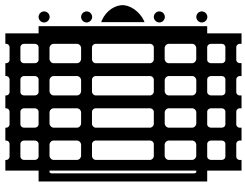
Technische Universität Chemnitz
Fakultät für Informatik
Professur Künstliche Intelligenz



TECHNISCHE UNIVERSITÄT
CHEMNITZ

Diplomarbeit

im Studiengang Angewandte Informatik

Vorgelegt von
Tolleiv Nietsch

Skalierbare Item Recommendation in Big-Data und Suchindexen

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende schriftliche Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keinem anderen Prüfer als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Vorname:	Tolleiv
Name:	Nietsch
Matrikelnummer:	172314

Chemnitz, den 27. August 2013

_____

Tolleiv Nietsch

Betreuung und Prüfung durch:

Prof. Dr. Fred Hamker,	Professur Künstliche Intelligenz, TU-Chemnitz
Dr. Johannes Steinmüller	Professur Künstliche Intelligenz, TU-Chemnitz

Technische Universität Chemnitz, Fakultät für Informatik
Straße der Nationen 62, 09107 Chemnitz

Zusammenfassung

tbw

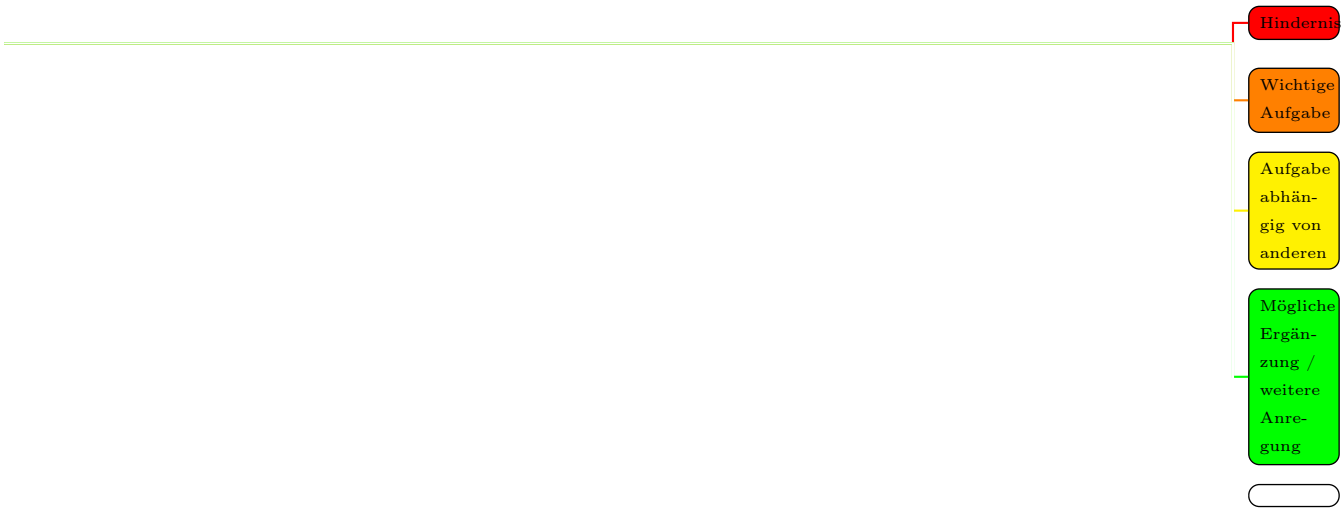
Inhaltsverzeichnis

*Liste der noch zu erledigenden Punkte

1 Einleitung

Aufbau der Arbeit:

- Grundlagen -> Kapitel 2
- Analyse+Design -> Kapitel 3
- Umsetzung -> Kapitel 4
- Auswertung -> Kapitel 5



2 Grundlagen

Im folgenden Abschnitt werden die notwendigen Grundlagen der verschiedenen Bestandteile einer personalisierten Suche beschrieben. Der erste Abschnitt beschreibt den Aufbau von Suchindexen und die Möglichkeiten zur Personalisierung der Ergebnisse. Der darauf folgende Abschnitt fasst Konzepte zur Bildung von Empfehlungen zusammen. Abschnitt 2.3 greift die Methode des kollaborativen Filterns nochmals auf und beschreibt die zugrunde liegenden Rechenmodelle. Der darauf folgende Abschnitt 2.4 beschreibt Herausforderungen die sich im praktischen Umgang mit kollaborativen Filtermodellen ergeben. Maßnahmen, um die vorgestellten Rechenmodelle auch auf über die Kapazitäten eines einzelnen Rechners hinausreichende Datenmenge anzuwenden, werden im Abschnitt 2.5 beschrieben. Methoden zum Vergleich bzw. zur Bewertung der Modelle werden im abschließenden Abschnitt 2.6 beschrieben.

2.1 Suchindexe

Mit dem Begriffen "Suche" und "Suchmaschinen" werden umgangssprachlich zahlreiche Methoden des Information Retrieval (IR) beschrieben, die durch Internet-Dienste wie Google[1] im Alltag vieler Menschen präsent sind. Im IR wird eine "Suche" formal als die Extraktion von Informationen aus einer Menge unstrukturierter Daten definiert. Innerhalb eines IR Systems liegen die Daten in Form von *Dokumenten* (Texte, Bilder, Videos) vor. Bei der Benutzung des IR Systems formuliert der Nutzer seinen *Informationsbedarf* mit Hilfe von *Anfragen*. Dokumente werden als *relevant* bezeichnet wenn die darin enthaltene Information dem Bedarf des Nutzers genügt. Im weiteren Sinne umfasst IR zudem das Filtern, Klassifizieren und Verarbeiten der gefundenen Dokumente.

Die durch den Nutzer formulierten Anfragen sind dabei, im Gegensatz zu Anfragen an strukturierte Datenbanken, nicht zwingend eindeutig. Sucht der Nutzer etwa nach "Fantasy Buch", kann "Harry Potter" in den Augen des Nutzers ein relevantes Dokument sein, ohne dass die Begriffe "Fantasy" oder "Buch" explizit darin vorkommen. [Manning u. a., 2008]

[1] http://www.google.com

2.1.1 Indexbildung

Da es bei einer großen Anzahl von vorhandenen Dokumenten sehr ineffizient wäre, wenn bei jeder Anfrage jedes Dokument geprüft werden müsste, verwendet man einen *invertierten Index* um Informationen zu Dokumenten abzubilden. Die Indexierung erfolgt dabei in vier Schritten, welche für zwei Beispieldokumente[2] wie folgt verlaufen:

1. Sammeln aller Dokumente, und Zuordnung von eindeutigen Bezeichnern (*docID*)

   ..., 20: Friends, Romans, countrymen, 21: So let it be with Caesar.,...

