Recommender System für das Open Journal System

Masterarbeit

von

Martin Grossegger

Betreuer:
Univ.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner
Dipl.-Ing. BSc Behnam Taraghi

Institut für Informationssysteme und Computer Medien
TU Graz

2011

Kurzfassung

Empfehlungssysteme treten im Internet immer häufiger auf. Die Einsatzmöglichkeiten sind vielfältig, oftmals sind sie ein wichtiger Bestandteil der Vermarktung von Produkten. Aber auch in nicht kommerziellen Aufgabenbereichen werden Empfehlungssysteme eingesetzt, beispielsweise um ein breiteres Angebot an nützlichen Informationen bieten zu können.
Im Bereich der Bereitstellung von Informationen bietet das Open Journal System (OJS) die Möglichkeit eine Zeitschrift online zu administrieren und zu veröffentlichen. Für dieses Journalsystem, welches als Open-Source-Software verfügbar ist, wird ein Plugin entwickelt, welches Empfehlungen erstellen kann. Diese Empfehlungen sollen dazu dienen, allen Benutzerinnen und Benutzern relevante Artikel in Abhängigkeit vom zuletzt gelesenen vorzuschlagen, wobei die Grenze einer Ausgabe überschritten werden soll, wenn eine inhaltliche Relevanz besteht. Um dies zu erreichen wird ein Plugin erstellt, welches Ansätze des Collaborative-Filterings mit Ansätzen des Content-Based-Filterings verbindet. Für das Collaborative-Filtering werden die Downloads der BenutzerInnen als Pfade betrachtet und gewichtet. Diese Gewichtung dient dazu, eine vorgegebene Navigationsstruktur abzuschwächen und somit Empfehlungen zu erstellen, die auf den Interessen der BenutzerInnen beruhen und gleichzeitig weniger von ihrer Position in der Navigationsstruktur beeinflusst werden.
Der Autor beschreibt in dieser Arbeit die Vorgehensweise für die Entwicklung dieses Plugins, beginnend mit der Analyse bestehender Daten einer OJS-Installation, über die Planung und Implementierung des Plugins, bis hin zu einer vergleichenden Auswertung des gewählten Algorithmus. Dieser Vergleich zeigt, dass der gewählte Ansatz in der Lage ist, relevante Empfehlungen zu erstellen.
Als Grundlage für die Analyse und Implementation des Plugins diente die OJS-Installation L3T, welche als e-learning-Journal von der TU Graz betrieben wird.

Abstract

Nowadays Recommender Systems (RS) are well known by all kind of internet users. Manifold fields of application for using a RS already exist, the best known purpose may be the recommendation process of products for customers. The usage of RS for non-commercial purposes, such as finding useful information for users dependent on their previous actions, is constantly growing.
Open Journal Systems (OJS) offers the possibility to create and publish a journal online. The purpose of this thesis is the development of a RS for OJS which is available as open source software. This RS should be implemented as a plugin and should help the users discover articles related to their interest.
Besides, this RS should help to discover articles beyound the borders of journal issues. To achieve this goal, the RS will combine collaborative filtering with content based filtering methods. The collaborative filtering part bases on user generated paths (a sequence of downloads) wich are weighted. This weight is used to generate recommendations which differ from an existing link structure, so the recommendations will depend more on the taste of the users than on the position of the element in the link structure.
In this thesis the complete process of developing a plugin which extends OJS by providing a RS is described. It starts with the analysis of existing data, followed by the design and implementation of the plugin itself. Finally an examination of the implemented algortihm is made and discussed.
The data for analysis and the environment for developing the plugin was provided by the L3T project, an e-learning journal at the Graz University of Technology.

Danksagung

Als erstes möchte ich Martin Ebner und Behnam Taraghi für die ausgezeichnete Betreuung während der Erstellung dieser Arbeit danken. Die regelmäßigen Treffen sorgten dafür, dass das angepeilte Ziel gut geplant und umgesetzt werden konnte. Auch wurde mir die Installation der fertigen Software auf drei Systemen ermöglicht, ein Dankeschön an Rene Derler für die aufgewendete Zeit bei der Einrichtung der Software und bei der Fehlersuche. Ein großes Dankeschön geht an meine Freundin Sara Ovis für ihre Unterstützung und Motivation. Weiters danke ich meinen Eltern, die mir das ausführliche Studium ermöglicht haben, speziell meinem Vater für die unermüdliche Lese- und Korrekturarbeit.

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …… ……..

(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…… ……..

date (signature)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Empfehlungssysteme sind Informationssysteme, die aus einer Fülle von Daten den BenutzerInnen für sie/ihn interessante Elemente vorschlagen. Ein solches Empfehlungssystem soll die BenutzerInnen bei den Interaktionen mit dem System dahingehend unterstützen, dass bewusst oder unbewusst gewünschte Ressourcen angezeigt werden. Die Bandbreite reicht dabei von Filmen, Büchern und Musik bis hin zu Reisezielen oder auch Nachrichtentexten. Es existieren vielfältige mathematische Modelle für die Umsetzung von Empfehlungssystemen, wobei die Wahl des Modells von den vorhandenen Daten (Daten über die Elemente, die empfohlen werden sollen und Daten über die BenutzerInnen des Systems) und von der Art der Anwendung (kommerziell oder nicht kommerziell) abhängt. Der Online-Handel ist eine treibende Kraft für die Entwicklung von Empfehlungssystemen, aber auch bei nicht kommerziellen Empfehlungssystemen finden sie eine breite Verwendung.

1.2 Problemstellung

Im Bereich der nicht kommerziellen Anwendungen sind Empfehlungssysteme noch nicht so stark im Einsatz, wie im kommerziellen Bereich. Der Grund dafür mag darin liegen, dass die Entwicklung eines Empfehlungssystems aufwendig ist und dass bei nicht kommerziellen Anwendungen die Ressourcen für die Entwicklung eines Empfehlungssystems fehlen. Ein Bereich, für den ein Empfehlungssystem eine wertvolle Ergänzung darstellt, ist der Anwendungsbereich der online publizierten Zeitschriften. Ein System für die Veröffentlichung einer Zeitschrift ist das „Open Journal Systems“ (OJS). Für dieses System gibt es die Möglichkeit, eine Erweiterung (ein sogenanntes „Plugin“) zu entwickeln, welche dann auf allen Installationen dieses Systems verwendet werden kann. Diese Arbeit beschreibt die Entwicklung eines Empfehlungssystems für OJS in Form eines Plugins.

1.3 Zielsetzung

Das Ziel der Arbeit ist die Entwicklung eines Empfehlungssystems in Form eines Plugins für OJS. Dafür müssen geeignete Modelle ausgewählt, implementiert und validiert werden. Diese Auswahl wird einerseits durch die vorhandenen Daten über die Inhalte und über die BenutzerInnen, andererseits auch durch die vorgegebenen Anforderungen eines Plugins beeinflusst. Durch die Umsetzung als Plugin sind gewisse Grenzen in Bezug auf Programmiersprache, Datenspeicherung und die Verwendung von externen Zusatzprogrammen gegeben. Letztendlich sollen für die BenutzerInnen interessante Empfehlungen auf Grund der bestehenden Elemente, sowie auf Grund des Leseverhaltens der Benutzerinnen und Benutzer erstellt und vorgeschlagen werden. Insbesondere sollen Artikel über die Grenzen von einzelnen Ausgaben einer Zeitschrift hinweg vorgeschlagen werden können.

2 Recommender Systems – Empfehlungssysteme

2.1 Überblick

Recommender Systems, im folgenden ***Empfehlungssysteme*** genannt, sind ein integraler Bestandteil heutiger kommerzieller Webseiten, wie etwa Amazon[1] oder Yahoo[2]. Empfehlungssysteme finden auch bei nicht kommerziellen Aufgaben immer weitere Verbreitung. Sie sollen Benutzerinnen und Benutzern helfen, für sie interessante Elemente ausfindig zu machen, und sie damit zu unterstützen. Die Anzahl an Dokumenten und Mediendaten, die online verfügbar sind, steigt ständig und somit sind neue Strategien zu entwickeln, um relevante Daten zu finden.
Empfehlungssysteme personalisieren eine Suchanfrage im Gegensatz zu Suchmaschinen, die sich mehr auf die Übereinstimmung mit dem Suchbegriff konzentrieren [11].
Personalisieren bedeutet, dass die Dienste und Leistungen an die/den BenutzerIn, besser gesagt, an Bedürfnisse der Benutzerinnen und Benutzer angepasst werden. Dieser Vorgang kann durch aktive Mithilfe (zum Beispiel durch die Angabe von persönlichen Daten, oder einer aktiven Einschränkung des Suchraumes) oder indirekt durch die Handlungen der Benutzerin oder des Benutzers und der Auswertung der gespeicherten Benutzerprofile erfolgen [14] S. 3.
Dies führt dazu, dass Empfehlungssysteme zuerst „background data“ (Daten, die das System bereits kennt) benötigen, bevor der Prozess der Empfehlungserstellung begonnen werden kann. Erweitert wird der Prozess mit „input data“ - Daten die aus der Kommunikation mit dem System entstehen, wie etwa Bewertungen oder signifikante Aktionen. Diese Datenquellen werden durch einen Algorithmus ermittelt, der eine versucht eine brauchbare Empfehlung zu erstellen [5].
Abbildung 1 gibt einen kurzen Überblick über die Einsatzmöglichkeiten von Empfehlungssystemen [22].

Einsatzgebiet	Experten- und Arbeitsplatzsuche	Partnersuche	Produkte und Dienstleistungen	Web-Materialien	Prozess-empfehlungen
Absicht	Arbeitsstellen bzw. Experten mit größtmöglicher Passung zu finden	Finden von Lebenspartnern	Unterstützung bei der Entscheidung für Produkte und Dienstleistungen	Empfehlungen für kostenlos zugängliche Materialien	diverse (u.a. Arzneiverschreibung, Unterstützung Lernender, Fernsehproduktion)
Verfahren	Suchanfragen und Ergebnisse aufgrund eigenschaftsbasierter Verfahren; aber auch Inhalts- und Netzwerkanalysen	Eingangsbefragung, Partnervorschläge aufgrund paarpsychologischer Erkenntnisse, eigenschaftsbasierte, teils fallbasierte Verfahren	Alle bekannten Empfehlungsverfahren sind im Einsatz	oft tagbasierte Empfehlungen und kollaboratives Filtern	Inhaltsbasierte und kollaborative Ansätze
Visualisierung	Ranglisten (v.a. Suchergebnisse)	Ranglisten mit Fotos, teils inkl. „Matching-Punkte“, auch Benachrichtung per E-Mail	u.a. Trefferlisten für Empfehlungen am Seitenrand, Ranglisten, E-Mail-Benachrichtigung	Markierung im Text, Ranglisten	Trefferlisten, Vorschläge, auch Signalton (Alarm!)

Abbildung 1: Einsatzgebiete von Empfehlungssystemen. ([22] Auszug aus Tabelle 10, S.36)

1 http://www.amazon.at (Abruf am 28.11.2011)
2 http://www.yahoo.com (Abruf am 28.11.2011)

2.2 Formale Definition [1]

Formal kann das Problem für die Erstellung von Empfehlungen folgendermaßen formuliert werden:

C sei die Menge der BenutzerInnen des Systems und *S* sei die Menge aller möglichen Daten, die empfohlen werden können, beispielsweise Bücher, Restaurants, Filme oder Links. Die Größenordnung des Raumes der Menge *S* kann beliebig groß sein, denn ein System hat die Möglichkeit nahezu unbeschränkt Daten zu speichern. Auf der anderen Seite kann der Benutzerraum ebenfalls eine beliebige Größenordnung erreichen.

Sei *u* eine Funktion („utility function"), welche die Nützlichkeit eines Elementes *s* für eine/einen BenutzerIn *c* berechnet: $C \times S \rightarrow R$ wobei R eine geordnete Menge darstellt (zum Beispiel keine negativen ganzen Zahlen oder reele Zahlen in einem bestimmten Bereich). Dann suchen wir für jede/jeden BenutzerIn $c \in C$ ein neues Element $s' \in S$ welches die Nützlichkeit für sie/ihn maximiert:

$$\forall c \in C, s'_c = \arg\max_{s \in S} u(c, s)$$

$$\begin{aligned} &s'_c : empfohlenes\ Element \\ &c : BenutzerIn \\ &s : Element \\ &u : utility\ function\,(Nützlichkeit\ von\ Elemnt\ s\ für\ BenutzerIn\ c) \end{aligned} \qquad (1)$$

Die Nützlichkeit eines Elements kann in Empfehlungssystemen oft dadurch ausgedrückt werden, wie jemand das Element bewertet hat („rating").
Die Definition der Elemente des Benutzerraumes *C* kann als Benutzerprofil festgelegt werden, welches charakteristische Daten in Bezug auf die/den BenutzerIn enthält wie Alter, Geschlecht oder Einkommen.
Die Elemente des Datenraumes *S* können auch durch eine Menge von charakteristischen Merkmalen definiert werden, bei Filmen wären das etwa die Länge, Titel, Genre, Schauspieler etc.

Das zentrale Problem von Empfehlungssystemen besteht darin, dass die Nützlichkeitsfunktion *u* nicht im gesamten $C \times S$ Raum definiert ist, sondern nur für eine Untermenge dieses Raumes.
Das bedeutet, dass *u* auf den gesamten Raum $C \times S$ extrapoliert werden muss.
In einem Empfehlungssystem für Filme beispielsweise bewerten Benutzerinnen und Benutzer die Filme, die sie bereits gesehen haben. Das vorher beschriebene Problem zeigt sich nun darin, dass neue Filme noch keine Bewertung bekommen haben. Für diese Filme soll das Empfehlungssystem nun vorhersagen, wer den Film interessant findet. Um so eine Vorhersage zu treffen, gibt es eine Vielzahl an Möglichkeiten und daraus ergeben sich mehrere Ansätze, wie ein Empfehlungssystem aufgebaut werden kann.

2.3 Unterteilung von Empfehlungssystemen

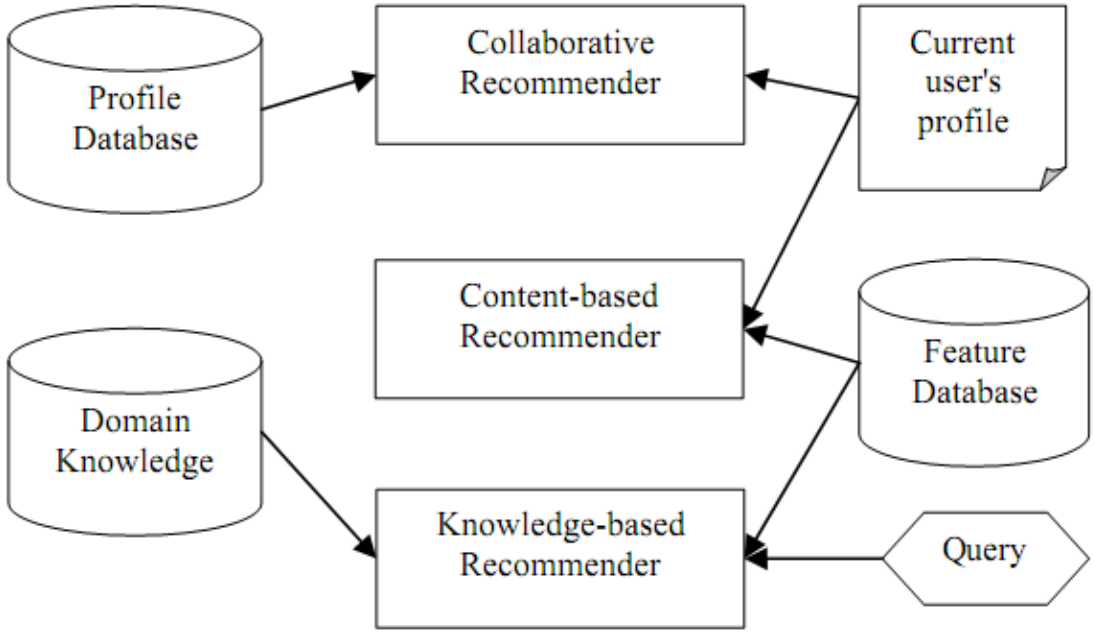


Abbildung 2: Einteilung von Empfehlungssystemen [5]

Empfehlungssysteme werden durch die Art der Empfehlungserstellung unterschieden:

- **„content based recommendations“:** Empfehlungen werden auf Grund der Ähnlichkeit zu Daten, die die/der BenutzerIn in der Vergangenheit als interessant empfunden hat, getroffen.

- **„collaborative recommendations“:** Empfehlungen werden auf Grund der Ähnlichkeit der Interessen der Benutzerin oder des Benutzers im Vergleich zu anderen Benutzerinnen oder Benutzern getroffen.

- **„knowledge based recommendations“:** Empfehlungen werden mit Hilfe einer Wissensbasis gebildet. Es muss zusätzlich eine explizit formulierte Anfrage („query“) an das System gesendet werden.

- **„hybrid approaches“:** Eine Kombination von Content-Based- und Collaborative-Systemen zur Erstellung von Empfehlungen.

Es gibt keinen allgemeingültigen Ansatz für die Erstellung eines Empfehlungssystems. Je nach Kontext und Anwendungsbereich muss untersucht werden, welches System, oder besser gesagt, welche Kombination von Systemen und Algorithmen den größten Erfolg („Nutzen“) für alle, die das Systems verwenden, bringt. Die meisten aktuellen Empfehlungssysteme verwenden Kombinationen verschiedener Ansätze, da damit die Schwächen der einzelnen Verfahren vermindert werden können.

2.3.1 Content-Based-Filtering

Empfehlungssysteme, die Content-Based-Filtering verwenden, basieren auf den Ähnlichkeiten zwischen den Vorlieben der BenutzerInnen und auf den Ähnlichkeiten der Ressourcen, die diesen Benutzerinnen und Benutzern zugeordnet sind. Solche Systeme empfehlen neue Inhalte auf Grund der Inhalte, die von ihnen in der Vergangenheit als interessant befunden wurden.
Dieser Ansatz des Content-Based-Filterings hat den Nachteil, dass Ressourcen, die als multimediale Information vorliegen, nicht automatisch zugeordnet werden können. Es gibt derzeit nur unzureichende Methoden um aus multimedialen Daten Merkmale zu extrahieren, welche für Vergleiche verwendet werden können. Zu solchen Multimediadaten gehören beispielsweise Musik-, Bild-, und Videodaten.
Zusätzlich kann so ein System nur Inhalte empfehlen, die eine gewisse Korrelation mit dem Benutzerprofil haben. Die/der BenutzerIn ist also auf Empfehlungen beschränkt, die eine hohe Ähnlichkeit mit den bereits bewerteten Inhalten haben [23].

Die Eigenschaften („features“) eines Elementes werden am häufigsten in Form von Vektoren abgespeichert. Dadurch wird der Einsatz von mathematischen Vergleichsmethoden ermöglicht (zum Beispiel kann die Ähnlichkeit zwischen zwei Feature-Vektoren mit Hilfe des Cosinus-Ähnlichkeitsmaßes bestimmt werden – vgl. Kapitel 2.4.1).
Im Bereich der Textanalyse bilden alle im Korpus vorkommenden Wörter die Dimensionen des Vektorraumes. Der Text wird also durch einen Vektor repräsentiert, der so viele Komponenten hat, wie der Korpus Wörter besitzt. Bei kurzen Texten bleiben natürlich viele Komponenten leer, da ja nicht alle verfügbaren Wörter verwendet werden. Dennoch sind Vektoren eine sehr effiziente Form der Eigenschaftsbeschreibung [14 S.57].

2.3.1.1 Vorteile von Content-Based-Filtering-Systemen [17]

- **Benutzerunabhängigkeit („user independance“):** Es werden nur die Bewertungen einer Benutzerin oder eines Benutzers zur Erstellung der Empfehlungen herangezogen - somit hängt die Qualität der Empfehlung nicht davon ab, wie viele Personen etwas bewertet haben.

- **Transparenz („transparency“):** Man kann anhand der verwendeten Eigenschaften der empfohlenen Elemente die Bildung der Empfehlung nachvollziehen. Collaborative-Filtering-Systeme (siehe Kapitel 2.3.2) dagegen wirken wie „black boxes“, man kann nicht so ohne weiteres nachvollziehen, wie es zur Empfehlung gekommen ist, da diese auf den Beziehungen und Ähnlichkeiten der BenutzerInnen untereinander basieren.

- **Neue Elemente („new item“):** Da Content-Based-Filtering-Systeme auf der Ähnlichkeit der Elemente basieren, können neue Elemente sofort vorgeschlagen werden (sobald die Eigenschaften des Elementes dem System bekannt sind).

2.3.1.2 Nachteile von Content-Based-Filtering-Systemen

- **Begrenzte Inhaltsanalyse („limited content analysis“):** Content-Based-Filtering-Systeme sind darauf angewiesen, dass die Elemente Eigenschaften besitzen, die vergleichbar sind. Diese Eigenschaften müssen nun entweder automatisiert gefunden werden, was bei Textdokumenten in Verbindung mit diversen Analysemethoden automatisch ablaufen kann, oder es benötigt die Mithilfe von Expertinnen und Experten (Meta-Daten) oder anderen Personen (Tagging) um dem System vergleichbare Eigenschaften zu liefern [17].
  Die Beschreibung mit Hilfe von Meta-Daten muss von Personen durchgeführt werden. Im Falle eines Systems für die Erstellung von Empfehlungen für Filme wären die Meta-Daten beispielsweise der Titel, der Regisseur, die Schauspieler, das Jahr etc. Der Film selbst ist für das System nicht einsehbar [5].

- **Neue BenutzerInnen („new user“):** Um eine aussagekräftige Empfehlung für eine/einen BenutzerIn treffen zu können, sind eine gewisse Anzahl von Bewertungen durch diese/diesen BenutzerIn notwendig, damit das System ihre/seine Vorlieben besser bestimmen kann [17].

- **Überspezialisierung („overspecialization“):** In Content-Based-Filtering-Systemen besteht die Gefahr, dass das System nur Elemente vorschlägt, die einen hohen Ähnlichkeitsgrad zu einem bestimmten Benutzerprofil aufweisen. Das führt nun dazu, dass die Empfehlungen auf Elemente begrenzt sind, die Ähnlichkeiten zu den bereits bewerteten Elementen einer Benutzerin oder eines Benutzers aufweisen. Die/der BenutzerIn wird also theoretisch nie Elemente vorgeschlagen bekommen, die aus ihr/ihm unbekannten Bereichen stammen. (Angenommen die/der BenutzerIn hat keinerlei Erfahrung und keine Bewertungen für griechische Küche im Benutzerprofil. Somit wird sie/er nie einen Vorschlag für ein griechisches Restaurant erhalten).
  Auf der anderen Seite gibt es Fälle, in denen Elemente auf Grund ihrer großen Ähnlichkeit zu bereits gesehenen Elementen nicht vorgeschlagen werden sollten. Als Beispiel können hier Nachrichtentexte angeführt werden, die sich auf identische, bereits gelesene Ereignisse beziehen [1].

Zusammenfassend kann man festhalten, dass eine gewisse Vielfalt bei der Genauigkeit der Empfehlungen anzustreben ist. Idealerweise wird der Benutzerin oder dem Benutzer anstelle einer homogenen Liste von Auswahlmöglichkeiten eine breite Palette von unterschiedlichen Empfehlungen angeboten [1].

2.3.2 Collaborative-Filtering

Bei Collaborative-Filtering wird vom Verhalten einer Gruppe auf das Verhalten eines Individuums geschlossen. Eine Benutzerin oder ein Benutzer wird dabei mit anderen Benutzerinnen und Benutzern des Systems verglichen, um Ähnlichkeiten und Nachbarschaften mit diesen anderen Personen zu finden. Diese Ähnlichkeiten ergeben sich aus den aufgezeichneten Daten über das Konsumverhalten oder über die abgegebenen Bewertungen. Ressourcen, die von „Nachbarn“ beziehungsweise „Interessensverwandten“ konsumiert oder bewertet wurden, sind für die betrachteten BenutzerInnen wahrscheinlich auch von Interesse [11].

Der Ansatz des Collaborative-Filterings umgeht den Schwachpunkt der begrenzten Inhaltsanalyse des Content-Based-Filterings, indem vom ähnlichen Verhalten von Personen auf das Verhalten der betrachteten Person geschlossen wird. Es ist also nicht notwendig den Inhalt der Ressourcen zu kennen, um eine Aussage darüber zu treffen, ob sie für jemanden interessant sind [23].
Ein Faktor für die Wertschätzung eines Empfehlungssystems durch die BenutzerInnen ist die Fähigkeit, überraschende, nicht erwartete Resultate zu erzeugen. Hierfür muss man zwischen „Neuheit“ („novelty“) und Überraschung („serendipity“) differenzieren: Ein neuer Film des eigenen Lieblingsregisseurs wird als Neuheit betrachtet, stellt aber keine große Überraschung für die BenutzerInnen dar. Wahrscheinlich hätte die/der BenutzerIn den Film auch ohne Hilfe entdeckt. Ein Film hingegen, der in der Liste der Empfehlungen nicht erwartet wurde und trotzdem der Benutzerin oder dem Benutzer zusagt, wird als Überraschung bewertet. Empfehlungssysteme, die auf Nachbarschaften der Personen, die das System verwenden, aufbauen („neighborhood based systems“), können dadurch überraschende Empfehlungen liefern [7].
Um Zusammenhänge zwischen der Testperson und den Benutzerinnen und Benutzern in der Datenbank zu finden, werden Vergleiche zu den gespeicherten Benutzerprofilen erstellt. Ein typisches Benutzerprofil eines solchen Systems wäre zum Beispiel ein Vektor aus Elementen und deren Bewertungen, die das System durch die Interaktion (Bewertung von Elementen) mit der Benutzerin oder dem Benutzer erstellt und gespeichert hat [5].

2.3.2.1 Vorteile von Neighborhood-Based-Systemen [7]

- **Einfachheit („simplicity“):** Diese Methoden sind oft intuitiv zu implementieren. Im einfachsten Fall hängt die Berechnung nur von der Anzahl der betrachteten BenutzerInnen mit gleichen Interessen wie die Testperson ab.

- **Begründung der Empfehlungen („justifiability“):** Mit der Verwendung von Neighborhood-Based-Systemen hat man die Möglichkeit, die Erstellung der Empfehlungen zu erklären. Zum Beispiel kann man die benachbarten Elemente und ihre Bewertungen als Begründung für die Empfehlung anzeigen. Die/der BenutzerIn hätte dann in weiterer Folge sogar die Möglichkeit, eine Gewichtung der BenutzerInnen mit gleichen Interessen durchzuführen, um für sie/ihn bessere Resultate zu erzielen.

– **Effizienz („efficiency“):** Neighborhood-Based-Systeme erfordern keinen aufwendigen Trainingsprozess, der in größeren Systemen immer wieder durchgeführt werden muss. Um die Dauer der Empfehlungserstellung zu reduzieren, können die Nachbarschaften offline vorausberechnet werden, wodurch der Prozess enorm beschleunigt wird.

2.3.2.2 Nachteile von Neighborhood-Based-Systemen

– **Überschneidungen („limited coverage“):** Auf Grund der Tatsache, dass die Korrelationen der BenutzerInnen untereinander an Hand der Elemente bestimmt wird, die alle betrachtenden BenutzerInnen bewertetet haben, können BenutzerInnen nur dann als BenutzerInnen mit gleichen Interessen betrachtet werden, wenn sie ähnliche bis identische Elemente bewertet haben [7].
Bei Systemen, die mit Ähnlichkeiten zwischen Personen arbeiten, kann es vorkommen, dass BenutzerInnen ähnliche Elemente nutzen (bewerten), die aber nicht identisch sind. Da hier mit Benutzerähnlichkeiten gearbeitet wird, werden die Ähnlichkeiten zwischen den Elementen nicht erkannt. Man spricht von einem „user based nontransitive association problem“ [11].

– **Spärlichkeit („sparsity“):** Die Anzahl der Bewertungen in einem Empfehlungssystem ist üblicherweise sehr klein im Vergleich zur Anzahl der Bewertungen, die vorhergesagt werden müssen. Somit müssen die Vorhersagen auf Grundlagen von nur wenigen Beispielen getroffen werden. Die Qualität eines Collaborative-Filtering-Systems hängt in weiterer Folge von einer kritischen Anzahl an Benutzerinnen und Benutzern ab. Als Beispiel sei hier ein Empfehlungssystem für Filme genannt: Wenn es viele Filme gibt, die aber nur von einer kleinen Anzahl von Personen bewertet wurden, werden diese Filme kaum vorgeschlagen, auch wenn sie sehr gute Bewertungen haben. Menschen, die einen eher ungewöhnlichen Geschmack haben, können zudem keine qualitativ guten Empfehlungen erhalten, da im System kaum Personen mit ähnlichen Vorlieben vorhanden sind.
Es gibt mehrere Ansätze, um die Spärlichkeit der Daten zu verbessern. Ein Ansatz besteht darin, die bekannten Daten der BenutzerInnen einzubeziehen. Beispielsweise kann das Geschlecht, das Alter, die Wohngegend (Postleitzahl) und die Schulbildung dazu verwendet werden, die BenutzerInnen in demographische Segmente einzuteilen. Dieser Vorgang wird auch „demographic filtering“ genannt. [1].

– **Kaltstart Problem („cold start problem“):**

Tritt auf bei:

1) Neue Benutzerin oder neuer Benutzer: Da die Erstellung der Empfehlungen auf dem Vergleich der Bewertungen der Testperson mit den Bewertungen der anderen BenutzerInnen des Systems beruht, ist eine neue Benutzerin oder ein neuer Benutzer, der noch sehr wenige Bewertungen durchgeführt hat, schwer zu

kategorisieren.
2) Neues Element: Ähnlich ist die Situation bei den Elementen: Elemente, die noch nicht oder kaum bewertet wurden, können nur schwer vorgeschlagen werden [5].

– **Lemming Effekt:** Dieser Effekt konnte beim Online-Shop „Amazon“ (http://www.amazon.com) beobachtet werden: Sehr populäre Produkte, die verständlicherweise von vielen Benutzerinnen und Benutzern erworben werden (dies führt im System zu einer positiven Bewertung für das Element) werden dann auch zusammen mit Produkten empfohlen, zu welchen kein erkennbarer Zusammenhang besteht. Der Grund dafür ist die Ausnahmesituation, dass sehr viele Personen dasselbe Produkt kaufen, und dieses somit für viele „Nachbarschaften“ empfehlenswert machen. Somit taucht das Produkt in Listen auf, wo man es nicht erwarten würde.
Wenn sich das System rein auf Bewertungen stützt, also das eigentliche Kaufverhalten außer acht gelassen wird, werden gut bewertete Produkte immer weiter gestärkt - besonders dann, wenn der alleinige Aufruf als implizite Datenquelle verwendet wird. Neue Objekte haben bei diesem Effekt kaum die Chance empfohlen zu werden [14 S.66, 67].

2.3.3 Knowledge-Based-Filtering

Knowledge-Based-Filtering-Systeme benötigen drei Arten von Wissen [5]:

- **„catalog knowledge“:** Wissen über die gespeicherten Elemente und deren Eigenschaften
- **„functional knowledge“:** Wissen über den Zusammenhang zwischen den Bedürfnissen einer Benutzerin oder eines Benutzers und den Elementen, die dazu passen könnten
- **„user knowledge“:** Wissen über die Bedürfnisse und Vorlieben einzelner BenutzerInnen.

Neben diesem großen Problem der Informationsbeschaffung besitzen Knowledge-Based-Filtering-Systeme aber auch Vorzüge. Beispielsweise gibt es keine Probleme in der Anfangsphase („cold start problem“) und die Bandbreite der Empfehlungen wird nur durch die Wissensbasis des Systems begrenzt [5].

Information kann als detailliertes Modell der BenutzerInnen realisiert werden, oder als Modell, welches den Auswahlprozess beschreibt, oder aber auch als Beschreibung des vorgeschlagenen Datenobjekts. Der komplexe und fehleranfällige Prozess der Extrahierung der benötigten Informationen und der Aufbau der Modelle („knowledge representation“) sind die größten Hürden bei diesem Ansatz [18].

Die Bildung der Informationsbasis erfolgt durch die Auswertung von semantischen Informationen, die über Personen und Ressourcen verfügbar sind oder gesammelt werden. Für diesen Zweck werden Ontologien erstellt, die Modelle der BenutzerInnen und der Ressourcen darstellen, indem Verbindungen zwischen wichtigen Konzepten hergestellt werden. Für die Ähnlichkeitsberechnung werden zwei Elemente (I, J) mit Hilfe einer Ontologie auf ihre semantische Ähnlichkeit hin überprüft:

$$SemSim(I,J) = \frac{count\_common\_desc(I,J)}{count\_total\_desc(I,J)} \qquad (2)$$

$I, J : Elemente$
$count\_common\_desc : Gemeinsame\ Beschreibungen$
$count\_total\_desc : Menge\ aller\ Beschreibungen$

Die Ähnlichkeit zweier Elemente I und J basiert auf dem Verhältnis der geteilten, gemeinsamen Beschreibungen in einem RDF (ressource description framework) Graphen und der Anzahl aller Beschreibungen in diesem Graphen [6].

2.3.4 Hybride Empfehlungssysteme [5]

Es gibt sieben Arten, wie Empfehlungssysteme miteinander kombiniert werden können.

1) **„Mixed“:** Die Ergebnisse von verschiedenen Systemen werden gemeinsam in einer kombinierten Liste gezeigt oder aber auch durch separate Listen präsentiert.

2) **„Weighted“:** Die Ergebnisse der Systeme werden durch eine Gewichtung kombiniert um eine Empfehlung zu erhalten.

3) **„Switching“:** Das System entscheidet mit Hilfe von bestimmten Kriterien, welches Empfehlungssystem in welchem Kontext verwendet wird. Dieses Kriterium kann als „Vertrauen“ in das Ergebnis des Systems gesehen werden. Idealerweise würde man nun bei den Systemen den „Vertrauenswert“ ermitteln und das System mit dem höchsten Wert wählen. Dies setzt jedoch die absolute Vergleichbarkeit der Systeme voraus, welche in der Praxis nicht gegeben ist.
   Als Alternative wählt man ein System aus, das als primäres System agiert. Wenn der erreichte Vertrauenswert einen Grenzwert innerhalb des Systems überschreitet, dann werden die Empfehlungen dieses System verwendet, ansonsten die Empfehlungen des sekundären Systems. Welches System nun als primäres und welches als sekundäres agieren soll, muss durch Testabläufe festgestellt werden.

4) **„Cascade“:** Ein Empfehlungssystem verfeinert die Ergebnisse eines anderen Empfehlungssystems. Wenn das primäre Empfehlungssystem unpräzise Ergebnisse liefert, dann kann ein sekundäres System in Kraft treten und versuchen die Ergebnisse zu verbessern. Dies funktioniert beispielsweise gut bei der Kombination von Collaborative-Filtering als Primärsystem und Content-Based-Filtering oder Knowledge-Based-Filtering als Sekundärsystem.

5) **„Feature Combination“:** Hier werden Eigenschaften, die mit einer Art von Empfehlungserstellung verknüpft sind (Collaborative-Filtering), in einen Algorithmus eingebracht, der auf einer unterschiedlichen Datenbasis operiert (Content-Based-Filtering). Zum Beispiel bilden die Bewertungen A-, B-, C+ drei neue Eigenschaften für ein Element.

6) **„Feature Augmentation“:** Das Ergebnis eines Vorverarbeitungsschrittes wird als Eigenschaft für die Weiterverarbeitung verwendet (zum Beispiel: „clustering“). Diese Erweiterung kann oft auch offline vorgenommen werden, um die Berechnung zu beschleunigen.

7) **„Meta-level“:** Ein Empfehlungssystem erstellt ein Modell, welches als Eingabe für ein zweites System dient.

Drei Möglichkeiten sind unabhängig von der Reihenfolge der kombinierten Systeme, nämlich „Weighted“, „Switching“ und „Feature Combination“. Ein „Weighted“-System mit der Reihenfolge Content-Based-Filtering / Collaborative-Filtering ist im erhaltenen Ergebnis identisch mit einem System mit Collaborative-Filtering / Content-Based-Filtering. (gilt auch für Kombinationen mit Knowledge-Based-Filtering).

2.4 Distanz und Fehlermaße

Ein Distanzmaß dient zur Bestimmung des Abstandes zwischen zwei Datenpunkten (oft als Vektoren dargestellt) oder anders betrachtet zur Bestimmung der Ähnlichkeit zweier Datenpunkte.
Die Wahl des Distanzmaßes hat kaum Einfluss auf die Genauigkeit der Vorhersagen des Empfehlungssystems [2]. Entscheidend ist die Interpretation der Ergebnisse und die Anwendbarkeit auf die gegebenen Daten. Auch die Komplexität der Berechnungen kann ein Kriterium für die Auswahl eines Ähnlichkeitsmaßes sein.

2.4.1 Cosinus-Ähnlichkeitsmaß

Das Cosinus-Ähnlichkeitsmaß (Cosine Similarity) wird für die Berechnung der Ähnlichkeit zweier n-dimensionaler Vektoren herangezogen:

$$similarity_{\cos}(A, B) = \cos(phi) = \frac{A \cdot B}{|A||B|} = \frac{\sum_{i}^{n} A_i \times B_i}{\sqrt{\sum_{i}^{n} (A_i)^2} \sqrt{\sum_{i}^{n} (B_i)^2}} \quad (3)$$

$A, B: Vektoren$

Die Ähnlichkeit zwischen den Vektoren A und B wird also durch den Winkel phi ausgedrückt, welcher wiederum aus der Division des Kreuzproduktes beider Vektoren durch das Produkt der Beträge beider Vektoren berechnet wird (Gleichung 3). Je ähnlicher die beiden Vektoren sind, desto geringer ist der Winkel zwischen ihnen. Dieses Ähnlichkeitsmaß findet vor allem in der Textanalyse Verwendung [14].

2.4.2 Pearson Korrelationskoeffizient

$$similarity_{pc}(A, B) = \frac{\sum_{i}^{n} ((A_i - \bar{A})(B_i - \bar{B}))}{\sqrt{\sum_{i}^{n} (A_i - \bar{A})^2 * \sum_{i}^{n} (B_i - \bar{B})^2}} \quad (4)$$

$A, B: Vektoren$

Beim Pearson Korrelationskoeffizienten werden bei den Vektoren A und B die Durchschnitte der Komponenten der Vektoren in die Gleichung (4) einbezogen [14]. Dabei wird von jeder Komponente des Vektors der Mittelwert aller Komponenten des Vektors abgezogen. Mit den Resultaten wird wieder, wie bei dem Cosinus-Ähnlichkeitsmaß, der Winkel zwischen den Vektoren berechnet.

2.4.3 Euklidischer Abstand

$$similarity_{ea}(A, B) = \sqrt{\sum_{i}^{n} (A_i - B_i)^2} \tag{5}$$

A, B: *Vektoren*

Der Euklidische Abstand ist eines der gebräuchlichsten Distanzmaße. Er bezeichnet den Abstand zweier Punkte in der Ebene (zum Beispiel der Satz von Pythagoras) oder Vektoren im Raum wie in Gleichung 5 dargestellt: Der Abstand zwischen den Vektoren A und B ist die Länge des Differenzvektors, wobei A_i und B_i jeweils die Komponenten der Vektoren darstellen [14].

2.4.4 Mean Absolute Error

Eine Maßgröße für die Genauigkeit von Empfehlungssystemen ist der „mean absolute error“ (MAE). Der MAE bezeichnet den Mittelwert aller Fehler, die bei der Vorhersage einer Bewertung gemacht wurden.

$$|E| = \frac{\sum_{i=0}^{N} |e_i|}{N} \tag{6}$$

N: *Anzahl Vorhersagen*
e: *Fehler bei einer Bewertung*
E: *gesamter Fehler*

In Gleichung 6 steht *N* für die Gesamtanzahl aller Vorhersagen, e_i bezeichnet den Fehler zwischen der Vorhersage der Bewertung und der tatsächlichen Bewertung. Je niedriger der MAE ist, desto genauer sind die Vorhersagen in Bezug auf die Bewertungen, die eine Benutzerin oder ein Benutzer vornehmen wird [12].

2.4.5 Precision und Recall

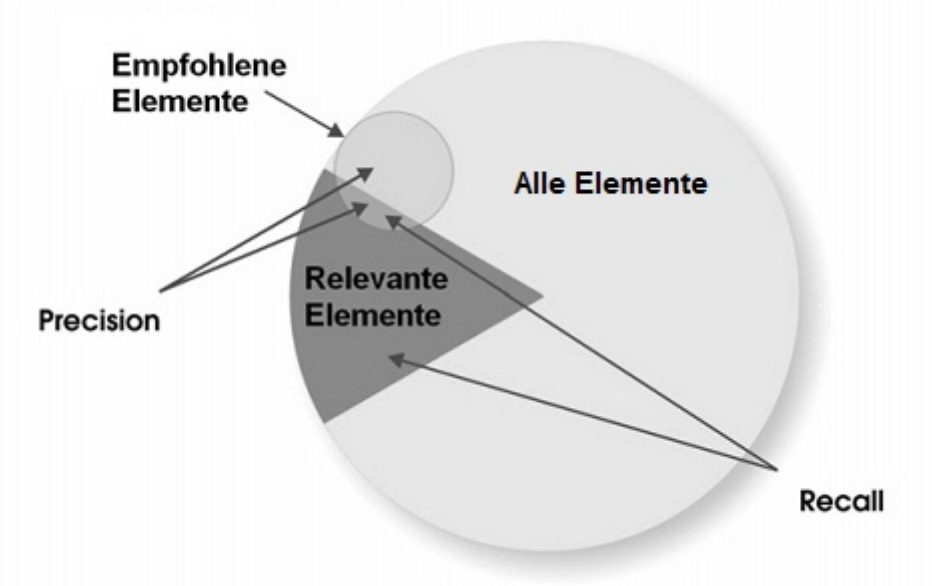


Abbildung 3: Precision und Recall [14]

Mit den Werten Precision (P) und Recall (R) errechnet sich ein theoretisches Maß für die qualitative Güte der erstellten Empfehlungen. Als „Precision“ wird das Verhältnis von relevanten zu insgesamt empfohlenen Texten bezeichnet, „Recall“ ist das Verhältnis von relevanten, empfohlenen Texten zu den insgesamt relevanten Texten.

$$P = \frac{\#\,Items_{relevant\,,\,recommended}}{\#\,Items_{recommended}} \tag{7}$$

$$R = \frac{\#\,Items_{relevant\,,\,recommended}}{\#\,Items_{relevant}} \tag{8}$$

Besteht beispielsweise die Menge der relevanten Elemente aus 10 Elementen und das System macht 15 Vorschläge unter denen sich 5 relevante befinden, so erhält man für Precision den Wert von 33% (Gleichung 7) und für Recall einen Wert von 50% (Gleichung 8) [14 S.40, 41].

