Institut für Maschinelle Sprachverarbeitung

Universität Stuttgart
Pfaffenwaldring 5b
D-70569 Stuttgart

Bachelorarbeit Nr. 337

Fokussiertes Webcrawling auf Basis von Aktivem Lernen

Alexander Bernhardt

Studiengang:	Informatik
Prüfer/in:	Sebastian Padó
Betreuer/in:	Roman Klinger
Beginn am:	1. Mai 2016
Beendet am:	31. Oktober 2016
CR-Nummer:	H.3.3, I.7.2, I.7.5

Zusammenfassung

In dieser Bachelorarbeit wird der Frage nachgegangen, inwieweit es möglich ist, Informationen zu einem bestimmten Themengebiet aus dem World Wide Web zu extrahieren, die den Vorstellungen eines Benutzers entsprechen und zugleich keine enorme Datenmenge angesammelt wird. Um dies herauszufinden wurde ein fokussierter Webcrawler entwickelt, der um eine Nutzerschnittstelle erweitert wurde, damit das Paradigma Aktives Lernen umgesetzt werden konnte und somit die Klassifizierung zusätzlich gesteuert werden kann. Anhand von durchgeführten Experimenten wurde das entwickelte System evaluiert. Dazu wurde eine Datenmenge an Webseiten durch einen Benutzer manuell klassifiziert und anschließend versucht, durch die Konfiguration von dem Threshold-Parameter, die identische Datenmenge zu erzielen. Eine Konfiguration ergab nahezu das angestrebte Ziel und zeigte somit, dass Aktives Lernen in Verbindung mit der Klassifizierung von Webseiten durchaus sinnvoll ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Dieses Kapitel soll den Leser an das Thema der Bachelorarbeit heranführen. Dabei wird in Kürze erläutert, weshalb dieses Thema untersucht wird und was das Ziel dieser Arbeit ist.

1.1 Motivation

Heutzutage hat das World Wide Web (kurz Web) den wohl größten Datenbestand der Welt. Jede Person ist in der Lage eine Webseite, die Dokumente und Links enthalten, zu erstellen. Das damit verbundene, unkontrollierte Wachstum führt zu einem sehr komplexen und riesigen Web [8]. Auch wenn das Web als gerichteter Graph betrachtet werden kann, wobei die Knoten den Dokumente und die Kanten den Hyperlinks entsprechen und damit die Konnektivität festgelegt wird, ist es durch die Dynamik und die enorme Größe des Webs nicht möglich, das ganze Web zu erfassen [8].

Um aus diesem komplexen Webgraphen Informationen für einen Nutzer, der diese über Suchmaschinen gewinnen möchte, gezielt herauszuholen, werden Webcrawler benötigt. Die Nutzung allgemeiner Webcrawler dient der breiten Websuche, also möglichst viele Informationen sollen erfasst werden. Dabei werden sehr große Datenmengen aufgebaut. Diese aufgebaute Datenmenge kann durch den Einsatz von fokussierten Webcrawlern, die für die vertikale Websuche eingesetzt werden, reduziert werden. Nicht enorme Datenmengen sollen durchsucht, gespeichert und indexiert werden, sondern nur bestimmte, für den Nutzer thematisch relevante Segmente des Webs. Gleichzeitig wird auch der Ressourcenaufwand verringert, da thematisch irrelevante Bereiche des Web unberücksichtigt bleiben.

1.2 Ziel der Arbeit

Im Laufe der Bachelorarbeit soll ein interaktiver, fokussierter Webcrawler basierend auf dem Paradigma *Aktives Lernen* entwickelt werden. Auf Grundlage der Entwicklung dieses Webcrawlers ist das primäre Ziel abzuschätzen, inwieweit sich der durch den Webcrawler aus dem Internet herausgefilterte Informationsgehalt mit den Vorstellungen eines Benutzers deckt. Es muss also der Frage nachgegangen werden, ob sich der Anteil relevanter Information, bei gleichzeitiger Reduktion der irrelevanten Information mit *Aktivem Lernen*, erhöhen lässt. Die erzeugten Ergebnisse werden daraufhin empirisch evaluiert, zum Beispiel durch die Durchführung einer Nutzerstudie oder ein Vergleich mit alternativen Webcrawlern.

2 Bisherige Arbeiten

Im Folgenden werden grundlegende Begriffe, Methoden und Verfahren erläutert, die im Laufe der Bachelorarbeit zum Einsatz gekommenen sind oder zukünftig verwendet werden können. Zuallererst wird ein kurzer Überblick über das Webcrawling gegeben und die Funktionsweise eines Webcrawlers erläutert. Daraufhin wird das Thema Klassifikation behandelt und ein spezieller Klassifikator vorgestellt. Zuletzt wird das Konzept Aktives Lernen beschrieben und anhand von Szenarien, sowie einer Query-Strategie näher betrachtet.

2.1 Webcrawling

Als Webcrawling wird der Prozess bezeichnet bei dem Webseiten aus dem Internet erfasst und gesammelt werden, um sie daraufhin zu indexieren [1]. Diese Art der Datengewinnung stellt die Grundlage für eine Suchmaschine dar, damit für das Beantworten von Suchanfragen eine möglichst aktuelle und umfassende Basis an Daten bereitstehen [1]. Das somit verfolgte Ziel von Webcrawling ist, möglichst viele nutzbringende Webseiten zusammen mit deren Linkstrukturen auf eine schnelle und effiziente Weise zu erfassen [1]. Die praktische Umsetzung erfolgt durch den Einsatz von sogenannten Webcrawlern, die vereinzelt auch *Spider* genannt werden. Diese sind Softwareprogramme, die das Web automatisch durchsuchen und die dabei gefundenen Webseiten analysieren. In Abbildung 2.1 wird die Crawler-Architektur eines konventionellen Webcrawler vereinfacht dargestellt, wie sie Heydon und Najork [3] in ihrer wissenschaftlichen Arbeit beschreiben. Die einzelnen Module und deren Funktionalität werden im Folgenden erläutert. Dafür wurde auf den Inhalt aus dem Buch von Manning et al. [1] zurückgegriffen.

Seeds

Die Menge an URLs, welche die Einstiegspunkte in das Web beschreiben und in der Frontier als Initialisierung gespeichert werden, werden Seed-URLs genannt. Dabei lässt sich erkennen, dass es sinnvoll ist, geeignete Einstiegsseiten zu wählen.

DNS

Dieses Modul umfasst die Umwandlung von einem Hostnamen in eine IP-Adresse.

Frontier

Die Frontier ist eine Datenstruktur, welche die darin gespeicherten URLs organisiert. Aus ihr werden die URLs geladen. Zu Beginn erfolgt die Initialisierung der Frontier durch die Menge der Seed-URLs.

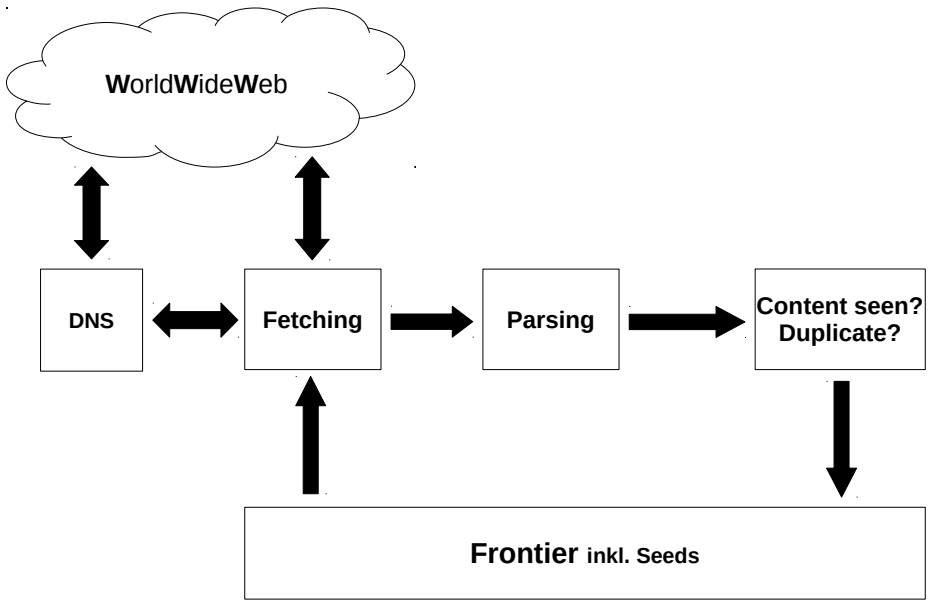

Abbildung 2.1: Die grundlegenden Komponenten und deren Zusammenspiel beschreiben einen konventionellen Webcrawler.