2. Extraktion der Dokumenten-Merkmale (z.B. Wörter)

   Friends | Romans | countrymen | Caesar

3. Normalisierung der Merkmale (z.B. Reduzierung auf Wortstämme)

   friend | roman | countryman | caesar

4. Indexbildung, Zuordnung der *docID* zu den Einträgen der sortierten Merkmalsliste

caesar	⟼	21
countryman	⟼	11 | 20
friend	⟼	15 | 20 | 73
roman	⟼	20 | 32

   ...

Die in den Schritten 1. bis 3. durchgeführten Verarbeitungsschritte sind immer abhängig vom gegebenen Kontext. Entspricht z.B. im herkömmlichen Verständnis jede Datei einem Dokument, so muss bei der Verarbeitung des MBox-Formates[3] jede Zeile einer Datei als einzelnes Dokument gesehen werden. Auch die Wahl einer geeigneten Methode zur Wortstammbildung (auch "Stemming") und das Filtern von nicht relevanten Wörtern mit Hilfe sog. "Stopwörtern" hängt vom gegebenen Kontext ab. (vgl. [Manning u. a., 2008, Kap. 2]).

Formuliert der Nutzer nun seine Anfrage, durchläuft diese ebenfalls die Schritte 2. und 3. bevor Dokumente mit Hilfe des Index gefunden werden können. Besteht die Anfrage aus mehreren Bestandteilen, wird die Liste der relevanten Dokumente aus der Schnittmenge der für die einzelnen Teile gefundenen Dokumentenmengen gebildet. Ergänzend existieren

[2] Ausschnitte aus "Julius Caesar" von William Shakespeare

[3] siehe RFC 4155 - http://tools.ietf.org/rfc/rfc4155.txt

verschiedene Erweiterungen des invertierten Index, um noch effizienter in sehr großen Dokumentenbeständen suchen zu können, um die Position der Merkmale innerhalb des Dokumentes nutzbar zu machen oder um den Umfang des Index einzuschränken. Diese werden u.a. in [Manning u. a., 2008, Kap. 3,4,5] beschrieben und hier zur Wahrung des Umfangs ausgelassen.

2.1.2 Relevanzberechnung

Die reine Generierung einer Dokumentenliste als Ergebnis der Anfrage genügt vor allem bei großen Dokumentenbeständen nicht. Mögliche Methoden, um die Listen entsprechend der Relevanz eines Dokumentes zu sortieren, sind zum einen das *Tf-idf Maß* und bei untereinander verknüpften Dokumenten der *PageRank*.

Tf-idf Maß Zur Bildung dieses Maßes wird die Relevanz des Terms i innerhalb des Dokumentes d und die Relevanz des Terms innerhalb des gesamten Dokumentenbestands ins Verhältnis gesetzt.

$$\mathrm{tf}(i,d) = \frac{\mathrm{freq}(i,d)}{\max_{z \in Z}(\mathrm{freq}(z,d))} \tag{1}$$

$$\mathrm{idf}(i) = \log \frac{N}{n(i)} \tag{2}$$

$$\text{tf-idf}(i,d) = \mathrm{tf}(i,d) * \mathrm{idf}(i) \tag{3}$$

Um die Relevanz eines Terms bezüglich eines Dokumentes abzubilden, wird die relative Häufigkeit mit der der Term innerhalb des Dokumentes vorkommt, genutzt. Die sog. *Termfrequenz* bildet sich entsprechend aus dem Verhältnis der Anzahl der Vorkommen des Terms innerhalb des Dokumentes ($freq(i,j)$) zur maximalen Anzahl aller anderen Terme Z im Dokument. Mit Hilfe der inversen Dokumentenfrequenz (IDF) wird die Termfrequenz (TF) eines Terms abgewertet wenn dieser in nahezu jedem Dokument vorkommt und aufgewertet wenn er nur selten genutzt wird. Die IDF wird aus dem Verhältnis der Gesamtdokumentenzahl N zur Anzahl der Dokumente die den Term i enthalten ($n(i)$) gebildet.

Das *tf-idf Maß* bildet sich entsprechend Formel (3) aus dem Produkt der beiden Teilmaße und ist:

- hoch: wenn der Term i oft in einer kleinen Anzahl von Dokumenten vorkommt und sich gut zur Unterscheidung von Dokumenten eignet
- niedrig: wenn der Term selten im Dokument vorkommt oder in vielen verschiedenen Dokumenten genutzt wird
- minimal: wenn der Term in nahezu jedem Dokument vorkommt

Die Summe der Relevanz eines Dokuments bezüglich aller Teilterme der Anfrage q bildet dann die Grundlage um die erzeugte Dokumentenliste zu sortieren.[Manning u. a., 2008]

$$\text{score}(q, d) = \sum_{t \in q} \text{tf-idf}(t, d) \tag{4}$$

PageRank Sind die Dokumente untereinander verknüpft, kann man auch die Popularität eines Dokumentes zur Grundlage der Anordnung in der Ergebnisliste machen. Diese Popularität wird i.d.R. in Form des PageRank ausgedrückt. Dieser korreliert mit der Wahrscheinlichkeit, dass ein zufällig über den Verknüpfungsgraphen laufender Nutzer ein bestimmtes Dokument erreicht. Dokumente, auf die häufig verwiesen wird, besitzen demnach einen hohe PageRank und Verweise von populären Dokumenten üben einen größeren Effekt auf den PageRank der verknüpften Dokumente aus.

$$R(d) = c \sum_{v \in B_d} \frac{R(v)}{N_v} + cE(d) \tag{5}$$

Berechnet wird der PageRank R einer Seite u mit Hilfe der Formel (5). Der Faktor $c < 1$ dient dabei zur Abstraktion des Verlustes durch Seiten ohne ausgehende Verweise. Der Vektor E bildet die Wahrscheinlichkeit ab, dass der Nutzer seinen Pfad unterbricht und zufällig bei Dokument d fortsetzt.[Page u. a., 1998; Manning u. a., 2008]

2.1.3 Personalisierung

Die Personalisierung der Dokumentenlisten kann über verschiedene Wege erreicht werden. Der offensichtliche ist, die der Anfrage entsprechende Dokumentenliste anhand der Präferenzen eines Nutzerprofils umzusortieren. Eine weitere Möglichkeit ist, die durch den Nutzer formulierte Anfrage vor der eigentlichen Verarbeitung mit Informationen des Nutzerprofils zu erweitern.