Für die Berechnung dieser Maßgrößen müssen die Daten auf die binäre Relation „relevant / nicht relevant“ abgebildet werden. Falls nun die Bewertungen nicht binär sind (zum Beispiel eine Skala von 1 – 5) muss ein Weg gefunden werden, dies auf die notwendige Relation zu reduzieren. Ein Weg wäre, Bewertungen mit 4 und 5 als relevant einzustufen, die Bewertungen von 1 – 3 als nicht relevant einzustufen. Hier zeigt sich in der Definition von „relevant“ ein Problem dieser Maßgrößen, da dies eine sehr subjektive Einschätzung ist. Eine Person kann nun den Wert 3 auf dieser Skala bereits als relevant betrachten, für jemand anderen erscheint er eher unwichtig.
Für den Wert von Recall ist der Relevanzbegriff noch gewichtiger. In der grundlegenden Bedeutung benötigt man für die Messung dieser Größe das Wissen über die Relevanz eines jeden Elements für jede Person, die das System verwendet.
Ein Weg, um Precision und Recall zu approximieren, besteht darin, die besten N Elemente

(„top N items“) zu bestimmen, für die Bewertungen vorhanden sind. Dazu werden die Daten der Bewertungen in eine Trainings- und eine Testmenge aufgeteilt. Der Algorithmus trainiert nun auf der Trainingsmenge und gibt Empfehlungen für die besten N Elemente aus. Unter der Annahme, dass die Verteilung von relevanten und nicht relevanten Elementen in dieser Vorhersage mit der Verteilung der Relevanz über alle Elemente des Systems für die Testperson übereinstimmt, erhält man gute Näherungen für Precision und Recall [8].

2.4.6 Qualität der Empfehlungen

Eine Schwäche der empirischen Maßgrößen zur Evaluierung von Empfehlungssystemen (zum Beispiel MSE) ist, dass damit nicht die Nützlichkeit oder Qualität einer Empfehlung gemessen werden kann. Ein Empfehlungssystem für einen Supermarkt, welches offensichtliche Güter vorschlägt, die die Konsumenten auf alle Fälle kaufen (Brot, Milch), erzielt natürlich hohe Werte in Bezug auf die Genauigkeit der Vorschläge – hilfreich, im Sinne einer Kaufunterstützung, sind diese aber nicht [1].
Abseits der Genauigkeit der Empfehlungen gibt es noch andere Größen, über die ein Empfehlungssystem bewertet werden kann:

- **Abdeckung („coverage“):** Diese Größe misst den Prozentsatz der Elemente, für die das System Empfehlungen abgeben kann. Der einfachste Weg um diese Größe zu messen besteht darin, eine zufällige Anzahl an Benutzer – Element Paaren zu wählen und für diese vom System Empfehlungen erstellen zu lassen. Der Prozentsatz, für wie viele Paare eine Empfehlung erstellt werden konnte, beschreibt nun die Abdeckung des Algorithmus.
  Beim Berechnen dieser Größe darf die Genauigkeit nicht außer acht gelassen werden. Wenn beispielsweise ein Element existiert, das für jeden, der das System verwendet, uninteressant ist, dann sollte es nicht empfohlen werden. Dies führt zwar zu weniger Abdeckung aber dafür zu höherer Genauigkeit [8].

- **Vertrauen („confidence“):** Die grundlegende Frage dieses Bereiches ist: „Wie sicher es, dass die Empfehlungen präzise und richtig sind?“ Bei Bewertungssystemen kann es vorkommen, dass hohe (und auch niedrige) Bewertungen auf sehr wenigen Daten beruhen und somit kaum Aussagekraft besitzen. Die Messung des Vertrauens in die Bewertungen ist schwierig, ein Weg wäre der Vergleich der Genauigkeit von Bewertungen mit hohen Vertrauenswerten mit Bewertungen mit niedrigen Vertrauenswerten [8].

- **Lernrate („learning rate“):** Die Größe misst wie schnell ein Algorithmus gute Empfehlungen erstellen kann (zum Beispiel in einer „Kaltstart“ Situation, ausgelöst durch eine neue unbekannte Benutzerin beziehungsweise einen neuen unbekannten Benutzer oder ein unbekanntes Element) [8].

- **Neuheit/Überraschung („novelty/serendipity“):** Eine Bewertung hinsichtlich der Neuheit der Empfehlungen. Eine Bewertung dieser Größen ist auch schwierig, da Überraschung ein Maß dafür ist, wie oft BenutzerInnen Elemente vorgeschlagen bekommen, die sie mögen und die gleichzeitig überraschend in der Empfehlungsliste auftauchen [8].

2.5 Erweiterungen und Kombinationen von Empfehlungssystemen

2.5.1 Performanceverbesserung bei Collaborative-Filtering-Systemen

Verschiedene Empfehlungssysteme verwenden verschiedene Ähnlichkeitsmaße um die größtmögliche Effizienz zu erreichen. Eine Strategie, die allgemeine Verwendung findet, besteht darin, alle Ähnlichkeiten zwischen den Benutzerinnen und Benutzern offline zu berechnen und diese Daten dann in einer Datenbank zur Verfügung zu stellen. Aus diesen gespeicherten Daten über die betrachteten Ähnlichkeiten kann dann sehr schnell eine Empfehlung erstellt werden. Von Zeit zu Zeit werden dann die gespeicherten Daten auf den neuesten Stand gebracht [1].

2.5.1.1 Information-Filtering

Um das Verhalten einer Benutzerin oder eines Benutzers vorherzusagen gibt es das sogenannte „information filtering“, also das Filtern anhand von Informationen über die Interessen von einzelnen Personen.
Es gibt für dieses Filtern zwei Ansätze. Zum einen den expliziten Ansatz, wo die/der BenutzerIn Interessen und etwaige Schlüsselwörter zur Verfügung stellt, zum anderen der implizite Ansatz, wo Schlüsselwörter und Interessengebiete aus den Handlungen der Benutzerin beziehungsweise des Benutzers gewonnen werden. Der explizite Ansatz ist natürlich begrenzt, denn es können nur Elemente empfohlen werden, die auf Grund von Ähnlichkeiten mit den Schlüsselwörtern, die die BenutzerInnen angegeben haben, gefunden wurden. Das heißt eine/ein BenutzerIn kann nur „gute“ Empfehlungen erhalten, wenn sie/er die Interessen wahrheitsgemäß angibt [10].

2.5.1.2 Default-Voting

Das „default voting“ ist eine Erweiterung zur Offline-Berechnung von Ähnlichkeiten. Collaborative-Filtering-Systeme sind darauf angewiesen, dass sich die Bewertungen der BenutzerInnen überschneiden, da ansonsten keine Ähnlichkeiten berechnet werden können: Für die Berechnung der Ähnlichkeit zwischen BenutzerIn *x* und BenutzerIn *y* benötigt man eine Menge von Elementen, die von beiden bewertet wurden. Die Genauigkeit der Empfehlungen kann nun dadurch gesteigert werden, indem man für die fehlenden Bewertungen einen Standardwert annimmt [1].

2.5.1.3 Multi-Dimensionalität von Empfehlungen

Ein Ansatz um genauere Empfehlungen zu erstellen ist die Erweiterung des zweidimensionalen Raumes ($Benutzer \times Element$) hin zu einem mehrdimensionalem Raum. Die Nützlichkeitsfunktion (utility function) *u* kann nun als Funktion über mehrere Dimensionen abgebildet werden:

$$u : D_1 \times ... \times D_n \rightarrow R$$

$$u : utility\ function \quad (9)$$
$$D : Dimension$$
$$R : Ergebnisraum$$

Am Beispiel eines Empfehlungssystems für Filme kann dies folgendermaßen aussehen: Die erste Dimension d_1 sind die Eigenschaften des Films, d_2 beschreibt die Person, die den Film sehen will, d_3 beschreibt, wo und wie der Film angesehen wird (im Kino, zuhause im TV, zuhause auf DVD), d_4 bezieht sich auf die Personen, mit denen man den Film sehen möchte (alleine, mit dem Partner, mit Freunden) und d_5 lässt die zeitliche Komponente einfließen, also wann der Film angesehen wird (am Wochenende, unter der Woche etc.).
Die Nützlichkeitsfunktion $u(d_{1,} d_{2,} d_{3,} d_{4,} d_5)$ kann dadurch sehr komplex werden [1].

2.5.1.4 Unaufdringlichkeit (Nonintrusiveness)

Viele Empfehlungssysteme sind dahingehend „aufdringlich“, dass sie die Mithilfe der BenutzerInnen verlangen, um gute Ergebnisse zu erzielen. Das System braucht oftmals viele Bewertungen der Elemente um daraus qualitativ gute Empfehlungen berechnen zu können.
Da man nicht von der Mithilfe der BenutzerInnen ausgehen kann und will, müssen Ansätze gewählt werden, die die Bewertungen implizit durch das Benutzerverhalten erschließen. Beispielsweise kann die Zeit, die jemand für das Lesen eines Artikels in einer Newsgroup aufwendet, stellvertretend für eine Bewertung des Artikels verwendet werden. Diese impliziten Beobachtungen haben allerdings nicht die gleiche Aussagekraft wie explizite Bewertungen durch diese Person und sollten eher als Ergänzung verwendet werden [1].

2.5.2 Einbeziehung des Kontextes bei Collaborative-Filtering-Systemen

Eine weitere Verbesserung der Qualität der Empfehlungen kann durch die Einbeziehung eines Kontextes erfolgen. Dabei wird die zweidimensionale Betrachtung („User“-“Item“) um die Dimensionen des Kontextes erweitert. Ein einfacher und immer verfügbarer Kontext wäre etwa die Jahreszeit, in der die Interaktion der Benutzerin oder des Benutzers mit dem System stattgefunden hat [20].

Es gibt drei grundlegende Möglichkeiten einen Kontext in ein Empfehlungssystem einzubeziehen [20]:

1. **contextual-pre-filtering:** Bewertungen, die nicht zum Kontext passen, werden von vornherein gefiltert.
2. **contextual-post-filtering:** Nach der Anwendung der klassischen zwei-dimensionalen Methoden zur Empfehlungserstellung werden die Resultate mit dem Kontext kombiniert.
3. **contextual-modeling:** Der Kontext wird in die Empfehlungserstellung direkt eingebunden.

2.5.2.1 Pre-Filtering [20]

Die Informationen des Kontextes werden dazu benutzt um Bewertungen (oder andere Aktionen der Benutzerin oder des Benutzers) aus der Menge der zu betrachtenden Bewertungen herauszufiltern. Das Empfehlungssystem wählt also aus der gesamten Menge der Bewertungen nur die aus, die dem Kontext *k* genügen. Die anschließend erstellte Matrix *Users x Items* enthält dann nur mehr Daten, die zum Kontext *k* passen. Auf diesem reduzierten Datenset werden dann die Empfehlungsalgorithmen ausgeführt.

2.5.2.2 Post-Filtering [20]

Bei diesem Ansatz werden die Informationen des Kontextes erst nach der Erstellung der Empfehlungen angewendet. Ein Ansatz, um die erhaltenen Empfehlungen mit dem Kontext zu verknüpfen, ist die Berechnung einer „kontextuellen Wahrscheinlichkeit“ (Gleichung 10).

$$Rating_k(i,j) = \begin{Bmatrix} Rating(i,j) & if & P_k(i,j) \geq P^* \\ 0 & if & P_k(i,j) < P^* \end{Bmatrix}$$

$k : Kontext$
$i : Person$
$j : Element$
$P_k : Wahrscheinlichkeit\ bei\ Kontekt\ k$
$P^* : Grenzwert$

(10)

Gleichung 10 sagt aus, mit welcher Wahrscheinlichkeit eine Person ein Element im Zusammenhang mit dem gegebenen Kontext auswählen würde. Bei einem Webshop beispielsweise wäre $P_k(i,j)$ die Wahrscheinlichkeit P im Kontext *k*, dass eine Person *i* ein Element *j* kaufen würde. Diese Wahrscheinlichkeit ergibt sich aus der Anzahl der Nachbarn der Person (Personen mit großer Ähnlichkeit zu Person *i*) die dasselbe Element *j* im selben Kontext *k* gekauft haben, geteilt durch die gesamte Anzahl der Nachbarn von Person *i*. Wenn diese Wahrscheinlichkeit größer als ein Schwellenwert P* ist, dann wird die Vorhersage der Bewertung beibehalten, ansonsten verworfen.

2.5.3 Clustering

Für die Effektivität von Empfehlungssystemen mit Collaborative-Filtering ist es sehr wichtig, dass die Nachbarinnen und Nachbarn einer Person, die das System verwendet, möglichst genau bestimmt werden. Nachbarinnen und Nachbarn sind jene BenutzerInnen mit ähnlich bis identisch bewerteten Elementen, wie die betrachtete Person selbst. Aus den Bewertungen der benachbarten BenutzerInnen kann man dann auf den Geschmack der betrachteten Person schließen.
Ab einer gewissen Datenmenge tauchen in Systemen Probleme auf, die Empfehlungen schnell genug zu berechnen, da die Anzahl der zu vergleichenden BenutzerInnen einfach zu groß wird. Eine Methode um dies zu umgehen ist „clustering“. Dabei werden die BenutzerInnen in Bereiche („cluster“) aufgeteilt. Die Nachbarschaft ist dann jener cluster mit der geringsten Distanz zur betrachteten Person [13].
Weitere Verfahren zur Klassifikation von Elementen wären: Naiver-Bayes-Klassifikator (berechnet die bedingte Wahrscheinlichkeit, dass ein Element einer Klasse angehört), ID3-Bäume (Klassifikation mit Entscheidungsbäumen) und das Minimum-Description-Length-Verfahren. Weiterführende Informationen zu diesen Techniken findet man bei [14].
Als Beispiele für Klassifikationsverfahren betrachten wir den k-means – Algorithmus, der zu den am häufigsten angewendeten Verfahren zählt, und den RNSA – Algorithmus.

2.5.3.1 k-means-Algorithmus

Bei der Clustering-Methode, die den *k-means*-Algorithmus verwendet, werden *k* Cluster gebildet, die aus Benutzerinnen und Benutzern bestehen, die untereinander ähnliche Eigenschaften (meistens Bewertungen von Elementen) aufweisen.
Im ersten Schritt werden *k* zufällige BenutzerInnen aus der Datenmenge ausgewählt. Diese bilden die ersten Zentren der *k* Cluster. Nun werden alle BenutzerInnen diesen Zentren zugeordnet, indem ihre Distanz zu diesem Zentrum bestimmt wird. Als Beispiel zur Distanzmessung kann hier der Pearson-Koeffizient (siehe Kapitel 2.4.2) verwendet werden [13].

Nach der Bildung der ersten Cluster wird der neue Mittelpunkt anhand der zugeordneten BenutzerInnen berechnet. Wenn für jeden Cluster das neue Zentrum gefunden wurde, werden wieder alle anderen BenutzerInnen mit den Distanzmaßen den neuen Zentren zugeordnet.
Diese Iteration wird solange durchgeführt, bis ein gewisser Schwellenwert erreicht ist. Dieser Schwellenwert besteht üblicherweise aus der Distanz zwischen dem neu gefundenen Zentrum und dem Zentrum in der vorherigen Iteration. Mit anderen Worten: Der Algorithmus terminiert, wenn die Bewegung der Zentren unter einen bestimmten Schwellenwert gefallen ist [12].

Wenn alle BenutzerInnen zugeordnet wurden und das Clustering somit abgeschlossen ist, wählt man den Cluster mit dem größten Pearson-Koeffizienten zwischen dem Clusterzentrum und der Testperson. Nun kann die Empfehlung für die Testperson auf Basis des gefundenen Clusters berechnet und somit auch für eine große Anzahl an Benutzerinnen und Benutzern erstellt werden. Die Qualität der Empfehlungen hängt als logische Folge stark von der Qualität der Clusterbildung ab [13].

2.5.3.2 Refined Neighbour Selection Algorithm (RNSA) [13]

Dieser Algorithmus zur Nachbarschaftssuche bildet ein Datenset als ungerichteten Graphen ab, in welchem die Knoten die einzelnen Personen darstellen und die gewichteten Kanten die Ähnlichkeit zwischen den Personen abbilden.

RNSA Algorithmus Zusammenfassung:

Input: Testperson *t*, Datenmenge *S*
Output: *Nachbarn*

1) Erstelle *k* Cluster aus *S* mit Hilfe des *k-means*-Algorithmus
2) Finde den ähnlichsten Cluster *C* für die Testperson *t*
3) Gib Testperson *t* zu Cluster *C* hinzu und bezeichne ihn als *v*
4) Gib v zu *Nachbarn*
5) Wenn es genug Nachbarn gibt, dann Ende. Ansonsten führe eine Breitensuche in *C* durch, ausgehend von *v*. Wenn einer der Knoten (Personen) die Kriterien erfüllt (Schwellenwert größer als *H* oder kleiner als *L*), dann sei die Person das neue *v*. Zurück zu Schritt 4.

Mit Hilfe des *k-means*-Algorithmus werden *k* Cluster aus der Datenmenge gebildet. Anschließend wird der Cluster *C*, welcher die größte Ähnlichkeit mit der Testperson aufweist, ausgewählt. Die Testperson *t* wird nun in *C* eingefügt und von dort aus eine Breitensuche gestartet. Wenn umliegende Knoten der Testperson die Kriterien erfüllen (Schwellenwerte für den Pearson-Koeffizienten), dann werden diese zu den Nachbarn der Testperson hinzugefügt. Ansonsten werden die Nachbarn der Nachbarn betrachtet und so weiter, bis genügend Nachbarn gefunden wurden.
Nun können diese Nachbarn zur Berechnung der Vorhersage einer Bewertung eines noch nicht bewerteten Elements herangezogen werden. Zusätzlich können auch noch die Attribute eines Elementes in die Bewertung einfließen.

2.5.4 Einbeziehung der Elementeigenschaften bei Collaborative-Filtering [12]

Systeme, die mit Collaborative-Filtering arbeiten, funktionieren in der Regel gut, da sie auf den Bewertungen von anderen Benutzerinnen und Benutzern basieren, welche ähnliche Charakteristika und Präferenzen aufweisen. Collaborative-Filtering-Systeme können aber nicht durchgehend sehr gute Empfehlungen abgeben, da solche Systeme eben **nur** auf der Grundlage von Bewertungen arbeiten und somit die Eigenschaften der einzelnen Elemente außer Acht lassen.
Um die Qualität der Empfehlungen zu erhöhen, können die Eigenschaften der Elemente („items“) miteinbezogen werden. Normalerweise können diese Eigenschaften direkt aus den Eigenschaften der Objekte in der wirklichen Welt abgeleitet werden.

Berechnung der Vorliebe einer Person für ein neues Element:

$$P_{a,i} = \bar{r}_a + \frac{\sum_k \{w_{a,k} \times (r_{k,i} - \bar{r}_k)\}}{\sum_k |w_{a,k}|} \qquad (11)$$

a, k : *Personen*
i : *Element*
r : *Bewertung*
w : *Übereinstimmung* (*Gleichung* 12)

$$w_{a,k} = \frac{\sum_j (r_{a,j} - \bar{r}_a)(r_{k,j} - \bar{r}_k)}{\sqrt{\sum_j (r_{a,j} - \bar{r}_a)^2 \sum_j (r_{k,j} - \bar{r}_k)^2}} \qquad (12)$$

a, k : *Personen*
i, j : *Elemente*
r : *Bewertung*

$P_{a,i}$ steht für die Vorliebe einer Person *a* in Bezug auf ein Element *i*. $\bar{r}_a$ und $\bar{r}_k$ stehen für den jeweiligen Durchschnittswert der Bewertungen von Person *a* und *k*.
$r_{k,i}$ und $r_{k,j}$ stellen die Bewertungen der Person *k* für die Elemente *i* und *j* dar, $r_{a,j}$ ist die Bewertung der Person *a* für Element *j*.
Wenn die Personen *a* und *k* ähnliche Bewertungen für ein Element abgegeben haben, dann ergibt $w_{a,k}$ einen Wert größer als 0. Der Betrag $|w_{a,k}|$ zeigt dann an, in wieweit Person *a* mit Person *k* in Bezug auf die Elemente, die beide bewertet haben, übereinstimmt. Man spricht in diesem Fall davon, dass Person *a* ein „positiver Nachbar“ von Person *k* ist (und umgekehrt).
Im Fall, dass die Personen gegensätzliche Bewertungen für ein Element abgegeben haben, wird der Koeffizient $w_{a,k} < 0$. Somit kann Person *a* als „negativer Nachbar“ zu

Person *k* bezeichnet werden. Der Betrag $|w_{a,k}|$ beschreibt somit, wie sehr sich die beiden Personen in der Meinung unterscheiden.
Wenn es keinen Zusammenhang zwischen den Personen gibt, gilt $w_{a,k}=0$.

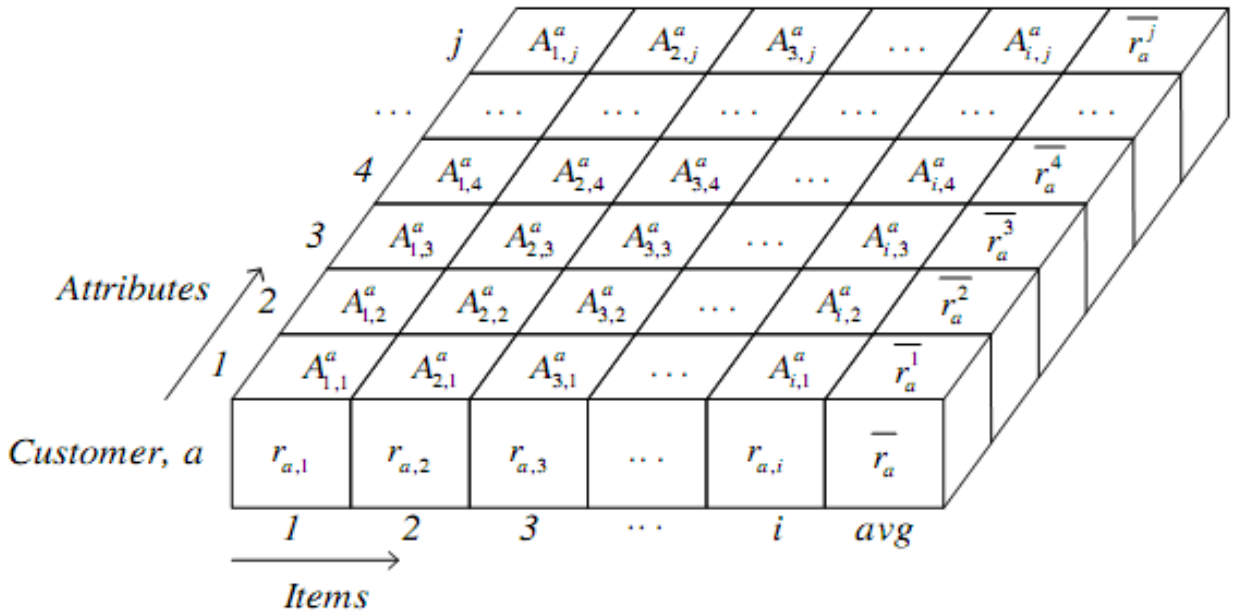


Abbildung 4: Item-Attribut-Matrix eines Users [12]

Um die Attribute eines Elements in den Prozess der Empfehlungserstellung einbeziehen zu können, kann man eine Item-Attribut-Matrix (Abbildung 4) verwenden. Diese Abbildung veranschaulicht die gespeicherten Informationen. $r_{a,i}$ bezeichnet hier die Bewertungen eines Person *a* für das Element *i*. Ein Element kann nun eine beliebige Kombination von *j* Attributwerten annehmen [12]:

$$Attribute(i) \subset \{1,2,3,\ldots,j\}$$

$$i: Element$$
$$j: Anzahl\ Attributwerte$$

(13)

Ob nun eine Eigenschaft für ein Element zutrifft, wird binär festgehalten [12]:

$$A^a_{i,j} \in \{0,1\}$$

$$a: Person$$
$$i: Element$$
$$j: Attribut$$

(14)

2.6 Die Empfehlung als Fallbeschreibung [18]

Case-Based-Reasoning, im folgenden CBR genannt, ist eine der erfolgreichsten Methoden für maschinelles Lernen, in welcher die Lösung eines Problemfalls („case“) auf der Analyse und Wiederverwendung bereits gelöster Problemfälle beruht.
In der einfachsten Umsetzung dieses Prinzips werden Produkte auf Grund von Ähnlichkeiten zu der Beschreibung des Produktes durch die BenutzerInnen vorgeschlagen. Die sogenannte Problemkomponente des Falles ist eine Sammlung von Produkteigenschaften, die von der Benutzerin beziehungsweise vom Benutzer eingegeben werden. Die Lösungskomponente ist das Produkt selbst.

Der Prozess zur Lösungsfindung wird durch ein Ähnlichkeitsmaß gesteuert, mit dessen Hilfe man die Beschreibung des Produktes mit den bekannten Produkten in der Fall-Datenbank („case base“) vergleichen kann. Die am besten passenden Produkte werden aus der Datenbank genommen und vorgeschlagen. Diese Ergebnisliste kann nun verfeinert werden, falls die gefundenen Produkte nicht den Wünschen entsprechen. Eine Verfeinerung löst einen neuen Erstellungszyklus für die Empfehlungen aus.

In Empfehlungssystemen, die fallbasiert arbeiten („Case Based Reasoning Recommender Systems“ oder kurz „CBR-RC“), hängt die Genauigkeit der generierten Empfehlungen von verschiedenen Faktoren ab:

- Von der Fähigkeit die Wünsche der BenutzerInnen mit den Produktbeschreibungen zu vergleichen, beziehungsweise in Verbindung zu bringen;

- Von den Werkzeugen, um die Verbindung sichtbar zu machen;

- Von der grafischen Benutzeroberfläche, die Inhalte für die BenutzerInnen zugänglich macht, sowohl in der Fallbeschreibung als auch in der Ergebnispräsentation.

2.6.1 Case-Based-Reasoning

Ein Fall („case“) beschreibt eine Erfahrung aus der Vergangenheit und besteht aus der Beschreibung des Problems sowie der angewendeten Lösung. Alle Fälle werden in einer Fall-Datenbank gespeichert. Wenn das System mit einem neuen Problem konfrontiert wird, wird die Datenbank nach ähnlichen Fällen durchsucht und ein dort gespeicherter Lösungsansatz wiederverwendet und gegebenenfalls modifiziert.

CBR ist ein zyklischer Prozess um Probleme zu lösen (Abbildung 5) und besteht hauptsächlich aus vier Schritten: Abfrage („retrieve“), Wiederverwendung („reuse“), Anpassung („adaption“) und Aufbewahrung („retain“). Die Anpassungsphase teilt sich in zwei Unterschritte, nämlich Korrektur („revise“) und Überprüfung („review“). Im Korrekturschritt wird die gefundene Lösung an das neue Problem angepasst und dann im nächsten Schritt überprüft.

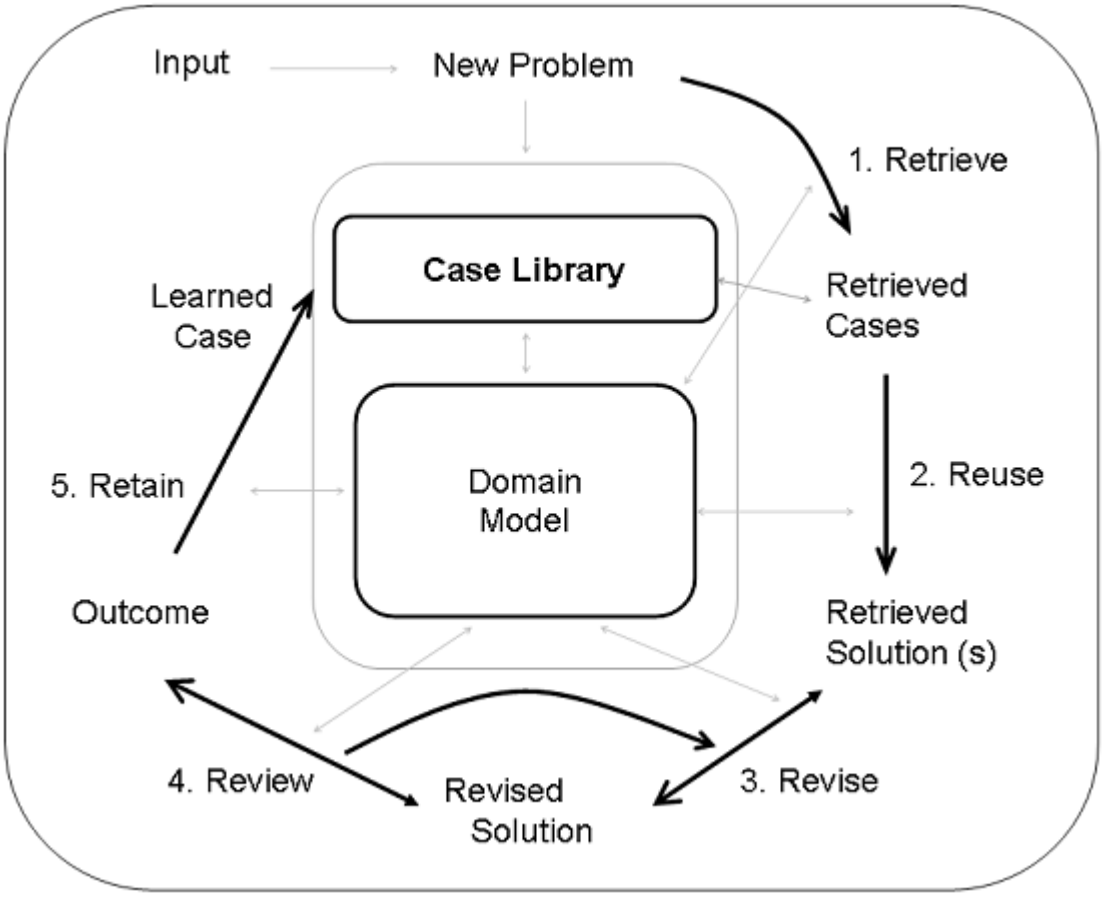


Abbildung 5: Case-Based-Reasoning-Problemlösungszyklus [18]

Als Beispiel aus der Praxis wird hier ein Diagnosesystem genannt. Die Symptome des Patienten werden an das System übermittelt (neuer Problemfall), welches nun in der Datenbank nach ähnlichen Fällen sucht. Manchmal kann die Lösung einfach verwendet werden, aber in der Mehrzahl der Fälle muss die Lösung an die neue Problemstellung angepasst werden. Nach dieser Anpassung erstellt das System einen neuen Fall (Problem und neue Lösung) und speichert ihn in der Datenbank ab.

Eine grundlegende Problemstellung von CBR ist die Modellierung einer Fallbeschreibung. Man muss alle notwendigen Attribute finden, aus denen eine Fallbeschreibung bestehen soll und welche Beschreibungssprache dafür in Betracht kommt. Der Prozess der Modellierung einer Fallbeschreibung besteht aus folgenden Schritten:

- Auswahl der relevante Attribute
- Definition von Indizes
- Strukturierung des Wissens

Die Definition von Indizes dient der Beschleunigung des Suchprozesses in der Fall-Datenbank, indem man sich auf die wichtigsten Dimensionen der Daten beschränkt. Mit Hilfe dieser Indizes werden also die Attribute eines Falles definiert und somit auch vergleichbar gemacht (Distanz zwischen den Attributen). Auf das vorher genannte Beispiel angewandt bedeutet das, dass Attribute wie zum Beispiel „Alter“, „Geschlecht“ oder „Beruf“

für die Diagnose der Krankheit viel weniger Gewicht haben als die Symptome des Patienten.

Viele Empfehlungssysteme implementieren einen Iterationsschritt als Erweiterung der fünf Grundschritte des CBR-Zyklus. In diesem Schritt kan die aktuelle Empfehlungsliste durch Aktionen angepasst werden. Die/der BenutzerIn kann die Empfehlungen bewerten oder die Angaben verfeinern, was einen neuen Zyklus auslöst.

Am Anfang eines Zyklus steht in jedem Fall die Benutzereingabe. Es gibt viele Strategien um mit der Benutzerin oder dem Benutzer in Kontakt zu treten, am öftesten wird eine dialogbasierte Strategie angewendet. Das System bietet durch Vorschläge oder Alternativen zu einem Produkt seine Hilfe an, um die Entscheidung zu erleichtern. Eine Auflistung von Produkteigenschaften, mit denen die/der BenutzerIn das gewünschte Produkt beschreiben kann, ist eine zusätzliche Möglichkeit der Eingabe.

Bei der Suche nach Produkten gibt es gewöhnlicherweise drei Situationen:

- Die/der BenutzerIn weiß genau, was sie/er möchte.
- Die/der BenutzerIn hat einen Wunsch, weiß aber nicht den Namen des Produkts.
- Die/der BenutzerIn weiß nicht genau, nach was sie/er sucht.

Um nun für alle diese Situationen brauchbare Empfehlungen abgeben zu können, benötigt das System ein gewisses Maß an Wissen, welches bei CBR-Empfehlungssystemen in der Fall-Datenbank gespeichert ist. Ein Fall kann Informationen über das empfohlene Produkt, über die Person, die die Empfehlung bekommen hat, über die Sitzung („session“), in der das Produkt empfohlen wurde und über Bewertungen der Empfehlung enthalten. Formal kann eine Fall-Datenbank so beschrieben werden:

$$CB \subseteq X \times U \times S \times E$$

$$\begin{aligned} &CB : Falldatenbank \\ &X : Modell\ eines\ Produktes \\ &U : Modell\ einer\ Benutzerin\ bzw.\ eines\ Benutzers \\ &S : Modell\ einer\ Sitzung \\ &E : Modell\ einer\ Evaluierung \end{aligned} \tag{15}$$

wobei *X* das Modell eines Produktes oder einer Ressource darstellt, *U* das Modell einer Benutzerin oder eines Benutzers, *S* das Modell einer Sitzung und *E* das Modell einer Evaluierung der Nützlichkeit der Empfehlung.
Ein allgemeiner Fall $c = (x, u, s, e) \in CB$ in einem allgemeinen CBR-Empfehlungssystem besteht aus vier (optionalen) Elementen (x, u, s, e) die jeweils Instanzen der Räume *X, U, S, E* abbilden. Jedes CBR-Empfehlungssystem adaptiert nun seine eigene Modellierung für die *X, U, S, E* Räume. Diese Räume können nun leer sein oder aus Vektoren, textuellen Listen, usw. bestehen.

Im folgenden wird eine kurze Betrachtung der einzelnen Modelle gegeben:

- **Modell des Produkts („content model“) *X*:** Das Produktmodell beschreibt das Produkt, welches vorgeschlagen wurde beziehungsweise vorgeschlagen werden soll. Die Beschreibung erfolgt gewöhnlicherweise durch Attributsvektoren, die das Produkt charakterisieren.

- **Modell der Benutzerin und des Benutzers („user model“) *U*:** Dieses Modell beinhaltet üblicherweise persönliche Daten, wie den Namen, die Adresse, oder aber auch Informationen zu der Nutzung des Systems durch diese eine Person. Auch die bevorzugten Produkte der Person können in diesem Modell abgebildet werden.

- **Modell der Sitzung („session model“) *S*:** Im Sitzungsmodell werden die Daten, wie es zur Empfehlung gekommen ist, gespeichert. Es kann der Prozess, der zur Lösung des Problems geführt hat, abgebildet werden. Als Anwendungsfall könnten alle Anfragen der Person und alle daraufhin ausgewählten Produkte in einer Sitzung gespeichert werden.

- **Modell der Evaluierung („evaluation model“) *E*:** Dieses Modell beschreibt den Erfolg der Empfehlung, also ob die Empfehlung nützlich war oder nicht. Dies kann beispielsweise a-posteriori mit der Bewertung der Ergebnisse erfolgen.

Jede Implementierung eines CBR-Empfehlungssystems verwendet andere Umsetzungen einer Fallmodellierung. Es gibt Systeme, die nur das Produktmodell verwenden, wieder andere konzentrieren sich auf die Auswertung von Sitzungen.

2.7 Beispiele für Empfehlungssysteme

2.7.1 Amazon

Amazon[3] ist einer der bekanntesten Online Shops unserer Zeit. In den letzten Jahren wurde das Angebot an Waren immer weiter ausgebaut, von Büchern bis hin zu Lebensmitteln.

Die Grundlage der Empfehlungen bei Amazon bildet eine „offline item-to-item“ Distanztabelle, die aufgrund des Benutzerverhaltens (Kaufverhaltens) vorausberechnet wird. Dies ist notwendig, um die riesigen Datenmengen verwalten zu können und in kurzer Zeit Ähnlichkeitsberechnung vornehmen zu können. Für die Distanzermittlung kommt das Cosinus-Ähnlichkeitsmaß (siehe Kapitel 2.4.1) zum Einsatz. Nachdem die Vektoren aller Elemente vorausberechnet wurden, hängt die Berechnung nur mehr von den Elementen ab, die in letzter Zeit besucht wurden [14 S. 95].

Amazon erstellt Empfehlungen, die auf den Aktionen der Benutzerinnen und Benutzer basieren. Mit Hilfe eines Benutzerkontos (das bei dieser Seite von jedem angelegt wird, der etwas bestellen möchte) können statistische Daten über die/den BenutzerIn gesammelt werden. Um das zu gewährleisten, bleibt man mit Hilfe von Cookies automatisch angemeldet. Somit wird die Wahrscheinlichkeit erhöht, dass man Daten über die Person sammeln kann.
Amazon verwendet noch weitere Techniken, um die Empfehlungen zu verbessern. Die Bereiche Vertrauen und Transparenz (siehe Kapitel 2.4.6) werden dadurch unterstützt, dass man sich den Weg, der zur Empfehlung geführt hat, ansehen kann. In weiterer Folge kann man sogar die Gewichtung verändern und so Einfluss auf die Erstellung der Empfehlungen nehmen (Abbildung 6).
Zusätzlich gibt es noch Bewertungen und Reviews durch die BenutzerInnen selbst.

Abbildung 6: Amazon: Empfehlungen mit Möglichkeit der Gewichtung

3 http://www.amazon.de (Abruf am 30.06.2011)

2.7.2 MovieLens

MovieLens[4] ist ein Empfehlungssystem für Filme. Zu Beginn muss man sich auf der Webseite anmelden und mindestens 15 Filme bewerten (siehe Abbildung 7). Auf Grund der großen Bandbreite der Datenmenge, die Filme gehen zurück bis in die 40er Jahre, ist es ein relativ langwieriger Prozess, da man kaum Filme findet, die man gesehen hat. Eine weitere Hürde ist die Sprache, denn es kann natürlich vorkommen, dass man den englischen Originaltitel noch nicht gehört hat und so einen Film übersieht, den man sehr wohl gesehen hat.
Je mehr Filme man bewertet, desto bessere Vorhersagen kann das System treffen, wie man einen ungesehenen Film bewerten würde.

Abbildung 7: MovieLens: Einstiegsseite zur Bildung des Benutzerprofiles

Das MovieLens-System ist ein hybrides System. Zum einen werden verwandte BenutzerInnen mit dem Pearson-Korrelationskoeffizienten (siehe Kapitel 2.4.2) ermittelt, zum anderen werden die Filme an Hand der Filmbeschreibungen verglichen.
Um das Problem der Spärlichkeit der Bewertungen zu umgehen, verwendet dieses System sogenannte „Bots“. Das sind Programme, die sich wie Personen verhalten und Filme bewerten. Diese „Bots“ agieren nach Genres: Ein „Bot“ für Kinderfilme wird einem Animationsfilm eine gute Bewertung geben, ein Horrorfilm erhält von diesem „Bot“ dafür keine Punkte [14 S.154].
Diese Art der Gewinnung von Bewertungen ist eine Alternative zu der bereits vorgestellten Methode des „default voting“ (siehe Kapitel 2.5.1.2).

4 http://www.movielens.org (Abruf am 30.06.2011)

3 Datenanalyse

Dieser Teil der Arbeit beschreibt die Auswertung der vorhandenen Daten einer online publizierten Zeitschrift. Die Daten werden dahingehend analysiert, ob sich Regelmäßigkeiten im Benutzerverhalten finden lassen, die für die Erstellung von Empfehlungen verwendet werden können. In den folgenden Kapiteln wird der Einfluss des Inhaltsverzeichnisses auf die Aktionen der BenutzerInnen analysiert. Des weiteren werden die von den Benutzerinnen und Benutzern erzeugten Pfade (ein Pfad ist eine Serie von Downloads von PDF-Dateien, die im Laufe eines Besuches erzeugt wird) auf ihre Aussagekraft für eine Empfehlung untersucht.

3.1 Werkzeuge und Datenbasis

Der folgende Abschnitt beschreibt den Vorgang der Analyse des vorhandenen Datenmaterials und die dabei verwendeten Werkzeuge.