Fetching

Das Laden einer aus der Frontier entnommenen URL übernimmt dieses Modul. Anschließend werden die erhaltenen Daten an das Parsing-Modul zur Weiterverarbeitung übergeben. Im Allgemeinen wird dabei das *http*- oder *https*-Protokoll verwendet.

Parsing

Nachdem die Daten einer Webseite geladen wurden, werden die darin enthaltenen Hyperlinks und der Text extrahiert. Die extrahierten Hyperlinks werden der Frontier hinzugefügt und der Prozess beginnt erneut mit dem Entnehmen, Laden und Parsen einer URL.

Duplicate? Content seen?

Die Überprüfung, ob eine extrahierte URL der Frontier hinzugefügt werden soll, übernehmen diverse Check-Module. So wird zum Beispiel durch die Umsetzung Duplikaterkennung ausgeschlossen, dass ein und dieselbe URL mehrmals geladen wird.

Der in 2.1 dargestellte Crawl-Vorgang kann wie folgt beschrieben werden: Zu Beginn wird die Frontier mit den Seed-URLs initialisiert. Ausgehend von diesen auserwählten URLs beginnt der eigentliche Crawl-Prozess. Eine URL wird der Frontier entnommen, gefetcht und geparsed, um daraufhin die Informationen und die Hyperlinks dieser Webseite zu extrahieren. Die extrahierten Hyperlinks werden wiederum der Frontier hinzugefügt. Dieser Vorgang wird nun solange wiederholt bis ein Abbruchkriterium erfüllt ist, beispielsweise bis die Frontier keine URLs mehr beinhaltet.

Nach der Vorstellung der konventionellen Crawler-Architektur und dessen Ablauf ist es wichtig zu überlegen, welche Anforderungen ein Crawler zu erfüllen hat. Manning et al. [1] unterscheiden solche Anforderungen, die erfüllt sein müssen und andere, die erfüllt sein sollten. Zu den Erstgenannten gehören Robustheit (robustness) und Höflichkeit (politeness). Ein Webcrawler wird als robust angesehen, sofern er nicht anfällig gegen *Spider traps* ist. Eine *Spider trap* wird durch einen im Web beinhalteten Server erstellt und führt dazu, dass ein Webcrawler in eine Endlosschleife gelangt. Der Webcrawler ruft dabei unendlich viele Webseiten der selben Domäne auf. Mit dem Begriff Höflichkeit wird die Eigenschaft eines Webcrawlers beschrieben, dass durch gewisse Richtlinien reguliert wird, wie und wann eine Webseite besucht werden kann. Zu den Anforderungen, die ein Webcrawler besitzen kann, gehören Eigenschaften wie beispielsweise Skalierbarkeit, Effizienz oder eine hohe Qualität. Das scheinbar einfache Traversieren durch den Webgraph ist aufgrund der vielen Anforderungen an ein Webcrawler-System in der Praxis sehr komplex. Man stelle sich vor, es werden 20 Billion Webseiten abgeholt. Dann bedeutet dies, dass der Webcrawler in der Lage sein muss 8000 Webseiten in einer Sekunde abzuholen. Neben einem konventionellen Crawler, der in erster Linie eine enorm große Datenmenge aufbaut, existiert eine weitere Art des Crawlens. Dies nennt sich fokussiertes Crawlen und hat das Ziel eine Datenmenge von einer bestimmten themenspezifischen Domäne aufzubauen. Im Umkehrschuss bewirkt dies eine Reduzierung der Datenmenge, die bei dem Crawl-Vorgang aufgebaut wird.

2.2 Klassifikator

Allgemein existiert bei einem Klassifikationsproblem eine Menge von Daten D und eine Menge von Klassen C. Nun ist eine Funktion, also der Klassifikator, gesucht, die jedem Element aus der Menge der Daten eine Klasse zuordnet und dabei bestimmte Kriterien erfüllt. Klassifikatoren lassen sich in zwei Kategorien einteilen. Dabei wird zwischen linearen und nicht-linearen Klassifikatoren unterschieden. Ein linearer Klassifikator entscheidet anhand einer Linearkombination von Merkmalen bezüglich einem Grenzwert b (Threshold) über die Klassenzugehörigkeit eines Objekts. Die Trennung der jeweiligen Klassen erfolgt durch eine lineare *Hyperebene*. Diese wird anhand von Trainingsdaten ermittelt, wobei es unendlich viele Hyperebenen für ein linear separierbares Problem gibt. Geometrisch gesehen entspricht die lineare Hyperebene im Falle von einem zweidimensionalen Raum einer Linie. Allgemein hat die Hyperebene die Form $\vec{w}^T\vec{x} = b$. Dabei beinhaltet der Vektor $\vec{x}$ die zu klassifizierenden Objekte, wohingegen $\vec{w}$ ein Gewichtsvektor beschreibt. Wird im Folgenden eine binäre Klassifikation in Betracht gezogen, das heißt es existieren nur zwei Klassen, so ergibt sich die Klassenzugehörigkeit von einem Objekt x anhand von folgendem Zuordnungskriterium: Das Objekt wird der Klasse c zugeordnet, falls $\vec{w}^T\vec{x} > b$ gilt und der Klasse $\bar{c}$ zugeordnet, sofern die Gleichung $\vec{w}^T\vec{x} < b$ erfüllt ist. Abbildung 2.2 zeigt eine Hyperebene, die Objekte in zwei Klassen aufteilt. Generell können es natürlich auch mehr als zwei Klassen sein. Der Maxmimum Entropy Klassifikator und der Naive Bayes Klassifikator sind Beispiele für lineare Klassifikatoren.

Die Trennung bei einem nicht-linearen Klassifikator entsteht aus mehreren linearen Segmenten, wobei die Form sehr komplex ist. In Abbildung 2.3 ist ein nicht lineares Problem

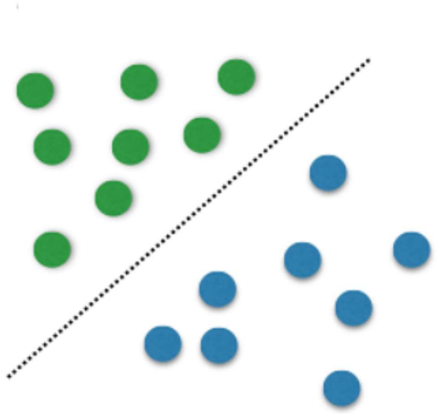

Abbildung 2.2: Ein lineares Problem, bei dem die grünen und blauen Objekte durch eine lineare Trennlinie (Trennfunktion) getrennt sind [6].

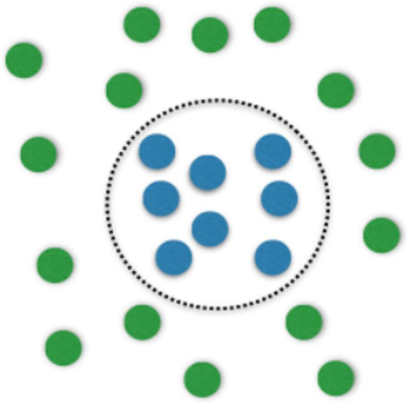

Abbildung 2.3: Ein nicht-lineares Problem, bei dem ein linearer Klassifikator die beiden Instanzmengen (grün und blau) nicht geeignet klassifizieren kann, da die Mengen nicht linear separierbar sind [6].

skizziert. Nicht-lineare Klassifikatoren sind mächtiger als lineare Klassifikatoren [1]. Aber bedeutet dies nun, dass ein nicht-linearer Klassifikator einem linearen Klassifikator bevorzugt werden soll? Die Antwort lautet Nein, denn es existiert ein *Trade-off* zwischen dem *Bias* und der *Varianz*, bekannt als *Bias-Varianz-Dilemma* [1].

Letzten Endes stellt sich die Frage, ob es einen „perfekten“ Klassifikator für jedes Problem gibt? Solch einen Klassifikator gibt es leider nicht [1]. Dazu müssen einige Fragen in Betracht gezogen werden, beispielsweise wie viele Trainingsdaten für ein Problem existie-

ren? Existieren zu wenige Trainingsdaten so kann es passieren, dass die Gesetzmäßigkeiten in den Trainingsdaten nicht sonderlich genau erfasst werden [1]. Dies hätte wiederum eine negative Auswirkung auf noch nicht gesehene Testdaten. Weitere aufkommende Fragen sind: Wie komplex ist das Klassifizierungsproblem oder wie stabil ist das Problem im Laufe der Zeit? Bei einem instabilen Problem sollte auf einen robusten und einfachen Klassifikator zurückgegriffen werden [1].

Der nächste Abschnitt beschäftigt sich mit einem speziellen probabilistischen Klassifikator, dem Maximum Entropy Klassifikator.