Die erste Methode wird zum Beispiel in [Durao u. a., 2012] beschrieben. Um die Dokumentenliste zu personalisieren, wird zunächst ein schlagwortbasiertes Nutzerprofil aufgebaut, welches die bevorzugten Schlagworte $t \in T_u$ in Bezug auf verschiedene Faktoren F und deren relative Frequenz $T_f \subset T_u$ beinhaltet. Die Gewichtung der Faktoren untereinander wird durch α_f realisiert. Zusätzlich werden auch zu jedem Dokument entsprechende Tags T_d gepflegt, so dass die Ähnlichkeit von Nutzerprofil und Dokument mit Hilfe der Kosinus-Ähnlichkeitsmaßes (siehe Abschnitt 2.3.1) berechnet werden kann. Da die initiale Dokumentenliste anhand des Tf-idf Maßes sortiert wird, ergibt sich die endgültige Bewertung für den Nutzer u aus: (vgl. [Durao u. a., 2012])

$$\text{score}(q, d, u) = \text{score}_{\text{tf-idf}}(q, d) * \sum_{f \in |F|} \alpha_f \frac{\overrightarrow{T_d}\overrightarrow{T_f}}{|\overrightarrow{T_d}||\overrightarrow{T_f}|} \tag{6}$$

Die Anpassung der Anfrage vor der Verarbeitung durch die Suche wird zum Beispiel in [Boughareb u. Farah, 2011] genutzt. Das Nutzerprofil wird dabei aus der Liste aller vorangegangenen Suchanfragen Q_s gebildet. Formuliert der Nutzer eine neue Anfrage, so wird diese um weitere relevante Schlüsselwörter aus ähnlichen Anfragen ergänzt. Zur Bestimmung der Ähnlichkeit zwischen Suchanfragen wird ebenfalls das Kosinus-Ähnlichkeitsmaß (siehe Abschnitt 2.3.1) genutzt. Die Relevanz der Schlüsselwörter wird an deren Dokumentenfrequenz (vgl. Abschnitt 4.1.3) innerhalb des Nutzerprofils Q_s bestimmt. Die Verarbeitung der Suche geschieht dann mit der erweiterten Anfrage wie in den vorangegangenen Abschnitten beschrieben.

In [Smyth u. a., 2005] wird gezeigt, dass die Erweiterung der Anfrage auch ohne explizites Nutzerprofil zur Verbesserung der Relevanz der gefundenen Ergebnisse beitragen kann.

Realisiert wird dies auf der Grundlage von kollaborativen- bzw. gruppenbasierten Methoden (vgl. Abschnitt 2.2.1 u. 2.2.2). Die zur Erweiterung genutzten ähnlichen Anfragen werden dabei aus der für die gesamte Suchmaschine genutzten Datenbasis gewonnen. Dies hat zudem den Vorteil, dass auch Nutzer ohne umfangreiches Nutzerprofil von den erweiterten Bewertungskriterien profitieren, allerdings ist der Grad der Personalisierung durch den Verzicht auf ein Nutzerprofil eingeschränkt. Auch die notwendige Homogenität der Nutzergruppe einer Plattform kann nicht beliebig auf andere übertragen werden. [Smyth u. a., 2005]

Auch der PageRank ermöglicht die personalisierte Sortierung der Dokumentenlisten. Wählt man für den Vektor E in Formel (5) nutzerspezifische "Absprungwahrscheinlichkeiten" so wird, wie in [Page u. a., 1998] beschrieben, der resultierende PageRank den Präferenzen des Nutzers entsprechen. Da eine vollständige Berechnung des PageRank pro Nutzer innerhalb großer Dokumentenbestände sehr unpraktisch ist, wurden zudem verschiedene Erweiterungen untersucht. In [Haveliwala u. a., 2003] werden mögliche Ansätze beschrieben. Im Kern aller Ansätze werden mehrere verschiedene "featurebasierte" PageRank-Werte pro Dokument berechnet. Während der Anfrage werden diese vorberechneten Werte entsprechend des Nutzerprofils gewichtet. [Haveliwala u. a., 2003]

ggf. um Beispiel ergänzen

2.2 Recommendation Konzepte

Die Auswahl von möglichst relevanten Empfehlungen für einen Nutzer kann auf sehr verschiedenen Wegen getroffen werden. Für die zahlreichen bekannten Techniken wird in der Literatur vorwiegend die folgende Gliederung genutzt [Ricci u. a., 2010, Kap. 1] [Burke, 2002] [Jannach u. a., 2010]:

- *Kollaboratives Filtern*, auch *Collaborative Filtering (CF)*, gewinnt relevante Elemente aus dem Vergleich des Nutzerprofils mit anderen (ähnlichen) Profilen.
- *Gruppen-basierte Empfehlungen*, bzw. *Community-based Filtering*, nutzen die Ähnlichkeit innerhalb von Gruppen, etwa in sozialen Netzwerken, um relevante Elemente zu finden.

- *Demographisch gestützte Empfehlungen* leiten sich von den Stereotypen, denen ein Nutzer zugeordnet wird, ab.
- *Inhaltsbasierte Empfehlungen* oder *Content-based Recommendations*, werden auf der Basis von, am Nutzerprofil gewichteten, Element-Eigenschaften getroffen.
- *Wissensbasierte Empfehlungen* bzw. *Knowledge-based Recommendations* werden durch zusätzliches domänenspezifisches Wissen generiert.
- *Utility-basierte Empfehlungen* bestimmen sich durch die Berechnung der "Nützlichkeit" der Elemente für den Nutzer mit Hilfe der sog. *Utility Function*.
- *Hybride Systeme* kombinieren verschiedene Techniken, um die Schwächen der einzelnen auszugleichen.

Die diesen Gruppen zugrunde liegenden Methoden werden in den nächsten Abschnitten näher erläutert. Dazu werden jeweils die zu erhebenden Daten, deren Verarbeitung und die Vor- und Nachteile der Methode beschrieben.

2.2.1 Kollaboratives Filtern

Der Grundgedanke beim kollaborativen Filtern ist, dass Nutzer die in der Vergangenheit gleiche Interessen hatten, diese auch in der Zukunft durch ähnliches Verhalten ausdrücken. So können Empfehlungen für einen Nutzer aus dem Verhalten ähnlicher Nutzer abgeleitet werden. Die Nutzerprofile bilden sich dabei ausschließlich aus Elementbewertungen (*Ratings*), Eigenschaften der bewerteten Elemente fließen nicht ein. Die Ähnlichkeit der Nutzer drückt sich entsprechend durch Gemeinsamkeiten in den Bewertungen aus. [Jannach u. a., 2010, Kap. 2]

Aus den Profilen aller Nutzer ergibt sich eine sog. *User-Item* Matrix, diese ermöglicht es, ähnliche Nutzer oder auch ähnliche Elemente im System zu finden. Zur Auswertung dieser Matrix, bzw. zur Generierung von Empfehlungen mit Hilfe dieser Matrix existieren verschiedene Strategien, welche in Abschnitt 2.3 näher beschrieben werden.