3.1.1 Open Journal Systems

Open Journal Systems (OJS) ist ein Zeitschriftenprojekt des „Public Knowledge Projektes“[5], eines gemeinsamen Projektes der „Faculty of Education at the University of British Columbia“, der „Simon Fraser University Library“, der „School of Education at Stanford University“, und dem „Canadian Centre for Studies in Publishing at Simon Fraser University“.
Das Ziel ist es ein einfach zugängliches System zur Veröffentlichung von Zeitschriften zu entwickeln, das die Möglichkeit von Peer-Reviews gestattet, also die Bewertung des Artikels durch Experten desselben Faches („peer“ Englisch für „der Ebenbürtige“, „der Gleichgestellte“). Hinzu kommt die Möglichkeit, das System mit Zusatzprogrammen, so genannten „Plugins“ zu erweitern.
Die Software ist Open-Source, daher frei nutzbar und kann auf jedem aktuellen Webserver installiert werden. Mit einer Installation ist es möglich mehrere Zeitschriften zu verwalten. Das System befindet sich weltweit im Einsatz, laut der Webseite des „Public Knowledge Projektes“ waren im Juli 2011 ungefähr 10000 Instanzen bekannt[6].

3.1.2 Piwik

Piwik[7] ist ein frei verwendbares Statistik- und Analyseprogramm. Es zeichnet die Vorgänge auf einer Webseite auf und bietet die Möglichkeit Besucherströme zu analysieren. Angefangen von Zugriffsstatistiken mit Informationen über die BenutzerInnen (gespeichert als IP-Adressen) bis hin zu visuellen Darstellungen bietet das Programm viele Funktionen. Die Daten werden dabei in einer Datenbank gespeichert. Da das Programm leider keine Funktion zum Export von sämtlichen in einer Periode aufgezeichneten Daten hatte, mussten die benötigten Daten direkt aus der Datenbank des Programms bezogen werden.

5 http://pkp.sfu.ca/?q=ojs (Abruf: 12.07.2011)
6 http://pkp.sfu.ca/ojs-user-numbers (Abruf am 28.11.2011)
7 http://piwik.org/ (Abruf: 12.07.2011)

3.1.3 Ruby on Rails

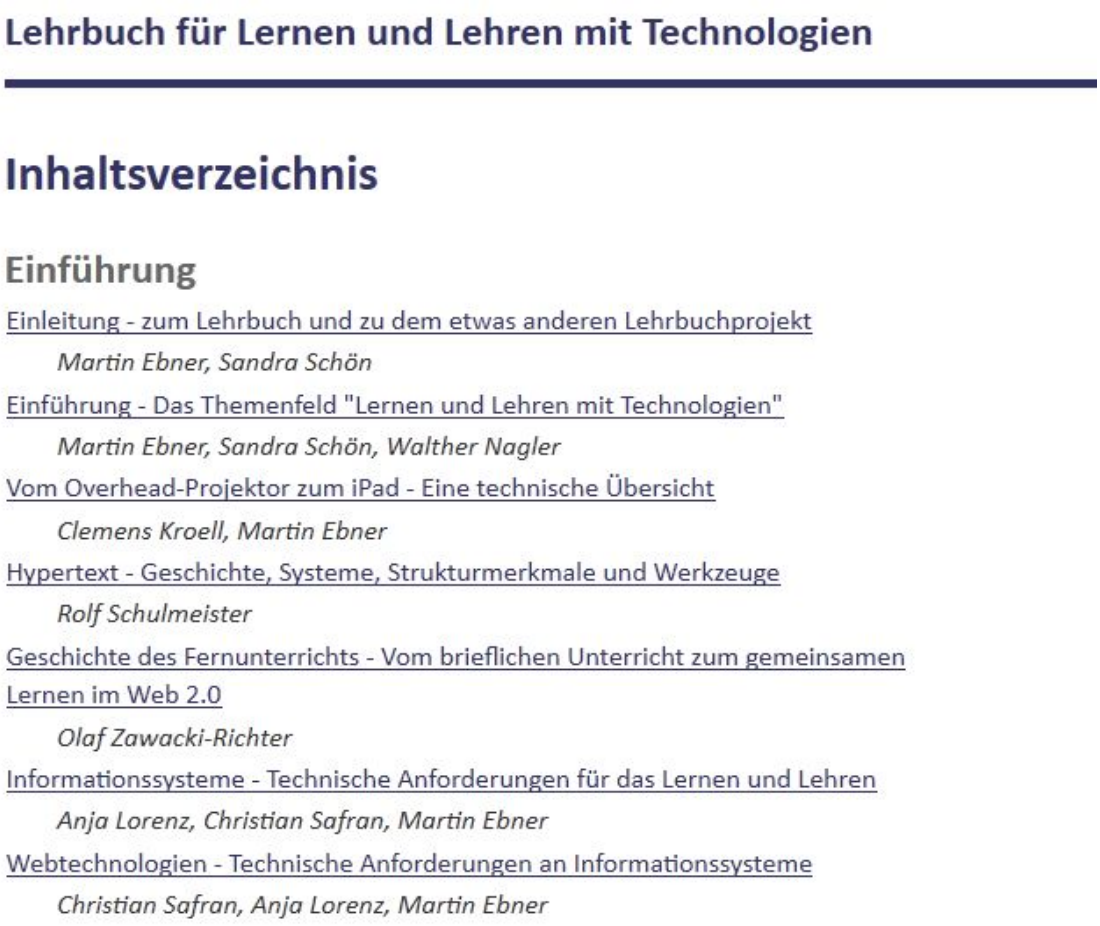
Ruby on Rails[8] ist eine Programmiersprache, die mit dem MVC („Model-View-Controller“) Prinzip arbeitet. Daher ist sie sehr gut zum Entwerfen von Webseiten, die die Daten aus einer Datenbank beziehen, geeignet.
Das „Model“ beschreibt das Modell der Datenbank und die Beziehungen der Daten untereinander, die „View“ beschreibt die Anzeige der ausgewählten Daten, und der „Controller“ stellt die Verarbeitung der Daten dar, damit sie in der „View“ angezeigt werden können.
Das Arbeiten mit dieser Struktur erfordert ein wenig Erfahrung und Einarbeitung, aber wenn man sich an das Konzept gewöhnt hat, hat es den enormen Vorteil, dass man sehr schnell ein funktionierendes System erstellen kann. Durch das Modell der Datenbank vermeidet man das Schreiben von expliziten Datenbank-Statements. Die einzige Anforderung ist, dass bestimme Namenskonventionen eingehalten werden. (Tabellen mit Daten tragen die Mehrzahlbezeichnung: „customers“, „users“; der Primärschlüssel hat die Bezeichnung „id“; Das Model trägt die Einzahlbezeichnung: customer, user).
Hat man nun diese Model-View-Controller-Struktur für eine Webseite angelegt, kann man die Daten einer Datenbank sehr effizient analysieren und darstellen. Einer der größten Vorteile von Rails liegt darin, dass sich die über die Modelle geladenen Daten wie Objekte verhalten und somit sehr gut weiterverarbeitet werden können.

3.1.4 Erzeugung der Datenbasis

Für die Analyse des Benutzerverhaltens wurden die Daten der Webseite l3t.tugraz.at[9] verwendet. Die Aktionen der BenutzerInnen auf der Webseite wurden mit der Gratissoftware Piwik aufgezeichnet. L3T ist ein Lehrbuch, welches als Buch mit einzelnen Kapiteln mit Hilfe einer Open-Journal-Systems-Installation veröffentlicht wird. Die Kapitel des Buches sind als Artikel der Zeitschrift verfügbar. Die einzige Ordnung der Artikel ist das Inhaltsverzeichnis (Abbildung 8).

Am Beginn stand die Frage, wie stark das Inhaltsverzeichnis den Weg der BenutzerInnen durch das Angebot der Artikel beeinflusst. Um dieser Frage nachzugehen, mussten die relevanten Daten aus der Sammlung von Piwik extrahiert werden.
Wie in Kapitel 3.1.2 bereits erwähnt, verfügt Piwik über keine Funktion, um die Daten über die Bewegungen einer Person auf der Webseite zu exportieren. Es gibt lediglich die Möglichkeit, 50 Besuche in eine CSV-Datei („comma separated values“ - die einzelnen Daten einer Tabelle werden in eine Textdatei geschrieben, wobei die Spalten durch Kommas eingezeichnet werden) zu exportieren. Angesichts der Menge von ca. 20.000 aufgezeichneten Besuchen ist dies eine denkbar unpraktikable Lösung.
Aus diesem Grund mussten die Daten direkt aus den Tabellen der Datenbank gewonnen werden. Die Wahl des Werkzeugs fiel auf Ruby on Rails, weil die Erzeugung eines Modells der benötigten Tabellen effizient möglich ist.

8 http://rubyonrails.org/ (Abruf 12.07.2011)
9 http://l3t.tugraz.at/index.php/LehrbuchEbner10 (Abruf 13.07.2011)



Lehrbuch für Lernen und Lehren mit Technologien

Inhaltsverzeichnis

Einführung

Einleitung - zum Lehrbuch und zu dem etwas anderen Lehrbuchprojekt
Martin Ebner, Sandra Schön

Einführung - Das Themenfeld "Lernen und Lehren mit Technologien"
Martin Ebner, Sandra Schön, Walther Nagler

Vom Overhead-Projektor zum iPad - Eine technische Übersicht
Clemens Kroell, Martin Ebner

Hypertext - Geschichte, Systeme, Strukturmerkmale und Werkzeuge
Rolf Schulmeister

Geschichte des Fernunterrichts - Vom brieflichen Unterricht zum gemeinsamen Lernen im Web 2.0
Olaf Zawacki-Richter

Informationssysteme - Technische Anforderungen für das Lernen und Lehren
Anja Lorenz, Christian Safran, Martin Ebner

Webtechnologien - Technische Anforderungen an Informationssysteme
Christian Safran, Anja Lorenz, Martin Ebner

Abbildung 8: Inhaltsverzeichnis des Projektes L3T

Drei Tabellen mussten in Rails modelliert werden. Die Tabelle „visits“ beinhaltet die Daten zu den einzelnen Besuchen wie zum Beispiel Datum, Dauer etc. Jeder Besuch besteht aus einer Anzahl an Aktionen, dies wird durch die Tabelle „actions“ ausgedrückt, welche die Daten über die durchgeführte Aktion speichert. In diesem Fall sind die angesehenen PDF-Dateien der einzelnen Artikel von großem Interesse. Die Artikel sind durch eine interne Nummerierung gekennzeichnet und gespeichert, welche auch bei den Analysen verwendet wurde. Die Einleitung beispielsweise (siehe Abbildung 8) wird intern als Nummer 89 geführt.
Verbunden werden diese beiden Tabellen mit einer „visit_actions“-Tabelle. Hier stehen die Verbindungen zwischen einem Besuch und den dazugehörigen Aktionen.
Nach der Abbildung dieser drei Tabellen (und der Bildung der Relationen zwischen den Tabellen) wurde das Modell einer Aktion dahingehend erweitert, dass erstens überprüft werden kann, ob es sich bei der Aktion um den Download eines PDF-Dokuments handelt, und zweitens, ob der Name (beziehungsweise die Nummer) des Artikels aus der Aktion extrahiert werden kann. Diese Methoden können direkt im Rails-Modell, welches die Datenbanktabelle abbildet, geschrieben werden und stehen somit für jedes Aktionsobjekt zur Verfügung.
Mit dieser Grundlage ist es möglich, aus den Daten von Piwik alle Besuche eines Zeitraumes und deren dazugehörige Aktionen zu finden und dahingehend zu filtern, ob die Aktion nun ein Download eines PDF-Dokuments war.

3.2 Auswertung

Die erste Auswertung der Daten hatte das Ziel herauszufinden, wie die Wahrscheinlichkeit der nächsten Downloads, ausgehend von einem bestimmten Artikel, aussieht. Man wählt also einen Artikel und betrachtet die Anzahl, mit der jeder andere Artikel als nächstes ausgewählt wurde. Es sollte damit empirisch die Frage beantwortet werden, welcher beziehungsweise welche Artikel die größte Wahrscheinlichkeit hat/haben als nächstes angesehen zu werden. Betrachtet man nun jeden Artikel, kann man eine Artikel-Matrix aufstellen, die in jeder Zeile die Häufigkeiten für die nächsten Artikel (ausgehend vom betrachteten Artikel) notiert.

3.2.1 Artikel-Matrix

Die Beschriftung der Matrix sind die extrahierten, internen Kapitelnummern. Das erste Kapitel hat im System beispielsweise die Nummer 89. Der Algorithmus für die Aufstellung der Matrix zählt nun alle Kapitel die im Anschluss an das Kapitel aufgerufen wurden. Dieses Zählen wird dadurch realisiert, dass alle Aktionen eines jeden Besuches einmal durchlaufen werden und die enthaltenen Aufrufe in eine Matrix eingetragen werden. Beispielsweise ist ein Pfad bei einem Besuch 89, 88, 43. Der Algorithmus sucht das Feld 89 → 88 (Zeile mit der Beschriftung 89 und Spalte mit der Beschriftung 88) in der Matrix und erhöht den Zähler um eins, als nächstes wird das Feld 88 → 43 gesucht und ebenfalls um eines erhöht.

Algorithmus im Pseudocode:

```
Für alle Besuche:
        Für alle Aktionen eines Besuchs:
                Wenn es mehr als eine Aktion gibt:
                        Wenn Aktion == Artikel
                                Trage Aufruf in die Matrix ein (aktueller Artikel, nächster Artikel)
                        end
                end
        end
end
```

Die Methode, die den Aufruf in die Matrix einträgt, überprüft als erstes, ob auch zwei Artikel übergeben werden. Wurde nur ein Artikel übergeben (der letzte Artikel einer Serie hat ja keinen Nachfolger), dann wird der Aufruf ignoriert. Anderenfalls wird der Aufruf in die Matrix eingetragen. Der Aufruf 89 → 88 etwa würde in der Zeile mit der Aufschrift „89“ den Zähler für die Spalte „88“ um eins erhöhen.
Für die Berechnung der prozentuellen Verteilung der Aufrufe wird auch mitgezählt, wie viele Aufrufe insgesamt erfolgen. Für diesen Zweck besitzt jede Zeile einen Zähler.

Label (TOC)	89 (1)	88 (2)	49 (3)	73 (4)	54 (5)	41 (6)	44 (7)	38 (8)	71 (9)
89 (1)		0.62	0.09	0.03	0.02	0.02	0.00	0.01	0.01
88 (2)	0.13		0.55	0.06	0.04	0.03	0.00	0.04	0.00
49 (3)	0.08	0.07		0.53	0.04	0.05	0.01	0.04	0.01
73 (4)	0.03	0.03	0.04		0.71	0.04	0.01	0.04	0.00
54 (5)	0.03	0.02	0.02	0.03		0.64	0.02	0.05	0.01
41 (6)	0.02	0.02	0.01	0.01	0.03		0.70	0.09	0.00
44 (7)	0.01			0.00		0.04		0.75	0.05
38 (8)	0.02	0.01	0.02	0.00	0.01	0.01	0.03		0.52
71 (9)	0.02		0.01	0.00		0.01	0.02	0.02	

Tabelle 1: Auszug aus der Kapitel-Matrix mit Inhaltsverzeichnis

Die vollständige Tabelle befindet sich im Anhang in Kapitel 8.1 (Tabelle 24).

In Tabelle 1 kommt die Dominanz des Inhaltsverzeichnisses gut zum Ausdruck: Jede Zeile stellt einen Ausgangspunkt, also eine angesehene PDF-Datei dar. Die Zahlen in den Spalten sind die Häufigkeiten, mit der - ausgehend vom Dokument in der Zeile - das Dokument in der Spalte als nächstes aufgerufen wurde. Als Beispiel die erste Zeile: Ausgehend vom Dokument mit der Nummer 89 wurde in 62% aller Fälle als nächstes Dokument Nummer 88 ausgewählt, Nummer 49 bereits nur mehr in 9% aller Fälle.

Da die Reihenfolge der Beschriftungen in dieser Tabelle der Reihenfolge des Inhaltsverzeichnisses entspricht, kann man sehr gut erkennen, dass die Mehrheit der BenutzerInnen dem Inhaltsverzeichnis folgt. Die höchsten Werte stehen neben der Hauptdiagonale und der Wert direkt rechts neben dem leeren Felde bezeichnet das nächste Dokument im Inhaltsverzeichnis.

Eine weitere Erkenntnis aus dieser Analyse ist, dass die gefundenen Häufigkeiten eine Vorhersage, welches Dokument wahrscheinlich als Nächstes aufgerufen wird, dahingehend beeinflussen, dass sehr oft einfach das nächste Element im Inhaltsverzeichnis empfohlen wird.

Dies führt uns zur Frage, wie sich diese Matrix verändern würde, wenn man die Häufigkeiten des Inhaltsverzeichnisses aus der Berechnung entfernt. In Tabelle 2 wurden die Aufrufe eines Dokuments bei der Zählung ignoriert, wenn das Dokument das nächste im Inhaltsverzeichnis war.

Label (TOC)	89 (1)	88 (2)	49 (3)	73 (4)	54 (5)	41 (6)	44 (7)	38 (8)	71 (9)
89 (1)		TOC	0.24	0.08	0.06	0.05	0.01	0.02	0.02
88 (2)	0.29		TOC	0.13	0.08	0.07	0.01	0.08	0.01
49 (3)	0.16	0.15		TOC	0.09	0.10	0.03	0.09	0.01
73 (4)	0.10	0.09	0.15		TOC	0.14	0.05	0.15	0.02
54 (5)	0.10	0.06	0.05	0.08		TOC	0.05	0.13	0.02
41 (6)	0.06	0.06	0.04	0.05	0.09		TOC	0.31	0.02
44 (7)	0.04			0.01		0.17		TOC	0.18
38 (8)	0.04	0.02	0.04	0.01	0.03	0.03	0.06		TOC
71 (9)	0.09		0.03	0.01		0.03	0.09	0.11	

Tabelle 2: Auszug aus Artikel-Matrix ohne Inhaltsverzeichnis

Die vollständige Tabelle befindet sich im Anhang in Kapitel 8.1 (Tabelle 25).

Die Eliminierung der direkt aufeinander folgenden Artikel (in Tabelle 2 mit TOC gekennzeichnet) führte zu einer gleichmäßigeren Verteilung der Häufigkeiten. Die höchsten Werte erreichen nun knapp die 30% Marke. Man kann erkennen, dass die größten Häufigkeiten entlang beziehungsweise rundum das Inhaltsverzeichnis auftreten. Dies trifft für die gesamte Matrix zu.
Zufolge Tabelle 2 kann folgendes angemerkt werden: Nehmen wir als Beispiel die Zeile mit der Artikelnummer 88. Die Spalte mit der Nummer 49 ist als TOC markiert, das heißt der Artikel mit der Nummer 49 folgt direkt auf den Artikel mit der Nummer 88 im Inhaltsverzeichnis (88 → 49). Die Zahlen links des TOC-Feldes beziehen sich auf die Artikel vorher im Inhaltsverzeichnis (das leere Feld ist der Artikel selbst), die Zahlen rechts von TOC auf die folgenden Artikel im Inhaltsverzeichnis. Diese Artikel sind in der Tabelle farblich gekennzeichnet: Blau für den Vorgängerartikel Nummer 89 (29%) und rot für den nachfolgenden Artikel mit der Nummer 73 (13%).

3.2.2 Pfadanalyse

Der nächste Versuch, um aus den vorhandenen Daten eine brauchbare Datenbasis zu schaffen, bestand darin, die Pfade der BenutzerInnen zu betrachten. Als theoretische Grundlage dient das Case-Based-Reasoning (Kapitel 2.6.1). Jeder Pfad einer Benutzerin oder eines Benutzers wird als Fall betrachtet. Wenn sich nun eine neue Benutzerin oder ein neuer Benutzer auf der Webseite bewegt, wird der Pfad aufgezeichnet (soweit es die Verwendung der PDF-Dokumente betrifft) und mit der Datenbasis verglichen, ob es bereits ähnliche Pfade gibt. Daraus können dann Empfehlungen generiert werden, welche Dokumente als nächstes interessant sind.

Im ersten Schritt werden die Pfade aus den Daten von Piwik extrahiert. Der verwendete Algorithmus iteriert in ähnlicher Art und Weise wie in der ersten Analyse über das vorhandene Datenmaterial. Es wird dieses Mal aber keine Matrix erzeugt, sondern ein Hash (eine Liste, deren Elemente aus Paaren bestehen, einem Schlüssel und einem Wert) generiert. Die Elemente des Hash sind paarweise Daten, der komplette Pfad und die Anzahl wie oft er vorgekommen ist.

Algorithmus im Pseudocode:

Für alle Besuche:
    Für alle Aktionen eines Besuchs:
        Wenn es mehr als eine Aktion gibt:
            Wenn Aktion == Artikel
                Füge Artikel zum Pfad hinzu
            end
        end
    end
    Trage den Pfad in den Hash ein, falls er noch nicht existiert
    Erhöhe den Zähler des Pfades
end

Mit diesem Algorithmus wurden nun um die 1700 Pfade identifiziert (ein Auszug aus den gefundenen Pfaden ist in Abbildung 9 dargestellt).

```
607  Path: |49|73|54 Number: 1
608  Path: |28|34|50|72|85|67|33|68|29|47|58|40|36|48|19|46|31|35|43 Number: 2
609  Path: |62|79|62 Number: 4
610  Path: |71|38|89|63|57|58|20|43|28|50|85|67|33|35 Number: 1
611  Path: |18|50 Number: 2
612  Path: |37|63|65|20|79|66|70|62|24|32|28|34|50|72|85|67|33|68|29|47|58|40|36|48|19|
613  Path: |88|49|73|41 Number: 1
614  Path: |45|22|61|51 Number: 1
615  Path: 28 Number: 33
616  Path: |43|35|31|46|19|48|36|40|58|47|29|68|33|85|72|50|34|28|32|24|62|70|66|79|20|
617  Path: |18|51 Number: 2
618  Path: |35|79 Number: 1
619  Path: |40|88|41|44|39|22|60|64|63|66|62|28|67|40|36|48 Number: 1
620  Path: |54|41|44|61|38|71 Number: 1
621  Path: |88|49|73|54|41|44|38|71|18|45|39|57|22|60|64|61|51|74|37|63|65|20|79|66|70|
622  Path: |18|89|19|85|29 Number: 1
```

Abbildung 9: Auszug aus den gefundenen Pfaden

Jede Zeile in Abbildung 9 steht für einen spezifischen Pfad. Beispielsweise Zeile 609 bildet eine Reihenfolge von drei Dokumenten: Nummer 62, Nummer 79 und Nummer 62 (62 → 79 → 62). In diesem Fall ist die/der BenutzerIn wieder zur Ausgangsseite zurückgekehrt. „Number: 4“ bedeutet, dass dieser Pfad insgesamt vier Mal im Zuge der Analyse vorgekommen ist.

Aus diesen Daten wurde nun eine neue Datenbank erstellt. Diese besteht aus drei Tabellen:

- Die Tabelle „traces“ beschreibt die einzelnen Pfade, deren Tiefe und Häufigkeit.
- Die Tabelle „labels“ beschreibt die Nummern der Dokumente und an welcher Stelle im Pfad sie vorkommen.
- Als Verbindung fungiert die „trace_lables“ Tabelle, in der festgehalten wird, welches Label zu welchem Pfad gehört.

Mit Hilfe dieser neuen Datenbasis kann nun untersucht werden, wie viele Pfade von einem Dokument ausgehen, oder wie die durchschnittliche Länge der Pfade (ausgehend von einem Link) aussieht.

Eine interessante Anwendung dieser Datenbasis ist der virtuelle Durchlauf durch die möglichen Pfade (Abbildung 10). Es wird eine Webseite erstellt, auf der man nach Auswahl des Startdokuments das nächste Dokument aus der Liste der verfügbaren Dokumente auswählt. Diese Liste wird durch die Nummern der Dokumente repräsentiert und ist in der Reihenfolge des Inhaltsverzeichnisses angeordnet. Wählt man nun einen Link aus, wird die Nummer des Dokumentes an den aktuellen Pfad angefügt und abgeschickt. Nun werden alle Pfade durchsucht und diejenigen, bei denen Übereinstimmungen gefunden wurden, angezeigt. Zusätzlich wird die Häufigkeit der nächsten Dokumente in diesen Pfaden angezeigt.

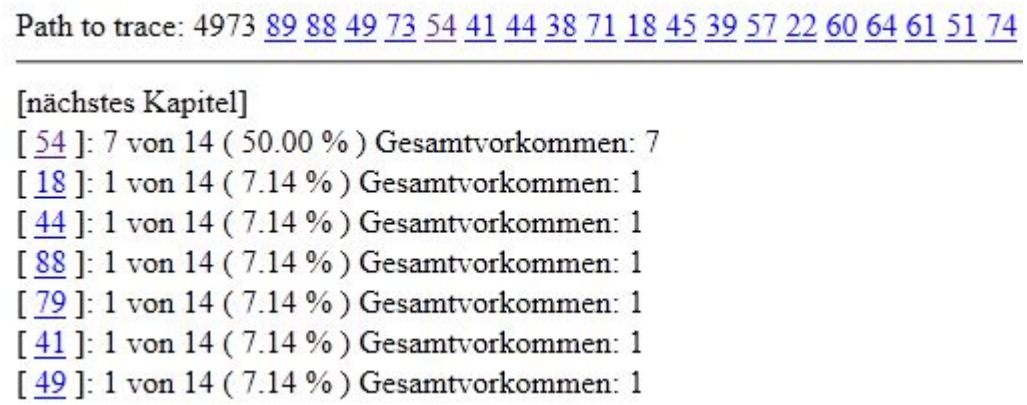


Abbildung 10: Pfadverfolgung

Abbildung 10 zeigt die Ausgabe der Pfadverfolgung. In der ersten Zeile ist der zu verfolgende Pfad notiert (4973 bedeutet den Teilpfad 49 → 73, also alle Pfade, die bei dem Dokument mit der Nummer 49 beginnen und als zweites Dokument die Nummer 73 beinhalten). Daneben stehen die Links für die Auswahl des nächsten Dokuments in der Reihenfolge des Inhaltsverzeichnisses.
Darunter wird die Auswertung für den gesuchten Pfad angezeigt. In den eckigen Klammern stehen die möglichen nächsten Kapitelnummern. Das sind jene Nummern, die in bekannten (in der Datenbank gespeicherten) Pfaden an nächster Stelle des Pfades vorkommen. In unserem Fall sind das alle Kapitelnummern, die in Pfaden, welche mit 49 → 73 beginnen, an dritter Stelle stehen. Für jede dieser Möglichkeiten wird nun angezeigt, wie oft sie vorkommen.
In unserem Beispiel wäre das Dokument mit der größten Häufigkeit die Nummer 54 (Teilpfad: 49 → 73 → 54), denn 7 von 14 unterschiedlichen Pfaden beginnen mit dieser Dokumentenfolge. Das Gesamtvorkommen notiert die Gesamtanzahl an Pfaden, die mit dem gesuchten Teilpfad beginnen. In Abbildung 10 ist das Gesamtvorkommen mit der Anzahl an gefundenen, unterschiedlichen Pfaden identisch, das heißt jeder gefundene Pfad kommt nur einmal vor.

Das Inhaltsverzeichnis gibt auch hier wieder die größten Häufigkeiten an, wie auch Abbildung 11 noch einmal verdeutlicht.

Path to trace: 898849 89 88 49 73 54 41 44 38 71 18 45 39 57 22 60 64 61 51 74

[nächstes Kapitel]
[73]: 111 von 138 (80.43 %) Gesamtvorkommen: 175
[88]: 8 von 138 (5.80 %) Gesamtvorkommen: 8
[89]: 6 von 138 (4.35 %) Gesamtvorkommen: 6
[54]: 3 von 138 (2.17 %) Gesamtvorkommen: 3
[18]: 2 von 138 (1.45 %) Gesamtvorkommen: 2
[38]: 2 von 138 (1.45 %) Gesamtvorkommen: 2
[41]: 2 von 138 (1.45 %) Gesamtvorkommen: 2
[68]: 1 von 138 (0.72 %) Gesamtvorkommen: 1
[66]: 1 von 138 (0.72 %) Gesamtvorkommen: 1
[39]: 1 von 138 (0.72 %) Gesamtvorkommen: 1

Abbildung 11: Pfadverfolgung Inhaltsverzeichnis

Der gesuchte Teilpfad (89 → 88 → 49) in Abbildung 11 besteht aus den ersten drei Kapitelnummern des Inhaltsverzeichnisses. Das Kapitel mit der größten Häufigkeit (80,43%) ist nun eindeutig das folgende Kapitel laut Inhaltsverzeichnis (Nummer 73). 111 der 138 gefundenen, unterschiedlichen Pfade weisen dieses Dokument als nächstes Dokument aus. Das Gesamtvorkommen der Pfade (175) unterscheidet sich hier von der Anzahl der unterschiedlichen Pfade (111), das bedeutet, dass manche Pfade öfter vorgekommen sind. Mit anderen Worten, verschiedene BenutzerInnen haben denselben Weg gewählt und so identische Pfade erstellt. Es erscheint logisch, dass dies entlang des Inhaltsverzeichnisses am öftesten auftritt.

Eine weitere Analyse ist die graphische Darstellung der Anzahl aller einzigartigen Pfade, die ihren Ursprung in einem Dokument haben. Abbildung 12 zeigt die Anzahl dieser Pfade, geordnet nach dem Inhaltsverzeichnis.

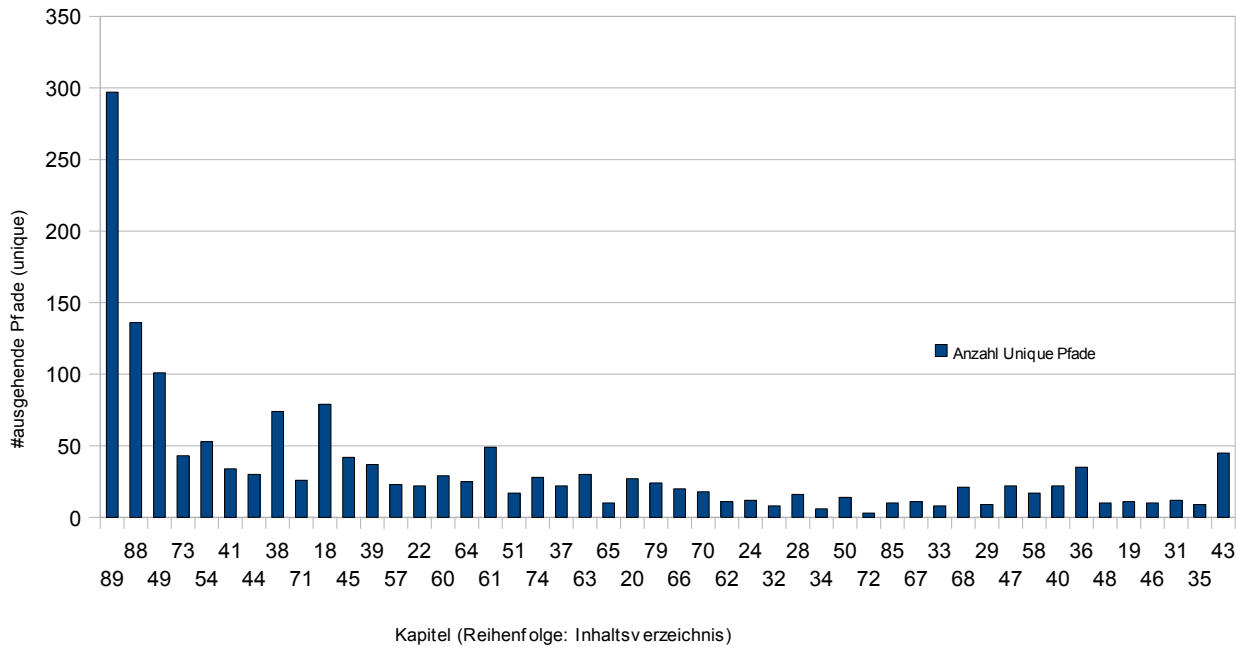


Abbildung 12: Vergleich der einzigartigen Pfade

Die Grafik (Abbildung 12) zeigt die Dominanz der ersten drei Kapitel des Inhaltsverzeichnisses. Diese drei Links sind der Ausgangspunkt der meisten Pfade. Links, die sich an späterer Stelle im Inhaltsverzeichnis befinden, werden seltener aufgerufen und sind somit auch viel seltener Ausgangspunkt eines Pfades.

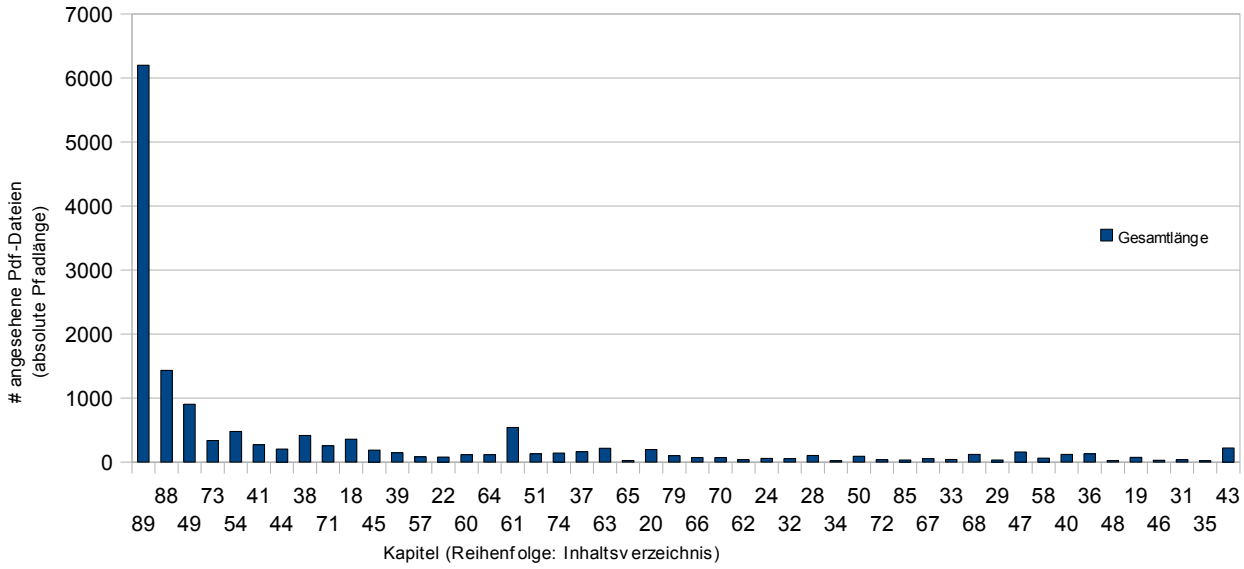


Abbildung 13: Vergleich der absoluten Pfadlängen

Abbildung 13 zeigt die Verteilung der absoluten Pfadlängen, also die Anzahl aller angesehenen PDF-Dateien in allen Pfaden. Der hohe Wert der ersten Seite lässt sich dadurch erklären, dass es erstens viele Pfade gibt, die mit der ersten Seite beginnen, und zweitens, dass diese Pfade eine hohe durchschnittliche Länge aufweisen (Abbildung 14) und drittens, dass mehrere BenutzerInnen denselben Pfad gewählt haben. Aus diesen Tatsachen resultiert dieses Ungleichgewicht zugunsten des ersten auswählbaren Dokuments im Inhaltsverzeichnis (siehe Abbildung 8).

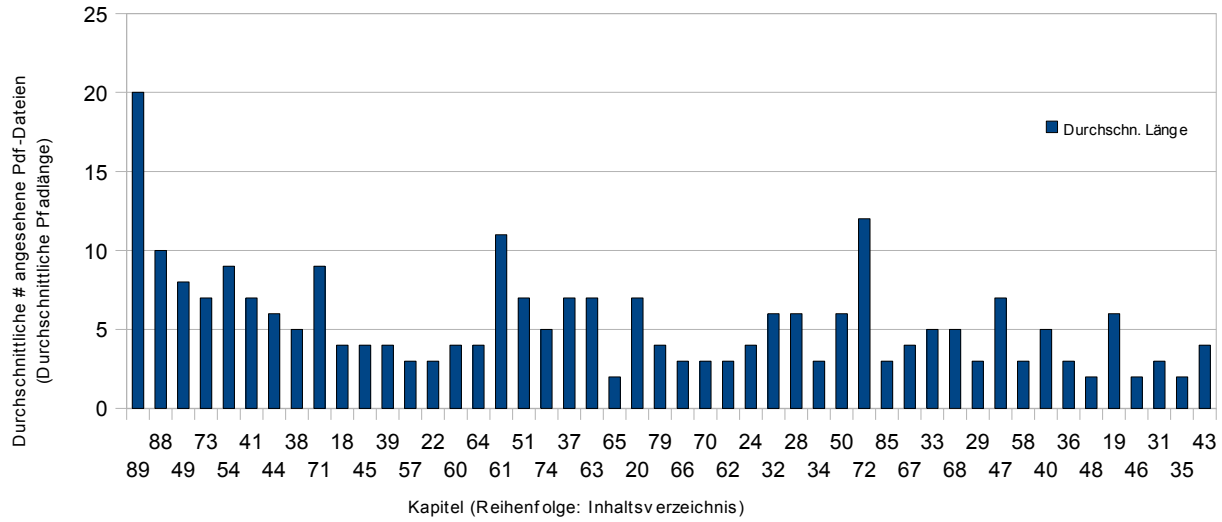


Abbildung 14: Vergleich der durchschnittlichen Pfadlängen

Der Vergleich der durchschnittlichen Pfadlängen, also die Anzahl der angesehenen PDF-Dateien aller Pfade (ausgehend von einem Dokument) geteilt durch die Anzahl der Pfade (ausgehend vom selben Dokument), zeigt wieder das starke erste Kapitel. Aber man kann erkennen, dass auch von Dokumenten, die weiter hinten im Inhaltsverzeichnis zu finden sind, trotzdem längere Pfade möglich sind. Diese Grafik muss in Relation zur nächsten Grafik (Abbildung 15) gesehen werden. Diese zeigt die Verteilung der Anzahl aller Pfade ausgehend von einem Dokument. Im Gegensatz zu Abbildung 12 werden hier alle Pfade gezählt, also auch jene, die öfter vorkommen.

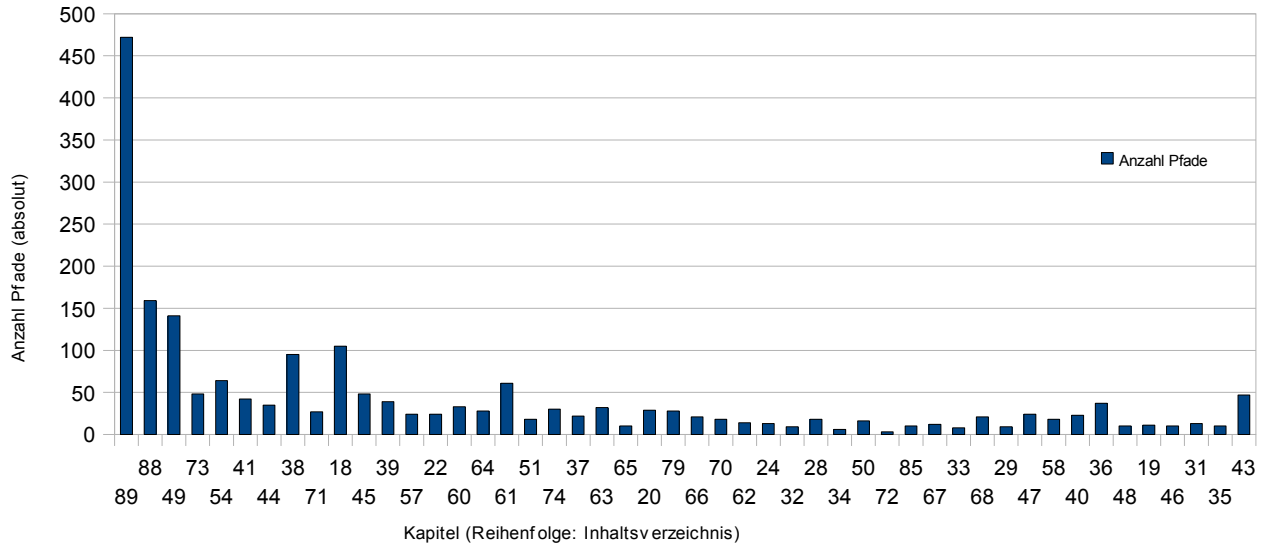


Abbildung 15: Vergleich der Anzahl der Pfade

Diese Grafik ähnelt stark der Abbildung 12 (Vergleich der einzigartigen Pfade), vor allem aus dem Grund, dass die Dokumente, die weiter hinten im Inhaltsverzeichnis zu finden sind, kaum der Ausgang für identische Pfade sind, während eben das Inhaltsverzeichnis ein starker Pfad ist, der auch von den Benutzerinnen und Benutzern verfolgt wird. Somit erzeugen die ersten Links auch die meisten identischen Pfade.

3.3 Schlussfolgerungen

Aus diesen Analysen ist ersichtlich, dass Empfehlungen generiert werden müssen, die sich vom Inhaltsverzeichnis abheben, beziehungsweise dass das bisherige Benutzerverhalten nur bedingt für die Voraussage geeignet ist. Die in Kapitel 3.2.1 gefundene Matrix zeigt die Dominanz einer vorgegebenen Navigationsstruktur.

Die Auswertung von Benutzerpfaden in Kapitel 3.2.2 kann eine Grundlage oder eine Ergänzung für die Erstellung von Empfehlungen sein. Man muss aber darauf achten, ob die berücksichtigten Pfade einer vorgegebenen Struktur folgen und die daraus erstellten Empfehlungen somit an „Überraschungsmoment“ verlieren. Denn falls eine Benutzerin oder ein Benutzer einen starken Pfad verfolgt (zum Beispiel erstes Dokument → zweites Dokument im Inhaltsverzeichnis), wäre das System nicht in der Lage abweichende Empfehlungen zu geben, da der Pfad, der durch die Struktur vorgegeben wird, unverhältnismäßig stark ist. Als unerwünschter Nebeneffekt wird dieser Pfad dadurch auch noch erneut gestärkt.