2.2.1 Maximum Entropy

Der Maximum Entropy Klassifikator, in der Literatur auch häufig MaxEnt genannt, gehört in die Kategorie der linearen Klassifikatoren. Er basiert auf dem Prinzip der maximalen Entropie, wobei Entropie ein Maß für den Informationsgehalt von Wörtern bezeichnet [1]. Die folgende Exponentialfunktion berechnet die maximale Entropie:

$$p_\lambda(y|\mathbf{x}) = \frac{e^{\sum_i \lambda_i f_i(y,\mathbf{x})}}{\sum_{y'} e^{\sum_i \lambda_i f_i(y',\mathbf{x})}} \tag{2.1}$$

$$= \frac{1}{Z(\mathbf{x})} e^{\sum_i \lambda_i f_i(y,\mathbf{x})} \tag{2.2}$$

Dabei entspricht $Z(\mathbf{x}) = e^{\sum_i \lambda_i f_i(y',\mathbf{x})}$ der normalisierten Exponentialfunktion.

Nachfolgend wird ein kurzer Überblick über die verwendeten Variablen aus der oben gezeigten Gleichung gegeben.

$\mathbf{x}$: gegebene Daten
y: Klassenvariable
$f_i(y, \mathbf{x})$: Merkmale (Beziehung zwischen einem Wort und einer Klasse)
λ_i: Parameter, der für die Gewichtung der Merkmale verantwortlich ist
$Z(\mathbf{x})$: normalisierte Exponentialfunktion

Wie in Formel 2.1 zu sehen ist gewichtet der Klassifikator jedes einzelne Feature (Wort). Dabei indiziert eine höhere Gewichtung eines Features eine stärkere Klassenzugehörigkeit [4]. Im Gegensatz zu Naive Bayes macht MaxEnt keine Annahmen über die Beziehungen zwischen den jeweiligen Features (Wörter).

Kuhlen und Semar [5] beschreiben des Klassifikators wie folgt: Die unbekannte Wahrscheinlichkeitsverteilung M^* soll sehr nahe an die wirkliche Merkmalsverteilung M gebracht. M ist die Verteilung, welche die Trainingsdaten T erzeugt. Da es mehrere Wahrscheinlichkeitsverteilungen M^* gibt, die der Merkmalsverteilung in T entsprechen, wird die Merkmalsverteilung M^* mit der maximalen Entropie ausgewählt. Somit hat M^* die wenigste externe Information, die überhaupt möglich ist.

2.3 Textklassifizierung

Die Inhalte für den folgenden Abschnitt wurden weitestgehend aus dem Buch von Manning et al. [1] entnommen.

Textklassifizierung befasst sich mit der inhaltlichen Analyse von Texten, um diese nach bestimmten inhaltlichen Kriterien zu klassifizieren [2]. Im einfachsten Fall existieren genau zwei Klassen, die zum Beispiel mit *relevant* und *nicht relevant* bezeichnet werden. Es können aber auch mehr als zwei Klassen existieren. Doch welche Möglichkeiten existieren, um Dokumente zu klassifizieren? Hier wird zwischen drei Ansätzen unterschieden. Bei der manuellen Klassifizierung, die Yahoo zu Beginn des Web verwendet hat, werden Dokumente durch einen Menschen klassifiziert. Handelt es sich hierbei um einen Experten, so kann eine sehr hohe Genauigkeit erzielt werden. Jedoch ist dies sehr aufwendig und kostspielig, da sehr viele Dokumente klassifiziert werden müssen. Deshalb wird bei vielen und vor allem bei größeren Problemen eine Methode der automatischen Klassifizierung bevorzugt. Der zweite Ansatz beruht auf Regeln, oftmals sind dies Boolsche Ausdrücke, die mit Hilfe von Entwicklungsumgebungen, wie z.B. Verity, erstellt werden können. Diese Vorgehensweise ist sehr mühsam und teuer, da die Genauigkeit erst mit der Präzision der formulierten Regeln wächst. Google Alerts ist ein Beispiel für die Nutzung dieses Ansatzes. Die dritte Kategorie besteht aus probabilistischen Ansätzen. Darunter fällt auch der zuvor erklärte Maximum Entropy Klassifikator, weshalb hier nicht weiter im Detail darauf eingegangen wird.

Formal gesehen lässt sich die Textklassifizierung durch drei Größen charakterisieren. Die erste Größe beschreibt den Dokumentenraum $\mathbf{X}$, in welchem alle Dokumente repräsentiert werden. Oftmals werden hochdimensionale Räume verwendet. Eine weitere Größe besteht aus einer festen Menge von Klassen $\mathbf{C} = \{c_1, c_2, \ldots, c_j\}$, die durch einen Menschen speziell für ein bestimmtes Anwendungsproblem definiert wird. Die dritte Größe beschreibt die Menge der Trainigsdokumente $\mathbf{D} = \{d_1, d_2, \ldots, d_i\}$. Jedes Element dieser Trainingsmenge ist schon zu Beginn einer bestimmten Klasse zugeordnet, d.h. $\langle d, c\rangle \in \mathbf{X} \times \mathbf{C}$. Mit Hilfe eines Lernverfahrens (Lernalgorithmus) wird nun ein Klassifikator γ trainiert, der daraufhin in der Lage ist Dokumente einer Klasse zuzuordnen Formal ausgedrückt bedeutet dies $\gamma : \mathbf{X} \to \mathbf{C}$. Diese Art des Lernens wird als überwachtes Lernen (supervised learning) bezeichnet: Die durch einen Menschen annotierten Daten sind dabei gegeben. Aus diesen Daten wird eine Funktion $y = f(x)$ gelernt, wobei x die Daten sind und y die zu prognostizierte Klasse beschreibt. Die Menge an Parametern eines antrainierten Systems wird als Modell bezeichnet. Zum überwachten Lernen hinzu, existieren noch diverse andere Lernverfahren, zum Beispiel das nicht überwachtes Lernen (unsupervised learning). Auf diese Art des Lernens wird im weiteren Verlauf nicht näher eingegangen. Bei all diesen Verfahren liegt jedoch das Bestreben darin, die Qualität bei der Klassifizierung von nicht gesehenen und analysierten Dokumenten zu maximieren.

Wird die Entwicklung eines wie oben beschriebenen Modells genauer betrachtet, so spielen dabei typischerweise drei Datenmengen eine Rolle: Trainingsdaten, Validierungsdaten und Testdaten. Die Menge an Trainingsdaten werden benutzt, um die Parameter eines Modells zu optimieren. Mit Hilfe der Validierungsdaten kann das Modell zwischenzeitlich bewertet werden, um gegebenenfalls Optimierungen vorzunehmen. Die Testdaten bestehen aus Daten, die zuvor keinen Einfluss auf die Entwicklung des Modells hatten und somit noch nicht betrachtet wurden. Mit den Daten wird letztlich das Modell getestet.

2.4 Aktives Lernen

Das Paradigma Aktives Lernen ist ein Teilgebiet des maschinellen Lernens, genauer gesagt der künstlichen Intelligenz und kann unter anderem Klassifizierungsverfahren erweitern [9]. Die Informationen für die folgenden Abschnitte zum Thema Aktives Lernen wurde aus dem Buch von Sickert [7] und aus dem technischen Bericht von Settles [9] entnommen.

2.4.1 Idee und Ziele

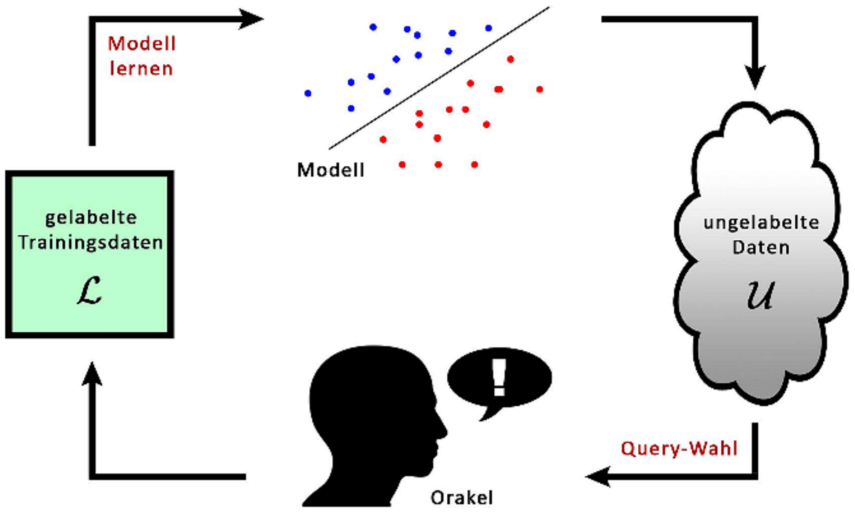

Abbildung 2.4: Der Zyklus eines aktiven Lernprozesses: Durch manuell gelabelte Trainingsdaten wird ein Modell erlernt. Anhand einer Query-Strategie und dem zuvor trainierten Klassifikator wird ein neues Objekt, das klassifiziert werden soll, gewählt und gelabelt [10].