Die Erhebung der Ratings kann sowohl auf explizite Weise, etwa mit einer 5-Punkte-

Likert-Skala, oder implizit, zum Beispiel durch die Aufzeichnung von Browsing-Verläufen, geschehen.

Ein wichtiger Vorteil des kollaborativen Filterns liegt darin, dass Empfehlungen unabhängig von Elementeigenschaften gebildet werden können. Dadurch ist es möglich, auch Elemente deren Inhalt nur schwer oder gar nicht gewonnen werden kann, in die Empfehlung einzubeziehen. Die zahlreichen Forschungsarbeiten und die große Zahl der daraus hervorgegangenen Filterstrategien ist ebenfalls ein Vorteil.

Problematisch ist die Verwendung bei Systemen, in denen der Nutzer (noch) kein oder nur ein sehr begrenztes Profil hat (*Cold Start*). Zudem ist es nicht in jedem Fall sinnvoll alle Eigenschaften der Elemente ausser Acht zu lassen, da so ggf. problemspezifische Entscheidungskriterien unbeachtet bleiben. [Ricci u. a., 2010; Burke, 2002]

2.2.2 Gruppen-basierte Empfehlungen

Gemäß [Sinha u. Swearingen, 2001] haben Nutzer ein größeres Vertrauen in Empfehlungen wenn sie von Freunden ausgesprochen werden. Diesem Ansatz folgend werden in gruppenbasierten Systemen Empfehlungen entsprechend der Präferenzen der Freunde eines Nutzers ausgesprochen. Das Nutzerprofil bildet sich daher aus einer Liste von Elementbewertungen und einer Liste von sozialen Verbindungen zu anderen Nutzern.

Da in Vergleichen mit reinen kollaborativen Systemen keine eindeutige Verbesserung der Empfehlungen nachgewiesen werden konnte, stellt das größere Vertrauen in die gebotenen Empfehlungen den wesentlichen Vorteil dieser Methode dar. Die gute Verbreitung und Verfügbarkeit der Daten über öffentliche Schnittstellen von bestehenden sozialen Netzwerken, wie etwa Facebook oder LinkedIn, sind ebenfalls positiv. Die Abwägung zwischen der Aufrechterhaltung der Privatsphäre und dem dadurch resultierenden Verlust an Genauigkeit ist ein wichtiges Problem (vgl. [Machanavajjhala u. a., 2011]). Auch das fehlende theoretische Fundament in anderen Bereichen, etwa beim Aufbau von Vertrauen und Misstrauen zwischen Nutzern, birgt mögliche Probleme bei der Umsetzung. [Victor u. a., 2011]

Ein etwas anderer gruppen-basierter Ansatz wird in [Smyth u. a., 2005] beschrieben.

Nimmt man an, dass eine bestimmte Plattform, wie zum Beispiel eine themenspezifsche Suchmaschine, nur von einer bestimmten Nutzergruppe (zB. Forscher) genutzt wird, sind die kollaborativ gewonnenen Empfehlungen ebenfalls gewinnbringend für die Gruppenmitglieder.

2.2.3 Demographisch gestützte Empfehlungen

Eine weitere Methode, um ähnliche Nutzer zu finden, ist die Gruppierung nach demographischen Eigenschaften. So können Gruppen zum Beispiel entsprechend des Alters, der Sprache oder des Geschlechts gebildet werden. Sie können allerdings auch mit Hilfe der Methoden des maschinellen Lernens aus bestehenden Transaktionsdaten gewonnen werden (vgl. [Burke, 2002]). Wie bei den vorangegangenen Methoden bildet sich auch hier das Nutzerprofil zunächst aus einer Liste von Elementbewertungen, ergänzt wird es durch die entsprechenden demographischen Eigenschaften. Die Empfehlungen für den einzelnen Nutzer ergeben sich aus seinen eigenen Präferenzen, die endsprechend der Gruppenzugehörigkeit gewichtet werden.

Arbeiten zu reinen demographischen Systemen gibt es kaum. In vielen Fällen, wie etwa [Vozalis u. Margaritis, 2007] werden kollaborative Ansätze ergänzt, um eine Verbesserung der Empfehlungsergebnisse zu erzielen bzw. um die Probleme bei Empfehlungen für neue Nutzer zu verringern. [Burke, 2002]

2.2.4 Inhalts-basierte Empfehlungen

Bei der inhalts-basierten Generierung von Empfehlungen werden die Element-Ratings eines Nutzers zur Erzeugung eines “Interessenprofils” genutzt. In diesem Profil drücken sich die Präferenzen des Nutzers für die inhaltlichen Eigenschaften der Elemente aus. Es kann direkt genutzt werden, um ihm Elemente mit ähnlichen Eigenschaften zu empfehlen. Hat ein Nutzer also zum Beispiel ein “Harry Potter” Buch positiv bewertet, so kann man leicht schlussfolgern, dass auch andere Fantasy-Bücher empfohlen werden könnten.

Neben der automatischen Erstellung des Profils ist es auch möglich, dieses explizit vom

Nutzer zu erfragen. Abhängig vom Problemfeld kann dies schneller zu guten Empfehlungen führen und zur Steigerung des Vertrauens in die erzeugten Empfehlungen beitragen, vgl. [Victor u. a., 2011].

Zur Bestimmung ähnlicher Dokumente, bzw. zur Extraktion der relevanten Eigenschaften (*Features*) werden abhängig vom Elementtyp verschiedene Methoden genutzt. Diese reichen von Entscheidungsbäumen über neuronale Netze bis hin zu Vektorraum-Verfahren (vgl. Abschnitt 2.3 und [Jannach u. a., 2010, Kap. 3]). Die große Anzahl der dafür zur Verfügung stehenden Verfahren, die damit verbundenen Erfahrungen und das daraus abgeleitete Problembewusstsein sind Vorteile. Wichtiger noch ist die Tatsache, dass inhalts-basierte Empfehlungen unabhängig von der Größe des Systems bzw. von der Anzahl der Nutzer generiert werden können. Ein weiterer Vorteil ist, dass für die so gewonnenen Empfehlungen auch leichter Erklärungen für den Nutzer generiert werden können, was wiederum ein wichtiger Faktor zur Steigerung des Vertrauens in die Qualität ist.