Eine Datenbank, welche die Bewegungen der BenutzerInnen aufzeichnet, kann durchaus als eine Grundlage für ein System, welches mit einem Collaborative-Filtering-Algorithmus arbeitet, verwendet werden. Um dies zu verwirklichen, ist es notwendig, einzelne BenutzerInnen zu identifizieren und in weiterer Folge durch die Analyse ihres Verhaltens einen Vergleich mit ihrer Nachbarschaft vorzunehmen. Dieses System kann jedoch nur als Zusatzsystem verwendet werden, da neue BenutzerInnen, die noch keine Aktionen durchgeführt haben, nicht mit anderen Benutzerinnen und Benutzern verglichen werden können. Daher benötigt man ein Primärsystem (im Fall von Textartikeln ein Content-Based-Filtering-System), welches die grundlegenden Empfehlungen erstellt, die dann gegebenenfalls vom Sekundärsystem (in diesem Fall ein System mit Collaborative-Filtering auf der Grundlage der verwendeten Elemente der BenutzerInnen) ersetzt oder erweitert werden.

4 Gegenüberstellung und Auswahl der Algorithmen

Im folgenden Abschnitt wird ein Überblick über die in Frage kommenden Algorithmen mit ihren Vor- und Nachteilen vorgenommen. Wie schon in der Analyse angeführt, ist es am sinnvollsten, als Primärsystem ein Content-Based-Filtering-System zu verwenden, da dieses unabhängig von den Aktionen der BenutzerInnen Empfehlungen erstellen kann.
Als Sekundärsystem soll ein Collaborative-Filtering-System verwendet werden, um die Empfehlungen des Primärsystems zu verbessern und zu erweitern.

4.1 Primärsystem – Content-Based-Filtering

4.1.1 Tagging

Tagging („Tag“: englisch für Kennzeichnung, Schild) bezeichnet den Vorgang des Kennzeichnens von Ressourcen mit einer Beschriftung. Idealerweise wird diese Kennzeichnung von Expertinnen oder Experten vorgenommen, die möglichst repräsentative Tags auswählen. Für die Erstellung der Empfehlungen können dann diese Tags verglichen werden. Eine identische Kennzeichnung zweier Texte signalisiert die Zugehörigkeit zu zusammengehörigen Themen.
Als Hilfestellung bei der Kennzeichnung kann ein Algorithmus automatisch Tags aus dem Text generieren und diese der Expertin beziehungsweise dem Experten vorschlagen. Solche Algorithmen arbeiten mit statistischen Methoden, wie zum Beispiel der Worthäufigkeit: Es werden die häufigsten Wörter im Text vorgeschlagen. Dabei ist zu beachten, dass sogenannte „Stopp-Wörter“ aus dem Text entfernt wurden, bevor der Algorithmus den Text bearbeitet. „Stopp-Wörter“ sind Wörter, die sehr häufig im Sprachgebrauch vorkommen, aber kaum inhaltliche Relevanz besitzen (zum Beispiel: in, auf, alle, als, zu etc.). Eine Liste für deutsche Stopp-Wörter kann hier gefunden werden: http://solariz.de/wp-content/files/stopwords.txt[10]

4.1.2 TF-IDF

TF-IDF steht für „Term Frequency – Inverse Document Frequency“ und bezeichnet ein statistisches Verfahren um die wichtigen Wörter eines Textes zu finden. Bei diesem Algorithmus wird die Wichtigkeit eines Wortes mit der Berechnung „Term Frequency x Inverse Document Frequency“ dargestellt. „Term Frequency“ steht für die Häufigkeit eines Wortes innerhalb eines Textes und „Inverse Document Frequency“ bezeichnet die inverse Häufigkeit des Wortes in allen Texten. Wörter, die in vielen Texten vorkommen, wie zum Beispiel Artikel oder Pronomen, sollen so als weniger wichtig eingestuft werden. Somit erhalten Wörter, die eine hohe Häufigkeit innerhalb des Textes aufweisen („Term Frequency“) und gleichzeitig in wenigen Dokumenten vorkommen („Inverse Document Frequency“) ein hohes Gewicht. Die Berechnung erfolgt durch die Multiplikation der beiden Terme [21].
Eine formale Beschreibung der Berechnung findet sich bei Klahod [14, S. 45].

10 http://solariz.de/wp-content/files/stopwords.txt (Abruf am 3.8.2011)

4.1.3 LSA

LSA steht für „Latent Semantic Analysis“. Bei diesem Verfahren wird zuerst eine Matrix gebildet, die die einzelnen Wörter mit den Texten in Beziehung setzt. Die Wörter bilden die Zeilen der Matrix. In den Spalten wird notiert, in welchem Textdokument das Wort vorgekommen ist. Als Resultat erhält man eine sehr große und spärlich besetzte Matrix. Um nun eine Matrix mit größerer Aussagekraft zu erhalten, wendet man ein Dimensionsreduktionsverfahren an, die sogenannte „Single Value Decomposition“, kurz SVD [14].
Dabei wird die Matrix auf *k* Dimensionen reduziert. Die optimale Größe von *k* kann nur durch empirische Versuche festgelegt werden. Da das Grundprinzip von LSA nicht darin liegt, die Ausgangsdaten exakt abzubilden, sondern die zu Grunde liegenden Relationen aufzudecken, besteht die optimale Wahl von *k* nicht in der besten Abbildung der Ausgangsdaten. Es müssen andere Qualitätskriterien eingeführt werden, wie zum Beispiel das Abschneiden des Algorithmus bei einem Synonymtest [16].
LSA betrachtet nun nicht die Häufigkeiten von einzelnen Termen in Dokumenten, sondern konzentriert sich auf das Aufspüren von Konzepten. Als Beispiel nehmen wir die Wörter „Auto“, „Fahrzeug“, „Fahrer“ und „Elefant“. „Auto“ und „Fahrzeug“ sind Synonyme, „Fahrer“ gehört zum gleichen Konzept und „Elefant“ ist in seiner konzeptuellen Bedeutung weit von den anderen Wörtern entfernt. Der auf *k* Dimensionen reduzierte Raum kann nun diese internen Beziehungen zwischen den einzelnen Konzepten abbilden. Grob gesagt werden die Wörter „Auto“ und „Fahrzeug“ sehr oft gemeinsam mit denselben Wörtern in Texten vorkommen (zum Beispiel mit: „Motor“, „Marke“, „KFZ“, „Hersteller“, „PS“, „Karosserie“, „Kombi“) und somit ähnliche Repräsentationen im *k* – Raum besitzen. Die Repräsentation für das Wort „Fahrer“ wird sich zu einem geringeren Grad überlappen und das Wort „Elefant“ wird hingegen eine gänzlich unterschiedliche Repräsentation besitzen [3].
Nach der Dimensionsreduktion kann nun die Korrelation der Dokumente untereinander berechnet werden. Dabei werden die Spalten der Matrix als Vektoren betrachtet, zwischen denen dann die Ähnlichkeit berechnet wird. Die Korrelation zwischen zwei Vektoren kann beispielsweise durch das Kreuzprodukt der Vektoren, geteilt durch das Produkt der Längen der Vektoren, bestimmt werden (siehe Kapitel 2.4.1).

4.1.4 Stemming

Als „Stemming“ wird ein Verfahren bezeichnet, mit dessen Hilfe der Kern einer Wortfamilie gefunden werden kann. Der Kern einer Wortfamilie ist üblicherweise kein existierendes Wort im Sprachgebrauch, denn er soll die größtmögliche Menge an abgewandelten Wörtern auf ein Mindestmaß reduzieren und trotzdem noch eindeutig sein. Dieses Verfahren ist ein nützlicher Vorverarbeitungsschritt, um die Wortmenge zu reduzieren und um zusammengehörige Wörter zusammenzufassen. Es gibt verschiedene Ansätze um den Kern eines Wortes zu finden: Einerseits gibt es Algorithmen, die Datenbanken verwenden, in denen die Kerne für jedes Wort verzeichnet sind, auf der anderen Seite gibt es Algorithmen, die an Hand von Regeln Wörter automatisch auf den Kern reduzieren. Einer der bekanntesten der zweiten Kategorie ist wohl der Porter-Stemmer[11], welcher Regeln für verschiedene Sprachen besitzt. Als Vorverarbeitungsschritt müssen hier wieder die Stopp-Wörter entfernt werden, da diese ohnehin zu kurz für das Stemming-Verfahren sind. Tabelle 3 zeigt die beispielhafte Anwendung des Algorithmus an Hand einiger Wörter.

11 http://snowball.tartarus.org/algorithms/german/stemmer.html (Abruf 1. 8. 2011)

Wort	Kern laut Porter Algorithmus
aufeinander	aufeinand
aufeinanderfolge	aufeinanderfolg
aufeinanderfolgen	aufeinanderfolg
aufeinanderfolgend	aufeinanderfolg
aufeinanderfolgende	aufeinanderfolg
aufeinanderfolgenden	aufeinanderfolg
aufeinanderfolgender	aufeinanderfolg
kategorisch	kategor
kategorische	kategor
kategorischen	kategor
kategorischer	kategor

Tabelle 3: Beispiel für Wortkerne laut Porter Algorithmus

4.1.5 Anwendung bei Open Journal Systems

Tagging kann als Grundlage für die Bestimmung der Ähnlichkeit von Texten herangezogen werden. Die Schwachstelle dieses Ansatzes liegt darin, dass man hier auf die Mithilfe von Expertinnen beziehungsweise Experten angewiesen ist, einerseits um automatisch generierte Tags auf ihre Richtigkeit hin zu überprüfen, andererseits um Überbegriffe zu finden, die nicht automatisch erstellt werden können. Die Qualität der Empfehlungen hängt dann sehr stark von der Qualität der gesetzten „Tags“ und von der Gewissenhaftigkeit der Expertinnen und Experten ab.

TF-IDF ist ein nützliches Werkzeug um Wörter eines Textes in Relation zu allen vorhandenen Texten des Systems zu betrachten und zu gewichten, jedoch beruht eine solche Gewichtung mittels TF-IDF auf das explizite Vorhandensein von einzelnen Ausdrücken in den Texten. So werden Synonyme beispielsweise nicht erkannt. Ein weiteres Manko dieses Algorithmus ist, dass er rechenintensiv ist: Für die Berechnung der inversen Dokumentenhäufigkeit muss jedes Wort in jedem Dokument gezählt werden.

LSA ist ein guter Ansatz um Dokumente auf ihre inhaltliche Ähnlichkeit hin zu untersuchen. Bei dem Hinzufügen von neuen Dokumenten hat man die Wahl zwischen der Neuberechnung der Matrizen oder einem Update [3]. Je größer die Grundmenge an Daten, also an Texten und somit Wörtern ist, desto besser können Konzepte gefunden und verglichen werden. Bei kurzen Texten, wie den Kurzfassungen, ist das Ergebnis schwer interpretierbar, weil es nur wenige Überschneidungen in den Daten gibt.

Stemming kann als Vorverarbeitungsschritt für TF-IDF eingesetzt werden, um den Korpus (die betrachteten Wörter eines Textes) zu reduzieren.

4.2 Sekundärsystem – Collaborative-Filtering

Collaborative-Filtering-Systeme analysieren das Verhalten von Benutzerinnen und Benutzern, indem Daten über deren Besuch auf der Webseite gesammelt werden. Aus diesen gesammelten Daten wird im ersten Schritt eine Benutzer-Element-Matrix erstellt, in welcher die relevanten Aktionen festgehalten werden. Diese Matrix kann nun sehr groß werden und wird mit großer Wahrscheinlichkeit auch spärlich ausgefüllt sein. Man benötigt nun Techniken um einerseits die Daten zu erfassen und andererseits die Daten effizient auswerten zu können.

4.2.1 Identifizierung der BenutzerInnen

Um Collaborative-Filtering einzusetzen, ist es erforderlich, einzelne BenutzerInnen zu identifizieren damit sie gegebenenfalls wiedererkannt werden können, wenn sie die Webseite erneut besuchen. Die zuverlässigste Methode für eine Benutzererkennung ist die Verwendung von Benutzerkonten, welche jede Benutzerin und jeden Benutzer mit einem Benutzernamen und einem Passwort verbinden. Zusätzlich gäbe es hier die Möglichkeit, demografische Daten über die/den BenutzerIn in den Empfehlungsprozess einzubeziehen.
Eine alternative Möglichkeit, BenutzerInnen erkennen zu können, ist die Verwendung von Cookies. Dies sind kleine Textdateien, die vom Browser verwaltet werden. Der Nachteil von Cookies besteht darin, dass sie unter der Kontrolle der BenutzerInnen stehen. Wird das Cookie gelöscht oder wird die Möglichkeit der Erstellung von Cookies lokal deaktiviert, ist eine Identifizierung der Benutzerin oder des Benutzers nicht mehr möglich.
Um möglichen Manipulationen durch die BenutzerInnen vorzubeugen, sollte nicht nur die Identifikationsnummer der Benutzerin beziehungsweise des Benutzers im Cookie gespeichert werden, sondern auch eine mit der Nummer verbundene Zufallszeichenkette.

4.2.2 K-means Alogrithmus

Die gesammelten Daten über die Beziehungen von Personen zu den verfügbaren Elementen sind naturgemäß sehr umfangreich. Um nun eine effiziente Weiterverarbeitung zu ermöglichen, ist es sinnvoll die Daten zu unterteilen. Der k-means Algorithmus (siehe Kapitel 2.5.3.1) unterteilt die BenutzerInnen eines Systems in Benutzergruppen, die einen ähnlichen Geschmack haben. Wenn nun eine neue Person hinzukommt, wird sie an Hand ihrer verwendeten Elemente einer Gruppe zugeordnet und erhält Vorschläge aus den Elementen dieser Gruppe. Diese Zuordnung wird erst ab einer bestimmten Anzahl von verwendeten Elementen zu einem sinnvollen Ergebnis führen.

4.2.3 Collaborative-Filtering mit Benutzerpfaden

Benutzerpfade haben eine eingeschränkte Verwendbarkeit, wie die Analyse in Kapitel 3 bereits gezeigt hat. Für die Anwendbarkeit von aufgezeichneten Benutzerpfaden für ein System mit Collaborative-Filtering ist es erforderlich, die uninteressanten Pfade zu

entfernen, beziehungsweise zumindest abzuschwächen. Im Fall einer Zeitschrift ist das Inhaltsverzeichnis jener Pfad, der abgeschwächt werden soll (siehe Kapitel 3.2.2).

Eine Methode um solche weniger interessanten Pfade zu kennzeichnen und zu entfernen ist, jedem Pfad ein spezifisches Gewicht zuzuordnen. Zu diesem Zweck wird aus den unerwünschten Relationen eine Tabelle erstellt, die dazu verwendet wird, eben diese Relationen zu zählen und in weiterer Folge zu „bestrafen“. Eine Relation ist in diesem System definiert als: [*Artikel A* ↔ *Artikel B*]. Das bedeutet, immer wenn Artikel A auf Artikel B folgt oder umgekehrt, wird das als Relation erkannt und entsprechend reagiert. Für L3T bedeutet das eine Menge von Relationen, die das Inhaltsverzeichnis beschreiben. Tabelle 4 illustriert diese gefundene Menge, die zur Berechnung der Gewichte der Pfade verwendet wird.

Relationen L3T
Inhaltsverzeichnis #1 (Artikelnummer: 89) ↔ Inhaltsverzeichnis #2 (Artikelnummer: 88)
Inhaltsverzeichnis #2 (Artikelnummer: 88) ↔ Inhaltsverzeichnis #3 (Artikelnummer: 49)
Inhaltsverzeichnis #3 (Artikelnummer: 49) ↔ Inhaltsverzeichnis #4 (Artikelnummer: 73)
…
Inhaltsverzeichnis #46 (Artikelnummer: 31) ↔ Inhaltsverzeichnis #47 (Artikelnummer: 35)
Inhaltsverzeichnis #47 (Artikelnummer: 35) ↔ Inhaltsverzeichnis #48 (Artikelnummer: 43)

Tabelle 4: Relationen zur Berechnung der Gewichte der Pfade

Mit Hilfe dieser Relationen Tabelle ist es möglich, jedem Pfad ein Gewicht zuzuweisen. Ein Pfad ist für die Erstellung von Empfehlungen umso weniger interessant, je ähnlicher er einer existierenden Struktur, wie zum Beispiel dem Inhaltsverzeichnis, ist. Für die Bewertung des Pfades werden hier die Vorkommen der unerwünschten Relationen gezählt und durch die Länge des Pfades (die Anzahl der Artikel, die in diesem Pfad gespeichert wurden) dividiert. Mit dieser Berechnung können auch kleine Sequenzen einer Struktur in einem Pfad wiedergefunden und bewertet werden:

$$weight_{path} = 1 - \frac{\#relations\_path}{path\_length} \tag{16}$$

Somit ist das maximale Gewicht eines Pfades 1. Das bedeutet, der Pfad beinhaltet keine Elemente einer definierten Struktur. Ein Wert kleiner als 1 bedeutet, dass der Pfad Teile der Struktur enthält. Auf der anderen Seite erhält ein Pfad sein Minimalgewicht, wenn der Pfad vollständig aus Elementen einer definierten Struktur besteht. In diesem Fall gilt $\#relations\_path = path\_length - 1$. Bei einem Pfad der Länge 6 sind dann alle 5 Relationen aus der Menge der definierten Relationen.

Beispiel: Berechnung des Gewichtes des Pfades $89 \rightarrow 88 \rightarrow 49 \rightarrow 64 \rightarrow 35 \rightarrow 31 \rightarrow 20 \rightarrow 72$:

Im ersten Schritt werden die Relationen, die in Tabelle 4 notiert wurden, im Pfad gezählt. Die Relationen sind: $(89 \rightarrow 88), (88 \rightarrow 49), (35 \rightarrow 31)$ - das ergibt eine Anzahl von 3 gefundenen Relationen innerhalb des untersuchten Pfades.
Die Länge des Pfades beträgt 8 Artikel. Eingesetzt in Formel 16 ergibt das

$$weight_{path} = 1 - \frac{3}{8} = 0{,}625.$$

Die gefundenen Gewichte werden nun dafür verwendet, um die Pfade, die für die Erstellung einer Empfehlung verwendet werden, zu bewerten. Durch die Gewichtung erfolgt eine Reihung der Pfade, aus denen die nächsten Elemente vorgeschlagen werden.

4.2.4 Anwendung bei Open Journal Systems

Die Anwendung eines Systems mit Collaborative-Filtering ist erst ab einem gewissen Grad an gesammeltem Wissen über die Beziehungen von Benutzerinnen und Benutzern zu den Artikeln sinnvoll. Das heißt, dass erst sinnvolle Empfehlungen erstellt werden können, wenn die/der BenutzerIn mehrere Artikel gelesen hat. Ab diesem Zeitpunkt ist dieses System eine interessante Ergänzung zum Primärsystem mit Content-Based-Filtering.

Die Identifizierung der BenutzerInnen wird mittels IP-Adresse und Cookies erfolgen, da die BenutzerInnen von OJS sich nicht zwingend anmelden müssen, um die Artikel lesen zu können. BenutzerInnen, die dem System unbekannt sind, bekommen eine interne Nummer zugeteilt, welche mit den Pfaden verbunden wird, die sie bei ihren Besuchen erzeugen. Diese Nummer wird als Cookie am Rechner einer Benutzerin oder eines Benutzers gespeichert, um sie oder ihn bei einem erneuten Besuch wiedererkennen zu können.

Die Verwendung von gewichteten Benutzerpfaden kann als Zusatzsystem zur Erstellung von Empfehlungen verwendet werden. Voraussetzung dafür ist einerseits, dass es Pfade im System gibt und andererseits, dass die aktuelle Benutzerin beziehungsweise der aktuelle Benutzer bereits einen Pfad generiert hat, der zum Vergleichen herangezogen werden kann. Um die Menge an gespeicherten Pfaden zu reduzieren, werden Duplikate der Pfade gelöscht. Jeder Pfad erhält einen Zähler wie oft er bereits gewählt wurde. Sobald Pfaddaten verfügbar sind, können sie zusätzlich zu den Empfehlungen des Content-Based-Filtering Systems angezeigt werden, beziehungsweise einzelne Empfehlungen ersetzen.

4.3 Tabellarische Übersicht der Algorithmen

Für das Plugin kommen folgende Methoden zum Einsatz: Stemming, TF-IDF und die Auswertung von Benutzerpfaden. Diese Methoden sind in Tabelle 5 farblich hervorgehoben.

Methode	Vorteile	Nachteile	Anwendbarkeit in OJS
Tagging	Bildung von Überbegriffen, inhaltliche Zuordnung	Abhängig von der Gewissenhaftigkeit der Experten, die Tags erstellen.	Auffinden von zusammengehörigen Texten
Stemming	Reduzierung der Wortmenge	Laufzeit abhängig von der Anzahl der Wörter	Vorverarbeitungs-schritt für TF-IDF, um die Wortmenge zu reduzieren
TF-IDF	Automatisiertes, statistisches Verfahren	Keine Synonym-erkennung; Update rechenintensiv	Auffinden von ähnlichen Texten über die Analyse der Abstracts
LSA	Automatisches Verfahren zum Finden der Ähnlichkeiten von Texten, Update-möglichkeit für das Einfügen von neuen Dokumenten	Bildung der Matrizen rechenintensiv; Matrizen werden sehr groß; Benötigt eine große Menge an Texten für aussagekräftige Resultate	Auffinden von ähnlichen Texten über die Analyse der Texte selbst
K-means	Einteilung der BenutzerInnen in Nachbarschaften, nach der Gruppen-bildung ist eine schnelle Erstellung der Empfehlungen möglich	Keine Empfehlungen für eine neue Person oder ein neues Dokument, Update der Benutzer-gruppen rechenintensiv, Externes Programm notwendig	Sekundärsystem zur Erweiterung der Ergebnisse wenn genug Daten vorhanden sind.
Pfaddaten	Einbindung des Verhaltens der BenutzerInnen in die Empfehlungs-erstellung	Erst nach einiger Zeit einsetzbar, da eine Grundmenge an Pfaddaten notwendig ist. Identifizierung einer Struktur für die Gewichtung der Pfade notwendig.	Erweiterung der Empfehlungen des Primärsystems, wenn Pfade zum Vergleich vorhanden sind. Gewichtung der Pfade durch das Inhaltsverzeichnis.

Tabelle 5: Vergleich der Algorithmen

5 OJS Plugin

5.1 Aufgabenstellung

Die Aufgabenstellung dieser Arbeit ist die Entwicklung eines Empfehlungssystems in Form eines Plugins für das Open Journal System (OJS). In einem Journalsystem wie OJS werden in periodischen Abständen neue Ausgaben einer Zeitschrift online gestellt. Die vorherigen Ausgaben sind dann nur mehr über den Link ins Zeitschriftenarchiv auffindbar. Ein Empfehlungssystem soll es den BenutzerInnen erleichtern, interessante und relevante Artikel zu finden, auch über die Grenzen einer Ausgabe einer Zeitschrift hinaus. Somit sollen auch ältere Artikel wieder besser zugänglich gemacht werden. Das Empfehlungssystem soll als Plugin realisiert werden, das bedeutet ein gewisses Maß an Einschränkungen, in Bezug auf die Abhängigkeit von externen Hilfsprogrammen und die Komplexität der Berechnungen. Das Plugin soll auf allen OJS Installation lauffähig sein beziehungsweise ohne allzu großen Aufwand installierbar und wartbar sein. Das Plugin soll des weiteren selbstständig Daten über das Benutzerverhalten sammeln und auf der Grundlage dieser Daten Empfehlungen für die BenutzerInnen erstellen. Die Datenquelle für die Empfehlungserstellung bilden die Downloads der Dokumente. Wenn eine Benutzerin oder ein Benutzer einen Artikel herunterlädt, dann wird diese Aktion vom Plugin aufgezeichnet. Durch das Speichern mehrerer Artikel erstellt die/der BenutzerIn einen Pfad, der vom Plugin für den Vergleich mit bestehenden Pfaden verwendet werden kann. Damit immer Empfehlungen erstellt werden können, auch wenn keine Daten über das Benutzerverhalten vorliegen, oder der Vergleich der Daten kein Ergebnis liefert, soll ein zusätzliches System auf Basis des Content-Based-Filterings helfen und Elemente empfehlen. Dieses zweite System bringt die Kurzfassungen der Artikel zueinander in Relation, da die eigentlichen Texte nur als PDF-Dokumente vorhanden sind.

Mit diesen Anforderungen soll nun ein Plugin entwickelt werden, welches in OJS unterhalb der Kurzfassung eines Artikels die erstellten Empfehlungen für weitere Artikel anzeigt. Je nachdem, welche Dokumente die/der BenutzerIn gespeichert hat, sollen sich die Empfehlungen an das Verhalten der Benutzerin oder des Benutzers anpassen.

In den vorangegangenen Kapiteln wurden die theoretischen Grundlagen für die verschiedensten Typen von Empfehlungssystemen behandelt, des weiteren eine Analyse der Benutzeraktionen einer bestehenden OJS Installation durchgeführt und eine Auswahl der Algorithmen für ein Empfehlungssystem in Form eines Plugins getroffen. Die folgenden Kapitel beschreiben den Aufbau und die technische Umsetzung dieses Empfehlungssystems in Form eines Plugins für das OJS und die Auswertung der Resultate des erstellten Plugins. Für diese Auswertung werden die Ergebnisse des Plugins den Ergebnissen eines Vergleichsalgorithmus gegenübergestellt und die Unterschiede diskutiert.

5.2 Plugins bei OJS

Um eine Installation des OJS zu erweitern, ist es möglich, Plugins zu entwickeln. Diese Programme werden, wenn sie aktiv sind, zusätzlich zum standardmäßigen Funktionsumfang des OJS geladen. Für das Erstellen eines Plugins für das OJS muss man einigen Spezifikationen folgen:

- Das Verzeichnis des Plugins muss eine vordefinierte Dateistruktur aufweisen.
- Die Dateinamen der php-Dateien sollten die Endung „.inc.php“ aufweisen, damit die Plugin-Funktion zum Inkludieren der Dateien verwendet werden kann.
- Es muss eine version.xml-Datei existieren, die Informationen zum Plugin beinhaltet (Name, Versionsnummer, Typ der Plugins, etc.)
- Wenn das Plugin eigene Datenbanktabellen verwendet, muss es eine install.xml und eine schema.xml Datei geben, damit die automatische Installation eines Plugins durch das OJS verwendet werden kann. Als Alternative kann das Datenbankschema auch händisch installiert werden, dafür braucht man aber direkten Zugriff auf den Server, auf dem das OJS installiert ist.

Damit das Plugin den Funktionsumfang des OJS erweitern kann, muss es möglich sein auf bestimmte Ereignisse zu reagieren. Für diesen Zweck kann man in der Hauptklasse des Programms auf sogenannte „Hooks“ zugreifen. Ein Hook (englisch für „Haken“) ist ein Einstiegspunkt, der aus einem Namen und einer Liste von Parametern besteht. OJS verwendet eine eigene Klasse um diese Hooks zu verwalten. Mit Hilfe dieser Klasse können an jeder beliebigen Stelle Einstiegspunkte registriert werden, zusammen mit den dazugehörigen Zustandsvariablen. Somit ist es möglich, mitten im Programmfluss eine eigene Methode zu registrieren und aufzurufen. Beispielsweise kann so eine Methode registriert werden, die darauf reagiert, wenn ein Artikel geöffnet wird. Der entsprechende Hook stellt für diese neue Methode nun das Objekt des aufgerufenen Artikels bereit, mit dessen Hilfe man auf die Daten dieses Artikels zugreifen kann, um sie nun auf die gewünschte Art und Weise zu verarbeiten.

5.3 Installation eines Plugins bei OJS

Um ein Plugin zu installieren gibt es innerhalb des OJS zwei Möglichkeiten: Entweder man kopiert den Ordner des Plugins mittels eines FTP-Programmes (FTP steht für „File Transfer Protocol“) in den richtigen Ordner, trägt die Versionsnummer in die Datenbanktabelle „versions“ ein und führt einen Skriptbefehl aus, welcher die benötigten Datenbanktabellen anlegt, oder man verwendet die integrierte Installationsmöglichkeit von OJS. Für die zweite Variante muss das Plugin als „.tar.gz“ - Datei (eine gepackte Form eines Dateiarchives) verfügbar sein. Das OJS entpackt diese Datei in das korrekte Verzeichnis, trägt die Versionsnummer in die Datenbank ein und installiert die benötigten Tabellen in der Datenbank. Bei beiden Methoden muss darauf geachtet werden, dass die vorgegebene Verzeichnisstruktur befolgt wird. Im Hauptverzeichnis des Plugins muss es eine „index.php“ Datei geben, die das Objekt des Plugins erzeugt und registriert. Hinzu kommt eine „version.xml“ Datei, die Informationen über das Plugin enthält, welche in die „versions“ Tabelle in der OJS Datenbank eingetragen werden. Wenn verschiedene Sprachen unterstützt werden, muss das Hauptverzeichnis ein „locale“ - Verzeichnis

enthalten, wo die Dateien für die jeweiligen Sprachen zu finden sind (zum Beispiel „en_US.xml“ für Englisch).
Nach der Installation der Dateien und Datenbanktabellen kann das Plugin auf der Pluginverwaltungsseite des Journals aktiviert werden.

5.4 Aufbau des Plugins

Das Plugin für die Erstellung von Empfehlungen trägt den Namen „RecommenderSystemPlugin“ und besteht aus mehreren Teilen, auf die in den folgenden Kapiteln genauer eingegangen wird:

- **RecommenderSystemPlugin**
  Diese Klasse ist die Hauptklasse des Plugins. Hier werden die benötigten Hooks und Callback-Methoden registriert, die Management-Funktionen des Plugins festgelegt und alle Objekte angelegt und verwaltet.

- **RecommenderSystemInstaller**
  Um die Empfehlungen aktuell zu halten, ist es notwendig in periodischen Abständen die Datenbank neu zu berechnen, damit die Distanzen zwischen den einzelnen Artikeln wieder auf den neuesten Stand gebracht werden. Diese Klasse implementiert die Verwendung eines Templates (englisch für „Schablone“, „Vorlage“), um den Installationsfortschritt anzeigen zu können. Ein Template bezeichnet eine Vorlage mit variablem Inhalt. Dieser Inhalt wird nun von der RecommenderSystemInstaller-Klasse erzeugt, an der richtigen Stelle im Template eingefügt und angezeigt.

- **RecommenderSystemInstallDAO**
  Diese Klasse beinhaltet die Zugriffsmethoden der Datenbank für die Installation und Aktualisierung der Datenbanktabellen. Die Abkürzung DAO steht für „Data Access Object“ und wird an alle Klassennamen angefügt, die Methoden zur Manipulation der Datenbank zur Verfügung stellen. Die Verbindungsherstellung und die grundlegenden Funktionen für die Verwendung der Datenbank in OJS wird von einer DAO-Basisklasse implementiert.

- **RecommenderSystemUserHandler**
  In dieser Klasse sind die Methoden zur Benutzeridentifizierung, Benutzerverwaltung und für die Aufzeichnung und Auswertung der Benutzerpfade gesammelt.

- **RecommenderSystemDAO**
  In dieser Klasse sind alle Zugriffsmethoden für die Datenbank enthalten, die nicht bei der Installation des Plugins benötigt werden. Die RecommenderSystemDAO Klasse ist von der DAO – Basisklasse abgeleitet.

- **RecommenderSystemRecommendations**
  Diese Klasse ist für die Zusammenstellung und die Ausgabe der Empfehlungen verantwortlich.

– **Hilfsklassen:**

RSTextAnalyzer: Erstellt aus diversen Filterklassen (Stoppwort-Filter, Porter-Stemmer-Filter, etc.) eine Filterkette zur Analyse von Texten.

CookieHelper: Diese Klasse ist eine Sammlung von Methoden zur Verwaltung von Cookies.

TimeHelper: Hier sind Methoden zur Erstellung von Zeitstempeln und Formatierung von Datums- und Zeitangaben gesammelt.

Debug: Diese Klasse ist für die Ausgabe von Fehlermeldungen zuständig. Als Parameter kann eine Datei angegeben werden, welche zur Ausgabe der Fehlermeldung genutzt werden soll. Im Live-Betrieb des Plugins ist die Fehlerausgabe ausgeschaltet.

5.4.1 RecommenderSystemPlugin

Diese Klasse (Abbildung 16) repräsentiert die Hauptklasse des Plugins. Hier wird festgelegt, wie auf bestimmte Aktionen der Benutzerin beziehungsweise des Benutzers reagiert werden soll. Die benötigten Objekte werden bei der Erzeugung des Plugins angelegt und in den jeweiligen Klassenvariablen gespeichert.

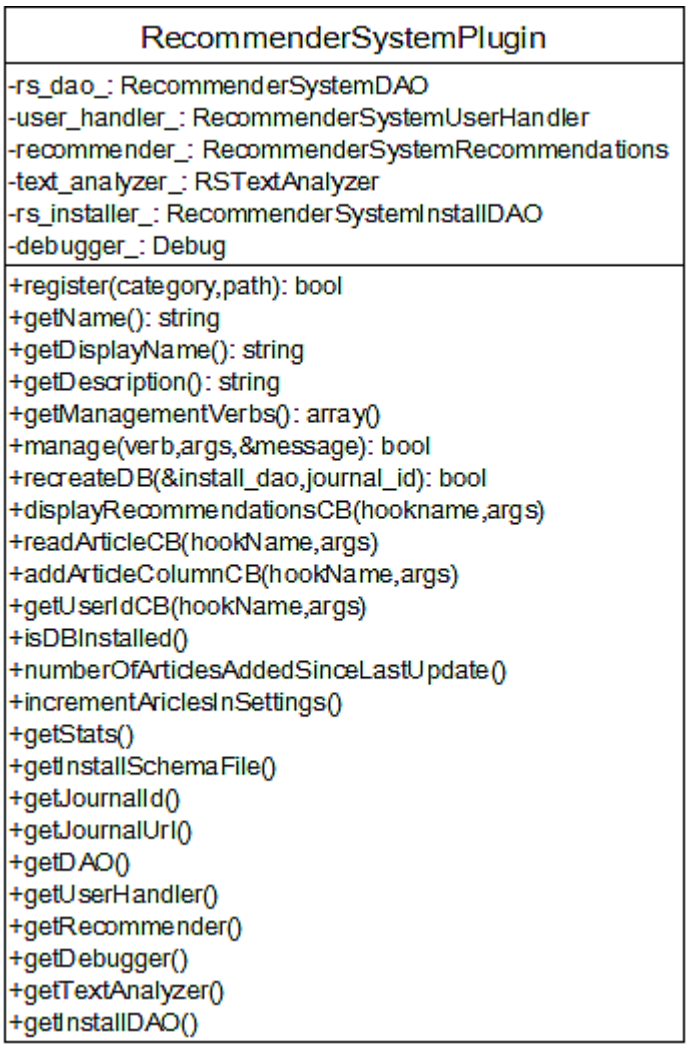


Abbildung 16: Klasse: RecommenderSystemPlugin

Um nun von außerhalb auf diese Objekt zugreifen zu können, stellt die Plugin-Klasse mehrere „get-Methoden“ zur Verfügung, mit deren Hilfe man Zugriff auf die Methoden der Objekte erhält. Beispielsweise liefert `getUserHandler()` den Zugriff auf das Objekt, das für die Erfassung der BenutzerInnen und Benutzeraktionen zuständig ist. Zusätzlich wird abgefragt, ob das Objekt existiert. Wenn das nicht der Fall ist, werden die benötigten Script-Dateien inkludiert und das Objekt angelegt.

Der Einstiegspunkt in das Plugin ist die `register()` Methode. Diese wird vom OJS System aufgerufen, wenn das Plugin aktiviert ist, oder wenn die Verwaltungsseite für OJS-Plugins aufgerufen wird. Der Zweck dieser Methode ist Callback-Methoden zu registrieren.

Dabei wird für jede gewünschte Aktion, auf die reagiert werden soll und für die ein Hook existiert, eben dieser Hook mit der entsprechenden Reaktion – der Callback-Methode – verbunden. Beim Aufruf des Hooks durch das Programm wird an einer zentralen Stelle (der sogenannten HookRegistry) nachgesehen, ob ein Hook durch ein Plugin registriert wurde. Wenn dies der Fall ist, wird die angegebene Callback-Methode des Plugins ausgeführt.

Für dieses Plugin sind folgende Aktionen von Bedeutung: Das Lesen eines Artikels (gleichbedeutend mit dem Download der pdf-Datei des Artikels), die Anzeige der Kurzfassung des Artikels und das Hinzufügen eines neuen Artikels. Diese Aktionen werden in der `register()` Methode mit den jeweiligen Callback-Methoden verbunden. Als Callback-Methode für den Download einer pdf-Datei wird `readArticleCB()` registriert, welche die zuständige Methode `readArticle()` in der UserHandler-Klasse aufruft.
Das Hinzufügen eines neuen Artikels ruft die Methode `addArticleColumnCB()` auf, welche folgende Aktionen auslöst: Um zu vermeiden, dass der Artikel doppelt eingefügt wird, überprüft die Methode als erstes ob die ID des Artikels bereits in der Wort-Artikel-Tabelle der Datenbank existiert. Wurde er noch nicht hinzugefügt, wird die Datenbanktabelle mit dem neuen Artikel aktualisiert (siehe Kapitel 5.4.3). Anschließend werden die Relationen des Inhaltsverzeichnisses neu berechnet und die Variable, die die Anzahl der eingefügten Artikel angibt, um eins erhöht.

Damit eine visuelle Ausgabe möglich ist, bieten die vorgefertigten Templates von OJS einige Einstiegspunkte, wie auch bei der Ansicht der Kurzfassung eines Artikels. Diese Anzeige wird mit der Methode `displayRecommendationsCB()` verbunden. Ein Hook welcher in einem Template registriert wird, bietet die Möglichkeit, an dieser Stelle eine Ausgabe anzuzeigen. In unserem Fall erstellt die Recommendations-Klasse die Empfehlungen (siehe Kapitel 5.4.8) und liefert eine Zeichenkette im HTML-Format, welche an das Template zur Anzeige übergeben wird.

Neben diesen Hooks und den dazugehörigen Callback-Methoden benötigt ein Plugin in OJS noch eine Reihe anderer Methoden, um korrekt zu arbeiten. Die Methoden `getName()`, `getDisplayName()` und `getDescription()` sind selbsterklärend, wobei letztere dazu verwendet werden kann, den Status des Plugins auf der Verwaltungsseite von OJS anzuzeigen. Dies ist zum Beispiel notwendig, wenn das Plugin eigene Datenbanktabellen verwendet. Für den Fall, dass nicht alle benötigten Tabellen bei der Installation des Plugins angelegt wurden, ist es für den Verwalter des Journals hilfreich wenn das Plugin eine entsprechende Nachricht anzeigen kann.

Zu den notwendigen Methoden gehört auch `getManagementVerbs()`, welche die Links für die Verwaltung des Plugins definiert und registriert. Dies wird durch ein Wertepaar realisiert, wobei der erste Wert den internen Namen der Funktion darstellt, wie zum Beispiel „enable“. Der zweite Wert ist der Name, der auf der Einstellungsseite angezeigt wird – für „enable“ wäre das beispielsweise „Plugin aktivieren“. Zusammen sieht das Wertepaar folgendermaßen aus: `array('enable', 'Plugin aktivieren').`
Anstelle einer Zeichenkette kann auch eine passende Übersetzung aus den Sprachdateien eingebunden werden.

Das Gegenstück zu `getManagementVerbs()` ist die Methode `manage()`. Hier wird auf die entsprechenden Links der Verwaltungsseite reagiert. Das RecommenderSystemPlugin behandelt folgende Anweisungen:

- *enable:* Das Plugin wird aktiviert und dieser Zustand wird in der Datenbank festgehalten. Für diesen Zweck wird die von OJS bereitgestellte Methode `updateSetting()` verwendet. Bei der ersten Aktivierung wird gleich die Datenbank für das Content-Based-Filtering gefüllt.
- *disable*: Das Plugin wird deaktiviert und der Zustand in der Datenbank aktualisiert.
- *install:* Dieser Aufruf ist für die Anzeige des Installations-Templates zuständig.
- *installdbdone:* Wenn das Installationsscript mit seiner Arbeit fertig ist, wird durch den Aufruf dieses Links die Installation beendet und dieser Zustand in der Datenbank festgehalten. Zusätzlich wird der Zähler für neu hinzugefügte Artikel wieder auf Null zurückgesetzt.
- *recreate:* Dieser Link aktiviert die erneute Installation der Datenbanktabellen für das Content-Based-Filtering. Die Benutzerpfade bleiben davon unberührt.

Bei der ersten Aktivierung des Plugins und bei der manuellen Aktualisierung der Datenbank kommt die Methode `recreateDB()` zum Einsatz. Als erste Aktion dieser Methode werden die Tabellen für das Content-Based-Filtering geleert. Im nächsten Schritt wird die Tabelle der Relationen für die Pfadgewichtung neu erstellt. Dadurch können etwaige Änderungen im Inhaltsverzeichnis berücksichtigt werden. Anschließend werden die Session-Variablen gesetzt, die dann im Installations-Template verwendet werden, um die Installation zu steuern.

