis G. Ipeirotis, Luis Gravano, and Mehran Sahami. Automatic classification of text databases through query probing. In *In Proceedings of the ACM SIGMOD Workshop on the Web and Databases (WebDB2000)*, 2000.

[Lin07] Aer Lingus. Flugsuche. Internet, 2007. http://www.aerlingus.com/.

[MWLK03] Xiaofeng Meng, Haiyan Wang, Chen Li, and Huizhong Kou. A schema-guided toolkit for generating wrappers. In *Proc. of WEBSA2003*, 2003.

[SE95] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the metacrawler. Proceedings of the 1995 World Wide Web Conference, October 1995.

[Wik08] Wikipedia. Recall und precision. Internet, April 2008. http://de.wikipedia.org/wiki/Recall_und_Precision.

[WYDM04] Wensheng Wu, Clement Yu, Anhai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *In SIGMOD Conference*, pages 95–106, 2004.

[ZMW+05] Hongkun Zhao, Weiyi Meng, Zonghuan Wu, Vijay Raghavan, and Clement Yu. Fully automatic wrapper generation for search engines. Proc. of 14th International World Wide Web Conference (WWW14), pp.66-75, Chiba, Japan, May 2005.

# C. Abbildungsverzeichnis

# D. Tabellenverzeichnis

# E. Listings