Schwierigkeiten bei der Erzeugung von Empfehlungen ergeben sich, wenn die für den Nutzer relevanten Eigenschaften nicht direkt "messbar" vorliegen. Zum Beispiel die Ästhetik eines Produktes oder die Nutzbarkeit einer Webseite lassen sich nur sehr schwer erfassen, können aber beim Vergleich zweier Elemente wichtiger sein als textuelle Eigenschaften. Wie auch beim kollaborativen Filtern ist es bei dieser Methode sehr schwer, gute Empfehlungen für Nutzer zu generieren, wenn diese kein oder nur ein unvollständiges Profil haben. Eine weitere Schwierigkeit ergibt sich daraus, dass Empfehlungen nur aus dem "bevorzugten" Interessenbereich des Nutzers gewonnen werden, dies kann zu sehr ähnlichen und kaum "überraschenden" Empfehlungen führen und zu einem Problem was als *more of the same* umschrieben wird (vgl. Abschnitt 2.4). [Jannach u. a., 2010, Kap. 3] [Lops u. a., 2011]

2.2.5 Wissensbasierte Empfehlungen

Wenn die Frequenz, mit der Nutzer ein Element brauchen oder konsumieren, sehr gering ist, wie es etwa bei Hauskäufen der Fall ist, ergibt sich für die bisher beschriebenen Methoden das Problem, dass nur selten umfangreiche Nutzerprofile zur Verfügung stehen oder

die darin enthaltenen Informationen schlicht veraltet sind. Oft gibt es zudem in vielen Bereichen Expertenwissen bzw. domänenspezifisches Wissen, welches zur Verbesserung von Empfehlungen bzw. zur Einschränkung der Kandidatenliste genutzt werden kann.

Um dieses vorhandene Wissen zur Generierung von Empfehlungen nutzbar zu machen, kann man es in eine Menge von Regeln überführen und mögliche Empfehlungen entsprechend der Regeln filtern. So kann man zum Beispiel aus der Information, dass der Nutzer auf der Suche nach einer Wohnung für seine fünfköpfige Familie ist, leicht ableiten, dass $40m^2$ Wohnungen nicht empfehlenswert sind und das solche mit zwei Bädern oder in einer ruhigeren Wohnlage empfohlen werden können.

Form und Inhalt des Nutzerprofils variieren hierbei in Abhängigkeit von der gewählten Wissens- bzw. Regelrepräsentation. Die Einbeziehung von Expertenwissen ermöglicht es auch, übliche Standards einzubeziehen und es erleichtert die Vervollständigung des Nutzerprofils durch die Auswahl sinnvoller Fragen bei der Interaktion mit dem Nutzer. Auch in Fällen, in denen keine Vorschläge gefunden werden konnten, haben regelbasierte Systeme Vorteile. Das System kann zum Beispiel eine Reihe von Vorschlägen zum Auslassen einzelner Regeln unterbreiten, wenn keine Empfehlungen für eine Anfrage gefunden werden. Dies hilft den Nutzen des Systems für den Nutzer zu steigern und baut Vertrauen in die Qualität des Systems auf. Nachteile ergeben sich, wenn das Expertenwissen und die darauf basierenden Regeln nicht an neue Entwicklungen angepasst werden, oder wenn für den Nutzer wichtige Features umbewertet bleiben. [Jannach u. a., 2010, Kap. 4]

2.2.6 Utility-basierte Empfehlungen

Ein zweiter Ansatz um domänenspezifisches Wissen zum Ausgangspunkt von Empfehlungen zu machen, ergibt sich, indem man die "Nützlichkeit" eines Elements mit Hilfe einer nutzerspezifischen Funktion (*Utility function*) berechnet. Dadurch kann zum Beispiel eine mögliche Toleranz des Nutzers gegenüber gewissen Produktmerkmalen direkt ins Verhältnis zur Dringlichkeit einer Bestellung gesetzt werden. Das Nutzerprofil ergibt sich dabei aus den Parametern der Funktion, welche i.d.R. explizit von Nutzer erfragt werden müssen.

	Item1	Item2	Item3	Item4	Item5	Item6	Item7
User1	5.0	3.0	2.5	?			
User2	2.0	2.5	5.0	2.0			
User3	2.5			4.0	4.5		5.0
User4	5.0		3.0	4.5		4.0	
User5	4.0	3.0	2.0	4.0	3.5	4.0	

Tabelle 1: Beispiel-Matrix für User-Item Ratings

Vor- und Nachteile sind ähnlich gelagert wie im vorangegangene Abschnitt. Vor allem der direkte Einfluss, den der Nutzer auf die Qualität der Ergebnisse hat, kann zur Steigerung das Vertrauens in die generierten Empfehlungen führen. [Ricci u. a., 2010, Kap. 1] [Burke, 2002; Victor u. a., 2011]

2.3 Filtermodelle

Will man die in Abschnitt 2.2.1 beschriebenen kollaborativen Filtermethoden nutzen, stellt sich das Problem, wie man die Ähnlichkeit von Nutzern oder Elementen bestimmen kann und wie man Empfehlungen für einen Nutzer erzeugt. Die dafür nötigen Modelle sollen in den folgenden Abschnitten näher erläutert werden.

Grundlage der im Folgenden beschriebenen Methoden ist eine *User-Item* Matrix R, welche die Bewertung aller Nutzer U für die Elemente (Produkte) P enthält. Die Wahl des Wertebereichs hängt dabei von der Applikation ab. Ein Beispiel für eine solche Matrix wird in Tabelle 1 gezeigt.

2.3.1 Ähnlichkeitsmaße

Euklidische Distanz Die naheliegendste Form zur Bestimmung der Ähnlichkeit zwischen zwei Spalten oder zwei Zeilen der User-Item Matrix ist es, deren Abstand im n-

dimensionalen euklidischen Raum, gem. Formel (7) zu nutzen.

$$dist(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{7}$$

$$sim(a,b) = \frac{1}{1 + dist(a,b)} \tag{8}$$

Hierbei ist n die Anzahl der Dimensionen und a_i bzw. b_i beziehen sich auf das i^{te} Attribut der Objekte, resp. die Ratings der Nutzer. Um den Distanzwert zu einem Maß der Ähnlichkeit mit einem Wertebereich von 1 (starke Korrelation) bis 0 (keine Korrelation) umzuformen, kann Formel (8) genutzt werden.