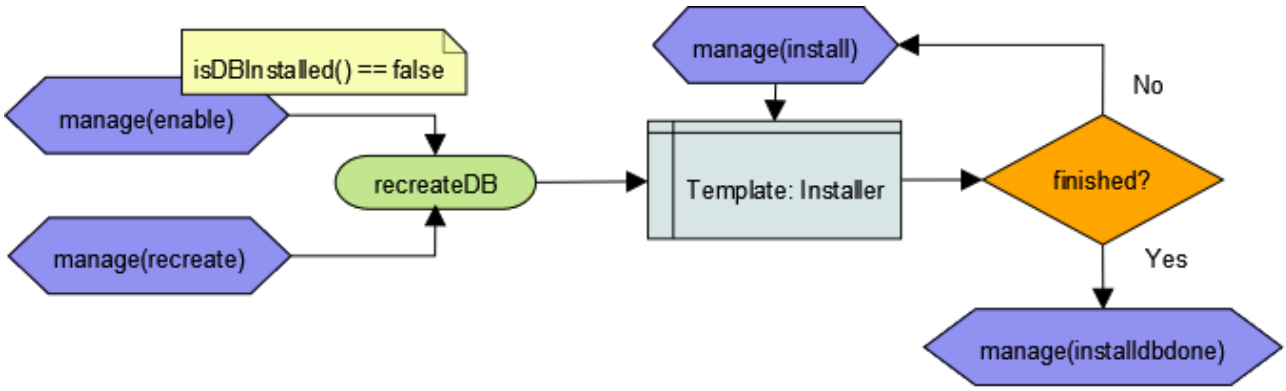


Abbildung 17: Sequenz: Datenbankinstallation

In Abbildung 17 ist die Installationssequenz für die Einrichtung der Datenbank illustriert. Zwei mögliche Events können den Prozess starten: `manage(enable)` und `manage(recreate)`. Das erste Ereignis tritt nur ein, wenn das Plugin aktiviert wird und die Datenbank noch nicht eingerichtet wurde (die Methode `isDBInstalled()` liefert den Wert `false`). Das zweite Ereignis wird durch einen Link aktiviert. Beide Events führen zum Aufruf der `recreateDB()` Methode, wie bereits erwähnt. Nach dem Setzen der Session-Variablen wird das Template für die Installation angezeigt. Der aktuelle Installationsfortschritt wird dabei in einer eigenen Session-Variable gespeichert. Solange dieser Wert nicht das Ende der Installation signalisiert, wird `manage(install)` ausgelöst, was einem Neuladen der Internetseite entspricht. Das Template wird erneut aufgerufen

und zwar mit den aktualisierten Werten. Dieser Kreislauf ist notwendig, da eine Anfrage an einen Webserver üblicherweise nach 30 Sekunden beendet wird. Je größer die Anzahl der zu bearbeitenden Texte in der Datenbank ist, desto länger dauert die Anfrage. Damit die Anfrage nun nicht unterbrochen wird, was zu einer unvollständigen Datenbank führen würde, besitzen die Installationsmethoden ein 15-Sekunden-Limit. Weiters speichern sie ihren internen Status in Sessionvariablen, damit der nächste Aufruf die Installation nahtlos weiterführen kann. Das Ende des Installationsvorganges wird durch `manage(installdbdone)` signalisiert. Nach Aufruf dieses Ereignisses wird in der Datenbank festgehalten, dass die Tabellen für die Verwendung des Empfehlungssystems einsatzbereit sind.

5.4.2 RecommenderSystemInstaller

Diese Klasse (Abbildung 18) implementiert die Anzeige des Installations-Templates. Im Konstruktor werden die Klassenvariablen gesetzt und die Vorlagendatei (in unserem Fall „install.tpl“) registriert. Für die Durchführung der Installation ist die Methode `installDB()` zuständig.

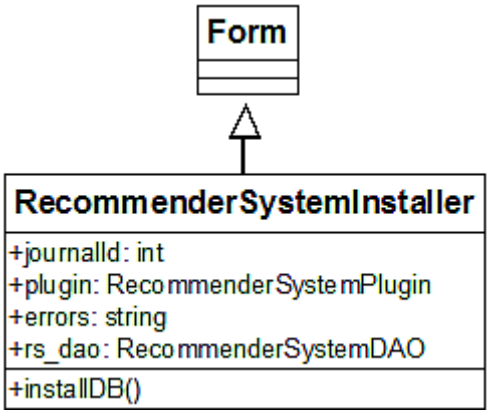


Abbildung 18: Klasse: RecommenderSystemInstaller

Anhand einer Zustandsvariable wird entschieden, welche Aktionen durchgeführt werden. Die möglichen Zustände sind:

- „`fillWordArticles`“: Die einzelnen Artikelbeschreibungen werden gesammelt, gefiltert und in die temporäre Wort-Artikel-Tabelle der Datenbank eingetragen.
- „`tfidf`“: Aus den Daten der temporären Wort-Artikel-Tabelle werden die TF-IDF Werte für jedes Wort gebildet und in die finale Wort-Artikel-Tabelle eingetragen.
- „`article_distances`“: In diesem Schritt werden die Distanzen zwischen den einzelnen Artikelvektoren bestimmt. Für jeden Artikel wird die Cosinusdistanz zu jedem anderen Artikel berechnet und gespeichert.
- „`finish`“: Dieser Zustand markiert das Ende der Installation. Im Template wird eine Schaltfläche für den Abschluss der Installation angezeigt und es werden die Session-Variablen gelöscht.

5.4.3 RecommenderSystemInstallDAO

In dieser Klasse (Abbildung 19) sind die Zugriffsmethoden für die Manipulation der Datenbanktabellen bei der Einrichtung und bei der erneuten Installation der Datenbanktabellen definiert.
Zwei Hilfsmethoden für die Erstellung der Tabellen für das Content-Based-Filtering sind `countArticles()` und `compute-ArticleIdArray()`.
Erstere Methode zählt die veröffentlichten Artikel im Journal, die zweite Methode erstellt eine Liste von Wertepaaren. Diese Paare bestehen aus der ID eines Artikels und der Länge des Wort-Artikel-Vektors für diesen einen Artikel. Diese Wertepaare stehen als Klassenvariable zur Verfügung, um die Längen der Wort-Artikel-Vektoren nicht ständig neu zu berechnen.

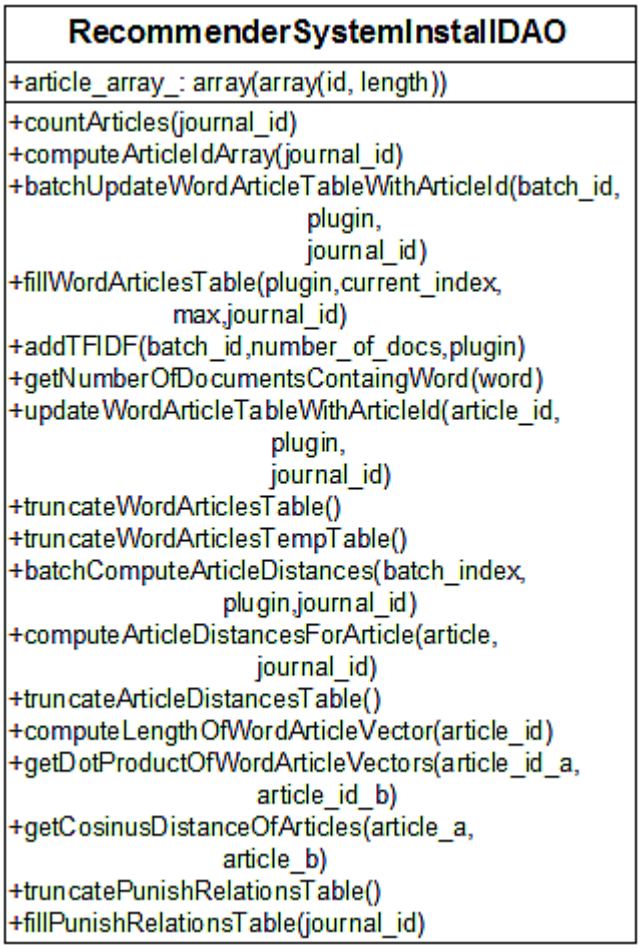


Abbildung 19: Klasse: RecommenderSystemInstallDAO

Um nun die besagten Wort-Artikel-Vektoren zu erzeugen, müssen die betrachteten Texte, in diesem System die Kurzfassungen der veröffentlichten Artikel, analysiert, zerlegt und als Worte mit ihren Häufigkeiten in die Datenbank eingetragen werden. Der Vorgang der Textanalyse wird in Kapitel 5.4.4 genauer beschrieben; das Eintragen der resultieren Worte übernehmen die beiden Methoden `fillWordArticles-Table()` und `batchUpdateWordArticle-TableWithArticleId()`.

Gesteuert wird das Einfügen von der ersten Methode. Die Ausführung dieser Methode unterliegt dabei einer zeitlichen Beschränkung, da eine Anfrage an den Webserver nach 30 Sekunden abgebrochen wird. Um zu gewährleisten, dass alle Artikel bearbeitet und eingefügt werden, wird wie in Kapitel 5.4.1 bereits angesprochen, ein Zeitlimit gesetzt: Wenn die Ausführung der Methode länger als 15 Sekunden dauert, wird die Durchführung beendet. Dabei werden die Zustände der Installation in Sessionvariablen gespeichert, damit sie beim nächsten Aufruf fortgesetzt werden kann. Für dieses Verhalten sorgt die RecommenderSystemInstaller Klasse, wie in Kapitel 5.4.2 beschrieben.
Innerhalb des Zeitlimits sorgt die vorher genannte Methode `batchUpdate-WordArticleTableWithArticleId()` für das Aufspalten eines Textes in seine Wörter und für die Zählung der Häufigkeiten der Wörter innerhalb dieses Textes. Anschließend wird aus den Wörtern und ihren Häufigkeiten eine INSERT – Anweisung für die Datenbank erstellt, die alles in eine temporäre Wort-Artikel-Tabelle einträgt. Temporär deshalb, weil das Einfügen von Daten in eine Datenbank sehr viel schneller ist als das Aktualisieren eines jeden Wertes.

Diese temporäre Tabelle kommt bei der Berechnung der TF-IDF-Werte zum Einsatz, denn bei diesem Schritt werden die benötigten Werte (das Wort, die Artikelnummer in dem das Wort vorkommt und die Häufigkeit des Wortes in diesem Artikel) aus der temporären Tabelle gelesen, für jedes Wort der TF-IDF Wert berechnet und schließlich in der endgültigen Wort-Artikel-Tabelle gespeichert. Diese Tabelle enthält nun das Wort, die Artikelnummer in dem das Wort vorkommt, die Häufigkeit des Wortes und den TF-IDF Wert des Wortes.
Im Plugin sind die Methoden `addTFIDF()` und `getNumberOfDocuments-ContainingWord()` an der Berechnung dieser TF-IDF-Werte beteiligt. Erstere Methode unterliegt wieder der zeitlichen Beschränkung von 15 Sekunden und speichert den Zustand der Berechnungen in eigenen Sessionvariablen, damit die Berechnung beim nächsten Wort fortgesetzt werden kann. Die zweite Methode dient zur Bestimmung der Anzahl von Dokumenten, die das betrachtete Wort beinhalten. Mit diesem Wert wird die inverse Dokumentenhäufigkeit („inverse document frequency“) berechnet:

$$idf_{word} = \log\left(\frac{\#\,documents}{\#\,documents\,with\,word}\right) \tag{17}$$

Der IDF-Wert eines Wortes ist demnach der Logarithmus des Verhältnisses der Anzahl der Dokumente die das Wort enthalten zu der Anzahl aller Dokumente im System. Das Resultat wird mit der Häufigkeit des Wortes im betrachteten Text („term frequency“) multipliziert und ergibt den TF-IDF-Wert [21]. Somit erhält man ein statistisches Maß für die Wichtigkeit eines Wortes. Wörter, die oft im Text vorkommen und zusätzlich in vielen anderen Texten vorkommen, erhalten nun ein niedrigeres Gewicht als Wörter die nur häufig im aktuellen Text vorkommen ohne in anderen Texten aufzutauchen [14 S.45].

Um nun die einzelnen Texte in Beziehung zu setzen, verwendet das Plugin die Berechnung der Cosinusdistanzen zwischen zwei Artikeln. Diese Berechnung geschieht in der Methode `getCosinusDistanceOfArticles(article_a, article_b)`. Die Parameter der Funktion werden hier erwähnt, weil jeder übergebene Parameter ein Wertepaar ist, bestehend aus der Artikel-ID und Länge des entsprechenden Wort-Artikel-Vektors. Die Berechnung dieser Länge erfolgt durch die Methode `computeLengthOfWorldArticleVector()` und entspricht der Definition des Betrags des Vektors:

$$|wordarticle| = \sqrt{tfidf_1^2 + tfidf_2^2 + ... + tfidf_n^2} \qquad tfidf_n : TF - IDF\ Wert\ von\ Komponente : n \tag{18}$$

Die Länge eines Wort-Artikel-Vektors ist also die Wurzel der Summe der Quadrate seiner TF-IDF Werte. Somit fehlt für die Berechnung nur mehr das Skalarprodukt, welches von der Methode `getDotProductOfWordArticleVectors()` geliefert wird. Das Skalarprodukt zweier Wort-Artikel-Vektoren A und B mit den TF-IDF Werten $(a_1, a_2, ..., a_n), (b_1, b_2, ..., b_n)$ wird wie folgt gebildet:

$$\begin{pmatrix} a_1 \\ a_2 \\ ... \\ a_n \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ b_2 \\ ... \\ b_n \end{pmatrix} = (a_1 \cdot b_1 + a_2 \cdot b_2 + ... + a_n \cdot b_n) \tag{19}$$

Es müssen nur die Einträge multipliziert werden, die in beiden Vektoren an der gleichen Stelle vorkommen, also die TF-IDF Werte der Wörter, die in beiden Artikeln vorkommen. Wenn ein Wort in einem Artikel nicht vorkommt, besitzt der Wort-Artikel Vektor an dieser Stelle eine „0“ - somit ergibt das Produkt mit diesem Wert wieder „0“. Also werden nur mit TF-IDF Werten Produkte gebildet, deren dazugehörige Wörter in beiden Artikeln vorkommen. Implementiert wird dieses Verhalten durch die Methode `getDotProductOfWordArticleVectors()`. Als erstes werden alle Wörter und die dazugehörigen TF-IDF-Werte eines Artikels gesucht. Für jedes dieser Wörter wird abgefragt, ob es auch im zweiten Artikel vorkommt, wenn ja, werden die TF-IDF-Werte des Wortes aus beiden Artikeln multipliziert. Diese Produkte werden dann aufsummiert, was dem Skalarprodukt der beiden Vektoren entspricht.

Die Ermittlung der Cosinusdistanzen aller Artikel ist die zeitlich aufwändigste Berechnung, da die Beziehung von jedem Artikel zu jedem Artikel ermittelt werden muss. Die Anzahl der Berechnungen wird halbiert, weil jede Beziehung bidirektional ist: Die Distanz von Artikel ***A*** zu ***B*** entspricht der Distanz von Artikel ***B*** zu ***A***:

$$similarity_{\cos}(A, B) = similarity_{\cos}(B, A) \tag{20}$$

Zuständig für diese Berechnungen ist die Methode `batchComputeArticleDistances()`, die auch wieder dem Zeitlimit von 15 Sekunden unterliegt und den Zustand der Installation in Sessionvariablen speichert. Bei einem durchschnittlichen Journal mit einer Anzahl von ungefähr 200 veröffentlichten Artikeln ergibt das ca. 40.000 Relationen (durch die Relation in beide Richtungen müssen nur ca. 20.000 berechnet und gespeichert werden). Die Anzahl der möglichen Relationen berechnet sich folgendermaßen:

$$\sum_{i=1}^{n} n-i \tag{21}$$

n: *Anzahl Artikel*
i: *berechnete Relationen*

Für den ersten Artikel von n Artikeln werden n-1 Relationen berechnet. Für den nächsten Artikel müssen bereits nur mehr n-2 Relationen berechnet werden, denn die Relation mit dem vorherigen Artikel wurde schon im vorherigen Durchlauf berechnet.
Bei einer beispielhaften Menge von fünf Artikeln wäre die Anzahl der zu berechnenden Relationen nach Formel 21: $(5-1)+(5-2)+(5-3)+(5-4)+(5-5) = 10$

Es werden alle notwendigen Relationen berechnet, jedoch nur Relationen mit einer Distanz größer „0“ in der Datenbank gespeichert. Ein Beispiel für die Anwendung der beschriebenen Methoden wird in Kapitel 5.4.5 behandelt.

Nach der Bestimmung und Speicherung der Relationen zwischen den Artikeln ist die Analyse der bestehenden Texte abgeschlossen und es können nun inhaltsbasierte Empfehlungen generiert werden. Dieser Vorgang wird in Kapitel 5.4.8 beschrieben.

Ein Journal ist nun aber kein fixer Datenbestand, es werden laufend neue Ausgaben des Journals mit zusätzlichen Artikeln erstellt werden. Um nun für einen neuen Artikel auch

Empfehlungen erstellen zu können, wird er analysiert und die Resultate der Berechnung seiner TF-IDF Werte in der Datenbank hinzugefügt. Dann werden nur die Distanzen der Artikel im System zum neuen Artikel berechnet und gespeichert, ohne etwas an den bestehenden Distanzen zu ändern. Dieser Vorgang wird von der Methode `updateWordArticleTable-WithArticleId()` erledigt.

Ein neuer Artikel beeinflusst die bestehenden Artikel dahingehend, dass er die gespeicherten TF-IDF-Werte beeinflusst und somit in weiterer Folge alle Distanzen zwischen den Artikeln verändert. In die Berechnung des IDF Teiles fließen die Anzahl der Dokumente im System und die Anzahl der Dokumente, die das aktuelle Wort enthalten, ein. Somit ändert ein neuer Artikel alle TF-IDF Werte einerseits durch die Erhöhung der Gesamtanzahl der Dokumente, andererseits durch die Erhöhung der Anzahl der bereits gespeicherten Wörter welche auch im neuen Text vorkommen.
Ein einzelner neuer Artikel wird die bestehenden Beziehungen aber nicht sehr stark beeinflussen – ähnliche Artikel sind auch weiterhin ähnlich zueinander – jedoch werden die Berechnungen für die Beziehungen neuer Artikel immer ungenauer. Aus diesem Grund ist es empfehlenswert, in periodischen Abständen die Datenbank erneut aufzubauen. Der Administrator des Systems wird dann auf der Management-Seite des Plugins informiert, dass eine Neuberechnung der Tabellen für das Content-Based-Filtering zu besseren Ergebnissen führt. Durch einen Link wird der Installationszyklus erneut ausgelöst, wie in Kapitel 5.4.2 Abbildung 17 dargestellt.

5.4.4 Textanalyse

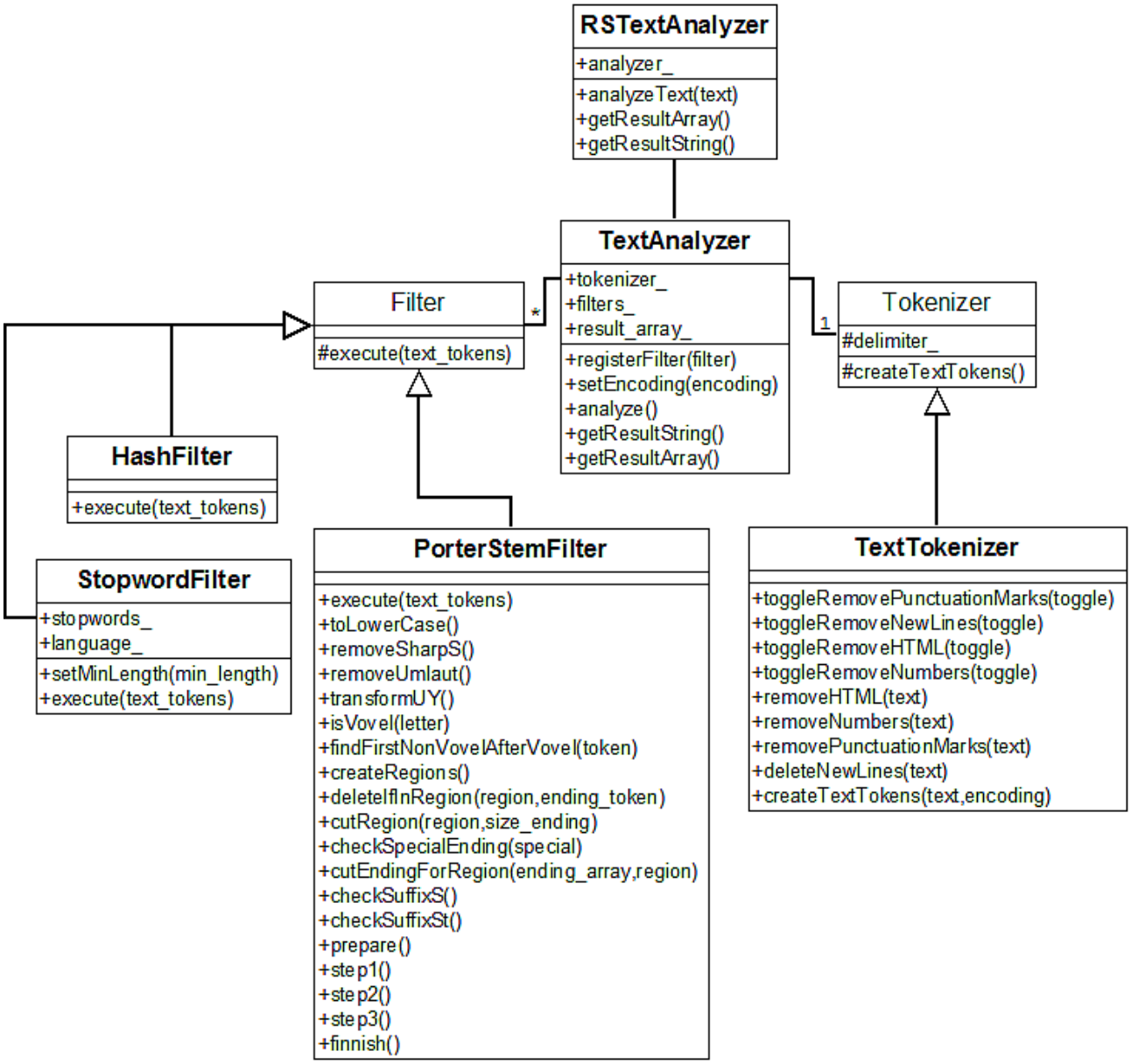


Abbildung 20: Klassendiagramm: Textanalyse

Das Analysieren eines Textes wird von einer selbst geschriebenen Methodenbibliothek erledigt, deren Struktur in Abbildung 20 dargestellt wird. Ziel dieser Bibliothek ist es, mit einem einzigen Aufruf einen Text in seine Wörter zerlegen zu können und mit diesen Wörtern verschiedene Operationen durchführen zu können. Die Reihenfolge der Operationen kann dabei selbst festgelegt werden. Erreicht wird dieses Verhalten durch die Verwendung von einzelnen Filterklassen, die in beliebiger Reihenfolge kombiniert werden können. Festgelegt wird die Reihenfolge im Konstruktor der Klasse „TextAnalyzer“. Dabei werden die einzelnen Filterobjekte in einer Liste gespeichert, die dann von der Analyseklasse abgearbeitet wird. In den folgenden Kapiteln werden die einzelnen Komponenten näher beschrieben.

5.4.4.1 Tokenizerklasse

Wird ein Text in kleine Teile zerlegt, nennt man diese Teile „Tokens“. Der Klassenname „Tokenizer“ leitet sich von dieser Bezeichnung ab. Im ersten Vorverarbeitungsschritt der Textanalyse soll ein beliebiger Text in seine Wörter aufgeteilt werden. Für diese Aufgabe existiert die TextTokenizer-Klasse, die von einer Basisklasse mit dem Namen Tokenizer abgeleitet ist. Diese Basisklasse ermöglicht die Implementierung unterschiedlich konfigurierter TextTokenizer-Klassen. In der Basisklasse wird das Trennzeichen festgehalten und die abstrakte Methode `createTextTokens()` notiert, damit sichergestellt ist, dass alle abgeleiteten Klassen diese Methode implementieren.

Die TextTokenizer-Klasse stellt verschiedene Methoden zur Verfügung, um unerwünschte Zeichen und Zeichenketten aus dem zu bearbeitenden Text entfernen zu können. So entfernt die Methode `removeHTML()` alle HTML-Anweisungen aus dem Text, die Methode `removePunctuationMarks()` alle Satzzeichen und Sonderzeichen und die Methode `removeNumbers()` löscht Zahlen, die aus mehr als vier Ziffern bestehen und somit höchstwahrscheinlich keine Jahreszahlen darstellen. Für die Entfernung der Absätze und der überflüssigen Leerzeichen ist die Methode `deleteNewLines()` zuständig. Jede dieser Methoden kann über eine Anweisung ein- oder ausgeschaltet werden. Für das Ausschalten zum Entfernen von Zahlen lautet der Aufruf beispielsweise: `toggleRemoveNumbers(„OFF“)`. Standardmäßig sind aber alle Bearbeitungsmethoden aktiv.

Nach dem Aufruf dieser Bearbeitungsmethoden besteht der anfängliche Text nun nur mehr aus Wörtern, die durch ein einziges Leerzeichen getrennt sind und keine Formatierungen oder Sonderzeichen mehr enthalten. Nun kann dieser Text mit Hilfe des gespeicherten Trennzeichens in einzelne Token zerlegt werden, welche in einer Liste für die weitere Verarbeitung gespeichert werden. Als Trennzeichen wird im Empfehlungssystem-Plugin das Leerzeichen verwendet, um den Text in einzelne Wörter zu teilen. Eine alternative Anwendungsart dieser TextTokenizer-Klasse ist zum Beispiel die Teilung eines Textes in einzelne Sätze. Um das zu erreichen, muss man das Trennzeichen in einen Punkt statt einem Leerzeichen ändern.

5.4.4.2 Filterklassen

Um eine dynamische Anordnung der diversen Filterklassen zu ermöglichen, werden alle Filter von einer Basisklasse abgeleitet, die eine abstrakte Methode `execute(text_tokens)` beinhaltet. Dadurch müssen alle Klassen, die von dieser Basisklasse abgeleitet werden, diese `execute(text_tokens)` - Methode implementieren. Der Parameter „text_tokens“ steht für die zu bearbeitende Liste von Elementen. Es muss darauf geachtet werden, dass nach der Anwendung aller gewünschten Operationen in der `execute()` - Methode eine Liste mit den modifizierten Elementen retourniert wird, damit der nächste Filter die Elemente weiterverarbeiten kann. Die Idee hinter dieser Implementierung ist es, eine Kette von Filtern zu erzeugen, die in beliebiger Reihenfolge angeordnet werden können.

Der erste Filter in der Filterkette des Plugins ist die StopwordFilter – Klasse. Beim

Erzeugen dieses Filters wird eine Liste mit Stoppwörtern inklusive der jeweiligen Sprache angegeben. Ein Stoppwort ist ein kurzes Wort einer Sprache, welches für die Textanalyse von geringer Bedeutung ist, wie zum Beispiel „der“, „die“, „das“, „ich“, „du“, etc. Die Stoppwörter sind in einer zweispaltigen Liste angelegt, in welcher die erste Spalte die Sprache bezeichnet und in der zweiten Spalte das Wort notiert ist. Somit können auch mehrsprachige Stoppwortlisten verwendet werden. Eine zusätzliche Funktion dieses Filters ist das Entfernen von Worten unter einer bestimmten Länge. Worte, die aus drei oder weniger Zeichen bestehen werden nicht verwendet. Anschließend werden alle Wörter, die auf der Stoppwortliste notiert sind, entfernt. Als Resultat des Filters erhält man eine Liste von Worten, die aus mindestens vier Zeichen bestehen und keine Stoppwörter sind.

Die Ausgabe des Stoppwortfilters dient als Eingabe für den nächsten Filter, nämlich die PorterStemFilter – Klasse. Stemming wurde in Kapitel 4.1.4 erklärt. Die Methoden dieser Klasse implementieren den Algorithmus Schritt für Schritt, wie er auf der Webseite http://snowball.tartarus.org[12] beschrieben ist.
Jedes Element der zu bearbeitenden Liste durchläuft den Algorithmus und wird auf den Wortkern laut Porter-Stemming-Algorithmus reduziert. Diese neue Liste repräsentiert die Ausgabe dieses Filters.

Der letzte Filter in der Filterkette des Plugins ist ein Hashfilter. Dieser bildet, wie der Name schon sagt einen Hash, also ein Array in der Form: Schlüssel → Wert. Der Schlüssel des Wertepaares ist das untersuchte Element (in Falle des Plugins der gebildete Wortkern) und der Wert ist die Häufigkeit des Vorkommens. Der Filter überprüft für jedes Element, ob in der Ergebnisliste bereits ein Schlüssel mit dem Namen des Elementes enthalten ist. Falls dem so ist, wird der Wert dieses Schlüssels um „1“ erhöht, anderenfalls wird der Schlüssel zusammen mit dem Wert „1“ an die Ergebnisliste angefügt. Somit erhält man als Resultat einen Hash, der die im Ausgangstext vorkommenden Wortkerne und die dazugehörigen Häufigkeiten abbildet.

5.4.4.3 TextAnalyzer-Klasse

Diese Klasse ist die Verbindung der bisher beschriebenen Elemente. Bei der Erstellung wird ein Tokenizer-Objekt gespeichert, welches für die Aufteilung des Textes zuständig ist. Um diverse Filter zu registrieren gibt es die Methode `registerFilter()`, welche einen Filter an eine bestehende Liste mit Filtern anhängt. Diese Liste stellt die Filterkette dar und wird in der `analyze()` Methode verwendet. Diese Methode verwendet nun alle bisher beschriebenen Objekte. Als erstes wird mit dem Tokenizer-Objekt eine Liste von einzelnen Elementen erzeugt. Diese Elemente werden an die Filterkette weitergereicht, wo jeder Filter in der Reihenfolge seiner Registrierung auf die Elementliste angewendet wird. Das Resultat dieses Vorgangs wird gespeichert und kann mit Hilfe von zwei „get“-Methoden abgerufen werden:
`getResultArray()` und `getResultString()`, wobei die erste Methode die Liste der Elemente nach der Anwendung der Filterkette zurück gibt und die zweite Methode diese Liste zuerst in eine Zeichenkette umwandelt und dann zurück liefert.

12 http://snowball.tartarus.org/algorithms/german/stemmer.html (Abruf 1. 8. 2011)

5.4.5 Beispiel: Installation der Content-Based-Filtering Tabellen

Die Erstellung der benötigten Tabellen wurde in den vorangegangenen Kapiteln beschrieben und soll nun mit Hilfe eines kleinen Beispiels demonstriert werden. Algorithmen, die Inhalte analysieren, sind von der Länge der Eingabedaten und somit von der Anzahl der Datenpunkte abhängig [1]. Die Texte des Beispiels sind absichtlich sehr kurz gehalten, um den Rechenweg zu vereinfachen.

Text 1: *„Empfehlungssysteme analysieren das Verhalten von Benutzern.“*

Text 2: *„Benutzer verhalten sich nicht wie vorhergesagt.“*

Text 3: *„Der Algorithmus benötigt eine gewisse Zeit.“*

Im ersten Schritt werden die Texte mit Hilfe der Textanalyse-Klassen bearbeitet, zerlegt und in eine temporäre Tabelle gespeichert. Anhand von Text 1 wird dieser Vorgang einmal durchgeführt:

1.) Tokenizer:
Eingabe: Empfehlungssysteme analysieren das Verhalten von Benutzern.
Ausgabe: Liste (Empfehlungssysteme, analysieren, das, Verhalten, von, Benutzern)

2.) Stoppwortfilter:
Eingabe: Liste (Empfehlungssysteme, analysieren, das, Verhalten, von, Benutzern)
Ausgabe: Liste (Empfehlungssysteme, analysieren, Verhalten, Benutzern)

3.) PorterStemFilter:
Eingabe: Liste(Empfehlungssysteme, analysieren, Verhalten, Benutzern)
Ausgabe: Liste(empfehlungssystem , analysi , verhalt , benutz)

4.) HashFilter:
Eingabe: Liste(empfehlungssystem , analysi , verhalt , benutz)
Ausgabe: Liste((empfehlungssystem, 1) , (analysi, 1) , (verhalt, 1) , (benutz, 1))

Dies wird für alle drei Texte durchgeführt und die Ergebnisse in die temporäre Wort-Artikel-Tabelle geschrieben. Das Ergebnis wird in Tabelle 6 dargestellt.

Die Spalte „id“ in Tabelle 6 bezeichnet die interne Nummer des Wortes innerhalb der Datenbanktabelle. In der Spalte „word“ ist der reduzierte Wortkern zu finden, „article_id“ notiert die Nummer des Artikels, in welchem der Wortkern vorkommt und „count“ steht für die Anzahl des Wortkerns im Artikel. Somit hat man alle Daten um im nächsten Schritt die TF-IDF Werte eines jeden Wortkerns ausrechnen zu können.

id	word	article_id	count
1	empfehlungssystem	1	1
2	analysi	1	1
3	verhalt	1	1
4	benutz	1	1
5	benutz	2	1
6	verhalt	2	1
7	vorhergesagt	2	1
8	algorithmus	3	1
9	benotigt	3	1
10	gewiss	3	1
11	zeit	3	1

Tabelle 6: Temporäre Wort-Artikel-Tabelle

Da für die Berechnung der Inversen-Dokument-Häufigkeit die Anzahl eines Wortes in allen Dokumenten gebraucht wird, kann die Berechnung der TF-IDF Werte erst dann erfolgen, wenn alle Wörter in die Datenbank eingetragen wurden. Um dennoch etwas Zeit zu sparen, werden die Datensätze bei den Abfragen alphabetisch geordnet. Dadurch muss für identische Wörter die Häufigkeit ihres Vorkommens nur einmal berechnet werden.

Die Berechnung des TF IDF Wertes für den Wortkern „benutz“ in Artikel 1:

$$tf_{word} = \#word\ in.document \tag{22}$$

$$idf_{word} = \log\left(\frac{\#documents}{\#documents\,with\,word}\right) \tag{23}$$

TF Wert für „benutz“ in Artikel 1 ist also die Häufigkeit des Wortes in Artikel 1 – dieser Wert ist in der Spalte „count“ von Tabelle 6 notiert. Das Wort kommt genau einmal vor, also ist der TF Wert auch „1“.

Spannender wird es da schon beim IDF-Wert. Dieser muss aus den Daten der Datenbank berechnet werden. Die Menge der Dokumente muss nur einmal für alle Wörter bestimmt werden, in diesem Fall existieren drei Dokumente. Das Wort „benutz“ kommt nun in allen analysierten Dokumenten insgesamt zweimal vor. In die Formel 23 für den IDF-Wert eingesetzt ergibt das:

$$idf_{benutz} = \log\left(\frac{3}{2}\right) = 0{,}405$$

Dieser Wert wird nun mit dem TF-Wert multipliziert:

$$tfidf_{benutz} = 0{,}405 \cdot 1 = 0{,}405$$

Nach der Berechnung aller TF-IDF Werte werden die Daten aus der temporären Tabelle in die endgültige Wort-Artikel-Tabelle geschrieben, zusammen mit den dazugehörigen TF-IDF Werten. Diese Form der Aktualisierung mit Hilfe einer temporären Tabelle begründet sich durch die viel höhere Geschwindigkeit einer Datenbank beim Einfügen großer Datenmengen im Gegensatz zur Aktualisierung von bestehenden Datenbanktabellen. Das Einfügen von ungefähr 15.000 Datensätzen in die endgültige Wort-Artikel-Tabelle benötigt wenige Sekunden, da es möglich ist viele Datensätze in einer INSERT-Anweisung zusammenzufassen. Eine Aktualisierung hingegen benötigt für jeden einzelnen Datensatz eine eigene UPDATE-Anweisung, die zeitintensiv ist.

Das Ergebnis der Berechnungen für die TF-IDF-Werte für die drei Beispielartikel ist in Tabelle 7 dargestellt. Alle Wörter außer „benutz“ und „verhalt“ erhalten das gleiche Gewicht, weil sie im Text jeweils nur einmal vorkommen. Je länger die Texte sind, desto besser können die Wörter gewichtet werden.

id	word	article_id	count	tf_idf
1	empfehlungssystem	1	1	1,0986
2	analysi	1	1	1,0986
3	verhalt	1	1	0,405
4	benutz	1	1	0,405
5	benutz	2	1	0,405
6	verhalt	2	1	0,405
7	vorhergesagt	2	1	1,0986
8	algorithmus	3	1	1,0986
9	benotigt	3	1	1,0986
10	gewiss	3	1	1,0986
11	zeit	3	1	1,0986

Tabelle 7: Wort-Artikel-Tabelle

Im letzten Schritt der Erzeugung der Tabellen für das Content-Based-Filtering werden die Cosinusdistanzen der einzelnen Dokumente untereinander berechnet. Bei drei Artikeln ergibt das sechs Distanzen, denn die Distanz eines Artikels zu sich selbst wird ignoriert. (Als Test für die Richtigkeit der Distanzberechnung kann die Distanz zu sich selbst aber verwendet werden, denn sie sollte „1“ ergeben). Von diesen sechs müssen nur drei betrachtet werden, da die Distanzen bidirektional sind, wie in Kapitel 5.4.3 bereits erwähnt.

Für die Distanzberechnung benötigt man zuerst die Länge des Wort-Artikel-Vektors. Für die Artikel des Beispiels ergibt das (Kapitel 5.4.3 , Formel 18):

$$\left|article_1\right| = \sqrt{1{,}0986^2 + 1{,}0986^2 + 0{,}405^2 + 0{,}405^2} = 1{,}6558$$

$$\left|article_2\right| = \sqrt{0{,}405^2 + 0{,}405^2 + 1{,}0986^2} = 1{,}2389$$

$$\left|article_3\right| = \sqrt{1{,}0986^2 + 1{,}0986^2 + 1{,}0986^2 + 1{,}0986^2} = 2{,}1972$$

Wie in Kapitel 5.4.3 beschrieben müssen für die Bestimmung des Kreuzproduktes nur die Einträge eines Vektors berücksichtigt werden, die in beiden Vektoren vorkommen. Als Vergleich zeigt Tabelle 8 die Vektoren der Artikel des Beispiels, wobei jede Spalte für den jeweiligen Wort-Artikel-Vektor steht:

Artikel 1	**Artikel 2**	**Artikel 3**
empfehlungssystem (1,0986)	benutz (0,405)	algorithmus (1,0986)
analysi (1,0986)	verhalt (0,405)	benotigt (1,0986)
verhalt (0,405)	vorhergesagt (1,0986)	gewiss (1,0986)
benutz (0,405)		zeit (1,0986)

Tabelle 8: Wort-Artikel-Vektoren in Tabellenform

Um die Berechnung des Kreuzproduktes zu verdeutlichen kann man die vollständige Wort-Artikel-Matrix aufstellen, welche in Tabelle 9 illustriert wird. Aus dieser Tabelle kann man ablesen, dass Tabelle 7 eine datenbankgerechte Schreibweise dieser vollständigen Wort-Artikel-Matrix ist.

Wortkern	**Artikel 1**	**Artikel 2**	**Artikel 3**
empfehlungssystem	1,0986	0	0
analysi	1,0986	0	0
verhalt	0,405	0,405	0
benutz	0,405	0,405	0
vorhergesagt	0	1,0986	0
algorithmus	0	0	1,0986
benotigt	0	0	1,0986
gewiss	0	0	1,0986
zeit	0	0	1,0986

Tabelle 9: Vollständige Wort-Artikel-Matrix

Die Bildung des Kreuzproduktes ist somit nur zwischen Artikel 1 und Artikel 2 möglich. Artikel 3 besitzt mit den anderen Artikeln keine gemeinsamen Werte, die Kreuzprodukte ergeben „0“ und somit ergeben auch die Distanzen 0. Das Kreuzproduktes zwischen Artikel 1 und Artikel 2 wird folgendermaßen (Kapitel 5.4.3 , Formel 19) gebildet:

$$article_1 \times article_2 = 1{,}0986 \cdot 0 + 1{,}0986 \cdot 0 + 0{,}405 \cdot 0{,}405 + 0{,}405 \cdot 0{,}405 + 0 \cdot 1{,}0986 + 0 \cdot 0 + \ldots = 0{,}32805$$

Die einzigen Produkte die nicht „0“ ergeben sind die Produkte für die Wortkerne „verhalt“ und „benutz“.

Zwischen Artikel 1 und Artikel 2 berechnet sich nun die Cosinusdistanz laut Formel 24 (siehe Kapitel 2.4.1):

$$similarity_{\cos}(A,B)=\cos(phi)=\frac{A\cdot B}{|A||B|}=\frac{\sum_{i}^{n} A_i\times B_i}{\sqrt{\sum_{i}^{n}(A_i)^2}\sqrt{\sum_{i}^{n}(B_i)^2}} \quad (24)$$

$A, B: Vektoren$

Einsetzen des Kreuzproduktes und der Längen der beiden Vektoren ergibt:

$$\frac{0{,}32805}{1{,}6558\cdot 1{,}2389}=0{,}1599$$

Dieser Wert wird nun in der Datenbanktabelle für die Distanz zwischen Artikel 1 und Artikel 2 gespeichert. Da die Kreuzprodukte der Wort-Artikel-Vektoren der Artikel 1 und 3 sowie der Artikel 2 und 3 „0“ ergeben, werden diese Werte erst gar nicht in der Datenbank gespeichert. Somit wird von vornherein ausgeschlossen, dass Elemente empfohlen werden, die überhaupt keine inhaltliche Gemeinsamkeit auf syntaktischer Ebene aufweisen. Das Erstellen von Empfehlungen für Elemente, die semantische Ähnlichkeiten aufweisen, ist die Aufgabe des Collaborative-Filterings.

5.4.6 RecommenderSystemUserHandler

Das Plugin benötigt nun Methoden für den laufenden Betrieb. Es müssen BenutzerInnen identifiziert werden, die Benutzerpfade müssen aufgezeichnet und aktualisiert werden. Für diese Aktionen gibt es die Klasse RecommenderSystemUserHandler (Abbildung 21).

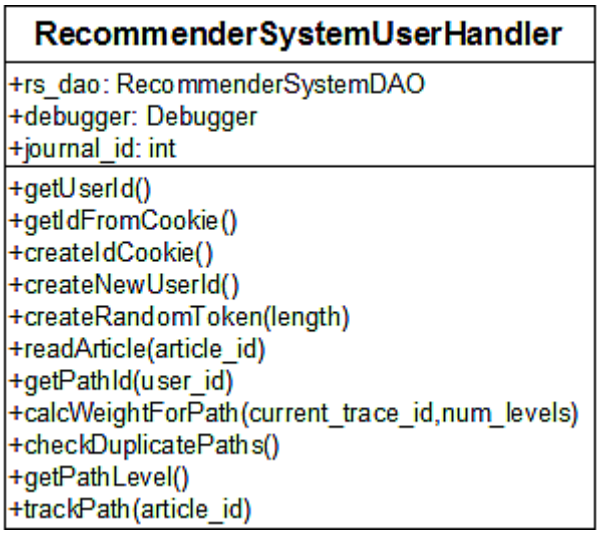


Abbildung 21: Klasse: RecommenderSystemUserHandler

Die erste Aufgabe dieser Klasse ist die Erkennung und Verwaltung von Benutzerinnen und Benutzern. Um nun jemanden zu identifizieren, wird in der aktuellen Session nachgesehen, ob die/der BenutzerIn schon bekannt ist (Methode `getUserId()`). Ist die/der BenutzerIn noch unbekannt, versucht die Methode `getIdFromCookie()` die ID der Person aus dem gespeicherten Cookie zu lesen. Existiert nun kein Cookie, oder ist die im Cookie gespeicherte ID ungültig, wird die IP-Adresse der Person mit den IP-Adressen, die in der Datenbank gespeichert sind, verglichen. Bei einem Treffer wird diese ID nun zurückgeliefert und im Cookie und in der Session gespeichert. Wird keine ID gefunden, erstellt die Methode `createIDCookie()` eine neue ID für die/den BenutzerIn und speichert diese zusammen mit der IP-Adresse der Benutzerin beziehungsweise des Benutzers in der Datenbank.