Aktives Lernen (kurz AL) macht die Annahme, dass zu Beginn des Lernprozesses meist nur wenige oder gar keine gelabelten *Instanzen* (die auch Muster oder Beobachtungen genannt werden) vorhanden sind. Infolgedessen erhöht AL in jedem Iterationsschritt die Menge der gelabelten Trainingsinstanzen mit dem Ziel, die Güte des Lernmodells mit Hilfe von „richtig“ gestellten Fragen zu maximieren. Gleichzeitig soll aber dieser Vorgang die kleinstmöglichen Kosten verursachen. Die erzielte Güte hängt zum einen von der gelabelten Trainingsmenge und zum anderen vom Typ des Klassifikators und dessen Parametrierung ab. Um diese Ziele zu erreichen wird eine sogenannte Selektionsstrategie verwendet. Die Idee dahinter ist unter Berücksichtigung des aktuellen „Wissensstands“ des Lernmodells hoch informative, nicht-gelabelte Instanzen zu wählen, die daraufhin durch ein Orakel (menschlicher Experte) gelabelt werden (d.h. mit passendem Zielwert versehen

werden). Die gelabelten Instanzen werden zur Trainingsmenge hinzugefügt und aktualisiert. Dieser Vorgang wird so oft wiederholt bis ein Abbruchkriterium erreicht ist. Der Ablauf dieses Zyklus ist in Abbildung 2.4 vereinfacht skizziert. Der oben erwähnte Begriff informativ bedeutet, dass mit sehr hoher Wahrscheinlichkeit die Güte des Lernmodells verbessert wird.

Wird ein genauerer Blick auf das Orakel geworfen, so wird schnell klar, dass dieses auch nicht immer zuverlässig ist. So kann ein Mensch beispielsweise von anderen Dingen abgelenkt werden oder müde sein. Dadurch lässt die Konzentration nach und das Ergebnis wird verfälscht. Die Qualität der Arbeit eines Orakels kann außerdem sehr variabel sein, da jeder Mensch unterschiedliche Ansichten und Auffassungen hat, ob nun etwas sehr wichtig, wichtig oder gar unwichtig erscheint.

Aktuelle Veröffentlichungen zeigen den zunehmenden Einsatz von AL bei bekannten Unternehmen, darunter IBM, Microsoft, Yahoo oder Mitsubishi. Diese verwenden AL hauptsächlich in den Bereichen Textklassifikation, Spracherkennung und Bildklassifikation. Microsoft und Yahoo nutzen AL zum Beispiel um „*Kommentare mit missbräuchlichen Inhalten in einem Newsportal zu erkennen*“. Weitere Anwendungsbereiche sind unter anderem Erkennung von Maleware oder Wirkstoffdesign.

2.4.2 Szenarien

Es gibt viele unterschiedliche Situationen, in denen Queries bzw. Anfragen gestellt werden können. Die drei wesentlichen Szenarien, Stream-Based Selective Sampling, Pool-Based Sampling und Membership Query Synthesis stellt Settles in seinem veröffentlichten technischen Bericht [9] vor. Die beiden Erstgenannten werden nun genauer erläutert. Für alle Szenarien wird die Annahme getroffen, dass Queries die Form von nicht-gelabelten Instanzen entsprechen.

Bei *Stream-Based Selective Sampling* wird zuvor angenommen, dass das Erhalten einer nicht-gelabelten Instanz keine Kosten oder zumindest nur sehr wenige Kosten verursacht. Diese Instanzen werden einzeln aus dem Instanzraum entnommen und individuell entschieden, ob sie dem Orakel übergeben oder nicht weiter berücksichtigt wird. Dabei kommt die Frage auf nach welchen Kriterien entschieden wird, ob die Instanz dem Orakel übergeben wird oder nicht. Eine Möglichkeit besteht darin das Maß für den Informationsgehalt als Kriterium zu nutzen. Dabei werden dem Orakel mit höherer Wahrscheinlichkeit Instanzen übergeben, die auch einen höheren Informationsgehalt aufweisen. Ein weiterer Ansatz basiert auf der Berechnung einer Region der Unsicherheit. Dies bedeutet, dass anhand des Informationsgehalts ein unterer Grenzwert (Threshold) vorab gesetzt wird. Liegt der berechnete Wert für eine Instanz über diesem Grenzwert, so wird sie dem Orakel übergeben und gefragt. Das Stream-Based Szenario findet eine hohe Bedeutung, wenn der Speicher oder die Rechenleistung stark limitiert sind.

Es gibt sehr viele Problemstellungen in der Praxis für die eine große Anzahl an nicht-gelabelten Daten auf einmal zur Verfügung gestellt werden können. Das *Pool-Based Sampling* nimmt an, dass wenige gelabelte Instanzen zur Verfügung stehen, aber eine große Menge an nicht-gelabelten Instanzen existieren. Im Gegensatz zu Stream-Based Selective Sampling wird bei Pool-Based Sampling die komplette Menge an nicht-gelabelten Instanzen nach ihrem Informationsgehalt bewertet und in eine Reihenfolge gebracht. Die besten

Queries werden dann dem Orakel übergeben. Das Pool-Based Szenario wurde schon sehr genau bei der Textklassifikation, Bild- und Videoklassifikation, sowie der Krebsdiagnostik untersucht.

2.4.3 Query-Strategie - Uncertainty Sampling

Da das Ziel verfolgt wird möglichst informative Queries zu stellen, werden Query-Strategien benötigt. Mit diesen erfolgt die Bestimmung des Informationsgehaltes von nicht-gelabelten Instanzen.

Die wohl einfachste Methode ist das *Uncertainty Sampling*. Dabei wird für jede Instanz berechnet, wie unsicher sich der aktuelle Klassifikator über deren Klassenzugehörigkeit ist. Die Instanz mit der höchsten Unsicherheit wird dann für das weitere Trainieren des Klassifikators ausgewählt. Unter der Annahme, dass der Wahrscheinlichkeitswert für die Klassifizierung einer Instanz zwischen null (0) und eins (1) dargestellt wird, bedeutet dies bei der Verwendung von einem probabilistischen Modell und einer binären Klassifikation, dass die Instanz ausgewählt wird, dessen Wahrscheinlichkeit 0.5 entspricht oder in dessen Nähe liegt. Dieser Wert entspricht damit der maximalen Ungewissheit.

3 Implementierung des interaktiven Webcrawlers

3.1 Crawler-Architektur

In Abbildung 3.1 ist die grundlegende Architektur des entwickelten Systems veranschaulicht. Die Architektur kann grob in drei Komponenten unterteilt werden. Die erste Komponente entspricht der Funktionalität eines Webcrawlers, wie er zuvor in Abschnitt 2.2.1 beschriebenen wurde und ist in Abbildung 3.1 grün umrandet. Die Klassifizierung von URLs in die beiden Klassen *relevant* und *nicht relevant* erfolgt durch die in Abbildung 3.1 rot umrandete Komponente. Zuletzt beinhaltet die Architektur mit der dritten Komponente, in Abbildung 3.1 blau markiert, noch eine Schnittstelle für die Interaktion zwischen dem System und einem Nutzer, so dass das Prinzip von Aktivem Lernen realisiert werden kann. Das Zusammenspiel dieser drei Komponenten kann als Pipeline aufgefasst werden.

Für das bessere Verständnis folgt nun eine kurze Erklärung von dem Ablauf dieser Pipeline. Die Grundlage besteht aus einer Datenstruktur, welche die Seed-URLs organisiert und verwaltet. Daraus werden die Seed-URLs entnommen und dem Fetching- und Parsing-Modul übergeben. Die Aufgabe von diesem Modul besteht darin, die erhaltenen URLs zu Laden, sie auf bestimmte Inhalte zu durchsuchen, sowie die darin enthaltenen Hyperlinks zu extrahieren. Diese Umsetzung entspricht nun exakt einem Crawler. Im Anschluss an die Crawler-Komponente folgt nun die Klassifizierung der übergebenen URLs mit Hilfe eines Klassifikators. Dieser berechnet für jede übergebene URL die Wahrscheinlichkeit mit der diese URL einer der zuvor definierten Klasse zugeordnet werden kann. Abhängig vom zuvor definierten Threshold wird eine URL direkt klassifiziert oder dem Nutzer übergeben, der diese dann manuell klassifizieren muss.

Liegt zum Beispiel der Threshold zwischen 0.7 und 0.8 (wie in Abbildung 3.1 als Beispiel verwendet wird) und der Klassifikator hat eine URL u zu 0.76 der Klasse *relevant* zugeordnet, so liegt dieser Wert im Bereich des Threshold. Da dies eine Unsicherheit des Klassifikators bedeutet wird bei diesem Szenario der Nutzer mit einbezogen. Dieser muss nun selbst entscheiden, ob diese URL für ihn relevant ist oder nicht und teilt dies dem System über die Konsole mit. Wurden alle URLs durch den Klassifikator und gegebenenfalls mit Hilfe des Nutzers in die beiden Klassen eingeteilt, so werden alle relevanten URLs herausgefiltert und der gerade eben beschriebene Prozess erfolgt erneut mit diesen URLs. Dieser Zyklus wird solange durchlaufen bis ein Abbruchkriterium erreicht ist.