Aus der Verallgemeinerung dieser Berechnung, der sog. *Lr-Norm* bzw. dem *Minkowski Abstand*, ergeben sich weitere Abstandsmaße. Die sog. *L1-Norm* (auch *City-Block-* oder *Manhattan-Distanz*) entspricht $r = 1$, $r = 2$ entspricht dem o.g. euklidische Abstand und $r = \infty$ entspricht dem *Tschebyscheff-Abstand*. [Amatriain u. a., 2009]

$$dist(a,b) = \sum_{i=1}^{n}(|a_i - b_i|^r)^{\frac{1}{r}} \tag{9}$$

Pearson-Korrelation Ein Problem bei der Berechnung mit der euklidischen Distanz ist, dass die Mittelwerte und Varianzen der Bewertungen einzelner Nutzer voneinander abweichen können, obwohl diese vergleichbare "Interessen" haben (vgl. [Segaran, 2007, Kap. 2]). Dieser Mangel wird mit Hilfe der *Pearson-Korrelation* (10) beseitigt. Ihr Wertebereich reicht von 1 (starke Korrelation) bis -1 (starke negative Korrelation). Vor Allem bei der Bestimmung von nutzerbasierten Ähnlichkeiten konnten mit ihr in vielen Fällen sehr gute Ergebnisse erzielt werden. Zudem existieren zahlreiche Erweiterungen, um zum Beispiel die Gewichtung von Übereinstimmungen bei der Bewertung von kontroversen Elementen stärker hervorzuheben. [Jannach u. a., 2010, Kap. 2.1] [Amatriain u. a., 2009]

$$sim(a,b) = \frac{\sum_{p\in P}(r_{a,p} - \bar{r_a})(r_{b,p} - \bar{r_b})}{\sqrt{\sum_{p\in P}(r_{a,p} - \bar{r_a})^2}\sqrt{\sum_{p\in P}(r_{b,p} - \bar{r_b})^2}} \tag{10}$$

Kosinus-Ähnlichkeit Ein weiterer Ansatz, der sich zum Standardmaß bei der Abbildung von Element- bzw. Item-Ähnlichkeit entwickelt hat, ist die *Kosinus-Ähnlichkeit* (11). Die Distanz zwischen zwei Vektoren entspricht dabei dem zwischen ihnen aufgespannten Winkel, entsprechend steigt die Ähnlichkeit von Vektoren wenn diese in die gleiche Richtung zeigen.

$$sim(a, b) = \frac{a \cdot b}{\|a\|\|b\|} \tag{11}$$

Der Wertebereich des erzeugten Ähnlichkeitsmaßes liegt zwischen 1 (starke Korrelation) und 0 (keine Korrelation) wenn die genutzten Ausgangsvektoren nur positive Werte haben. Dies ist zum Beispiel der Fall bei den oft üblichen 5 Stern Rating-Skalen oder beim Vergleich von Textdokumenten anhand der Vorkommen einzelner Wörter. Das Maß reicht bis -1 für starke negative Korrelationen, wenn auch negative Werte genutzt werden. [Jannach u. a., 2010][Kap. 2.2]

Jaccard-Koeffizient Liegen Ratings nur als binäre Werte vor, kann die Ähnlichkeit zweier Elemente durch das Verhältnis der Schnittmenge zur Vereinigungsmenge dieser definiert werden. Der Wertebereich des sog. *Jaccard-Koeffizienten* (12) liegt ebenso zwischen 1 und 0. Verwendung findet er auch wenn die Werte wenig Informationen tragen, die Information, ob ein Nutzer eine Bewertung abgegeben hat, im Zentrum der Betrachtung steht oder durch die Rating-Werte Beziehungen zwischen Nutzern und Elementen (im Sinne eines Graphen) ausgedrückt werden. Erweitert wird der Jaccard-Koeffizent vom *Tanimoto-* und vom *Dice-Koeffizienten* (vgl. [Bogers u. van den Bosch, 2009]). [Jannach u. a., 2010, Kap. 3.1] [Segaran, 2007]

$$sim(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{12}$$

Welches der Distanzmaße für eine konkrete Anwendung genutzt werden sollte, kann nicht pauschal beantwortet werden. Bei der Bestimmung von nutzerbasierten Ähnlichkeiten stellt in vielen Fällen die Peason-Korrelation einen guten Ausgangspunkt dar, beim Vergleich von Elementen ist die Kosinus Ähnlichkeit oft eine gute Wahl, aber in jedem Fall muss die Wahl eines Maßes immer mit einer entsprechenden Evaluation gegenüber anderen Maßen kontrolliert werden (vgl. Abschnitt 2.6 u. 5).

MAE dazu?

	User2	User3	User4	User5
User1	0.203	0.286	0.667	0.472

Tabelle 2: Aus Tabelle 1 mit der euklidischen Distanz abgeleiteter Ähnlichkeitsvektor für User1

2.3.2 Nachbarschaftsmodelle

Nutzer-basierte Modelle Geht man nun davon aus, dass ähnliche Nutzer auch in der Zukunft eine ähnliche Meinung zu einem Element haben werden, kann man für einen Nutzer u aus den vorliegenden Bewertungen ähnlicher Nutzer $\mathcal{N}_i(U)$ eine Bewertung für ein Element i voraussagen ($pred(u, i)$). Die dabei in Betracht gezogenen anderen Nutzer werden auch als "Nachbarschaft" des Nutzers bezeichnet. Da diese zudem i.d.R. auf eine bestimmte Größe k oder einen bestimmten Ähnlichkeits-Schwellwert limitiert wird, wird die Methode als *k–nearest-neighbors* (k-NN) bezeichnet.

Um Empfehlungen für einen Nutzer aus den in Tabelle 1 gezeigten Ausgangsdaten abzuleiten, wird mit Hilfe der schon vorliegenden Ratings zunächst die Ähnlichkeit dieses Nutzers zu anderen berechnet (siehe Tabelle 2). Um $pred(u, i)$ aus diesen abzuleiten, werden die Ratings anderer Nutzer für dieses Element $r_{v,i}$ entsprechend der Ähnlichkeit zwischen den Nutzern aufsummiert und normiert:

$$pred(u, i) = \frac{\sum_{v \in \mathcal{N}_i(U)} sim(u, v) * r_{v,i}}{\sum_{v \in \mathcal{N}_i(U)} sim(u, v)} \tag{13}$$