Die zweite Aufgabe dieser Klasse ist die Aufzeichnung der Benutzerpfade. Zu diesem Zweck existiert die Methode `readArticle()`, welche in der Hauptklasse des Plugins mit dem Hook für den Download eines PDF-Dokuments verbunden wird. Ein Download eines PDF-Dokuments löst die Aufzeichnung des Benutzerpfades aus. Als erstes wird in der Methode `trackPath()` überprüft, ob der aktuelle Artikel nicht bereits zuvor eingefügt wurde. Damit wird verhindert, dass ein Artikel zwei- oder mehrmals direkt hintereinander in einem Pfad vorkommt. Nachdem das ausgeschlossen wurde, wird die ID des ausgewählten Artikels an den aktuellen Pfad der Benutzerin oder des Benutzers angefügt.

Die letzten Aufgaben dieser Klasse sind das Finden von Pfadduplikaten und die Aktualisierung der Pfadgewichte. BenutzerInnen können im Laufe der Zeit identische Pfade erzeugen. Diese Pfade müssen aufgespürt und zu einem Pfad vereinigt werden, da sie ansonsten das Resultat der Empfehlungsberechnung verfälschen. Um nun nicht ständig die gesamte Datenbank der Pfade nach Duplikaten zu untersuchen, bekommen alle Pfade eine Markierung, ob sie bereits untersucht wurden. Die Methode `checkDuplicatePaths()` lädt nun alle als unbearbeitet markierten Pfade aus der Datenbank und überprüft nur diese neuen Pfade, ob es bereits einen identischen Pfad in der Datenbank gibt. Wird ein solcher identischer Pfad gefunden, wird der Zähler dieses Pfades um eins erhöht. Dann wird bei der Verbindung zwischen untersuchtem Pfad und BenutzerIn die ID des untersuchten Pfades mit der ID des gefundenen Pfad ersetzt und somit die/der BenutzerIn dem bestehenden Pfad zugewiesen. Dann wird der untersuchte Pfad gelöscht. Andererseits, wenn kein identischer Pfad für den untersuchten Pfad existiert, wird der untersuchte Pfad als „bearbeitet“ markiert.

Es wurde festgelegt, wann ein Pfad beginnt und wann er endet. Als Beginn eines Pfades wurde der Beginn einer neuen Session bestimmt. Innerhalb einer Session wird der Pfad einer Benutzerin beziehungsweise eines Benutzers aufgezeichnet. Um das Auffinden von Pfadduplikaten zu automatisieren, löst die Erstellung einer neuen Session die zuvor beschriebene Methode `checkDuplicatePaths()` aus. Damit nun kein Pfad unterbrochen wird, werden nur Pfade untersucht, die mindestens einen Tag alt sind. Somit wird den Benutzerinnen und Benutzern genug Zeit gegeben, den Pfad zu vervollständigen.

Das Kernstück der Empfehlungsberechnung mit Hilfe von Benutzerpfaden ist die Berechnung der Pfadgewichte. Als einzig mögliche Stelle für die Durchführung dieser Berechnung ist direkt im Anschluss an die Prüfung auf Duplikate. Einerseits wird durch das Löschen der identischen Pfade verhindert, dass unnötige Berechnungen vorgenommen werden, andererseits bekommt so jeder Pfad sicher ein Gewicht. Die Berechnung der Gewichte wird, wie in Kapitel 4.2.3 beschrieben, in der Methode `calcWeightForPath()` implementiert. Zuerst werden alle Elemente eines Pfades aus der Datenbank geladen und als aufeinanderfolgende Elementpaare betrachtet. Für jedes dieser Elementpaare wird überprüft, ob dieses Paar in der Tabelle für die Bestrafung der Pfadrelationen notiert ist. Jedes Vorkommen eines Paares wird gezählt und diese Summe schließlich durch die Anzahl aller Elemente eines Pfades dividiert. Das Ergebnis wird von „1“ abgezogen und ergibt dann die Gewichtung dieses Pfades.

Nachdem ein Pfad als bearbeitet markiert, sein Gewicht berechnet und in der Datenbank gespeichert wurde, kann er für die Berechnung von Empfehlungen verwendet werden.

5.4.7 RecommenderSystemDAO

In der Klasse RecommenderSystemDAO (Abbildung 22) sind alle Methoden gesammelt, die dazu benötigt werden, um die Datenbank des Plugins im laufenden Betrieb zu verwenden.
Die benötigten Daten für das Erkennen, Überprüfen und Anlegen einer Benutzerin oder eines Benutzers werden mit den jeweiligen Methoden aus der Datenbank gelesen und zurückgeliefert.
Die meisten der hier notierten Methoden sind reine Datenmanipulationsmethoden. Das heißt sie nehmen Parameter entgegen und speichern sie in der Datenbank oder sie lesen Daten aus der Datenbank und liefern diese Daten zurück.
Beispielsweise liefert die Methode `getNearestArticleIds()` eine frei wählbare Anzahl von Artikeln, geordnet nach ihrem Abstand zu einem bestimmten Artikel. Auf Grund der Vorverarbeitungsschritte bei der Einrichtung der Content-Based-Filtering Tabellen (Kapitel 5.4.3 und 5.4.4) ist die Erzeugung von Empfehlungen in Bruchteilen einer Sekunde nur durch die Verwendung dieser einen Methode möglich. Die Qualität der Empfehlungen hängt natürlich von den bei der Textanalyse angewandten Methoden ab.

Abbildung 22: Klasse: RecommenderSystemDAO

Neben den Methoden für das Content-Based-Filtering sind in dieser Klasse vor allem die Methoden für die Verwaltung und Auswertung von Benutzerpfaden gesammelt. Dies umfasst Methoden für das Anlegen und Löschen von Pfaden, für die Änderung der Gewichte, für die Erkennung von Duplikaten und natürlich für die Auswahl der Pfade, die zur Erstellung von Empfehlungen verwendet werden.

Für das Auffinden von Duplikaten von Pfaden existiert die Methode `findDuplicatePaths()`, die wie der Name schon ausdrückt, identische Pfade für einen bestimmten Pfad aufspürt. Als erstes werden alle Elemente des zu untersuchenden Pfades aus der Datenbank gelesen. Um nun ein Duplikat zu finden wird der Pfad Element für Element mit den gespeicherten Pfaden verglichen. Jeder Durchlauf schränkt die Menge der Ergebnisse weiter ein. Der erste Durchlauf liefert alle Pfade, die dieselbe Länge wie der zu untersuchende Pfad aufweisen und an der ersten Stelle identisch mit diesem Pfad sind. Der zweite Durchlauf befasst sich nur mehr mit den Pfaden des Ergebnisses des ersten Durchlaufes und vergleicht die zweite Stelle der Pfade. Dies wird so lange wiederholt, bis keine Pfade mehr als Ergebnis vorhanden sind oder bis der zu

untersuchende Pfad vollständig durchlaufen wurde. Das Ergebnis des letzten Durchlaufes sind nun die Menge an Pfaden, die mit dem zu untersuchenden Pfad vollkommen identisch sind.
Ein Beispiel für die Suche nach identischen Pfaden für den Pfad $P=\{A\rightarrow B\rightarrow C\rightarrow D\}$ sieht folgendermaßen aus:

Folgende Pfade sind im System gespeichert:

$P_1=\{A\rightarrow B\rightarrow C\rightarrow D\}$
$P_2=\{B\rightarrow C\rightarrow D\}$
$P_3=\{A\rightarrow B\rightarrow C\rightarrow E\}$
$P_4=\{A\rightarrow D\rightarrow E\rightarrow B\}$

Suche alle Pfade mit gleicher Länge zu P und deren erste Stelle identisch mit der ersten Stelle in Pfad P ist *(P[0] = A)*.

Resultat: $R=\{P_{1,}P_{3,}P_4\}$

Suche in R alle Pfade deren zweite Stelle identisch mit P an der zweiten Stelle ist *(P[1] = B).*

Resultat*:* $R=\{P_{1,}P_3\}$

Suche in R alle Pfade deren dritte Stelle identisch mit P an der dritte Stelle ist *(P[2] = C).*

Resultat: $R=\{P_{1,}P_3\}$

Suche in R alle Pfade deren vierte Stelle identisch mit P an der vierten Stelle ist *(P[3] = D).*

Resultat: $R=\{P_1\}$

Pfad P hat keine weiteren Elemente, somit ist das Resultat R die Menge aller zu Pfad P identischen Pfade $P\equiv R\equiv\{P_1\}$.
Wie zuvor beschrieben werden die Verbindungen von P in der Datenbank durch P_1 ersetzt, der Zähler von P_1 wird um ein erhöht, und als letztes wird P gelöscht.

Für die Berechnung von Empfehlungen ist es notwendig, herauszufinden, ob sich die/der BenutzerIn auf einem bereits gespeichertem Pfad bewegt. In der Praxis funktioniert dies ähnlich wie das Finden von Duplikaten, bis auf den Unterschied, dass die Länge nicht verglichen wird. Ansonsten wird wieder der zu untersuchende Pfad Element für Element mit den Pfaden der Datenbank verglichen. Wurde der Pfad bis zu seinem Ende verglichen, sind die im Resultat R verbliebenen Pfade die Menge der Pfade, aus denen eine Empfehlung erstellt werden kann. Um den Berechnungsaufwand zu reduzieren, wird das Resultat R in einer Sessionvariable gespeichert. Somit muss nur mehr eine Datenbankabfrage durchgeführt werden.
Die gefundene Menge an Pfaden wird dann nach ihrem Gewicht geordnet und aus den Pfaden mit den größten Gewichten werden die nachfolgenden Elemente vorgeschlagen. Angenommen der zu untersuchende Pfad besitzt sechs Elemente, dann wird das siebente Element aus den Pfaden mit dem größten Gewicht vorgeschlagen. Anschließend werden

etwaige Duplikate aus dem Ergebnis entfernt. Für die Implementierung dieses Verhaltens auf der Datenbankebene sind die Methoden `findAllTracesForPath()` und `findArticleIdsFromTraces()` zuständig.

Da es nun passieren kann, dass kein passender identischer Pfad in der Datenbank verfügbar ist, greift der Algorithmus auf eine weitere Anwendung des Collaborative-Filterings zurück. Als erstes werden alle Pfade gesucht, die die Elemente des zu untersuchenden Pfades beinhalten. Das Prinzip ist dasselbe wie bei dem Aufspüren von identischen Pfaden, nur wird hier die Position der Elemente im Pfad nicht berücksichtigt. Somit erhält man eine viel größere Grundmenge an Pfaden. Jeder Pfad aus dieser gefundenen Menge muss nun mit dem zu untersuchenden Pfad verglichen werden. Als Distanzmaß kommt auch hier die Cosinusdistanz zum Einsatz. Im Gegensatz zu den Vektoren des Beispiels in Kapitel 5.4.5 bestehen die Vektoren hier aus den einzelnen Elementen eines Pfades. Um einen Pfad vergleichen zu können, muss auch die Anzahl der einzelnen Elemente ermittelt werden. Ein Pfad $P_1 = A \rightarrow B \rightarrow C \rightarrow D \rightarrow B$ wird umgeformt zu $P_1 = \{A(1), B(2), C(1), D(1)\}$ und kann dann dann für die Berechnung der Cosinusdistanz verwendet werden.

Als Beispiel die Berechnung der Distanz zwischen P_1 und dem Pfad $P_2 = A \rightarrow B \rightarrow E$:

$$P_1 = \{A(1), B(2), C(1), D(1)\} \qquad Länge: \sqrt{(1^2+2^2+1^2+1^2)} = 2{,}6458$$

$$P_2 = \{A(1), B(1), E(1)\} \qquad Länge: \sqrt{(1^2+1^2+1^2)} = 1{,}732$$

$$P_1 \times P_2 = (P_{1A} \cdot P_{2A} + P_{1B} \cdot P_{2B} + P_{1C} \cdot P_{2C} + P_{1D} \cdot P_{2D} + P_{1E} \cdot P_{2E}) = (1 \cdot 1 + 2 \cdot 1 + 1 \cdot 0 + 1 \cdot 0 + 0 \cdot 1) = 3$$

$$similarity_{\cos} = \frac{3}{2{,}6458 \cdot 1{,}732} = 0{,}6547 \qquad \text{(Eingesetzt in Gleichung 3)}$$

Diese Umformungen und Berechnungen werden mit allen zu vergleichenden Pfaden durchgeführt. Nach der Ermittlung der Distanzen zu dem Pfad der Benutzerin beziehungsweise des Benutzers werden die Pfade mit der geringsten Distanz – oder anders betrachtet – mit der größten Ähnlichkeit für die Erstellung einer Empfehlung verwendet. Dabei wird aus jedem dieser Pfade das nachfolgende Element des Elements, für das eine Empfehlung gesucht wurde, bestimmt und dem Ergebnis hinzugefügt. Angenommen der bisher aufgezeichnete Pfad entspricht dem Pfad $P_2 = A \rightarrow B \rightarrow E$ und die/der BenutzerIn wählt das Element D aus. Nun werden die Pfade mit der größten Ähnlichkeit zu $P_{user} = (A \rightarrow B \rightarrow E \rightarrow D)$ gesucht und aus diesem Ergebnis die Elemente empfohlen, die auf das Element D folgen.

Dieses Verhalten wird in der Methode `getNearesttArticleFromAllTraces()` implementiert. Um den Berechnungsaufwand zu minimieren, werden die ID-Nummern der gefundenen Pfade in einer Sessionvariable gespeichert, sodass bei einem neuen Element des Benutzerpfades nur die Schnittmenge zwischen den bereits gespeicherten Pfad-IDs und den Pfad-IDs, die das neue Element beinhalten, gebildet werden muss. Das Ergebnis ist dann die Menge an Pfaden, die die Elemente des bisherigen Benutzerpfades plus das neue Element beinhalten. Für den Beispielpfad $P_{user} = A \rightarrow B \rightarrow E$ und das neue Element D sind das alle Pfade, die die Elemente (A, B, E, D) enthalten.

5.4.8 RecommenderSystemRecommendations

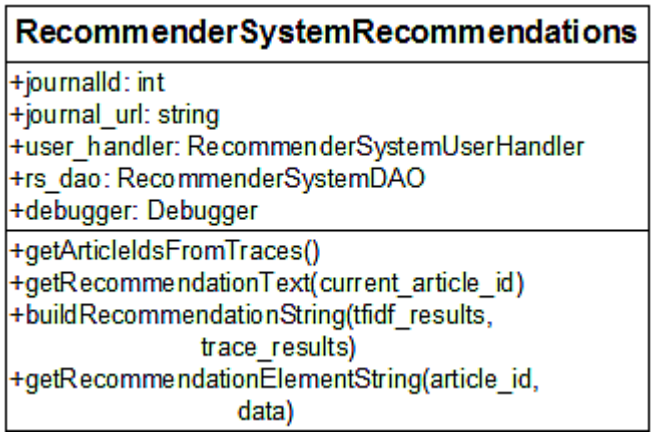


Abbildung 23: Klasse: RecommenderSystemRecommendations

Diese Klasse ist für die Steuerung der Berechnungen zur Empfehlungserstellung verantwortlich. Mit dem Aufruf der Methode `getRecommendationText()` wird die Erstellung der Empfehlungen gestartet und das Resultat als HTML-Zeichenkette zurückgeliefert. Die Methode erwartet als Parameter die ID eines Artikels. Für diesen Artikel werden in der Datenbank die sechs Artikel mit der geringsten Distanz gesucht und in einer Liste gespeichert.
Anschließend werden die Resultate des Collaborative-Filterings erstellt. Dabei wird als erstes mit Hilfe der Methode `getArticleIdsFromTraces()` versucht, passende Pfade zum Pfad der Benutzerin oder des Benutzers zu finden. Wenn die/der BenutzerIn noch keinen Pfad in seiner aktiven Sitzung generiert hat, wird versucht den letzten generierten Pfad der Benutzerin oder des Benutzers zu laden. Existiert auch kein älterer Pfad der Benutzerin oder des Benutzers in der Datenbank, gibt es keine Vergleichsmöglichkeit und es wird eine leere Liste zurückgeliefert. Wird nun jedoch ein Pfad gefunden, sucht der Algorithmus nach allen Pfaden in der Datenbank, die mit dem gefundenen Pfad beginnen. Diese Ergebnispfade werden nach ihrem Gewicht geordnet und aus dem Pfad mit dem größten Gewicht werden die Nummern der Artikel zurückgeliefert, die als nächstes im Ergebnispfad notiert sind.
Als zusätzliches Ergebnis wird noch in allen Pfaden nach den ähnlichsten Pfaden zum aktuellen Pfad der Benutzerin beziehungsweise des Benutzers gesucht. Aus dieser Menge werden wiederum die Nachfolger des gesuchten Elementes zurückgeliefert.

Nachdem diese Daten gesammelt wurden, werden noch Duplikate aus den Empfehlungen entfernt. Dann sorgt der Aufruf der Methode `buildRecommendationString()` für die Ausgabe der Empfehlungen. Als Parameter erhält diese Methode zwei Listen, erstens die Resultate des Content-Based-Filterings und zweitens die Resultate der Pfadsuche. Die erste Liste enthält sechs Elemente, die immer angezeigt werden können. Nun wird aus den drei Ergebnislisten (Content-Based-Filtering, Collaborative-Filtering mit bekannten Pfaden und Collaborative-Filtering durch die Distanz der Pfade) eine Liste mit Empfehlungen erstellt. Als erstes werden die Ergebnisse des Collaborative-Filterings mit bekannten Pfaden eingetragen. Gibt es weniger als sechs Resultate, dann wird die Liste mit den Empfehlungen des Collaborative-Filterings der Distanz der Pfade aufgefüllt. Sind anschließend wiederum weniger als sechs Elemente in der Liste, dann wird mit den Elementen des Content-Based-Filterings ergänzt.

Zusammenfassend kann gesagt werden:
Sobald Ergebnisse des Collaborative-Filterings verfügbar sind, werden diese angezeigt. Ansonsten greift der Algorithmus auf die Resultate des Content-Based-Filterings zurück.

5.4.9 Ablaufdiagramm: Benutzeridentifikation

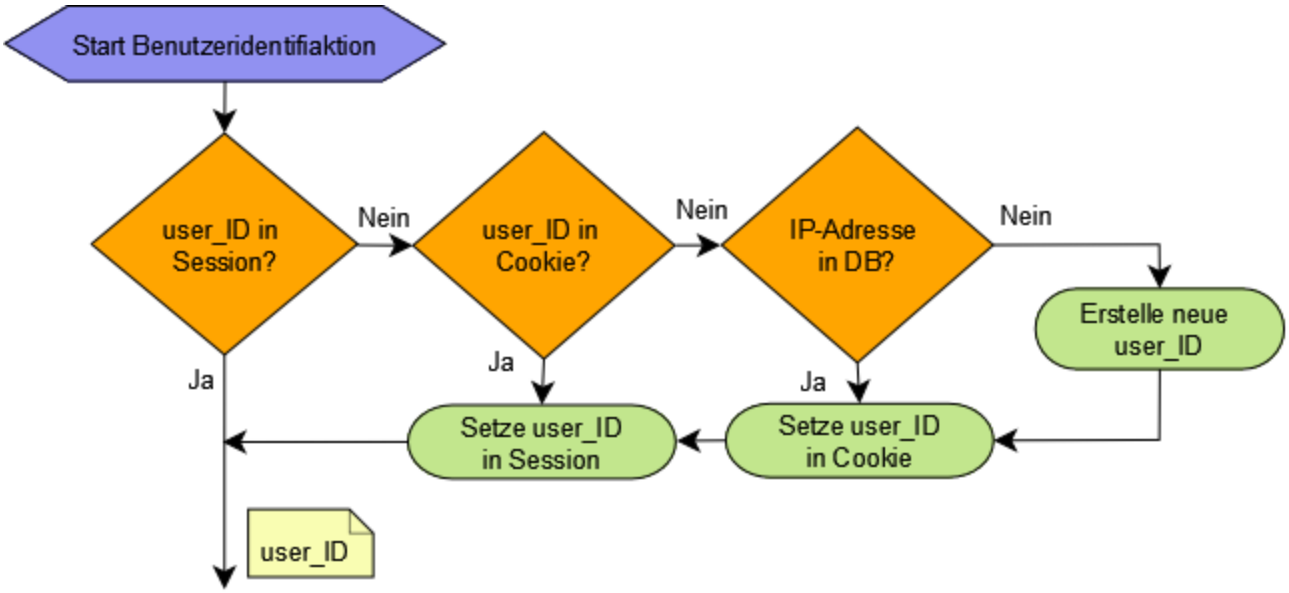


Abbildung 24: Benutzeridentifikation

Abbildung 24 zeigt den Vorgang zur Erkennung einer Benutzerin beziehungsweise eines Benutzers. Die Benutzeridentifikation wird bei jedem Aufruf einer Seite durchgeführt, um die BenutzerInnen erkennen zu können und somit die Möglichkeit zu bieten, Empfehlungen aus ihren gespeicherten Pfaden zu erzeugen, auch wenn sie während der aktuellen Sitzung noch keinen Pfad erzeugt haben. Andererseits wird es durch die Erkennung möglich, den Pfad der Benutzerin oder des Benutzers aufzuzeichnen und diese Daten zu Erstellung von Empfehlungen zu verwenden. In Abbildung 24 kann man die Entscheidungen ablesen, die dem System die Benutzer – ID liefern.

Die erste Abfrage richtet sich an die aktuelle Session. Wurde die/der BenutzerIn bereits erkannt, wird die in der Sessionvariable gespeicherte ID zurückgeliefert. Wurde jedoch noch keine Variable gesetzt, wird überprüft, ob ein Cookie mit der Benutzer-ID gespeichert ist. Hat das System ein solches Cookie gefunden, wird die ID aus dem Cookie gelesen, in der Sessionvariable gespeichert und anschließen zurückgeliefert.

Existiert nun auch kein Cookie, vergleicht das System die IP-Adresse der Benutzerin oder des Benutzers mit gespeicherten IP-Adressen in der Datenbank. Scheint die gesuchte IP-Adresse in der Datenbank auf, wird die dazu gespeicherte Benutzer-ID zurückgeliefert, ein Cookie mit dieser ID erzeugt und die Sessionvariable gesetzt. Existieren nun gar keine passenden Aufzeichnungen über die aktuelle Benutzerin oder den aktuellen Benutzer, wird eine neue Benutzer-ID erstellt, diese zusammen mit der IP-Adresse in der Datenbank gespeichert und anschließend in Cookie und Session festgehalten. Somit sollten die meisten wiederkehrenden BenutzerInnen erkannt werden können.

5.4.10 Ablaufdiagramm: Lesen eines Artikels

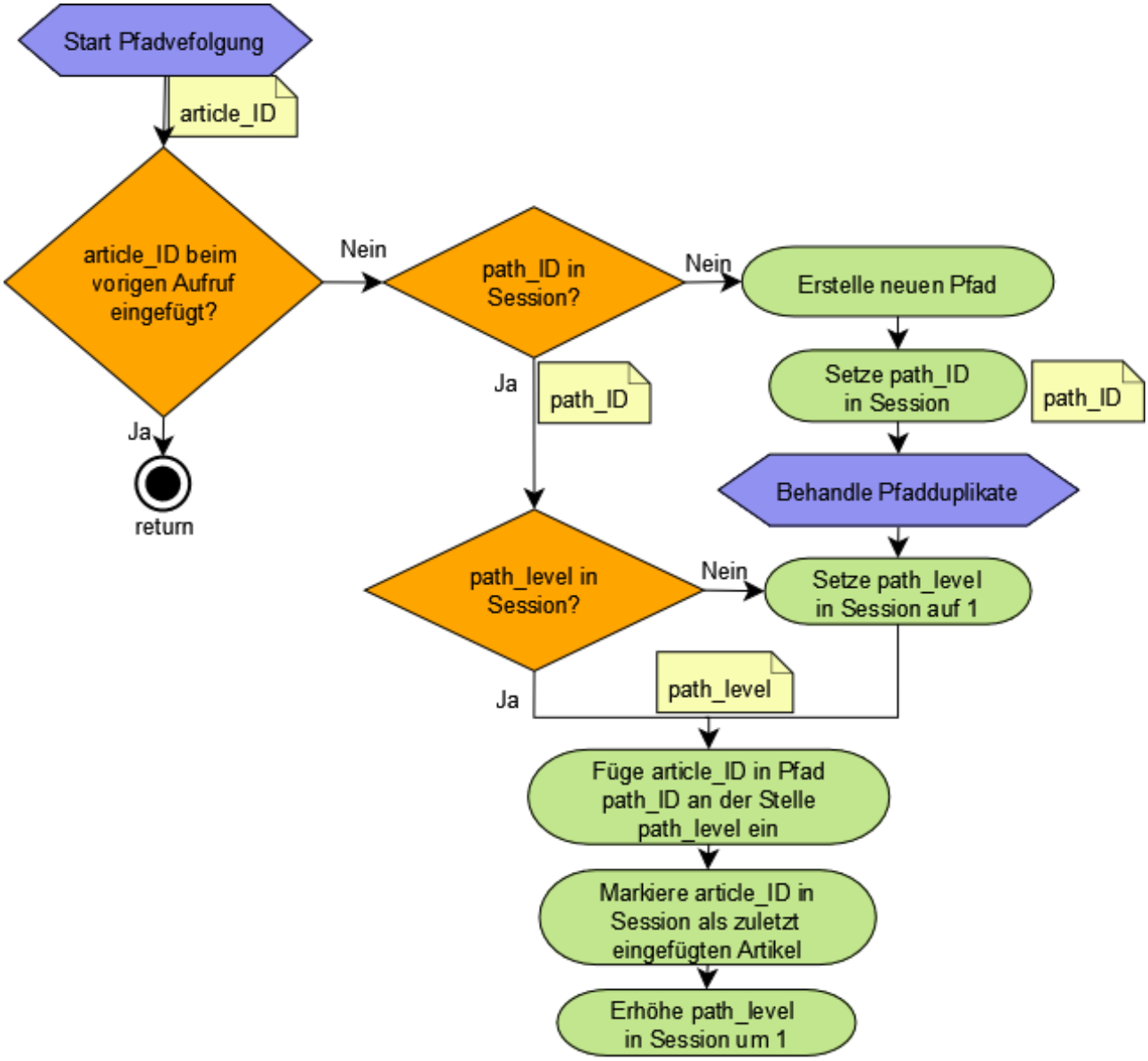


Abbildung 25: Lesen eines Artikels - Pfadaufzeichnung

Der Download eines PDF-Dokuments löst den Vorgang der Pfadverfolgung aus. Dabei wird mit der ID des Artikels überprüft, ob diese ID bereits in der aktuellen Session als zuletzt eingefügtes Element gekennzeichnet ist. Diese Kennzeichnung verhindert ein doppeltes Einfügen derselben ID direkt hintereinander. Falls der zuletzt eingefügte Artikel identisch mit dem aktuellen Artikel ist, wird keine weitere Aktion ausgeführt. Ist der aktuelle Artikel nicht identisch mit dem zuletzt eingefügten, dann beginnt der Vorgang der Pfadaufzeichnung mit dem Versuch, die ID des Pfades der Benutzerin oder des Benutzers aus der Session zu lesen. Ist so ein Pfad notiert, dann wird versucht die Stelle, an welcher der Artikel eingefügt werden soll, aus der Session zu lesen. Andererseits, wenn kein Pfad notiert ist, muss für die/den BenutzerIn ein neuer Pfad angelegt werden. Die ID dieses Pfades wird in der Session vermerkt. An dieser Stelle wird nun die Behandlung von Duplikaten gestartet, wie in Kapitel 5.4.6 und 5.4.7 beschrieben.

Nachdem alle Duplikate gefunden und behandelt wurden, wird die Stelle für den Artikel im neuen Pfad in der Session abgelegt. Da es sich um einen neuen Pfad handelt, ist der Wert dieser Variable „1“.
Nun kann der Artikel in den identifizierten Pfad an die jeweilige Stelle geschrieben werden. Dann wird die Variable für die Stelle des Pfades um eins erhöht, und der aktuelle Artikel wird in der Session als zuletzt eingefügter Artikel vermerkt.

5.4.11 Ablaufdiagramm: Erstellung von Empfehlungen

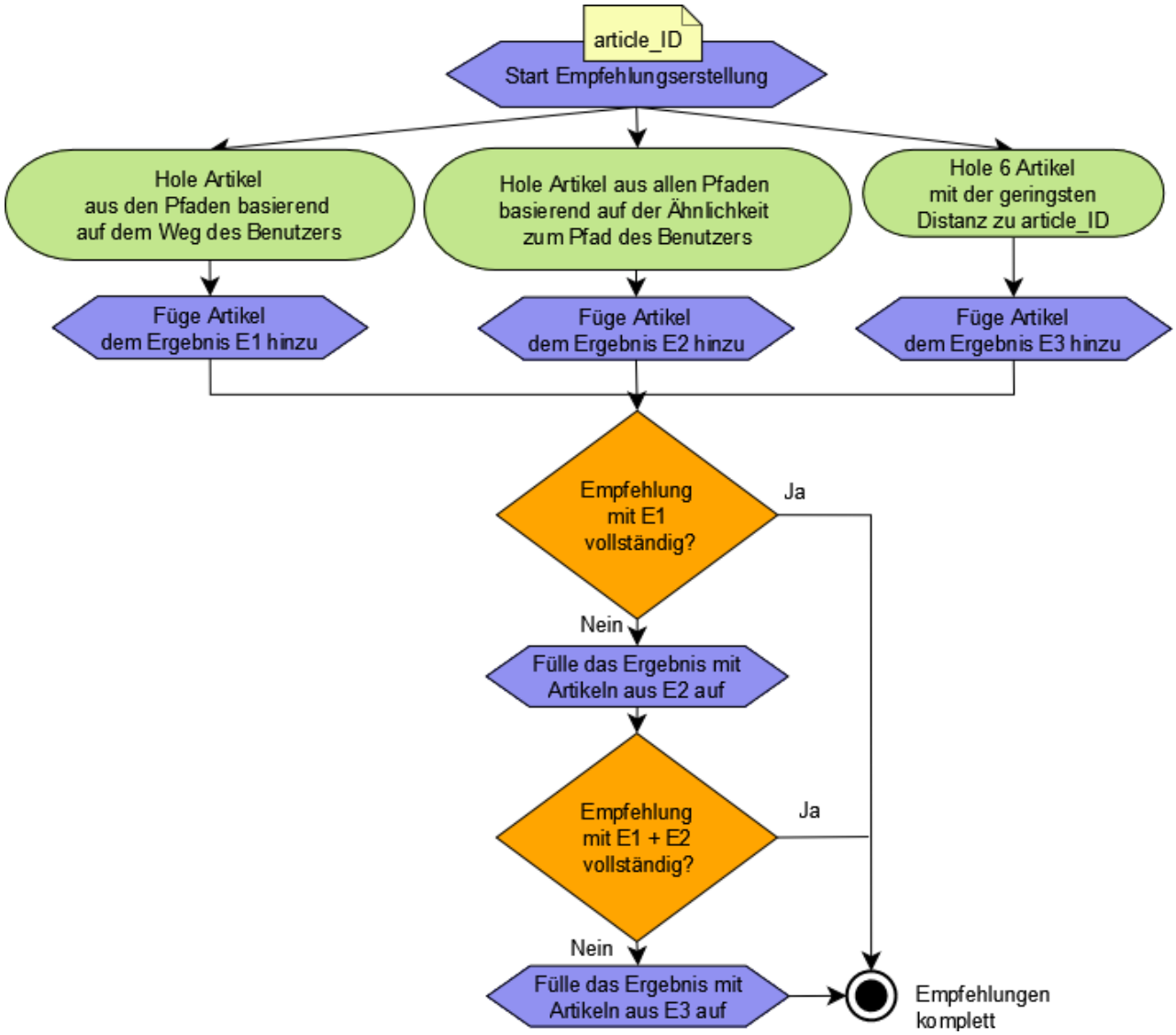


Abbildung 26: Empfehlungserstellung

Die Auswahl eines Artikels löst die Sequenz zur Erstellung von Empfehlungen aus. Ausgehend vom aktuellen Artikel werden die Ergebnislisten für die einzelnen Verfahren erstellt. Das Resultat der Suche nach Artikeln in Pfaden, die mit dem gesuchten Pfad beginnen, bildet das Ergebnis E1. Die Suche nach Artikeln in ähnlichen Pfaden zum gesuchten Pfad liefert das Ergebnis E2. Ergänzt werden diese Ergebnislisten durch die Liste E3, welche die Ergebnisse des Content-Based-Filterings beinhaltet. Die Empfehlungen werden aus diesen drei Listen erstellt, wobei das Endergebnis durch Auffüllen in der Reihenfolge E1, E2, E3 gebildet wird, wie in Kapitel 5.4.8 beschrieben.

5.5 Datenbankschema

ID	token	ip
1	oOcDGMNDp2QfiHi9ePjl	127.0.0.1

Tabelle 10: Datenbanktabelle: BenutzerInnen

In Tabelle 10 werden die Daten der BenutzerInnen festgehalten, wobei die erste Spalte „ID“ die interne Nummer des Datensatzes speichert. Diese Spalte existiert in allen Tabellen und wird bei den nachfolgenden Beschreibungen nicht mehr explizit erwähnt. Die Spalte „token“ speichert eine Zufallszeichenkette, die zusammen mit dem Wert der Spalte ID im Cookie abgespeichert wird. Die Spalte „ip“ speichert die dazugehörige IP-Adresse.

ID	quantity	levels	journal_id	weight	date	duplicate_check
17	1	12	2	0.8	1318410464	1

Tabelle 11: Datenbanktabelle: Pfad

In Tabelle 11 werden die Daten eines Pfades gespeichert. Die Spalte „quantity“ zählt die Häufigkeit des Pfades, „levels“ erfasst die Länge des Pfades, „journal_id“ ist die Nummer des Journals in OJS, in welchem der Pfad erzeugt wurde, „weight“ ist das Gewicht des Pfades, „date“ beinhaltet den Zeitstempel, wann der Pfad erzeugt wurde und „duplictate_check“ hält fest, ob der Pfad bereits auf Duplikate überprüft wurde.

ID	trace_id	article_id	level
9	17	43	5

Tabelle 12: Datenbanktabelle: Pfadelement

Tabelle 12 zeigt die Speicherung eines Pfadelementes. Die Spalte „trace_id“ beinhaltet die Nummer des Pfades, zu welchem das Element gehört. In der Spalte „article_id“ wird gespeichert, welcher Artikel zu diesem Element gehört. Die letzte Spalte „level“ ist die Position des Elements im Pfad. Das Beispiel in Tabelle 12 bedeutet: Das fünfte Element in Pfad Nummer 17 war der Artikel mit der Nummer 43.

ID	user_id	trace_id
4	1	17

Tabelle 13: Datenbanktabelle: Verbindung BenutzerInnen und Pfad

Die Verbindung von BenutzerIn und Pfad wird in Tabelle 13 dargestellt. In der Spalte „user_id“ wird die Nummer der Benutzerin beziehungsweise des Benutzers abgelegt und durch den Eintrag der Pfadnummer in der Spalte „trace_id“ mit diesem Pfad verbunden. Eine Benutzerin oder ein Benutzer kann somit mehrere Pfade zugewiesen bekommen. Auf

der anderen Seite kann ein Pfad auch mehrere BenutzerInnen haben. Dies ist zum Beispiel der Fall, wenn ein Pfad öfter als einmal vorkommt.

ID	word	article_id	count
567	abschnitt	35	4

Tabelle 14: Datenbanktabelle: Temporäre Wort-Artikel-Tabelle

Tabelle 14 beschreibt die temporäre Wort-Artikel-Tabelle die bei der Installation des Plugins benötigt wird (siehe Kapitel 5.4.3). In der Spalte „word“ wird das analysierte Wort gespeichert, in der Spalte „article_id“ die Nummer des Artikels, der das Wort beinhaltet und die Spalte „count“ zählt die Vorkommen des Wortes im Artikel.

ID	word	article_id	count	tf-idf
18	abschnitt	35	4	4.13454

Tabelle 15: Datenbanktabelle: Wort-Artikel-Tabelle

Tabelle 15 beschreibt die Wort-Artikel Tabelle, die aus der temporären Wort-Artikel-Tabelle gewonnen wird. Neu hinzugekommen ist die Spalte „tf-idf“, welche den Wert der TF-IDF Berechnung für das Wort in der Spalte „word“ enthält.

ID	article_id_left	article_id_right	distance	journal_id
659	40	88	0.10610270512259	3

Tabelle 16: Datenbanktabelle: Artikeldistanzen

Das Resultat des Content-Based-Filterings wird in Tabelle 16 gespeichert. Die Spalten „article_id_left“ und „article_id_right“ bezeichnen dabei die beiden Seiten der Relation zwischen zwei Artikeln. Die Spalte „distance“ speichert die errechnete Cosinusdistanz und in der Spalte „journal_id“ ist wieder die Nummer des Journals gespeichert, in welchem die Artikel enthalten sind. Das Beispiel in Tabelle 16 liest sich folgendermaßen: Zwischen den Artikeln mit den Nummern 40 und 88 beträgt die Distanz 0.106.

ID	issue_id	article_id_from	article_id_to
1	7	89	88

Tabelle 17: Datenbanktabelle: Relationen des Inhaltsverzeichnisses

In Tabelle 17 sind die Relationen für die Berechnung der Gewichte gespeichert. Damit die Relationen nur innerhalb einer Ausgabe betrachtet werden, speichert die Spalte „issue_id“ die Nummer der Ausgabe, in welcher die Relationen gefunden wurde. Die Spalte „article_id_from“ speichert den Ausgangsartikel der Relation, die Spalte „article_id_to“ speichert den Zielartikel der Relation. Das Beispiel in Tabelle 17 bedeutet, dass in Ausgabe sieben eine Relation 89 → 88 existiert.

6 Analyse und Diskussion

Um eine Aussage über die Qualität des Algorithmus treffen zu können, ist es notwendig, ihn hinsichtlich der Genauigkeit in Bezug auf die Relevanz der Ergebnisse als auch im Hinblick auf die Nützlichkeit der Ergebnisse für die BenutzerInnen zu untersuchen. Des weiteren soll ein Vergleich mit einem Algorithmus durchgeführt werden, der Empfehlungen auf eine andere Art und Weise erstellt. Für diese Aufgabe wurde ein Algorithmus implementiert, der mit bedingten Wahrscheinlichkeiten arbeitet. Die beiden Algorithmen bekamen dieselben Daten als Eingabe um Empfehlungen zu erstellen. Anschließend wurden die Ergebnisse hinsichtlich ihrer Relevanz mit Hilfe der Maßgrößen Precision und Recall analysiert. Die folgenden Kapitel beschreiben die Durchführung dieser Analyse und diskutieren anschließend die Ergebnisse für die beiden Algorithmen.

Ein Schlagwort bei der Analyse von Empfehlungen ist die „Relevanz“. Dieser Begriff ist schwer zu messen, denn für jeden Menschen können andere Dinge relevant sein. Hinzu kommt die Schwierigkeit eine Grenze in Bezug auf den Umfang des Begriffs „Relevanz“ zu ziehen. Meistens handelt es sich um eine einfache Ja/Nein Entscheidung, wie zum Beispiel die Entscheidung ob ein Artikel für eine Benutzerin oder einen Benutzer interessant ist oder nicht. Es kann aber auch innerhalb eines Wertebereichs entschieden werden, ob ein Element durch das Überschreiten eines bestimmten Schwellwertes relevant wird [8].

Um dennoch eine Aussage über die Relevanz einer Ausgabe eines Empfehlungssystems treffen zu können, existieren die Maßgrößen „Precision“ und „Recall“ (siehe Kapitel 2.4.5). Kurz wiederholt bezeichnet Precision den Prozentsatz der relevanten Elemente innerhalb der Empfehlung und Recall den Prozentsatz der gefundenen relevanten Elemente in Bezug auf alle existierenden relevanten Elemente. In dieser Definition steckt auch gleichzeitig die Schwäche dieser Maßgröße, denn um diese Werte berechnen zu können, braucht man eine Referenz, welche Elemente in einem Datenbestand relevant zu einer Suchanfrage sind.

Die Bestimmung des Wertes für die Precision ist durch eine Auswertung der Empfehlungen für jedes System durchführbar, wo hingegen der Wert für Recall speziell bei großen Datenbeständen nur näherungsweise bestimmt werden kann. Precision und Recall hängen zusammen – ein Wert von 100% von Recall ist einfach zu erreichen, indem man alle Dokumente als Ergebnis liefert, dafür wird der Wert für Precision stark sinken – und sind für BenutzerInnen je nach Anwendung unterschiedlich wichtig. Bei Suchergebnissen ist ein hoher Wert für Precision gewünscht, bei der Durchsuchung der eigenen Festplatte ein hoher Wert für Recall [19].

Im Fall von L3T hat man es nur mit einer relativ kleinen Menge an Elementen zu tun. Zusätzlich existiert eine Expertinnen- und Expertenmeinung über die Relevanz der einzelnen Artikel zueinander, welche in Form einer Mindmap (Abbildung 27) die Beziehungen der einzelnen Artikel zueinander abbildet. Neben den Beziehungen sind auch Leseempfehlungen für jeden Artikel, die als Pfeile dargestellt sind, abgebildet. Diese Mindmap bildete die Grundlage der Analyse hinsichtlich Precision und Recall für die beiden Algorithmen. Artikel, die in der Mindmap direkt aneinander grenzen oder die einem gemeinsamen Themengebiet untergeordnet sind, wurden als relevant zueinander eingestuft. Ebenso wurden die Leseempfehlungen als relevant eingestuft.