Da die Vorgehensweise beim Erforschen des World Wide Web einer Breitensuche entspricht, wird in jedem Zyklus eine Ebene der Breitensuche abgedeckt. Anmerkung: Die Seed-URLs und die extrahierten URLs werden in zwei unterschiedlichen Datenstrukturen gespeichert.

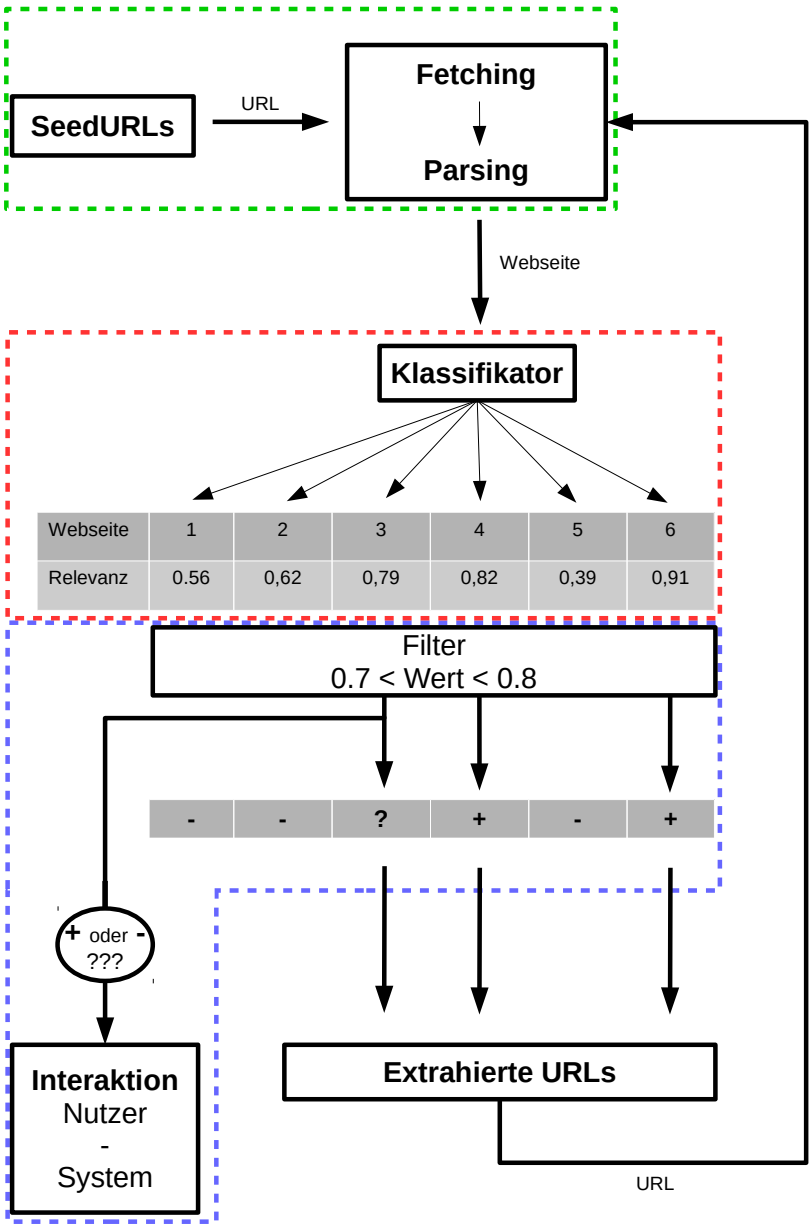

Abbildung 3.1: Die Architektur des entwickelten Systems.

3.2 Das entwickelte System

Für die Umsetzung des interaktiven Webcrawler wurde die Programmiersprache Java gewählt. Dabei wurde auf zwei Java-Bibliotheken zurückgegriffen, die sowohl die Umsetzung der Module *Fetching* und *Parsing*, als auch die Klassifizierung erleichtert haben. Zum einen wurde *Jsoup* für das Extrahieren der Informationen und Hyperlinks aus dem HTML-Code verwendet, zum anderen kam die Bibliothek *Mallet* zum Einsatz. Diese wurde benutzt um den Klassifizierungsprozess zu realisieren.

Für den Verbindungsaufbau zu einer URL wurde bedacht, dass derselbe Server nicht all zu oft hintereinander Anfragen erhalten soll, wie in Abschnitt 2 erklärt. Falls dies doch der Fall sein sollte wird eine bestimmte Zeitspanne gewartet bis die nächste Anfrage erfolgen kann.

Als Merkmale wurden alle Wörter einer Webseite betrachtet, die im HTML-Code unter dem Tag *body* stehen. Diese entsprechen allen sichtbaren Wörtern einer Webseite. Für die Repräsentation einer Webseite wird das Bag-of-Words Modell verwendet. Dabei wird ein Wörterbuch aus den Wörtern, die in den Trainingsdaten vorkommen, erstellt, wobei keine Duplikate berücksichtigt werden. Nun wird jedes Wort aus den einzelnen Webseiten mit jedem Eintrag aus dem Wörterbuch verglichen. Falls ein Wort aus dem Wörterbuch in einem Dokument enthalten ist, so erhält es den Wert *true* (1), ansonsten *false* (0). Ein Merkmalsvektor, der eine Webseite repräsentiert, ist also ein 2-dimensionaler Vektor, der jedem Wort einen Wert (1 oder 0) zuordnet.

Die genaue Umsetzung von Aktivem Lernen in dem entwickelten System wird im nächsten Abschnitt erklärt.

3.2.1 Klassen des Systems und deren Funktionalitäten

Im Folgenden werden die Klassen, welche für die Umsetzung des interaktiven Webcrawlers implementiert wurden, aufgeführt und deren Funktionalitäten genauer erläutert.

Website

Diese Klasse umfasst alle notwendigen Angaben einer Webseite und beinhaltet darüber hinaus Methoden, die der Verarbeitung von URLs dienen und für die Informationsgewinnung verantwortlich sind. Der HTML-Code, die URL und die Domain, sowie der textuelle Inhalt und das Klassenlabel einer Webseite werden gespeichert. Mithilfe von *getDomainName(URL)* erhält man die Domain einer übergebenen URL. Die beiden Methoden *seedURLsToWebsites(URLs)* und *urlsToWebsites(URLs)* transformieren URLs in Webseiten, wobei in *seedURLsToWebsites(URLs)* das Klassenlabel gesetzt wird, da es von den Seed-URLs schon zu Beginn bekannt ist. Die eigentliche Informationsgewinnung erfolgt durch die Methoden *getURLinformation(URL)* und *extractHyperLinks(URL)*. Beide Methoden erzeugen eine Verbindung zum Server der URL und extrahieren jeweils die gewünschten Informationen. *getURLinformation(URL)* extrahiert wie oben beschrieben Angaben der Webseite. Der textuelle Teil entspricht dem sichtbaren Teil der Webseite (*body-Tag*). Wohingegen *extractHyperLinks(URL)* die Extrahierung aller Hyperlinks (*href-Tag*) vornimmt, um diese im weiteren Verlauf zu klassifizieren. Diese Hyperlinks werden normalisiert, „da oftmals die HTML-Kodierung von einem Link einer Webseite p

das Ziel relativ zu p angibt“ [7]. Aus relativen URLs werden somit absolute URLs kreiert. Mittels *similarity(Webseite)* wird unter Verwendung des Jaccard-Index überprüft, inwieweit sich der Inhalt einer Webseite mit schon besuchten Webseiten ähnelt, ohne dass es dieselbe URL ist. Somit wird vermieden, dass eine sehr ähnliche Webseite weiterhin berücksichtigt und verarbeitet wird. Diese Methode entspricht einem Filter. Der Wert von diesem Index kann zwischen 0 und 1 liegen. Im Falle einer Überschreitung des Wertes 0.95 wird die Webseite aufgrund von zu hoher Ähnlichkeit verworfen.