Wie [Herlocker u. a., 2002] zeigen, muss zudem ein weiterer Unterschied zwischen einzelnen Nutzern in Betracht gezogen werden. Auch wenn Nutzer generell ähnliche Interessen bzw. Meinungen haben, so kann es durchaus sein, dass Mittelwert und Varianz der Ratings dieser Nutzer sehr verschieden sind. Diese Gewichtung am Rating-Mittelwert $\overline{r_u}$ und der Varianz σ_u der Nutzer, dem sog. *Z-score*, wird aus diesem Grund in den erweiterten Formeln (14) und (15) beachtet. [Desrosiers u. Karypis, 2011; Huete u. a., 2012]

$$pred(u, i) = \overline{r_u} + \frac{\sum_{v \in \mathcal{N}_i(U)} sim(u, v) * (r_{v,i} - \overline{r_v})}{\sum_{v \in \mathcal{N}_i(U)} sim(u, v)} \tag{14}$$

$$pred(u, i) = \overline{r_u} + \sigma_u \frac{\sum_{v \in \mathcal{N}_i(U)} sim(u, v) * \frac{r_{v,i} - \overline{r_v}}{\sigma_v}}{\sum_{v \in \mathcal{N}_i(U)} sim(u, v)} \tag{15}$$

	Item1	Item2	Item3	Item5	Item6	Item7
Item4	0.387	0.472	0.204	0.586	0.667	0.500

Tabelle 3: Aus Tabelle 1 mit der euklidischen Distanz abgeleiteter Ähnlichkeitsvektor für Item4

Element-basierte Modelle Neben der Bestimmung von ähnlichen Nutzern, kann mit den in der *User-Item* Matrix vorliegenden Daten auch die Ähnlichkeit von Elementen bestimmt werden. Die Ähnlichkeiten bzw. Nachbarschaften der Elemente $\mathcal{N}_u(I)$ zu anderen kann dann, analog zu Formel (13), wie folgt zur Voraussage der Bewertungen genutzt werden:

$$pred(u,i) = \frac{\sum_{j \in \mathcal{N}_u(I)} sim(i,j) * r_{u,j}}{\sum_{j \in \mathcal{N}_u(I) \mathcal{N}_i(U)} sim(i,j)} \tag{16}$$

Wie bei den nutzerbasierten Modellen sollte auch hier der Einfluss verschiedener Bewertungs-Mittelwerte und Varianzen ausgeglichen werden. Dies geschieht analog zu Formel (14) und (15).

Quelle das "Normalisierung" auch bei item-item relevant ist

Ergebnisse der Rechnung einfügen

Für die Abwägung zwischen nutzer- und elementbasierten Methoden gibt [Desrosiers u. Karypis, 2011] die folgenden Kriterien an:

- *Genauigkeit*: Abhängig von der Menge der Nutzer und Elemente im System schwankt die Zuverlässigkeit der Nachbarschaften. Ist die Anzahl der Nutzer im System größer als die der Elemente, so kann man davon ausgehen, dass elementbasierte Methoden kleinere aber zuverlässigere Nachbarschaften für die einzelnen Elemente produzieren und damit bessere Ergebnisse liefern und umgekehrt (vgl. auch [Huete u. a., 2012] u. [Herlocker u. a., 2002])

- *Effizienz* - Das Verhältnis von Nutzern und Elementen beeinflusst auch den Umfang der notwendigen Berechnung bei der Bestimmung von Nachbarschaften. [Linden u. a., 2003] zeigt zum Beispiel, dass die Grenzen der Skalierbarkeit schnell erreicht werden, wenn die Zahl der Nutzer die der Elemente stark überschreitet.

- *Stabilität* - Die Änderungshäufigkeit in der Menge der bewerteten Elemente bzw. die Fluktuation der Nutzer beeinflusst wie stabil Ähnlichkeiten sind. Ist die gewählte Basis ausreichend stabil können Ähnlichkeiten vorberechnet werden.

- *Erklärbarkeit* - Die der Berechnung von elementebasierten Ähnlichkeiten zugrunde liegenden Elemente lassen sich leicht zur Erklärung der Ergebnisse nutzen und ermöglichen ggf. eine darauf basierende Interaktion mit dem Nutzer. Bei nutzerbasierten Modellen ist dies i.d.R. auch aus Gründen des Datenschutzes erheblich schwerer.
- *Zufälligkeit* - Die im ersten Punkt beschriebene Genauigkeit führt i.d.R dazu, dass bei elementbasierten Modellen oft weniger überraschende Vorschläge erzeugt werden. Bezieht man bei nutzerbasierten Modelle nur wenige Nachbarn zur Erzeugung der Vorschläge ein, ist die Wahrscheinlichkeit einer "Überraschung" höher. (vgl. Abschnitt 2.4)

Nachbarschaftsgrößen Unabhängig von der Methodenwahl muss die Größe der betrachteten Nachbarschaft k begrenzt werden. Wählt man feste Werte, so sind Größen zwischen 20 und 50 (vgl. [Herlocker u. a., 2002]) üblich. Kleinere Nachbarschaften werden wegen ihrer Anfälligkeit für Ausreißer nicht empfohlen, bei Größeren wiederum steigt i.d.R. die Fehlerquote.

Neben festen Werten wird oft alternativ die Wahl eines Ähnlichkeits-Schwellwertes vorgeschlagen. Dabei werden alle Nachbarn einbezogen, deren Ähnlichkeit einen Maximalabstand nicht überschreitet. Da die Wahl des Schwellwertes nicht nur Auswirkungen auf den Umfang der Nachbarschaft, sondern auch auf die Abdeckung der berechenbaren Bewertungsvorhersagen hat, wird i.d.R (vgl. [Herlocker u. a., 2002, 1999]) von der alleinigen Verwendung abgeraten.

Die Wahl der konkreten Größe ist bei beiden Methoden zudem in jedem Fall ein Kompromiss zwischen Genauigkeit, Zufälligkeit und Effizienz.

Vorteile von Nachbarschaftsmodellen Dass zum Vergleich mit anderen Methoden, keine langwierige Trainingsphase notwendig ist, ist ebenso wie die leichte Nachvollziehbarkeit der Methodik ein Vorteil von Nachbarschaftsmodellen. Die Möglichkeit, Empfehlungen durch eine Erklärung zu ergänzen, ist ein weiterer Vorteil dem bei der Bildung des Nutzervertrauens besonders viel Gewicht zukommt (vgl. [Tintarev u. Masthoff, 2011]). Die

Stabilität der zugrunde liegenden Ähnlichkeitsmatrizen und der Effizienzgewinn durch die mögliche Vorberechnung der Nachbarschaften sind zudem bei großen Systemen von Vorteil (vgl. [Linden u. a., 2003]).[Desrosiers u. Karypis, 2011]

Weiterführend: Regression vs. Classification aus HB04 evt. ergänzen

ggf. SlopeOne ergänzen

2.3.3 Matrixfaktorisierung

Merkmal-basierte Empfehlungen Bei den Methoden der Matrixfaktorierung geht man davon aus, dass zusätzliche Merkmale der Nutzer und Elemente (*Features*) aus den Einträge der *User-Item* Matrix abgeleitet werden können. Die Bandbreite erstreckt sich von sehr anschaulichen Merkmalen, wie etwa dem Genre bei Büchern, Filmen oder Musik, über schwer definierbare Merkmale wie etwa "Qualität" bis hin zu gänzlich uninterpretierbaren. Der Erfolg dieses Ansatzes wurde zum Beispiel beim 2006 ausgeschriebenen Netflix-Preis, zur Generierung von Filmempfehlungen, unter Beweis gestellt (vgl. [Koren u. a., 2009]).