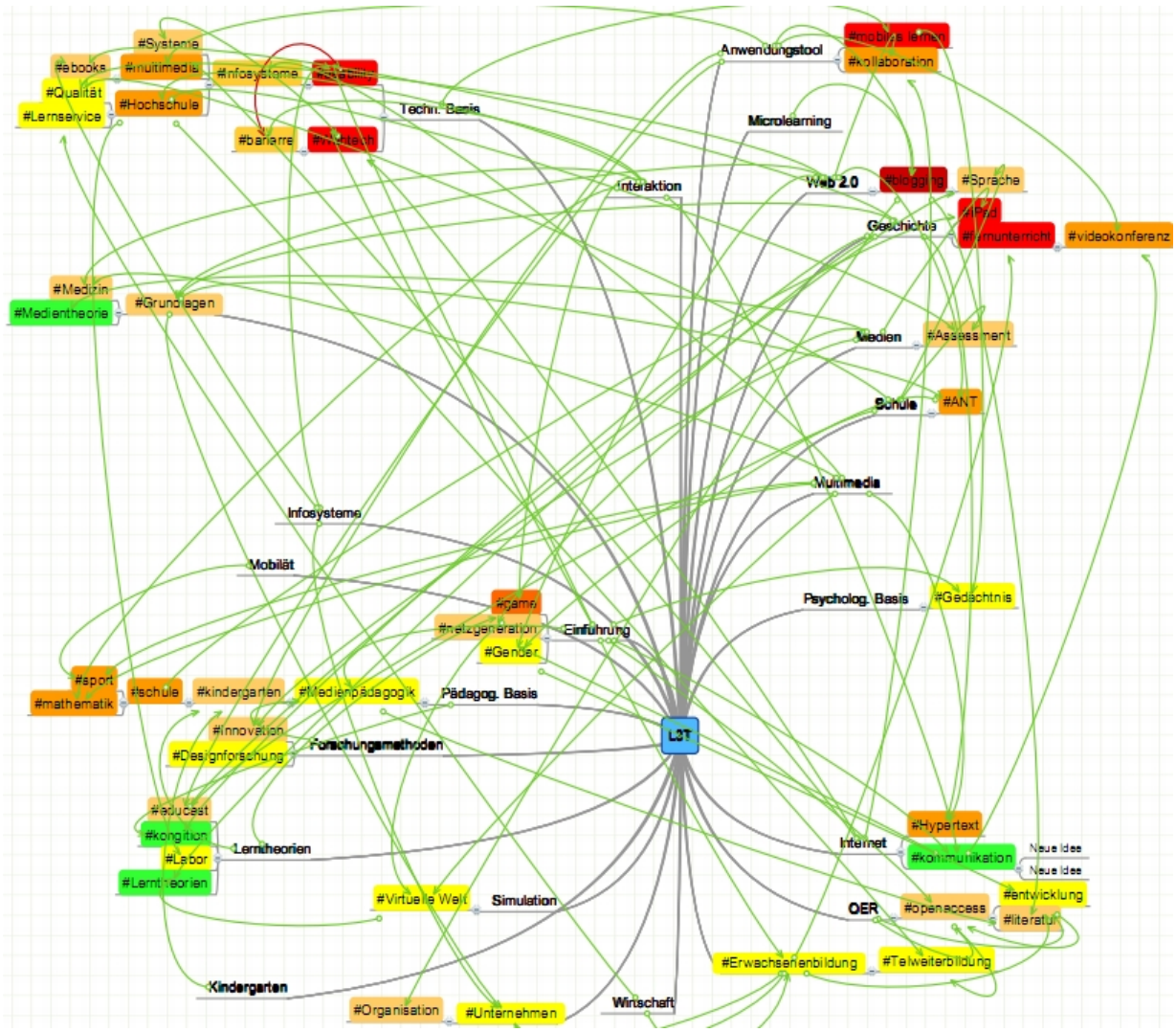


Abbildung 27: L3T Mindmap: Beziehungen und Leseempfehlungen

Das Ergebnis der Auswertung dieser Grafik war eine Liste mit relevanten Artikeln für jeden einzelnen Artikel. Tabelle 18 stellt einen Auszug dieser Vergleichsliste dar – die vollständige Liste befindet sich im Anhang (Kapitel 8.2). Zur besseren Lesbarkeit sind in der Tabelle neben den IDs der Artikel auch die „Hashtags“ der einzelnen Artikel notiert. Ein Hashtag ist eine Kurzbezeichnung, welcher ein # (engl. „Hash“) vorangestellt ist.

Artikel	Dazu relevante Artikel-IDs
#grundlagen (ID: 88)	68 19 28 40 65 73 18 49 54 63 66 70 72
#ipad (ID: 49)	54 68 85 37 61 47 88 73 65 18
#hypertext (ID: 73)	57 74 41 19 35 31 32 65 88 18 49 54
#fernunterricht (ID: 54)	49 24 36 65 88 18 49 73
#infosysteme (ID: 41)	73 19 35 31 32 71 36 39 58
#webtech (ID: 44)	34 71 39 45 51 57 62 36 79 61 72 66 38 74

Tabelle 18: Auszug Relevanzliste für L3T

Jeder dieser Hashtags ist eindeutig und wird auch in der Mindmap zur Bezeichnung der Artikel verwendet.

Eine Technik um die Qualität eines Systems zu messen ist die sogenannte „k-fold cross-validation“ [15]. Dabei werden die Daten zufällig in k Teile aufgeteilt. Ein Teil k wird zur Validierung des Systems verwendet, während k-1 Teile für den Aufbau des System zur Verfügung stehen. Dieser Validierungs- und Trainingsprozess wird nun k-mal durchgeführt, wobei jedes Mal ein anderes der k Teile für die Validierung verwendet wird. Die Resultate der einzelnen Iterationen können nun wie gewünscht kombiniert werden. Ein Beispiel hierfür ist die Bildung des Durchschnitts der Ergebnisse.

Für die Durchführung der Analyse wurde eine große Datenmenge benötigt. Alle verwendeten Daten stammten aus den Aufzeichnungen, die mittels Piwik erstellt wurden und in Kapitel 3 bereits für die erste Datenanalyse herangezogen wurden. Für die Auswertung der Ergebnisse der Algorithmen dieser Arbeit wurde ein „10-fold cross-validation“ Verfahren eingesetzt. Hierfür wurden als erstes die Daten in zehn einzelne Teilpakete zu je 2000 Besuchen („visits“) zerlegt. Von diesen zehn Paketen wurden neun zur Erstellung der Pfaddatenbank und der Wahrscheinlichkeiten verwendet, das zehnte Paket diente zur Simulation der Benutzeraktionen. Bei jeder Iteration wurde das Testpaket vertauscht und die Datenbasis mit den restlichen neun Paketen neu berechnet:

Iteration 1: Datensets: 2-10; Simulation mit Set 1
Iteration 2: Datensets: 1, 3-10; Simulation mit Set 2
Iteration 3: Datensets: 1-2, 4-10; Simulation mit Set 3
…
Iteration 10: Datensets: 1-9; Simulation mit Set 10

Diese Iterationen wurden für die beiden Algorithmen durchgeführt und die Ergebnisse jeder Iteration aufgezeichnet. Die Durchschnitte der Werte für Precision und Recall wurden anschließend ermittelt und grafisch aufbereitet (siehe Kapitel 6.3).

6.1 Vergleichsalgorithmus

Die Ergebnisse des Algorithmus für das Plugin müssen mit den Ergebnissen eines anderen Algorithmus verglichen werden, um eine Aussage über die Qualität des Algorithmus treffen zu können. Als Vergleichsalgorithmus wurde ein Algorithmus gewählt, der mit bedingten Wahrscheinlichkeiten arbeitet. Die dafür benötigten Übergangswahrscheinlichkeiten wurden bereits in Kapitel 3.2.1 bei der Datenanalyse mit Hilfe der Artikelmatrix aufgezeichnet. Für die Berechnung von bedingten Wahrscheinlichkeiten benötigt man außerdem noch die Wahrscheinlichkeit für das Auftreten eines jeden Artikels.

Eine bedingte Wahrscheinlichkeit berechnet die Wahrscheinlichkeit eines Ereignisses unter der Voraussetzung, dass ein bestimmtes Ereignis vorausgegangen ist. Mit anderen Worten: Es wird die Wahrscheinlichkeit für ein Ereignis B berechnet, unter der Voraussetzung, dass B auf das Ereignis A folgt ($A \rightarrow B$ oder $(B|A)$).
Diese Wahrscheinlichkeit kann man mit dem Satz von Bayes berechnen [4]:

$$P_1(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A|B) \cdot P(B)}{P(A)} \qquad (25)$$

A, B: *Ereignisse*
P: *Wahrscheinlichkeit*

In der Vorbereitungsphase wurden die Wahrscheinlichkeiten und die Übergangswahrscheinlichkeiten der Artikel aus den jeweiligen neun Datensets erstellt und in der Datenbank gespeichert. Zusätzlich zu den Wahrscheinlichkeiten wurden auch die dazugehörigen Artikelnummern in jeder Zeile vermerkt. Tabelle 19 zeigt zwei Zeilen aus der Datenbank.

ID	article_from	probability_from	article_to	probability_to	probability_trans
1	89	0.0657574	88	0.0486362	0.616364
2	89	0.0657574	49	0.0454469	0.0963636

Tabelle 19: Datenbanktabelle der Wahrscheinlichkeiten (Auszug)

Die Spalte „ID“ bezeichnet die Nummerierung der Datensätze innerhalb der Datenbank, die Spalte „article_from“ speichert die Nummer des Ausgangsartikels, „probability_from“ ist die Wahrscheinlichkeit seines Vorkommens. Daneben befinden sich die Nummer des Zielartikels („article_to“) und dessen Wahrscheinlichkeitswert („probability_to“). In der letzten Spalte ist die Wahrscheinlichkeit des Übergangs gespeichert.

Diese erste Zeile bedeutet, dass Artikel 89 mit einer Wahrscheinlichkeit von 6,5% vorkommt, Artikel 88 mit 4,8 % und dass ein Übergang 89 → 88 mit einer Wahrscheinlichkeit von 61% vorkommt. Mit dieser Datenbank können die benötigten Wahrscheinlichkeiten schnell berechnet werden. Der Algorithmus berechnet für ein Ereignis die Wahrscheinlichkeiten für alle nachfolgenden Ereignisse und retourniert die besten sechs Ergebnisse, welche dann empfohlen werden.

6.2 Testdurchführung

Für die Durchführung der Tests wurde eine eigene Klasse implementiert, die die Recommender-Klasse des Plugins verwendet, um Empfehlungen zu erstellen. Als erstes wurden die Pfade der Testdaten aus der Datenbank gelesen. Diese Pfade simulieren die Bewegung einer Benutzerin oder eines Benutzers. Anschließend wurde für jeden Artikel die Menge der für ihn relevanten Artikel in Form einer Liste innerhalb der Testklasse gespeichert. Diese Liste entspricht der Tabelle der relevanten Artikel im Anhang (Kapitel 8.2 , Tabelle 26). Die Testmethode bekam jeden Pfad der Testdaten als Eingabe um eine Person zu simulieren, die das System verwendet. Für jeden Artikel im Testpfad wurden Empfehlungen erstellt. Diese Empfehlungen wurden mit der Liste der für diesen Artikel relevanten Artikel verglichen und die Werte für Precision und Recall berechnet.
Anschließend wurden die Empfehlungen dahingehend überprüft, ob auch der nächste Artikel im Pfad in den Empfehlungen enthalten war. Ein solches Vorkommen wird als „Hit“ bezeichnet. Die Division aller „Hits“ durch die Anzahl der Artikel, für die eine Empfehlung erstellt wurde, ergibt die „Hit-Ratio“. Dieses Verfahren findet in ähnlicher Form Anwendung bei [9].

Abbildung 28 zeigt ein Beispiel für die Ausgabe, die vom Testprogramm während einer Station des Testpfades erstellt wurde. Zu sehen sind das Element, für welches eine Empfehlung generiert wurde („last_label“), die sechs Empfehlungen für dieses Element („recommendation“) und das nächste Element des Pfades („current_label“).

```
last_label: 71, recommendation: 18 = current_label: 18  <--
last_label: 71, recommendation: 45, current_label: 18,
last_label: 71, recommendation: 44, current_label: 18,
last_label: 71, recommendation: 38, current_label: 18,
last_label: 71, recommendation: 34, current_label: 18,
last_label: 71, recommendation: 50, current_label: 18,
---------------------------
Article: 71 Precision: 0.5 Recall: 0.21428571428571
```

Abbildung 28: Beispiel einer Ausgabe des Testprogramms

Für den Artikel mit der ID 71 wurden folgende Empfehlungen erstellt: (18, 45, 44, 38, 34, 50). Die erste Empfehlung wurde im aufgezeichneten Pfad als nächstes verwendet und wird somit als „Hit“ gezählt.
Der Wert für Precision liegt bei 0,5, das bedeutet, dass die Hälfte der empfohlenen Elemente auch in der Liste der Expertenmeinungen vorkommen. Da sechs Elemente für einen Artikel vorgeschlagen werden ergibt das (Formel 7 Kapitel 2.4.5): 3/6 = 0,5.
Ein Wert für Recall von ca. 0,21 sagt aus, dass 21 % aller verfügbaren Elemente aus der Expertenmeinung für diesen Artikel empfohlen wurden. Laut Tabelle 26 im Anhang existieren für den Artikel mit der Nummer 71 eine Menge von 14 relevanten Artikeln. Daher ergibt sich der Wert (Formel 8 Kapitel 2.4.5): 3/14 = 0,21428.
Diese Berechnungen wurden für alle Elemente der Pfade in den Testdaten durchgeführt und ausgewertet. Am Ende eines jeden Durchlaufs eines Testdatensatzes wurden die Durchschnittswerte für Precision und Recall gebildet und zusammen mit der Hit-Ratio ausgegeben.

6.3 Ergebnisse

Die Durchläufe der 10 Testserien wurden für jeden Algorithmus durchgeführt, zusätzlich wurden für jede Testserie drei Durchläufe mit unterschiedlichen maximalen Längen der Pfade der Testdaten durchgeführt. Die Längenbegrenzungen betrugen 10, 20 und 50 Elemente in einem Pfad. Zusätzlich wurden nur Pfade ab einer gewissen Länge betrachtet. Hier gilt anzumerken, je höher die minimale Länge der Pfade gewählt wird, desto weniger unterschiedliche Pfade gibt es. Setzt man die minimale Länge hingegen zu niedrig an, erhält man viele sehr kurze Pfade. Mit diesen Pfaden kann man kein aussagekräftiges Ergebnis erzeugen, denn es existieren sehr viele Pfade, die die Elemente dieser kurzen Pfade beinhalten. In den Testserien hat sich gezeigt, dass Pfade ab einer Länge von sechs Elementen gute Ergebnisse liefern. In Zahlen bedeutet das, dass für die einzelnen Testläufe Pfade in den Grenzen von (min. 6 – max. 10), (min. 6 – max. 20), und (min. 6 – max. 50) Elementen als Eingabe verwendet wurden. Wie im vorherigen Abschnitt erläutert, werden diese Testpfade durchlaufen und für jeden Schritt eine Empfehlung erstellt, die dann verglichen wird. Im ersten Schritt enthält der zu vergleichende Pfad ein Element, im zweiten Schritt zwei Elemente, im dritten drei Elemente usw., so als ob eine Person die Artikel in der Reihenfolge des Testpfades heruntergeladen hätte.

#Predictions:			
#Test	L < 10	L < 20	L < 50
1	170	492	1969
2	184	477	1506
3	143	373	1001
4	174	405	975
5	97	243	879
6	124	361	787
7	111	188	526
8	86	194	571
9	109	270	516
10	90	160	366
Summe	**1288**	**3163**	**9096**

Tabelle 20: Testdurchläufe

In Tabelle 20 ist die Anzahl der Empfehlungserstellungen für jede Testserie notiert. Eine Empfehlungserstellung ist definiert als die Ausgabe von sechs Vorschlägen ausgehend vom aktuell besuchten Artikel der Benutzerin oder des Benutzers. Die Empfehlungen der Testdurchläufe der einzelnen Grenzen (L<10, L<20, L<50) sind jeweils in den den Empfehlungen der höheren Grenze enthalten. Die Empfehlungen von L<10 sind also eine Teilmenge von L<20, welche wiederum eine Teilmenge von L<50 sind.

Insgesamt wurden 13547 Empfehlungen generiert und ausgewertet. Die Auswertungen wurden getrennt durchgeführt, um die Auswirkung der Pfadlängen auf das Ergebnis zu demonstrieren. Dazu wurden die Empfehlungen mit der Expertenmeinung in Tabelle 26 verglichen und die durchschnittlichen Werte der zehn Iterationen für Precision (Tabelle 21 und Abbildung 29) und Recall (Tabelle 22 und Abbildung 30) berechnet. Die vollständigen Tabellen mit den Ergebnissen für alle Iterationen finden sich im Anhang (Kapitel 8.3).

Anmerkung: In den Abbildungen dieses Kapitels wird das Plugin mit „Traces“ bezeichnet, der Wahrscheinlichkeitsalgorithmus mit der Abkürzung „WSK“.

	Precision (WSK)	Precision (Traces)
T<10	0,42787	0,34364
T<20	0,41939	0,33623
T<50	**0,39552**	**0,32961**

Tabelle 21: Vergleich Precision

	Recall (WSK)	Recall (Traces)
T<10	0,24742	0,20515
T<20	0,24406	0,20409
T<50	**0,23882**	**0,20517**

Tabelle 22: Vergleich Recall

Der Vergleich der Werte für Precision zeigt einen Unterschied von ca. 7% zugunsten des Wahrscheinlichkeitsalgorithmus. In relevanten Artikeln ausgedrückt ergibt das bei einer Menge von sechs empfohlenen Artikeln pro Empfehlung einen Unterschied von 0.42 relevanten Artikeln pro Empfehlungserstellung. Dieser Unterschied liegt in der unterschiedlichen Funktionsweise der Algorithmen. Der Wahrscheinlichkeitsalgorithmus kann immer eine Empfehlung abgeben, wenn eine Beziehung der Artikel in der Datenbank besteht. Beim Algorithmus des Plugins, welches den Testpfad mit den gespeicherten Pfaden vergleicht, kann es natürlich vorkommen, dass kein passender Pfad in der Datenbank vorhanden ist. In diesem Fall greift das Content-Based-Filtering ein, um dennoch eine Empfehlung erstellen zu können. Nun unterscheiden sich die Ergebnisse des Content-Based-Filterings von der Expertenmeinung stärker, da der Vergleich auf der begrenzten Datenmenge der Kurzfassungen der Texte stattfindet. Oft ist die Beziehung, die in der Expertenmeinung notiert ist, aus der Analyse der Texte nicht ersichtlich und führt somit zu schlechteren Precision und Recall Werten. In Bezug auf die Werte für Recall beträgt der Unterschied ca. 4%. Der Wert für Recall ist – bedingt durch die Experteneinschätzung – um einiges kleiner als der Wert für Precision, da in den meisten Fällen mehr als sechs relevante Artikel für einen Artikel definiert sind.

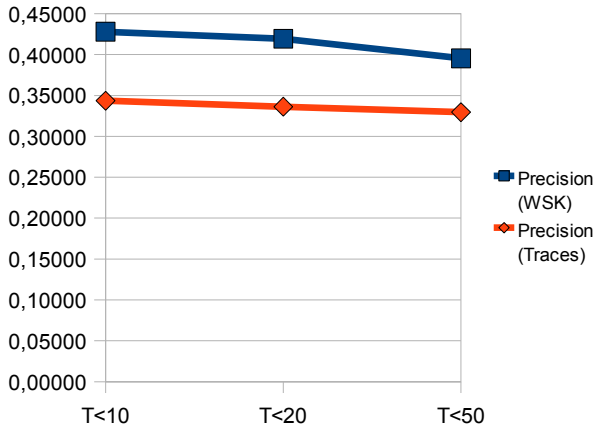


Abbildung 29: Vergleich Precision

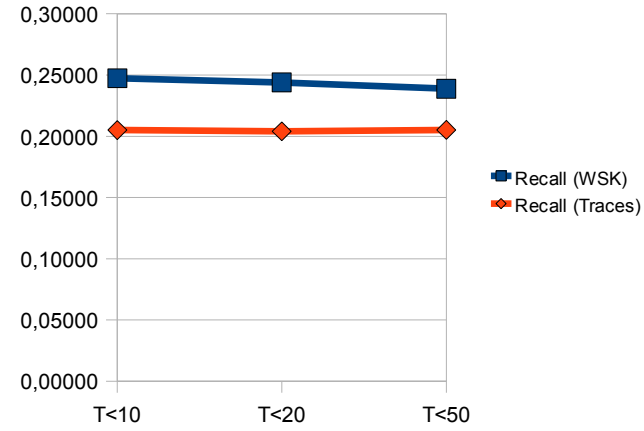


Abbildung 30: Vergleich Recall

Abbildung 29 und 30 zeigen den Verlauf von Precision und Recall mit steigender

Maximallänge der Testpfade. Längere Pfade folgen eher der vorgegebenen Navigationsstruktur (in diesem Fall dem Inhaltsverzeichnis) und führen zu schlechteren Werten in Bezug auf Precision und Recall.

Im Gegensatz dazu steigt die Hit-Ratio mit der Pfadlänge an (Tabelle 23). Die Vorhersage, welchen Artikel die/der BenutzerIn als nächstes auswählt, wird von der maximalen Länge der Pfade beeinflusst. Kürzere Pfade divergieren mehr als längere Pfade, zudem enthalten längere Pfade mehr Elemente des Inhaltsverzeichnisses, wie bereits in Kapitel 3.2.2 (Abbildung 11) beschrieben wurde.

	Hit (WSK)	Hit (Traces)
T<10	0,54429	0,38483
T<20	0,66259	0,50084
T<50	**0,82814**	**0,72492**

Tabelle 23: Vergleich Hit Ratio

Abbildung 31 zeigt den Verlauf der Hit-Ratio bei wachsender Pfadlänge. Beide Algorithmen liefern steigende Ergebnisse. Längere Pfade folgen, wie bereits ausgeführt, eher einer Struktur. Dies führt dazu, dass in der Datenbank gefundene, ähnliche Pfade eben diese Struktur abbilden. Die Hit-Ratio steigt zwar an, hat aber für längere Pfade daher eine geringe Aussagekraft. Auf den Algorithmus des Plugins hat die Pfadlänge des Testpfades einen stärkeren Einfluss in Bezug auf die Hit-Ratio, als auf den Wahrscheinlichkeitsalgorithmus.

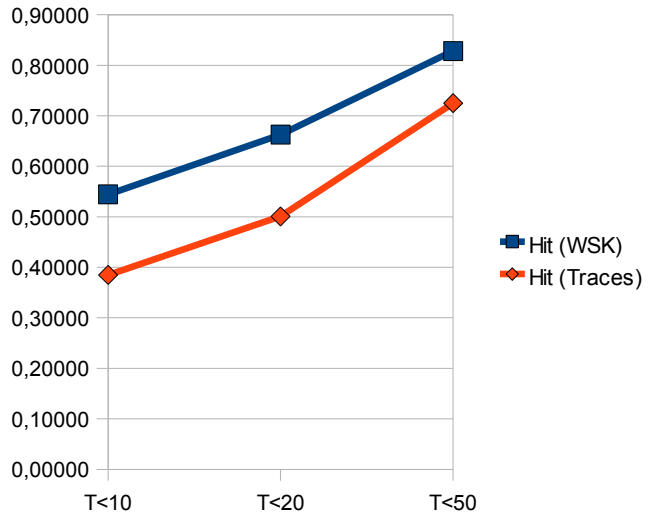


Abbildung 31: Vergleich Hit-Ratio

Die betrachteten Größen Precision, Recall und Hit-Ratio sind auch von der Anzahl der empfohlenen Artikel pro Empfehlungserstellung abhängig. Vor allem die Hit-Ratio steigt mit mehr empfohlenen Artikeln stark an. Für diese Analyse wurde eine Menge von sechs empfohlenen Artikeln festgesetzt, vor allem um den BenutzerInnen eine angenehme

Menge an Auswahlmöglichkeiten zu präsentieren, ohne dass das Ergebnis unübersichtlich wird.

Die unterschiedlichen Arbeitsweisen der beiden Algorithmen wird durch die Verteilungen der empfohlenen Artikel illustriert. Abbildung 32 zeigt die Verteilung des Wahrscheinlichkeitsalgorithmus, Abbildung 33 zeigt die Verteilung des Algorithmus des Plugins. Die Artikel sind in den Abbildungen nach der Häufigkeit (in %), wie oft sie von jedem Algorithmus empfohlen wurden, geordnet.

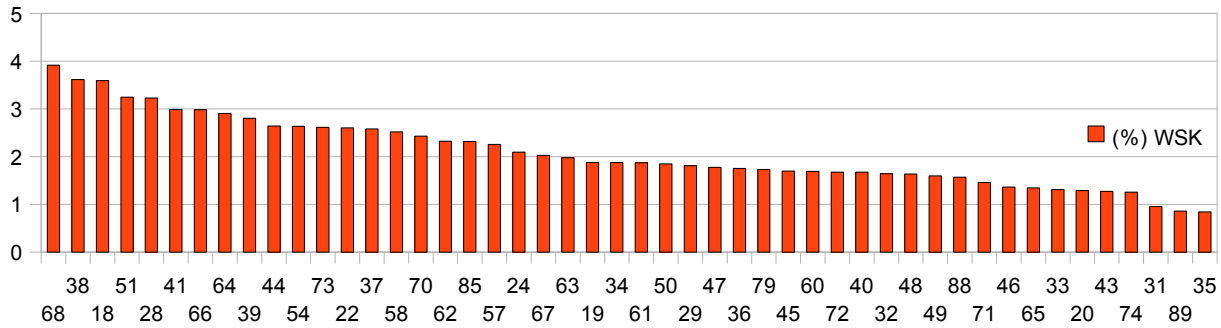


Abbildung 32: Prozentuale Verteilung der empfohlenen Artikel: Wahrscheinlichkeitsalgorithmus

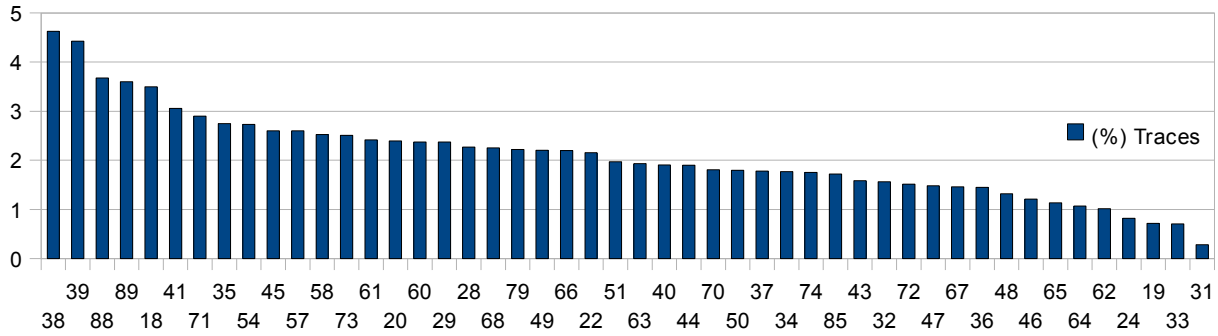


Abbildung 33: Prozentuale Verteilung der empfohlenen Artikel: Plugin-Algorithmus

Die Verteilungen der Artikel ähneln sich, jedoch werden bei jedem Algorithmus die Artikel mit einer anderen Häufigkeit empfohlen. Durch die Gewichtung der Pfade (Kapitel 4.2.3) versucht das Plugin, die Dominanz des Inhaltsverzeichnisses abzuschwächen und den Benutzerinnen und Benutzern interessante Elemente abseits der vorgegebenen Reihenfolge vorzuschlagen, falls ein passender Pfad in der Datenbank gespeichert ist.

Es werden bei beiden Algorithmen alle Artikel im System empfohlen. Die Spitzenwerte der Häufigkeiten heben sich nicht sehr vom Rest der Häufigkeiten ab, beim Wahrscheinlichkeitsalgorithmus erreicht der Artikel 68 eine Häufigkeit von 3,92 % (Abbildung 32), beim Algorithmus des Plugins erreicht der am öftesten empfohlene Artikel 38 eine Häufigkeit von 4,63 % (Abbildung 33).

Abbildung 34 fasst die Ergebnisse beider Algorithmen zusammen und zeigt die Häufigkeiten jedes Artikels (wieder in %) in der Reihenfolge des Inhaltsverzeichnis an.

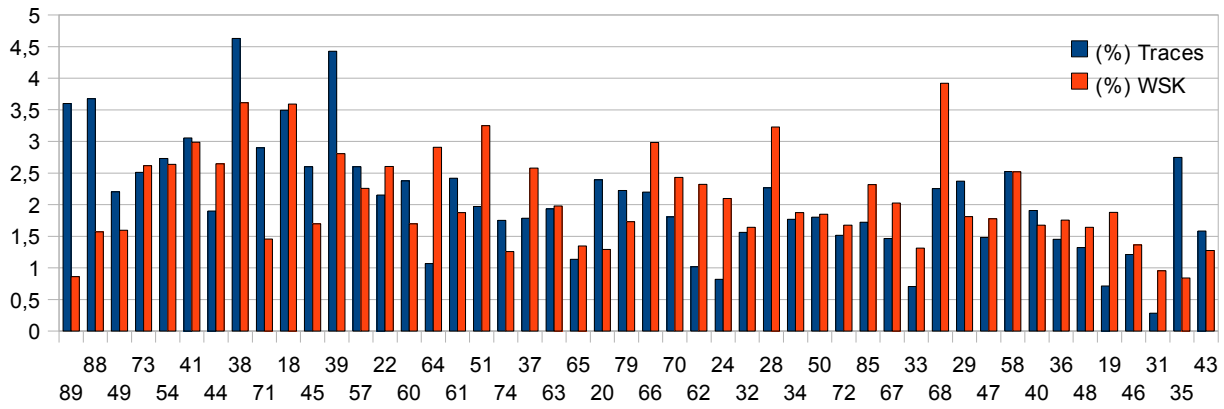


Abbildung 34: Verteilung der empfohlenen Artikel in beiden Algorithmen

Ein großer Unterschied zwischen beiden Ansätzen ist die Behandlung neuer Artikel. Der Ansatz des Plugins ermöglicht durch die Anwendung von Content-Based-Filtering das Empfehlen dieses neuen Artikels auch dann, wenn er noch von keiner Person heruntergeladen wurde. Beim Collaborative-Filtering-Teil des Plugins ist das Vorkommen des neuen Artikels in einem Pfad die Voraussetzung für eine Empfehlung. Es genügen bereits wenige Pfade, damit das neue Element vorgeschlagen werden kann.

Wie gut ein neues Element im Fall des Wahrscheinlichkeitsalgorithmus empfohlen werden kann, hängt davon ab, wie groß das Verhältnis der bestehenden Übergangswahrscheinlichkeiten zu den Übergangswahrscheinlichkeiten zum neuen Artikel ist. Ein Artikel, der erst später in ein bestehendes System eingefügt wird, muss erst von einer bestimmten Menge an Benutzerinnen und Benutzern verwendet werden, um dadurch die Wahrscheinlichkeiten für diesen neuen Artikel solange zu verbessern, bis er empfohlen wird.

Die Analyse hat die Unterschiede des Plugins zu einem alternativen Algorithmus beleuchtet und die unterschiedlichen Arbeitsweisen der Algorithmen gezeigt. Trotz der unterschiedlichen Herangehensweise liefern beide Algorithmen ähnliche Ergebnisse. Der Wahrscheinlichkeitsalgorithmus liefert geringfügig bessere Precision und Recall Werte hinsichtlich der Experteneinschätzung. Auf der anderen Seite ist das Plugin in der Lage Elemente zu empfehlen, die dem Wahrscheinlichkeitsalgorithmus noch unbekannt sind, weil sie neu in das System integriert wurden, oder weil sie noch von zu wenigen Personen heruntergeladen wurden.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde die Entwicklung und Erstellung eines Empfehlungssystems für OJS beschrieben. Kapitel 2 widmete sich den theoretischen Grundlagen, um einen Einblick in die Welt der Empfehlungssysteme zu erhalten und auf die verschiedenen Arten von existierenden Systemen einzugehen. Es wurden die Stärken und Schwächen der Ansätze des Content-Based-Filterings, des Collaborative-Filterings und des Model-Based-Filterings behandelt und Kombinationsmöglichkeiten besprochen. Des weiteren wurde eine Auflistung von verschiedenen Fehlermaßen und Qualitätskriterien vorgenommen. Das erste Kapitel wurde mit der Vorstellung von zwei Empfehlungssystemen aus der Praxis abgeschlossen.

Im dritten Kapitel wurde der Vorgang der Datenerfassung und Datenanalyse behandelt. Aus den Rohdaten der Aufzeichnungen von Piwik wurde eine Datenbank erstellt, die die Grundlage für die weiteren Analysen bildete. Der erste Schritt war die Erzeugung einer Artikelmatrix, in welcher die Häufigkeiten der Wechsel von einem Artikel zum Nächsten dargestellt wurden. Aus dieser Matrix war ersichtlich, dass die BenutzerInnen häufig einer vorgegebenen Navigationsstruktur folgten. Der nächste Schritt war die Entwicklung eines Programms um die Pfade der BenutzerInnen aus den Rohdaten zu extrahieren und anschließend verfolgen zu können. Diese Pfaddaten wurden analysiert und grafisch aufbereitet. Als Schlussfolgerung wurde festgehalten, dass die gefundenen Häufigkeiten und Pfade für ein Empfehlungssystem nur eingeschränkt verwendbar sind, weil die Navigationsstruktur einen sehr großen Einfluss auf die Auswahl der Artikel durch die BenutzerInnen hat.

Kapitel 4 lieferte einen Vergleich von Algorithmen, die für die Umsetzung in OJS verwendet werden könnten. Das Primärsystem sollte ein System mit Content-Based-Filtering sein, weil für die Empfehlungserstellung außerhalb des Benutzerverhaltens in OJS nur die Artikel selbst zur Verfügung standen. Zusätzlich kann ein Content-Based-Filtering System auch Empfehlungen abgeben, ohne auf die Mithilfe der BenutzerInnen angewiesen zu sein. Die Wahl der Algorithmen fiel auf Stemming und TF-IDF, weil diese Verfahren unabhängig von zusätzlichen Informationen wie Meta-Tags sind und auch bei den kleineren Wortmengen der Kurzfassungen noch gute Ergebnisse erzielen.
Das Sekundärsystem sollte ein System mit Collaborative-Filtering sein, das die Ergebnisse des Primärsystems ersetzten oder erweitern kann. Bei OJS sind nur begrenzte Informationen über eine Benutzerin beziehungsweise einen Benutzer verfügbar, vor allem die Wiedererkennung ist schwierig, da es keine zwingende Anmeldung über ein Benutzerkonto gibt. Die Identifikation der BenutzInnen musste also über Cookies und über die IP-Adresse erfolgen. Damit die vorhandenen Daten der Benutzerpfade für eine Empfehlungserstellung verwendet werden konnten, wurden sie dadurch gewichtet, wie sehr sie der Navigationsstruktur ähnelten. Somit sollten solche Pfade für die Empfehlungserstellung weniger wichtig gemacht werden.

Die Planung und Implementierung des Plugins wurde in Kapitel 5 beschrieben. Zuerst wurden die Rahmenbedingen für die Erstellung eines Plugins für OJS betrachtet, denn ein Plugin muss bestimmten Konventionen folgen, damit es in OJS funktioniert. Jedes Plugin benötigt eine Hauptklasse, welche die Funktionen des Plugins steuert. Gleichzeitig ermöglicht diese Klasse auch die Registrierung der benötigten Einstiegspunkte und die

Verbindung dieser Einstiegspunkte mit den dazugehörigen Callback-Methoden. Eine Schwierigkeit war die Installation der benötigten Datenbank für das Content-Based-Filtering, da ein Zeitlimit von 30 Sekunden einzuhalten war. Somit musste eine eigene Installationsroutine entwickelt werden, um die Datenbank zu füllen.
Diese Installation verwendete eine eigens entwickelte Textanalyse-Klasse, um diverse Filter auf verschiedenste Arten zu kombinieren und so eine Filterkette für die Textanalyse zu bilden. Das Ergebnis dieser Filterkette diente als Grundlage für die Berechnung der TF-IDF Werte und in weiterer Folge für die Berechnung der Cosinusdistanzen zwischen den einzelnen Artikeln. Für die Implementierung des Collaborative-Filterings wurden Klassen zur Benutzerverwaltung und Pfadverfolgung erstellt. Um die Menge an Pfaden zu reduzieren, wurde eine Routine zur Erkennung und Behandlung von Pfadduplikaten entwickelt.

Das **Collaborative-Filtering** basiert auf den **gewichteten Pfaden** (Kapitel 4.2.3), die die BenutzerInnen bei der Verwendung des Systems erstellen. Durch die Gewichtung wird versucht, die dominante Struktur des Inhaltsverzeichnisses abzuschwächen und somit Empfehlungen zu erstellen, die auf Grund des Interesses der Benutzerinnen und Benutzer zustande gekommen sind und nicht auf Grund der Reihenfolge im Inhaltsverzeichnis. Der erste Teil des Collaborative-Filterings sucht Pfade in der Datenbank, die mit dem Pfad der Benutzerin oder des Benutzers beginnen und erstellt aus diesen, nach ihrem Gewicht geordneten Pfaden, eine Empfehlung für den nächsten Artikel. Wurden in der Datenbank keine passenden Pfade gefunden, sucht der zweite Teil des Collaborative-Filterings nach Pfaden, die Elemente des gesuchten Pfades enthalten und erstellt daraus Empfehlungen. Der in Kapitel 6.3 durchgeführte Vergleich mit einem Wahrscheinlichkeitsalgorithmus zeigt, dass der Ansatz des Plugins in der Lage ist, relevante Empfehlungen zu treffen, die sich zusätzlich von einer vorgegebenen Struktur abheben.

Für die Erstellung der Empfehlungen wurde schließlich eine Klasse implementiert, die die Ergebnisse beider Systeme auf folgende Weise verband: Sind Empfehlungen auf Basis des Collaborative-Filterings verfügbar, dann werden diese angezeigt und mit Empfehlungen des Content-Based-Filterings aufgefüllt. Kann das Collaborative-Filtering keine Ergebnisse erzeugen, werden nur die Empfehlungen des Content-Based-Filterings angezeigt.

Das sechste Kapitel befasste sich mit der Auswertung des Algorithmus. Um einen Vergleich durchführen zu können, wurde ein zweiter Algorithmus implementiert, der mit bedingten Wahrscheinlichkeiten arbeitet. Als Maßgrößen wurden Precision, Recall und die Hit-Ratio beider Algorithmen ermittelt. Um Precision und Recall bestimmen zu können, wurde eine Experteneinschätzung der Relevanz der Artikel untereinander als Referenz gewählt. Der Vergleichsalgorithmus erreichte bessere Werte als der Algorithmus des Plugins (Precision +7%, Recall +4%, Hit-Ratio +10% bis +16%), bezogen auf diese Experteneinschätzung. Auf der anderen Seite besitzt das Plugin die Möglichkeit, Elemente zu empfehlen, die neu im System sind und noch nicht von Benutzerinnen oder Benutzern verwendet wurden. Der Vergleich der Verteilungen der erstellten Empfehlungen zeigte die unterschiedliche Arbeitsweise der beiden Algorithmen. Durch die Verwendung der gewichteten Benutzerpfade und den Einsatz von Content-Based-Filtering beim Algorithmus des Plugins wurden bei diesem Ansatz Artikel mit unterschiedlicher Häufigkeit im Vergleich zum Wahrscheinlichkeitsalgorithmus empfohlen.

Es wurde gezeigt, dass diese Kombinationen von Content-Based-Filtering auf Basis der Kurzfassungen der Artikel und die Aufzeichnung und Analyse der gewichteten Benutzerpfade, die durch den Download von Dokumenten entstehen, in Form eines Plugins für OJS eingesetzt werden kann und dass relevante Ergebnisse erstellt wurden. Das Plugin wurde auf drei OJS-Systemen installiert (L3T, Zeitschrift für Hochschulentwicklung[13] und Bildungsforschung[14]) und ist dort im Einsatz.

Möglichkeiten zur Weiterentwicklung bietet die Analyse der bestehenden Pfade. Speziell die Rolle der Teilpfade kann hier betrachtet werden, denn bei langen Pfaden gibt es kaum Vergleichsmöglichkeiten. Hier muss untersucht werden, ab welcher Länge die Teilpfade für einen Vergleich herangezogen werden können.