Classification

Die eigentliche Klassifizierung, also das Annotieren einer Webseite, erfolgt in dieser Klasse. Dazu wird anhand der *buildPipe()*-Methode eine sogenannte *Pipe* kreiert. Diese dient der Modifizierung der ursprünglich, übergebenen Rohdaten und läuft wie folgt ab: Nach dem Anlegen der beiden Klassen *relevant* und *nicht relevant* werden die Stoppwörter eliminiert und alle Buchstaben klein geschrieben. Daraufhin werden die restlichen Wörter transformiert. Ein Wort, also eine *CharacterSequence* wird in eine *TokenSequence* umgewandelt, diese dann zu einer *FeatureSequence* überführt und im letzten Schritt daraus ein *FeatureVector* erstellt. Die oben beschriebene Pipe verarbeitet nur *Instance*-Elemente, das heißt es wird eine Methode für den Übergang von einer Webseite, die in der Klasse *Website* erstellt wird, zu einer *Instance* benötigt. Diese Transformation erfolgt in den beiden Methoden *generateInstanceList(Webseiten)* und *generateInstance(Webseite)*. Die daraus resultierenden *Instances* können nun verwendet werden um mit der *trainClassifier(Instances)*-Methode einen Klassifikator zu trainieren.

Die Java-Bibliothek *Mallet* bietet eine Vielzahl an Klassifikatoren an, beispielsweise den MaxEnt- oder den NaiveBayes-Klassifikator. Mit der *printLabelings(Classifier, Instance)*-Methode kann, abhängig vom übergebenen Klassifikator, eine *Instance* in eine der angelegten Klassen klassifiziert werden. Außerdem wird in dieser Methode der Parameter festgelegt, wann ein Nutzer zur Klassifizierung einer Webseite gefragt wird.
Das Attribut *RequestUserCounter* beschreibt die Anzahl von Webseiten, wie oft der Nutzer zur manuellen Klassifizierung einer Webseite gefragt wurde.

UserInteraction

Die *UserInteraction*-Klasse dient der Interaktion zwischen dem Nutzer und dem System. Die Methode *evaluateWebsite(Instance)* wird für die manuelle Klassifizierung durch den Nutzer verwendet. Besteht bei dem Klassifikator eine Unsicherheit bezüglich einer Webseite, so wird der Nutzer über die Relevanz der Webseite gefragt. Dazu wird dem Nutzer die Webseite über den Webbrowser mit der Methode *openBrowser(URL)* geöffnet. Der Nutzer kann daraufhin die Webseite anschauen und dem System über die Konsole die Bewertung mitteilen. Falls die besuchte Webseite für den Nutzer relevant ist, so gibt er den Buchstaben *Y* für Yes ein. Andernfalls besteht die Eingabe aus dem Buchstaben *N* für No. Bei einer fehlerhaften Eingabe, also einem Zeichen ungleich Y oder N, wird der Nutzer noch einmal aufgefordert die Webseite zu bewerten.

GetStaticClassifier

Die Aufgabe dieser Klasse besteht lediglich darin den aktuellen Stand des Klassifikators abzuspeichern.

Frontier

Die Klasse *Frontier* verwaltet alle URLs und Webseiten. Die Liste *SeedURLs* speichert die zuvor durch den Nutzer festgelegten Seed-URLs, um ausgehend von diesen den Crawl-Prozess zu starten. *relevantURLs* und *irrelevantURLs* beinhalten jeweils relevante beziehungsweise nicht relevante URLs bezüglich einer zuvor festgelegten Domäne. All diese Listen und deren Größe werden am Ende des Crawl-Prozesses dem Nutzer auf der Konsole angezeigt.

Crawler

Die einzige Methode *crawlWebsite(Webseiten)* in dieser Klasse umfasst den kompletten Ablauf des Crawlens. Dabei werden viele der oben vorgestellten Methoden aus den anderen Klassen verwendet. In 3.2 ist dies grob skizziert.
Ausgehend von den übergebenen Webseiten beginnt der Crawl-Prozess. Dieser wird solange wiederholt bis eine gewünschte Anzahl an relevanten URLs erreicht ist. Der Parameter für die Anzahl von diesem Abbruchkriterium lässt sich in der Klasse *Test* verändern. Hierbei werden die zuvor vorgestellten Klassen größtenteils benutzt. Das Attribut *counterIteration* speichert die Anzahl ab, wie oft der Nutzer während dem Programmablauf bezüglich einer Webseite gefragt wurde. Außerdem werden in jedem Iterationsschritt die Anzahl der relevanten und irrelevanten Webseiten ausgegeben.

Test

Zu Beginn werden in dieser Klasse die Seed-URLs manuell angelegt. Diese werden daraufhin in die Frontier geschrieben. Durch die *seedURLsToWebsites(Seed-URLs)*-Methode werden die Seed-URLs in die entsprechenden Webseiten transformiert, wie in der Klasse *Website* zuvor erklärt wurde und ebenfalls in der Frontier abgespeichert. Diese Webseiten werden anschließend der Methode *generateInstanceList(Webseiten)* übergeben, um daraus die *Instances* zu kreieren, die für das Trainieren des Klassifikators notwendig sind. In dieser Methode werden ebenfalls die die Seeds gelabelt, da diese zu Beginn schon bekannt sind. Je nach Klassenlabel werden diese Webseiten in die entsprechende Liste in der Frontier gespeichert. Von den relevanten Webseiten werden anschließend die Hyperlinks über die Methode *extractHyperLinks(URL)* extrahiert. Mittels *crawlWebsite(Webseiten)* wird der Crawl-Prozess so lange wiederholt bis die zuvor eingegebene Anzahl an gewünschten, relevanten URLs erreicht ist. Dies ist also das Abbruchkriterium für den das Crawling. Das Abbruchkriterium kann zu jeder Zeit geändert werden.

3.2.2 Parameter

Mit Hilfe von zwei Parametern ist es möglich das Verhalten dieses Systems in gewisser Weise zu steuern. Der erste Parameter entspricht der Auswahl eines Klassifikators. *Mallet* bietet diverse Klassifikatoren an (MaxEnt, NaiveBayes oder DecisionTree), auf die zurückgegriffen werden kann. Die Konfiguration von diesem Parameter erfolgt in der Methode *trainClassifier()*, die sich in der *Classification*-Klasse befindet. Die *printLabeling()*-Methode, die sich ebenfalls in der Klasse *Classification* befindet, beinhaltet den zweiten, fast wichtigeren Parameter dieses Systems, der *Threshold*. Dieser besteht aus

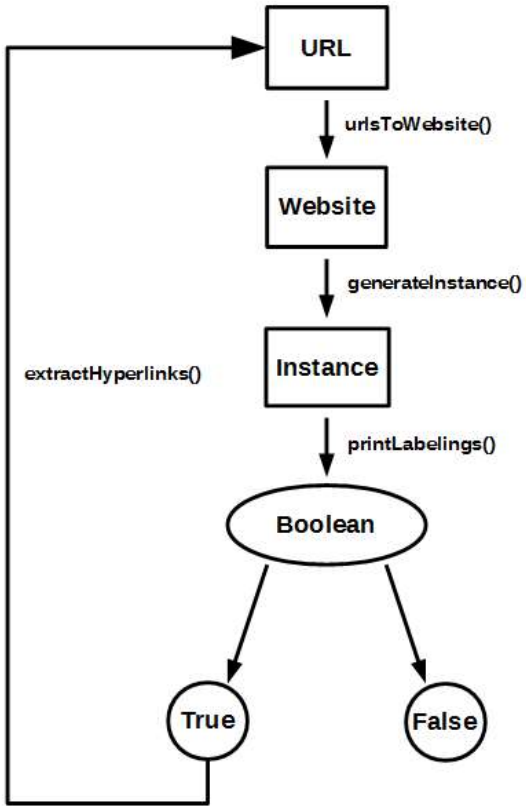

Abbildung 3.2: Vereinfachtes Ablaufmodel der Methodenaufrufe, um von einer URL zu deren Klassifizierung zu gelangen.

einem minimalen und einem maximalen Wert. Beide Werte liegen zwischen 0 und 1. Außerdem muss gelten, dass der minimale Wert kleiner als der maximale Wert ist. Der Bereich zwischen diesen Werten gibt die Unsicherheit des Klassifikators an. Bezüglich einer URL und einer Klassen gibt der Klassifikator einen Wert zwischen 0 und 1 zurück, welcher der Wahrscheinlichkeit entspricht mit der die URL zu dieser Klasse gehört. Liegt jedoch dieser Wert im Bereich der Unsicherheit, so bedeutet dies, dass der Nutzer nun selbst über die Relevanz der URL entscheidet und teilt dies dem System über die Konsole mit. Je größer dieser Threshold-Bereich ist, desto häufiger wird der Nutzer gefragt. Angenommen der minimale Wert liegt bei 0 und der maximale Wert liegt bei 1. Dann bedeutet dies, dass der Nutzer bei jeder URL selbst entscheiden muss, ob sie für ihn relevant ist oder nicht und damit der Klassifikator bei jeder Webseite unsicher ist.

4 Experimente

Um das implementierte System zu evaluieren wurden Experimente durchgeführt. Das dabei verfolgte Ziel war abzuschätzen, inwieweit sich der Informationsgehalt, welchen der entwickelte Webcrawler aus dem World Wide Web herauszieht, mit den Vorstellungen eines Benutzers deckt.

Der Aufbau der durchgeführten Experimente wird im nächsten Abschnitt erklärt. Im darauffolgenden Abschnitt werden die daraus gewonnenen Ergebnisse dargestellt und zum Abschluss werden die möglichen Gründe für die zuvor gewonnen Resultate diskutiert.

4.1 Aufbau der Experimente

Als Grundlage für alle durchgeführten Experimente wurden zwei Klassen manuell erstellt. Die relevante Klasse umfasst URLs von dem Themengebiet ***Haartransplantation***. Die Klasse der nicht relevanten URLs umfasst Webseiten über das Thema ***Regenwald***. Die entsprechenden Seed-URLs der beiden Klassen wurden mit Hilfe der Suchmaschine *google* ermittelt. Dabei wurden die ersten 8 Suchergebnisse der beiden Suchanfragen als Seed-URLs verwendet. Insgesamt ergeben sich 16 Trainingsdaten (8 relevante, 8 nicht relevante). Anhand dieser Trainingsdaten erfolgt das Trainieren eines ausgewählten Klassifikators. Für diese Experimente kam der in 2.2.1 erklärte MaxEnt-Klassifikator zum Einsatz. Um zu überprüfen, inwieweit der Webcrawler den Informationsbedarf eines Benutzers deckt, wurde als Basis eine Datenmenge an Webseiten durch einen Benutzer festgelegt. Ihm wurden dabei 100 Webseiten nacheinander angezeigt, die er daraufhin zu klassifizieren hatte. Somit ergab sich eine Menge an relevanten und nicht relevanten Webseiten. Für die Umsetzung wurde der Threshold-Parameter, also das Gebiet der Unsicherheit, so konfiguriert, dass bei jeder Webseite der Nutzer gefragt wird und er selbst die Klassifizierung dieser Webseite vornimmt. Für dieses Szenario wurde der minimale Wert des Threshold-Parameters auf 0.0 und der maximale Wert auf 1.0 gesetzt. Das Ziel ist nun den Threshold-Parameter so einzustellen, dass die darauf resultierende Ergebnismenge von Webseiten bestmöglich der durch den Benutzer zuvor annotierten Menge an Webseiten entspricht. Dazu wurde der maximale und minimale Wert von dem Threshold-Parameter 11-Mal verändert (zwischen 0.70 und 0.98) und für jedes Experiment ein Vergleich zur manuell annotierten Menge hergestellt, indem die Precision (Genauigkeit), der Recall (Vollständigkeit) und die Anzahl der Anfragen an den Nutzer notiert wurde.

Die Precision gibt an, wie groß der Anteil an relevanten Webseiten zur Gesamtmenge ist. Der Recall hingegen gibt den Anteil der gefundenen, relevanten Webseiten im Verhältnis zur Gesamtmenge der relevanten Webseiten an, die gefunden wurden.

4.2 Ergebnisse

Die Ergebnisse der durchgeführten Experimente werden in 4.1 dargestellt. Außerdem sind diese Werte in den Abbildungen 4.2, 4.3 und 4.4 aufgezeichnet.

Der Tabelle kann entnommen werden, dass bei der Einstellung eines niedrigen Threshold-Parameter der Recall sehr hoch ist. Jedoch sind die Werte der Precision, mit Ausnahme der Einstellung 0.78 - 0.88, etwas niedriger. Tendenziell kann gesagt werden, dass bei steigendem Threshold-Wert der Recall abnimmt, aber gleichzeitig die Precision ansteigt. Der Verlauf dieser beiden Werte ist also gegensätzlich. Betrachtet man zusätzlich noch die Anzahl, wie häufig der Benutzer gefragt wurde, so lässt sich erkennen, dass der Nutzer einen großen Einfluss auf die Precision hat. Einzige Ausnahme ist das Experiment mit der Einstellung 0.90 - 0.95. Dabei wurde trotz seltenem Fragen eine hohe Precision erzielt. Das beste Ergebnis, mit einem Recall von 0.91 und einer Precision von 0.98, erzielte das Experiment mit der Einstellung 0.80 - 0.90. Dies entspricht nahezu der Klassifizierung des Benutzers.

Setzt man die Anzahl der Anfragen an den Nutzer in Relation zu der Anzahl der klassifizierten Webseiten, so lässt sich erkennen, dass bei den meisten Experimenten im Durchschnitt jede 6. Webseite durch den Benutzer klassifiziert wurde. Die daraus resultierenden Ergebnisse zeigen eine überwiegende Überdeckung der erzielten Datenmenge anhand der Experimente mit den Vorstellungen des Benutzers.

Threshold	Recall	Precision	# Anfragen an Nutzer
0.70 - 0.75	0.98	0.69	6
0.75 - 0.80	0.93	0.73	4
0.78 – 0.83	0.93	0.75	6
0.78 – 0.88	0.93	0.98	21
0.80 - 0.85	0.89	0.71	3
0.80 - 0.90	0.91	0.98	17
0.83 – 0.88	0.90	0.97	16
0.83 – 0.93	0.84	1.00	19
0.85 - 0.90	0.82	0.97	16
0.90 - 0.95	0.79	0.97	4
0.95 - 0.98	0.72	1.00	16

Abbildung 4.1: Die Werte für Precision und Recall bei entsprechender Wahl des minimalen und maximalen Wertes von dem Threshold-Parameter. Die letzte Spalte entspricht der Anzahl, wie oft der Nutzer bei dem jeweiligen Experiment gefragt wurde, um eine Webseite zu klassifizieren.

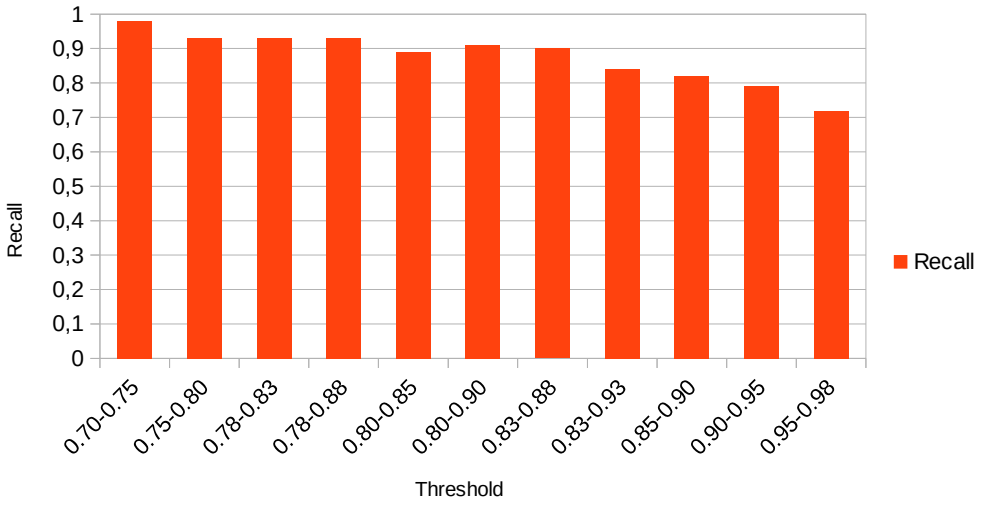

Abbildung 4.2: Diagramm: Recall von jedem Experiment bezüglich der jeweiligen Einstellung von dem Threshold-Parameter.

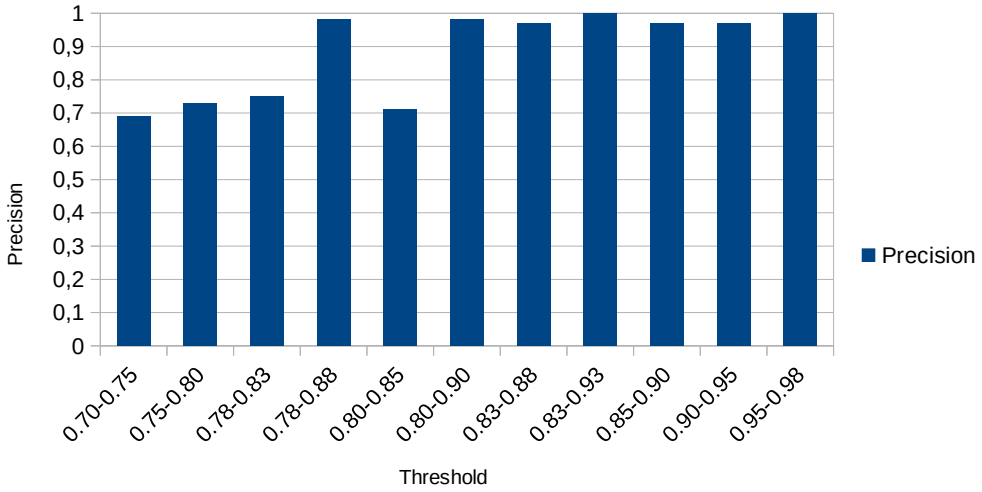

Abbildung 4.3: Diagramm: Precision von jedem Experiment bezüglich der jeweiligen Einstellung von dem Threshold-Parameter.