13 http://www.zfhe.at (Abruf am 21.11.2011)
14 http://www.bildungsforschung.org (Abruf am 21.11.2011)

8 Anhang

8.1 Artikelmatrizen

Label (TOC)	89	88	49	73	54	41	44	38	71	18	45	39	57	22	60	64	61	51	74	37	63	65	20	79
89		0.62	0.09	0.03	0.02	0.02	0.00	0.01	0.01	0.03	0.01	0.01	0.00	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.00	0.01	0.00
88	0.13		0.55	0.06	0.04	0.03	0.00	0.04	0.00	0.03	0.01	0.01	0.01	0.00	0.00	0.01	0.00	0.00	0.01	0.00		0.01	0.00	0.00
49	0.08	0.07		0.53	0.04	0.05	0.01	0.04	0.01	0.06	0.00	0.01	0.00	0.00	0.01	0.01	0.01		0.02		0.00		0.01	0.01
73	0.03	0.03	0.04		0.71	0.04	0.01	0.04	0.00	0.02	0.00	0.01	0.01	0.00	0.00	0.00	0.00		0.00	0.00	0.00			0.01
54	0.03	0.02	0.02	0.03		0.64	0.02	0.05	0.01	0.03	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.01	0.01	0.00	0.00	0.00
41	0.02	0.02	0.01	0.01	0.03		0.70	0.09	0.00	0.03	0.01	0.01	0.01	0.01	0.00	0.00	0.00		0.00	0.00				0.01
44	0.01			0.00		0.04		0.75	0.05	0.04	0.02	0.02	0.01	0.01	0.00	0.01	0.01					0.00		
38	0.02	0.01	0.02	0.00	0.01	0.01	0.03		0.52	0.16	0.02	0.04	0.02	0.01	0.01	0.02	0.01	0.02	0.01			0.00	0.00	0.00
71	0.02		0.01	0.00		0.01	0.02	0.02		0.78	0.03	0.01	0.01	0.00	0.01	0.00	0.00	0.01	0.01		0.00			0.00
18	0.03	0.02	0.01	0.01	0.01	0.01	0.01	0.04	0.02		0.53	0.06	0.03	0.03	0.01	0.02	0.01	0.02	0.01	0.00	0.01	0.01	0.01	0.01
45	0.01	0.01	0.01		0.01		0.00	0.02	0.00	0.04		0.68	0.04	0.02	0.02	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01
39	0.01	0.01	0.00	0.01	0.01	0.00	0.02	0.01	0.00	0.03	0.02		0.67	0.05	0.02	0.03	0.01		0.01	0.02	0.01	0.00	0.01	0.00
57	0.01	0.01				0.01	0.01	0.01	0.00	0.01	0.01	0.03		0.72	0.03	0.04	0.01	0.02	0.01	0.00	0.01	0.00	0.01	
22	0.01				0.00		0.01		0.01	0.01	0.00	0.01	0.02		0.70	0.03	0.03	0.04	0.02	0.00	0.01	0.01	0.01	
60	0.00	0.01	0.00	0.00	0.01	0.01		0.00	0.00	0.01	0.00	0.01	0.01	0.03		0.75	0.03	0.01	0.01	0.01	0.01	0.01	0.01	0.01
64	0.02	0.01	0.01	0.01	0.00	0.00	0.01	0.01		0.01	0.02	0.01	0.01	0.02	0.05		0.59	0.03	0.02	0.03	0.01	0.01	0.01	0.01
61	0.02	0.01	0.00		0.01	0.00	0.00	0.01		0.01	0.01	0.00		0.00	0.00	0.00		0.71	0.03	0.02	0.03		0.01	0.02
51	0.01			0.00	0.00		0.01	0.01	0.01	0.01	0.01		0.01	0.00	0.01	0.01	0.01		0.71	0.03	0.00	0.01	0.02	0.03
74	0.01	0.01	0.02		0.01	0.01		0.01		0.01	0.00	0.01	0.01	0.00			0.01	0.02		0.67	0.05	0.02	0.01	0.02
37	0.01		0.00	0.00				0.01		0.01	0.01	0.00	0.00	0.00			0.00	0.01	0.02		0.73	0.05	0.03	0.01
63	0.02	0.00	0.00	0.00	0.01			0.00	0.00	0.01	0.00		0.01	0.00	0.01	0.00	0.01	0.01	0.01	0.02		0.69	0.05	0.01
65	0.00		0.01		0.00	0.00		0.00				0.00	0.00	0.01			0.00	0.00	0.01	0.01	0.02		0.78	0.03
20	0.01	0.01	0.00		0.00			0.01		0.01				0.00				0.01	0.01	0.01	0.02	0.01		0.70
79	0.01	0.01			0.00	0.00		0.00	0.01	0.01			0.00	0.00	0.00			0.01	0.01	0.00	0.01	0.01	0.02	
66	0.01		0.00		0.01		0.00		0.00	0.00	0.01	0.01	0.01	0.00			0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.01
70	0.01	0.00			0.01		0.00	0.00	0.00	0.01		0.00		0.00				0.01	0.01	0.01	0.01		0.01	
62			0.00	0.01		0.00		0.01			0.00					0.00				0.00	0.01	0.01	0.01	0.02
24	0.01	0.01	0.00		0.00						0.00	0.00				0.00	0.01		0.01			0.00	0.00	0.00
32	0.02	0.01	0.00		0.01	0.00				0.00	0.00		0.00			0.00	0.02	0.02	0.00	0.00		0.00		0.01
28	0.01	0.01	0.00		0.01			0.00		0.01		0.01			0.00	0.00		0.01	0.01			0.00	0.00	
34		0.00		0.00							0.01				0.01		0.01				0.00		0.00	
50	0.01	0.01	0.00		0.00	0.00		0.00			0.00		0.00	0.01				0.00	0.00		0.00	0.00		
72		0.01		0.00	0.00	0.00								0.00							0.00			
85		0.01	0.00				0.00		0.00					0.00			0.00	0.01	0.00	0.00	0.00		0.01	0.00
67	0.01	0.00	0.00						0.00	0.01	0.00			0.01	0.00			0.00	0.00			0.00		
33	0.01		0.00					0.00	0.00			0.00		0.00	0.00			0.00	0.01	0.01				
68	0.01	0.01	0.02	0.01	0.00	0.01		0.00	0.00	0.01	0.01	0.01			0.00		0.01		0.00	0.01	0.00	0.00	0.00	0.00
29	0.01	0.00	0.01					0.00		0.01	0.01	0.00		0.00			0.00	0.00						
47	0.01	0.01	0.01	0.00		0.00		0.01	0.01	0.02					0.00		0.00	0.00	0.00		0.01		0.01	
58	0.01	0.00			0.00			0.01	0.00		0.00	0.00	0.00		0.01	0.00	0.00			0.00		0.00	0.00	
40	0.00	0.01	0.01		0.00	0.00		0.01		0.01						0.00		0.00			0.00		0.01	0.00
36	0.02	0.02	0.01		0.01	0.01	0.01	0.01	0.00	0.01		0.01	0.00	0.00	0.00		0.01	0.00	0.00	0.01			0.01	0.00
48	0.00		0.01		0.00					0.00											0.00			
19		0.01	0.00					0.00							0.00	0.01				0.00			0.00	0.00
46	0.01		0.01			0.00		0.00			0.01	0.00								0.00				
31	0.00	0.00	0.01	0.00				0.00		0.01	0.00				0.00					0.00				
35	0.02			0.01	0.00			0.01	0.01	0.00					0.01									0.00
43	0.08	0.03	0.02		0.01	0.01	0.02	0.02	0.01	0.02		0.02	0.02		0.03	0.01	0.02		0.02	0.02	0.01	0.02	0.01	0.02

66	70	62	24	32	28	34	50	72	85	67	33	68	29	47	58	40	36	48	19	46	31	35	43
0.00	0.00	0.00	0.00	0.01	0.00		0.00		0.00	0.00		0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01
0.00	0.00			0.00	0.00		0.00		0.00			0.00		0.00	0.00	0.00	0.00		0.00	0.00			0.00
0.01	0.00		0.00	0.00	0.00	0.00	0.00	0.00		0.00		0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		0.00	0.00
			0.00	0.00	0.00			0.00		0.00										0.00			0.00
0.01	0.00		0.00		0.00		0.00			0.00		0.00				0.00	0.01				0.00		0.01
			0.00	0.00	0.00	0.00				0.00		0.00			0.00						0.00		0.00
0.00		0.00			0.00	0.01	0.00					0.00		0.00	0.00								0.00
0.01	0.00	0.00					0.01		0.00			0.00		0.00	0.00		0.01			0.00	0.00	0.00	0.01
	0.00		0.00		0.00	0.01	0.01	0.00	0.00			0.00	0.00				0.00		0.01		0.00	0.00	
0.01	0.01	0.00	0.01	0.01	0.00		0.01	0.00	0.00		0.00	0.00	0.00	0.00	0.00	0.01	0.01			0.00		0.00	
0.01		0.00	0.00				0.00		0.00			0.01	0.01	0.01	0.00	0.00	0.00	0.00		0.00	0.00		0.00
0.01	0.01		0.00	0.00	0.01		0.00		0.00	0.00	0.01			0.01	0.01	0.00					0.01		0.01
0.01	0.01		0.00		0.00	0.00	0.00		0.00			0.01			0.01						0.00	0.00	0.01
0.01	0.01	0.00	0.01	0.00		0.00	0.01		0.00	0.01		0.00			0.01	0.01	0.01	0.00		0.00			
0.01		0.01	0.00		0.01	0.00			0.01	0.00		0.00	0.00	0.00	0.00	0.00	0.01					0.00	0.01
0.01	0.01		0.02	0.00	0.02	0.01		0.00	0.00	0.00				0.01		0.02	0.00	0.00	0.00			0.00	0.01
0.01	0.00	0.00	0.01	0.00	0.00	0.01		0.00	0.00			0.01	0.01	0.01			0.01				0.01		0.01
0.02		0.01	0.01	0.01	0.01	0.01	0.01		0.00	0.01		0.01		0.00		0.01	0.00						0.00
0.02	0.01	0.00	0.00	0.00	0.01		0.01					0.00	0.00	0.01		0.01	0.01				0.00		0.00
0.02	0.02	0.00	0.00	0.01	0.01	0.00		0.01	0.00			0.00		0.00	0.00		0.00		0.01		0.00		0.01
0.03	0.02	0.01	0.01		0.01	0.00		0.00	0.01	0.00	0.00		0.00	0.01	0.00	0.01	0.01			0.00	0.00		0.01
0.02	0.02	0.01	0.00	0.01	0.01	0.00	0.00					0.00	0.00	0.00	0.01	0.01					0.00	0.00	
0.05	0.01	0.02	0.02	0.00	0.02	0.00	0.00		0.01		0.01	0.01	0.01	0.00	0.00	0.01	0.01	0.00			0.00		0.02
0.69	0.02	0.04	0.01	0.05	0.02	0.00	0.01	0.01	0.01			0.01	0.01	0.01		0.00		0.00					0.00
	0.66	0.04	0.04	0.02	0.02	0.00	0.01	0.01	0.01	0.02	0.00	0.01		0.01	0.01	0.00	0.01			0.01	0.01		0.01
0.02		0.72	0.03	0.02	0.01	0.01	0.01	0.01	0.01	0.01		0.01	0.00	0.01	0.01	0.00	0.00		0.01		0.00		0.01
0.01	0.03		0.77	0.01	0.02	0.01	0.01			0.00		0.01		0.01	0.01	0.00	0.01		0.00				0.00
0.03	0.02	0.02		0.71	0.04	0.02	0.01		0.01	0.00		0.02	0.00	0.01	0.00	0.01	0.01	0.00	0.00	0.01			0.01
0.01	0.01	0.01	0.07		0.63	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.00	0.01	0.00	0.01		0.00	0.01	0.00		0.01
0.01		0.01	0.01	0.01		0.70	0.06	0.02	0.02	0.01	0.00	0.03	0.01	0.01	0.01	0.00		0.00		0.00	0.00		
0.00	0.00	0.00		0.00	0.04		0.81	0.03	0.01		0.00	0.02	0.01			0.00	0.00					0.00	
				0.00	0.00	0.01		0.74	0.05	0.02	0.00	0.03	0.00	0.00	0.01	0.02	0.00	0.00	0.00	0.00	0.00		0.01
0.00						0.00	0.01		0.86	0.04		0.01	0.00	0.01	0.01	0.00				0.00	0.00		
0.01	0.00		0.02			0.01	0.01	0.02		0.74	0.03	0.02	0.01		0.02	0.03	0.02	0.00		0.00	0.01		0.01
0.01			0.00	0.01	0.00	0.00	0.00	0.00	0.01		0.75	0.08	0.02	0.01	0.00	0.01	0.01	0.01					0.00
			0.01		0.01	0.00		0.00	0.01	0.00		0.82	0.02	0.02	0.00	0.01	0.00	0.00	0.00			0.00	
0.00	0.01		0.00		0.01	0.01	0.01		0.01	0.04	0.02		0.61	0.04	0.05	0.03	0.02	0.01		0.01	0.00		0.01
	0.00		0.00		0.00			0.00	0.01	0.02		0.03		0.80	0.02	0.02	0.00	0.01	0.00		0.00		0.00
0.01		0.00			0.01						0.00	0.02	0.03		0.71	0.02	0.03	0.00	0.00	0.01	0.03	0.00	0.03
0.00	0.00	0.01			0.00		0.01			0.01	0.00	0.01	0.01	0.03		0.74	0.06	0.02	0.01	0.01		0.00	0.01
	0.01	0.01	0.01	0.00		0.00		0.00	0.01	0.00		0.01	0.01	0.00	0.03		0.76	0.03	0.01	0.00		0.00	0.01
0.01		0.01	0.00	0.00	0.01		0.00				0.00	0.02	0.00		0.00	0.05		0.64	0.03	0.02	0.01	0.00	0.05
0.00	0.00		0.00			0.00					0.00	0.01	0.00	0.01	0.01	0.01	0.02		0.80	0.04	0.00	0.01	0.02
				0.00	0.00		0.00		0.01		0.00	0.02				0.01	0.00	0.01		0.84	0.01	0.02	0.01
0.00			0.00	0.01				0.00			0.00			0.00	0.02	0.00		0.01	0.03		0.83	0.01	0.02
		0.00	0.00	0.00			0.01					0.00		0.01	0.00		0.00		0.00	0.03		0.79	0.08
0.00										0.01	0.00	0.00	0.00				0.01	0.00			0.02		0.86
0.02	0.01	0.02	0.02	0.02	0.05	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.04	0.05	0.02	0.07	0.07	0.01	0.02	0.03	0.04	0.08	

Tabelle 24: Artikelmatrix

Label (TOC)	89	88	49	73	54	41	44	38	71	18	45	39	57	22	60	64	61	51	74	37	63	65	20	79
89		TOC	0.24	0.08	0.06	0.05	0.01	0.02	0.02	0.08	0.03	0.02	0.01	0.02	0.01	0.03	0.02	0.00	0.02	0.02	0.01	0.01	0.02	0.01
88	0.29		TOC	0.13	0.08	0.07	0.01	0.08	0.01	0.07	0.03	0.02	0.02	0.01	0.00	0.01	0.00	0.01	0.03	0.01		0.01	0.01	0.01
49	0.16	0.15		TOC	0.09	0.10	0.03	0.09	0.01	0.12	0.01	0.01	0.00	0.01	0.02	0.01	0.01		0.03		0.01		0.01	0.01
73	0.10	0.09	0.15		TOC	0.14	0.05	0.15	0.02	0.06	0.01	0.03	0.03	0.01	0.01	0.01	0.02		0.01	0.02	0.02			0.03
54	0.10	0.06	0.05	0.08		TOC	0.05	0.13	0.02	0.08	0.03	0.02	0.01	0.03	0.03	0.01	0.04	0.02	0.01	0.02	0.03	0.01	0.01	0.01
41	0.06	0.06	0.04	0.05	0.09		TOC	0.31	0.02	0.09	0.02	0.04	0.02	0.02	0.01	0.02	0.01		0.01	0.02				0.02
44	0.04			0.01		0.17		TOC	0.18	0.15	0.10	0.09	0.04	0.02	0.01	0.03	0.03					0.01		
38	0.04	0.02	0.04	0.01	0.03	0.03	0.06		TOC	0.33	0.04	0.08	0.04	0.03	0.02	0.04	0.02	0.03	0.02			0.01	0.01	0.01
71	0.09		0.03	0.01		0.03	0.09	0.11		TOC	0.15	0.05	0.07	0.01	0.03	0.01	0.01	0.04	0.03		0.01			0.01
18	0.06	0.03	0.02	0.02	0.02	0.02	0.02	0.08	0.04		TOC	0.13	0.07	0.06	0.03	0.04	0.02	0.05	0.03	0.00	0.03	0.01	0.02	0.02
45	0.02	0.03	0.02		0.02		0.01	0.05	0.01	0.13		TOC	0.11	0.07	0.05	0.05	0.06	0.04	0.04	0.02	0.02	0.02	0.02	0.04
39	0.02	0.03	0.01	0.02	0.03	0.01	0.05	0.02	0.01	0.08	0.05		TOC	0.15	0.07	0.08	0.03		0.04	0.07	0.04	0.01	0.02	0.01
57	0.05	0.02				0.02	0.02	0.02	0.01	0.03	0.04	0.11		TOC	0.12	0.14	0.04	0.08	0.02	0.01	0.05	0.01	0.02	
22	0.02				0.01		0.02		0.03	0.05	0.01	0.05	0.07		TOC	0.09	0.08	0.13	0.07	0.01	0.04	0.03	0.02	
60	0.01	0.02	0.01	0.01	0.02	0.02		0.01	0.01	0.04	0.01	0.02	0.02	0.13		TOC	0.13	0.05	0.04	0.02	0.06	0.02	0.02	0.05
64	0.06	0.01	0.02	0.03	0.01	0.01	0.01	0.02		0.01	0.04	0.01	0.03	0.05	0.12		TOC	0.07	0.04	0.07	0.03	0.01	0.01	0.01
61	0.06	0.02	0.01		0.03	0.01	0.01	0.04		0.02	0.02	0.01		0.01	0.01	0.01		TOC	0.09	0.06	0.11		0.04	0.07
51	0.02			0.01	0.01		0.02	0.02	0.02	0.04	0.02		0.02	0.01	0.02	0.04	0.04		TOC	0.10	0.01	0.03	0.06	0.09
74	0.04	0.03	0.05		0.03	0.03		0.03		0.04	0.01	0.04	0.02	0.01			0.04	0.07		TOC	0.14	0.05	0.04	0.05
37	0.02		0.01	0.01				0.02		0.02	0.02	0.01	0.01	0.01			0.01	0.05	0.07		TOC	0.17	0.10	0.04
63	0.05	0.01	0.01	0.01	0.02			0.01	0.01	0.02	0.01		0.04	0.01	0.02	0.01	0.04	0.02	0.03	0.07		TOC	0.16	0.03
65	0.02		0.03		0.02	0.02		0.02				0.02	0.02	0.03			0.02	0.02	0.03	0.05	0.09		TOC	0.16
20	0.02	0.02	0.01		0.01			0.02		0.04				0.01				0.03	0.02	0.04	0.07	0.03		TOC
79	0.03	0.04			0.01	0.01		0.01	0.02	0.03			0.01	0.01	0.01			0.02	0.02	0.01	0.03	0.04	0.06	
66	0.02		0.01		0.02		0.01		0.01	0.01	0.03	0.02	0.02	0.01			0.03	0.03	0.03	0.01	0.03	0.02	0.04	0.03
70	0.03	0.01			0.03		0.01	0.01	0.01	0.04		0.01		0.01				0.03	0.03	0.03	0.03		0.05	
62			0.02	0.03		0.02		0.06			0.02					0.02				0.02	0.03	0.06	0.03	0.09
24	0.03	0.02	0.01		0.01						0.01	0.01				0.01	0.02		0.02			0.01	0.01	0.01
32	0.05	0.02	0.01		0.02	0.01				0.01	0.01		0.01			0.01	0.05	0.05	0.01	0.01		0.01		0.04
28	0.02	0.04	0.01		0.02			0.01		0.04		0.02			0.01	0.01		0.04	0.02			0.01	0.01	
34		0.02		0.02							0.04				0.04		0.04				0.02		0.02	
50	0.05	0.03	0.02		0.02	0.02		0.02			0.02		0.02	0.05				0.02	0.02		0.02	0.02		
72		0.06		0.03	0.03	0.03								0.03							0.03			
85		0.03	0.01				0.01		0.01					0.01			0.01	0.03	0.01	0.01	0.01		0.03	0.01
67	0.05	0.02	0.02						0.02	0.03	0.02			0.03	0.02			0.02	0.02			0.02		
33	0.05		0.03					0.03	0.03			0.03		0.03	0.03			0.03	0.05	0.05				
68	0.02	0.02	0.05	0.02	0.01	0.02		0.01	0.01	0.03	0.02	0.02			0.01		0.02		0.01	0.02	0.01	0.01	0.01	0.01
29	0.04	0.02	0.06					0.02		0.04	0.04	0.02		0.02			0.02	0.02						
47	0.03	0.04	0.03	0.01		0.01		0.04	0.03	0.06					0.01		0.01	0.01	0.01		0.03		0.03	
58	0.03	0.01			0.01			0.03	0.01		0.01	0.01	0.01		0.03	0.01	0.01			0.01		0.01	0.01	
40	0.02	0.03	0.05		0.02	0.02		0.03		0.06						0.02		0.02			0.02		0.03	0.02
36	0.06	0.05	0.02		0.03	0.02	0.02	0.04	0.01	0.02		0.02	0.01	0.01	0.01		0.02	0.01	0.01	0.03			0.02	0.01
48	0.02		0.04		0.02					0.02											0.02			
19		0.06	0.03					0.03							0.03	0.06				0.03			0.03	0.03
46	0.05		0.08			0.03		0.03			0.05	0.03								0.03				
31	0.02	0.02	0.04	0.02				0.02		0.04	0.02				0.02					0.02				
35	0.14			0.07	0.03			0.07	0.07	0.03					0.07									0.03
43	0.08	0.03	0.02		0.01	0.01	0.02	0.02	0.01	0.02		0.02	0.02		0.03	0.01	0.02		0.02	0.02	0.01	0.02	0.01	0.02

66	70	62	24	32	28	34	50	72	85	67	33	68	29	47	58	40	36	48	19	46	31	35	43
0.01	0.00	0.01	0.01	0.02	0.01		0.01		0.00	0.00		0.01	0.02	0.01	0.01	0.02	0.01	0.00	0.01	0.01	0.00	0.01	0.02
0.01	0.01			0.00	0.00		0.00		0.01			0.00		0.01	0.00	0.01	0.01		0.00	0.00			0.01
0.01	0.01		0.00	0.00	0.00	0.00	0.00	0.00		0.00		0.01	0.00	0.01	0.00	0.01	0.01	0.01	0.00	0.01		0.00	0.01
			0.01	0.01	0.01			0.01		0.01										0.01			0.01
0.03	0.01		0.01		0.01		0.01			0.01		0.01				0.01	0.03				0.01		0.03
			0.01	0.02	0.01	0.02				0.02		0.01			0.01						0.01		0.01
0.01		0.01			0.01	0.02	0.01					0.01		0.01	0.01								0.01
0.02	0.01	0.00					0.01		0.01			0.00		0.00	0.00		0.01			0.00	0.01	0.00	0.01
	0.01		0.01		0.01	0.03	0.03	0.01	0.01			0.01	0.01				0.01		0.03		0.01	0.01	
0.02	0.02	0.00	0.02	0.01	0.01		0.03	0.00	0.00		0.00	0.00	0.00	0.00	0.00	0.02	0.03			0.01		0.00	
0.02		0.01	0.01				0.01		0.01			0.02	0.02	0.02	0.01	0.01	0.01	0.01		0.01	0.01		0.01
0.03	0.02		0.01	0.01	0.02		0.01		0.01	0.01	0.02			0.02	0.02	0.01					0.02		0.02
0.04	0.02		0.01		0.01	0.01	0.01		0.01			0.02			0.04						0.01	0.01	0.02
0.02	0.05	0.01	0.03	0.01		0.01	0.03		0.01	0.02		0.01			0.02	0.05	0.02	0.01		0.01			
0.04		0.04	0.01		0.02	0.01			0.02	0.01		0.01	0.01	0.01	0.01		0.02					0.01	0.02
0.03	0.02		0.05	0.01	0.04	0.01		0.01	0.01	0.01				0.02		0.04	0.01	0.01	0.01			0.01	0.03
0.03	0.01	0.01	0.04	0.01	0.01	0.04		0.01	0.01			0.04	0.03	0.03			0.03				0.02		0.02
0.06		0.04	0.03	0.03	0.05	0.03	0.03		0.01	0.03		0.04		0.01		0.04	0.01						0.01
0.05	0.04	0.01	0.01	0.01	0.02		0.04					0.01	0.01	0.03		0.03	0.03					0.01	0.01
0.06	0.07	0.01	0.01	0.02	0.04	0.01		0.02	0.01			0.01		0.01	0.01		0.01		0.04		0.01		0.05
0.09	0.05	0.02	0.02		0.03	0.01		0.01	0.02	0.01	0.01		0.01	0.03	0.01	0.03	0.02			0.01	0.01		0.04
0.11	0.08	0.03	0.02	0.03	0.03	0.02	0.02					0.02	0.02	0.02	0.03	0.03					0.02	0.02	
0.17	0.03	0.07	0.06	0.01	0.05	0.01	0.01		0.02		0.02	0.04	0.03	0.01	0.01	0.03	0.02	0.01			0.01		0.05
TOC	0.05	0.12	0.02	0.17	0.07	0.01	0.02	0.04	0.02				0.03	0.03	0.03		0.01		0.01				0.01
	TOC	0.12	0.10	0.05	0.05	0.01	0.03	0.02	0.02	0.05	0.01	0.02		0.03	0.02	0.01	0.03				0.02	0.02		0.03
0.09		TOC	0.13	0.08	0.04	0.03	0.05	0.04	0.03	0.04		0.04	0.01	0.03	0.03	0.01	0.01		0.03		0.01			0.03
0.05	0.11		TOC	0.05	0.08	0.03	0.05			0.02		0.06		0.03	0.03	0.02	0.06		0.02				0.02
0.11	0.06	0.07		TOC	0.15	0.07	0.02		0.05	0.01		0.08	0.01	0.02	0.01	0.02	0.05	0.01	0.01	0.02			0.02
0.02	0.03	0.03	0.19		TOC	0.04	0.05	0.02	0.04	0.03	0.03	0.02	0.03	0.01	0.04	0.01	0.02		0.01	0.02	0.01		0.04
0.02		0.02	0.02	0.04		TOC	0.21	0.06	0.06	0.02	0.01	0.10	0.04	0.02	0.04	0.01		0.01		0.01	0.01		
0.02	0.02	0.02		0.02	0.20		TOC	0.15	0.07		0.02	0.13	0.07			0.02	0.02					0.02	
				0.02	0.02	0.05		TOC	0.21	0.06	0.02	0.11	0.02	0.02	0.03	0.08	0.02	0.02	0.02	0.02	0.02		0.05
0.03						0.03	0.10		TOC	0.26		0.10	0.03	0.06	0.06	0.03				0.03	0.03		
0.03	0.01		0.06			0.03	0.03	0.06		TOC	0.13	0.07	0.04		0.06	0.10	0.07	0.01		0.01	0.03		0.03
0.03			0.02	0.03	0.02	0.02	0.02	0.02	0.05		TOC	0.31	0.06	0.05	0.02	0.05	0.03	0.03					0.02
			0.05		0.05	0.03		0.03	0.08	0.03		TOC	0.13	0.10	0.03	0.05	0.03	0.03	0.03				0.03	
0.01	0.02		0.01		0.02	0.02	0.02		0.03	0.09	0.06		TOC	0.09	0.12	0.07	0.05	0.02			0.02	0.01		0.03
	0.02		0.02		0.02			0.02	0.06	0.08		0.15		TOC	0.10	0.08	0.02	0.04	0.02			0.02		0.02
0.04		0.01			0.03						0.01	0.06	0.09		TOC	0.06	0.10	0.01	0.01	0.03	0.09	0.01		0.09
0.01	0.01	0.03			0.01		0.03			0.03	0.01	0.04	0.03	0.12		TOC	0.22	0.06	0.03	0.03		0.01	0.04
	0.03	0.05	0.05	0.02		0.02		0.02	0.03	0.02		0.06	0.05	0.02	0.12		TOC	0.12	0.03	0.02		0.02	0.05
0.02		0.02	0.01	0.01	0.02		0.01				0.01	0.06	0.01		0.01	0.14		TOC	0.07	0.05	0.02	0.01	0.14
0.02	0.02		0.02			0.02					0.02	0.04	0.02	0.07	0.07	0.07	0.11		TOC	0.20	0.02	0.04	0.11
				0.03	0.03		0.03		0.06		0.03	0.12				0.06	0.03	0.06		TOC	0.09	0.12	0.06
0.03			0.03	0.05				0.03			0.03			0.03	0.11	0.03		0.05	0.16		TOC	0.05	0.13
		0.02	0.02	0.02			0.04					0.02		0.06	0.02		0.02		0.02	0.15		TOC	0.36
0.03										0.07	0.03	0.03	0.03				0.07	0.03			0.17		TOC
0.02	0.01	0.02	0.02	0.02	0.05	0.01	0.01	0.01	0.01	0.02	0.02	0.02	0.04	0.05	0.02	0.07	0.07	0.01	0.02	0.03	0.04	0.08	

Tabelle 25: Artikelmatrix ohne Inhaltsverzeichnis

8.2 Relevanzliste L3T

Artikel	Dazu relevante Artikel-IDs
#grundlagen (ID: 88)	68 19 28 40 65 73 18 49 54 63 66 70 72
#ipad (ID: 49)	54 68 85 37 61 47 88 73 65 18
#hypertext (ID: 73)	57 74 41 19 35 31 32 65 88 18 49 54
#fernunterricht (ID: 54)	49 24 36 65 88 18 49 73
#infosysteme (ID: 41)	73 19 35 31 32 71 36 39 58
#webtech (ID: 44)	34 71 39 45 51 57 62 36 79 61 72 66 38 74
#multimedia (ID: 38)	41 39 58 65 63 24 20 85 74 66 44 39
#usability (ID: 71)	34 65 40 41 44 74 45 51 57 62 36 79 61 72
#lerntheorie (ID: 18)	88 73 65 49 20 67 46 54 32 29 79 46
#medienpaedagogik (ID: 45)	44 51 57 62 79 36 61 72 71 60 67 29 85 50
#systeme (ID: 39)	41 58 44 38
#kommunikation (ID: 57)	24 73 44 51 45 62 36 79 61 72 19 31 34 58 79
#organisation (ID: 22)	36 40 63
#literatur (ID: 60)	63 36 48 62 45
#telweiterbildung (ID: 64)	36 62 58
#netzgeneration (ID: 61)	68 85 37 49 72 79 32 45 51 57 62 36 44 71
#gedaechnis (ID: 51)	44 45 57 62 36 79 61 72 74 71
#mobil (ID: 74)	73 51 66 71 44 38 24 20 85 63
#assessment (ID: 37)	49 68 85 61 58 43 28 70 72
#blogging (ID: 63)	38 24 20 85 74 66 43 28 70 88 72 60 22
#ebook (ID: 65)	38 71 40 88 73 18 49 54
#educast (ID: 20)	38 24 63 85 74 66 67 46 18 43 29 79 28
#game (ID: 79)	61 72 44 45 57 62 36 51 19 31 34 58 20 29 67 46 18 71
#kollaboration (ID: 66)	74 38 24 20 85 63 28 70 88 72 44
#qualitaet (ID: 70)	37 43 28 72 63 66 28 72 70 33
#openaccess (ID: 62)	44 51 57 45 36 79 61 72 71 60 48 64
#videokonferenz (ID: 24)	54 57 63 38 20 74 85 66

#virtuellewelt (ID: 32)	73 19 35 31 41 61 18
#ant (ID: 28)	63 66 70 88 20 37 43 72
#barrierefrei (ID: 34)	44 71 79 31 19 58 57
#designforschung (ID: 50)	85 45
#gender (ID: 72)	57 61 79 48 37 43 28 70 63 66 88 28 44 45 62 36 51 71
#innovation (ID: 85)	50 54 68 37 61 38 24 20 63 74 66 45
#kognition (ID: 67)	20 46 18 29 79 45 29
#lernservice (ID: 33)	36 70 58
#medientheorie (ID: 68)	88 19 54 85 37 61
#kindergarten (ID: 29)	67 45 47 20 79 67 46 18
#schule (ID: 47)	49 29 45 35 31
#hochschule (ID: 58)	64 41 38 39 57 70 46 33 37 79 19 34 31
#unternehmen (ID: 40)	88 22 65 71 36
#erwachsenenbildung (ID: 36)	22 64 54 60 33 40 44 51 45 62 79 61 72 71
#entwicklungszusammenarbeit (ID: 48)	60 62 72
#medizin (ID: 19)	68 88 73 41 35 32 79 31 34 58 57
#labor (ID: 46)	20 67 18 29 79 58
#mathematik (ID: 31)	73 41 19 32 47 35 79 34 58 57
#sport (ID: 35)	73 41 19 31 32 47
#sprache (ID: 43)	63 37 28 70 72 20

Tabelle 26: Relevanzliste - Experteneinschätzung

8.3 Testergebnisse

Traces: Precision			
#Test	L < 10	L < 20	L < 50
1	0,34701	0,34461	0,33178
2	0,35595	0,33423	0,33390
3	0,35670	0,34216	0,33110
4	0,33095	0,33631	0,33252
5	0,34137	0,34146	0,33266
6	0,33673	0,33166	0,32947
7	0,40341	0,37553	0,34419
8	0,30238	0,30622	0,31660
9	0,30645	0,31938	0,32399
10	0,35539	0,33071	0,31987
Summe	3,43635	3,36229	3,29606
Durchschnitt:	**0,34364**	**0,33623**	**0,32961**

Tabelle 27: Ergebnis Precision: Plugin-Algorithmus

WSK: Precision			
#Test	L < 10	L < 20	L < 50
1	0,42179	0,40779	0,38351
2	0,49643	0,45115	0,41751
3	0,45545	0,45647	0,41055
4	0,45607	0,45146	0,42170
5	0,41784	0,41434	0,39875
6	0,41477	0,41209	0,39124
7	0,46537	0,43827	0,40009
8	0,41791	0,42688	0,39503
9	0,37037	0,37773	0,37051
10	0,36275	0,35772	0,36631
Summe	4,27875	4,19390	3,95521
Durchschnitt:	**0,42787**	**0,41939**	**0,39552**

Tabelle 28: Ergebnis Precision: Wahrscheinlichkeitsalgorithmus

Traces: Recall			
#Test	L < 10	L < 20	L < 50
1	0,20419	0,21220	0,20631
2	0,19914	0,20037	0,20584
3	0,20652	0,20012	0,20169
4	0,19290	0,20316	0,20913
5	0,21252	0,21013	0,20620
6	0,20756	0,20349	0,20579
7	0,22403	0,21943	0,21200
8	0,18179	0,18312	0,19832
9	0,19057	0,20022	0,20402
10	0,23227	0,20863	0,20236
Summe	2,05149	2,04087	2,05166
Durchschnitt:	**0,20515**	**0,20409**	**0,20517**

Tabelle 29: Ergebnis Recall: Plugin-Algorithmus

WSK: Recall			
#Test	L < 10	L < 20	L < 50
1	0,24466	0,23919	0,23482
2	0,27908	0,25735	0,25042
3	0,25010	0,24670	0,23747
4	0,25540	0,25598	0,24857
5	0,25114	0,25108	0,24531
6	0,25565	0,25234	0,24525
7	0,25060	0,24089	0,23520
8	0,23829	0,24511	0,24092
9	0,22137	0,22631	0,22468
10	0,22791	0,22559	0,22559
Summe	2,47421	2,44055	2,38823
Durchschnitt:	**0,24742**	**0,24406**	**0,23882**

Tabelle 30: Ergebnis Recall: Wahrscheinlichkeitsalgorithmus

Traces: Hits			
#Test	L < 10	L < 20	L < 50
1	0,42353	0,62195	0,83240
2	0,40217	0,47170	0,74768
3	0,40559	0,59786	0,79421
4	0,41954	0,55062	0,73744
5	0,37113	0,44033	0,75540
6	0,37903	0,50970	0,69504
7	0,45045	0,54787	0,74335
8	0,30238	0,41237	0,70928
9	0,39450	0,51852	0,64147
10	0,30000	0,33750	0,59290
Summe	3,84833	5,00841	7,24917
Durchschnitt:	**0,38483**	**0,50084**	**0,72492**

Tabelle 31: Ergebnis Hit-Ratio: Plugin-Algorithmus

WSK: Hits			
#Test	L < 10	L < 20	L < 50
1	0,57059	0,72764	0,87709
2	0,61413	0,68763	0,87052
3	0,53147	0,73995	0,88811
4	0,67816	0,76543	0,88205
5	0,53608	0,62551	0,85552
6	0,50000	0,67590	0,81194
7	0,59459	0,66489	0,80988
8	0,43023	0,57732	0,80736
9	0,53211	0,67407	0,77132
10	0,45556	0,48750	0,70765
Summe	5,44292	6,62585	8,28144
Durchschnitt:	**0,54429**	**0,66259**	**0,82814**

Tabelle 32: Ergebnis Hit-Ratio: Wahrscheinlichkeitsalgorithmus

Literaturverzeichnis

[1] G. Adomavicius, A. Tuzhilin, Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. In: IEEE Transactions on Knowledge and Data Engineering, Vol. 17, No. 6, 2005

[2] X. Amatriain, A. Jaimes, N. Oliver, and J. Pujol: Data Mining Methods for Recommender Systems. In: F. Ricci, L. Rokach, B. Shapira, Paul B. Kantor (Eds.): Recommender System Handbook, Springer Verlag New York pp. 39-71 2011

[3] M. W. Berry, S. T. Dumais, G.W. O'Brien: Using Linear Algebra For Intelligent Information Retrieval, Computer Science Departement, CS-94-270 1994

[4] G. L. Bretthorst: An Introduction to Parameter Estimation using Bayesian Probability Theory. In: P. F. Fougère (Eds.): Maximum Entropy and Bayesian Methods, Kluwer Academic Publishers, The Netherlands 1990 pp. 53-79

[5] R. Burke: Hybrid Systems for Personalized Recommendations. In: B. Mobasher and S. S. Anand (Eds.): ITWP 2003, LNAI 3169, pp. 133-152, Springer Verlag Berlin Heidelberg 2005

[6] Z. Chedrawy and S. S. R. Abidi: A Web Recommender System for Recommending Predicting and Personalizing Music Playlists. In: G. Vossen, D. D. E. Long and J. X. Yu (Eds.): WISE 2009, LNCS 5802, pp. 335-342, Springer Verlag Berlin Heidelberg 2009

[7] C. Desrosiers and G. Karypis: A Comprehensive Survey of Neighborhood-based Recommender Methods. In: F. Ricci, L. Rokach, B. Shapira, P.B. Kantor (Eds.): Recommender Systems Handbook, Springer Verlag New York 2011

[8] J. L. Herlocker, J. A. Konstan, L. G. Terveen and J. T. Riedl, Evaluating Collaborative Filtering Recommender Systems. In: ACM Transactions on Informations Systems, Vol. 22, No. 1, Jannuary 2004, pp 5-53

[9] X. Jin, Y. Zhou, B. Mobasher: A Maximum Entropy Web Recommendation System: Combining Collaborative and Content Features, In: Proc. IEEE International Conference on Information Technology Coding and Computing, Las Vegas 2005

[10] K. Kawazu, M. Murao, T. Ohta, M. Mase and T. Maneo: A System to Construct an Interest Model of User Based on Information in Browsed Web Pages by User. In: J. A. Jacko (Eds.): Human-Computer Interaction, Part III, HCII 2009, LNCS 5612, pp. 404-313, Springer Verlag Berlin Heidelberg 2009

[11] B. M. Kim, Q. Li, J. W. Kim and J. Kim: A New Collaborative Recommender System Addressing Three Problems. In: C. Zhang, H. W. Guesgen, W. K. Yeap (Eds.): PRICAI 2004, LNAI 3157, pp. 495 - 504, Springer Verlag Heidelberg 2004

[12] T.-H. Kim and S.-B. Yang: Using Attributes to Improve Prediction Quality in Collaborative Filtering. In: K. Bauknecht, M. Bichler and B. Pröll (Eds.): EC-Web 2004, LNCS 3182, pp. 1-10, Springer Verlag Berlin Heidelberg 2004

[13] T.-H. Kim and S.-B. Yang: An Effective Recommendation Algorithm for clustering-Based Recommender Systems. In: S. Zhang and R. Jarvis (Eds.): AI 2005, LNAI 3809, pp. 1150-1153, Springer Verlag Berlin Heidelberg 2005

[14] A. Klahod: Empfehlungssysteme - Recommender Systems - Grundlagen, Konzepte und Lösungen, Vieweg und Teubner Verlag / GWV Fachverlag Gmbh, Wiesbaden 2009

[15] R. Kohavi: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: C.S. Mellish (Eds.): Proceedings of the 14th International Joint Conference on Articial Intelligence, Morgan Kaufmann Publishers, Inc. 1995 pp. 1137 - 1143

[16] T. K. Landauer, P. W. Foltz, D. Laham: Introduction to Latent Semantic Analysis, In: Discourse Processes, 25, S. 259 - 284, 1998

[17] P. Lops, M. De Gemmis and G. Semerano: Content-Based Recommender Systems: State of the Art and Trends. In: F. Ricci, L.Rokach, B. Shapira, P.B. Kantor (Eds.): Recommender Systems Handbook, Springer Verlag New York 2011

[18] F. Lorenzi, F. Ricci: Case-Based Recommender System: A Unifying View. In: B. Mobasher and S. S. Anand (Eds.): ITWP 2003, LNAI 3169, pp. 89-113, Springer Verlag Berlin Heidelberg 2005

[19] C. D. Manning, P. Raghavan, H.Schütze: An Introduction to Information Retrieval, Cambridge University Press 2009, pp. 154 - 157

[20] U. Panniello, M. Gorgolione and C. Palmisano: Comparing Pre-filtering and Post-filtering Approach in a Collaborative Contextual Recommender System: An Application to E-Commerce. In: T. di Noia and F. Buccafurri (Eds.): EC-Web 2009, LNCS 5692, pp. 348-359, Springer Verlag Berlin Heidelberg 2009

[21] J. Ramos: Using TF-IDF to Determine Word Relevance in Document Queries, Department of Computer Science, Rutgers University, 23515 BPO Way, Piscataway, NY, 08855 2002

[22] S. Schaffert, T. Bürger, W. Hilzensauer, C. Schneider und D. Wieden-Bischof: Empfehlungen im Web. Konzepte und Realisierungen, Salzburg Research 2010

[23] J.-H. Yoo, K.-S. Ahn, J. Jun and P.-K. Rhee: A Recommendation System for Intelligent User Interface: Collaborative Filtering Approach. In: M. Gh. Negoita et al. (Eds.): KES 2004, LNAI 3215, pp. 869-879, Springer Verlag Berlin Heidelberg 2004