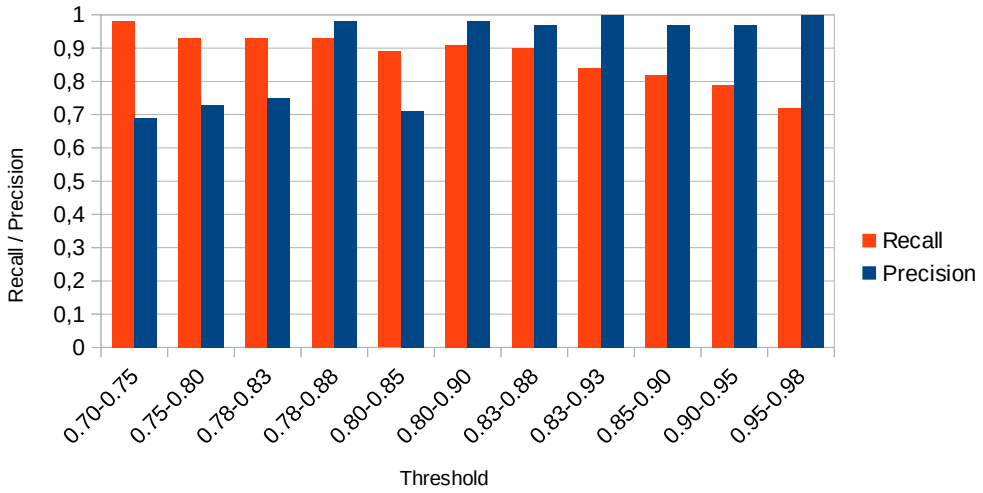

Abbildung 4.4: Diagramm: Recall und Precision von jedem Experiment bezüglich der jeweiligen Einstellung von dem Threshold-Parameter.

4.3 Diskussion

Die gewonnenen Ergebnisse der durchgeführten Experimente lassen erkennen, dass die Kombination aus Klassifizierung und Aktivem Lernen durchaus eine Möglichkeit darstellt, um die gewünschten Informationen eines Benutzers zu erhalten. Die möglichen Gründe für die erzielten Ergebnisse werden im Folgenden diskutiert.

Generell muss in Betracht gezogen werden, dass die Empfindung über die Relevanz einer Webseite von Benutzer zu Benutzer variieren. Um zu schauen, inwieweit die Empfindungen auseinandergehen, wurden zwei Benutzer ausgewählt, welche die Basismenge für die weiterführenden Experimente manuell klassifizierten. Die Folge waren zwei unterschiedliche Datenmengen und somit wäre das Training des Klassifikators unterschiedlich abgelaufen. Zum Beispiel war für Benutzer 1 die Homepage einer Klinik völlig irrelevant, da es auf dieser Webseite Links zu dem Thema Haartransplantation gab, jedoch die Seite selbst keine Information über dieses Thema brachte. Benutzer 2 fand diese Webseite relevant, da er über einen weiteren Link zum Thema Haartransplantation gelangen konnte. Dies zeigt, dass die Menschen auf unterschiedliche Merkmale einer Webseite schauen.

Ein wichtiger Punkt, der zu diesen Ergebnissen führt, ist die Wahrscheinlichkeitsverteilung der besuchten und zu klassifizierten Webseiten durch den Klassifikator. Im Bereich 0.78 - 0.98 lagen sehr häufig die Wahrscheinlichkeiten, dass eine Webseite als relevant angesehen wird. Dementsprechend wurde auch in diesem Gebiet sehr häufig der Benutzer für die Klassifizierung gefragt. Dies hat wiederum eine positive Auswirkung auf das Ergebnis, da der Nutzer „immer richtig klassifiziert“.

Da die Anzahl der zu klassifizierenden Webseiten in den Experimenten auf 100 und die Anzahl der Seed-URLs auf 8 gesetzt wurde, ist man nicht zu sehr in die Tiefe beim Webcrawlen gekommen (2 Iterationen). Dies hat zur Folge, dass man häufig in der selben Domäne geblieben ist und nicht all zu viele weitere Webseiten aufgespürt hat. Wird das Experiment mit den Einstellungen 0.95 - 0.98 betrachtet, zu konnten bei diesem Experiment keine 100 Webseiten klassifiziert werden, da durch das Abbruchkriterium, das Webcrawling zu früh terminierte.

5 Ausblick

Durch den interaktiven Webcrawler wurde zu einem bestimmten Thema der Informationsgehalt extrahiert. Diese Informationen lassen sich größtenteils mit den Vorstellungen eines Benutzers decken, das anhand der Ergebnisse der durchgeführten Experimente gezeigt wurde. Das entwickelte System kann durch Erweiterungen und Ergänzungen verbessert, optimiert und benutzerfreundlicher gestaltet werden. Des weiteren können durchaus weitere Evaluierungsformen experimentell getestet werden. Im Folgenden werden einige Überlegungen erwähnt.

Für die Optimierung des Systems kann in Betracht gezogen werden, dass nicht nur ein Crawler das Web erforscht, sondern mehrere Webcrawler erstellt werden (über Threads) und sich somit die Performanz des Systems verbessern lässt. Dazu wäre noch ein Crawler-Manager notwendig, der die erzeugten Webcrawler verwaltet.

Für eine bessere Übersicht würde sich die Entwicklung einer grafischen Oberfläche (GUI) anbieten. Dabei kann das System um weitere Funktionalitäten erweitert werden. So wäre es sinnvoll ein Textfeld auf der GUI zu erzeugen, damit der Nutzer das für ihn relevante Thema eingibt. Ausgehend von seiner Eingabe werden dann die ersten Treffer einer Suchmaschine automatisch als Seed-URLs gespeichert und der Crawl-Vorgang von diesen URLs gestartet. Somit müssten die Seed-URLs nicht manuell eingeben werden.

Außerdem wäre es sinnvoll, die Anzahl der Seed-URLs zu erhöhen, um somit die Terminierung des Crawl-Prozesses nicht zu früh eintreten zu lassen. Damit besteht die Möglichkeit weiter in die Tiefe zu crawlen. Um den Terminierungsprozess ebenfalls nicht all zu früh eintreten zu lassen, ist die Überlegung ein anderes Abbruchkriterium zu wählen. So kann beispielsweise so lange das Web erforscht werden bis eine bestimmte Anzahl an relevanten Webseiten erreicht wird.

Des weiteren wäre zu überlegen, noch weitere Query-Strategien und Selektionsstrategien in das System einzubauen und diese dann in Kombination mit verschiedenen Klassifikatoren zu evaluieren. Möglicherweise werden dabei bessere Ergebnisse erzielt.

Am meisten Gedanken kann man sich über die Auswahl der Features machen. Welche Features charakterisieren eine Webseite am genauesten? Ist es der Text? Sind es die Überschriften oder gar die Kombination mit Bildern?

Um das System noch genauer zu evaluieren, besteht die Möglichkeit den Vergleich zu einer konventionellen Suchmaschine wie google herzustellen. Dabei wird dann verglichen, inwieweit beispielsweise die ersten 100 Suchergebnisse dieser Suchmaschinenanfrage mit der Ergebnismenge des entwickelten Webcrawlers übereinstimmen.

Literaturverzeichnis

[1] H. S. Christopher D. Manning, Prabhakar Raghavan. *An Introduction to Information Retrieval*. Cambridge University Press, 2009.

[2] K. L. et al. Itwissen. Website. http://www.itwissen.info/definition/lexikon/Textklassifizierung-text-classification.html; abgerufen am 8. Oktober 2016.

[3] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[4] B. Pang, L. Lee, and S. Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[5] D. S. Rainer Kuhlen, Wolfgang Semar. *Grundlagen der praktischen Information und Dokumentation: Handbuch zur Einführung in die Informationswissenschaft und -praxis*, volume 6, illustriert. Walter de Gruyter, 2013.

[6] S. Raschka. Naive bayes and text classification. Website. http://sebastianraschka.com/images/blog/2014/.

[7] T. Reitmaier. *Aktives Lernen für Klassifikationsprobleme unter der Nutzung von Strukturinformationen*, volume 5. kassel university press GmbH, 2015.

[8] H. J. . A.-L. B. Réka Albert. Internet: Diameter of the world-wide web. Website, 1999. http://www.nature.com/nature/journal/v401/n6749/full/401130a0.html.

[9] B. Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[10] Sickert. Strategien des aktiven lernens. Website. http://www.inf-cv.uni-jena.de/dbvmedia/de/Sickert.

Die Webseiten wurden zuletzt am 20.10.2016 besucht.

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, den 31.10.2016

Unterschrift