PureSVD erläutern?

Um die in den Einträgen der Matrix "verborgenen" Merkmale zu berechnen, werden diese in einem f-dimensionalen Raum als Produkt von Nutzer- und Elementvektoren abgebildet. Die Dimension des Raumes entspricht der Anzahl der Merkmale, beim o.g. Beispiel lag diese zwischen 100 und 500. Die Einträge des Elementvektors $q_i \in \mathbb{R}^f$ drücken aus, zu welchem Grad — positiv oder negativ — dessen Eigenschaften dem Merkmal entsprechen. Der Nutzervektor $p_u \in \mathbb{R}^f$ korreliert entsprechend die "Interessen" des Nutzers mit diesen Merkmalen. Bezieht man zudem den Rating-Mittelwert μ, sowie element- und nutzerspezifische Abweichungen b_x ein, so ergibt die mögliche Bewertung eines Nutzer u für ein Element i aus:

$$pred(u, i) = \mu + b_i + b_u + q_i^T p_u \tag{17}$$

Die dafür benötigten Parameter q, p und b werden in einer Trainingsphase aus den vorhandenen Bewertungen gelernt. Dies geschieht mit den Methoden der Singular Value Decomposition (SVD) [Golub u. Kahan, 1965] durch die Minimierung der mittleren quadratischen Abweichung zwischen existierenden und vorausberechneten Bewertungen:

$$\min_{q*,p*,b*} \sum_{(u,i)\in\mathcal{K})} (r_{ui} - \mu - b_i - b_u - p_i^T q_u)^2 + \lambda(\|q_u\|^2 + \|p_i\|^2 + b_u^2 + b_i^2) \tag{18}$$

Die Menge $\mathcal{K}$ entspricht dabei allen vorliegenden Bewertungen r_{ui}. Die Konstante λ wird genutzt um das sog. *overfitting* zu verhindern, sie stellt also sicher, dass das abgeleitete Modell generisch bleibt. [Koren u. a., 2009; Koren u. Bell, 2011]

Trainingsmethoden Zur Durchführung des Trainings kann einer der beiden im Folgenden beschriebenen Methoden verwendet werden.

Beim *stochastischen Gradientenverfahren* [Funk, 2006] wird das Minimierungsproblem durch einen Gradientenabstieg gelöst. Die Richtung des Abstieges ergibt sich mit Hilfe der vorliegenden Trainingsdaten aus der Abweichung e_{ui} zwischen den tatsächlichen und vorausberechneten Werten (siehe Formal (19) - (23)). Der Faktor γ entspricht dabei der Lernrate bzw. Schrittweite. Entsprechend der Anzahl der zu ermittelnden Merkmale wird das Training f-mal, jeweils bis zur Konvergenz, durchgeführt. [Funk, 2006; Langford u. a., 2009; Koren u. Bell, 2011]

$$e_{ui} = r_{ui} - pred(u, i) \tag{19}$$

$$q_i' \leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i) \tag{20}$$

$$p_u' \leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u) \tag{21}$$

$$b_i' \leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \tag{22}$$

$$b_u' \leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \tag{23}$$

Die Methode der *Alternating Least Squares* [Bell u. Koren, 2007] verfolgt einen anderen Ansatz zur Lösung des Minimierungsproblems. Da $p_i^T q_u$ nicht konvex ist, kann das resultierende Gleichungssystem nicht vollständig gelöst werden. Nimmt man aber p_i oder q_u als konstant an, ist eine Approximation der nicht konstanten Werte durch die Methode der kleinsten Quadrate möglich . Im Trainingsverlauf werden deshalb abwechselnd die p_i und q_u konstant gehalten um die jeweils anderen anzupassen. Wegen des aufwendigeren Ablaufs konvergiert das Verfahren langsamer als das zuvor beschriebene Gradientenverfahren, der Mangel wird bei größen Datenmengen aber durch eine bessere Parallelisierbarkeit ausgeglichen. [Bell u. Koren, 2007; Koren u. Bell, 2011]

Göhler S. 89 unten

Pseudocode aus Loup-pe10 ?

Temporale Effekte dazu ?

Vorteile der Matrixfaktorisierung Ein wichtiger Vorteil der durch die Matrixfaktorisierung erzeugten Modelle ist die Approximation der in der *User-Item* Matrix enthaltenen Informationen in erheblich kompakterer Form. Zudem ermöglicht sie, dass man über die Anzahl der zu lernenden Merkmale die Größe des resultierenden Modelles steuern kann. Die Erweiterbarkeit der Ausgangsformel (17) ist ein zusätzlicher Vorteil. So konnten mit Erweiterungen, die zum Beispiel temporale Effekte oder nicht-lineare Zusammenhänge in den Daten auszunutzen, in verschiedenen Anwendungsfällen zusätzlich verbesserte Ergebnisse erzielt werden. [Koren u. Bell, 2011; Vozalis u. Margaritis, 2007]

Iterative Methoden ggf. ergänzen bzw. Informationen zu iterativen Updates der Modelle hinzufügen - siehe Entwurf->Problemstellun

2.4 Schwierigkeiten von Recommendern

Alle Methoden des kollaborativen Filterns haben ihre Stärken und Schwächen, die wichtigsten Herausforderungen werden im folgenden Absatz zusammengefasst.

Weitere Ansätze ergänzen? zB. Graphen-Modelle? Bolzmann-Dings?

Cold-Start Existieren von Nutzern keine oder nur wenige Bewertungen im System, können die im vorangegangenen Abschnitt beschriebenen Methoden den neuen Nutzer nur schwer zu vorhandenen in Relation setzen. Daraus folgt, dass keine oder nur sehr ungenaue Empfehlungen generiert werden können. Ähnliches gilt für Elemente für die nur wenige Bewertungen vorliegen. Dieses Problem des sog. *Cold-Start* kann durch die Gewinnung von zusätzlichem impliziten Informationen oder durch die Ergänzung von domänenspezifischem Wissen verringert werden (vgl. [Claypool u. a., 1999]). In dem von [Steck, 2010] entwickelten Ansatz wird zudem auch in fehlenden Bewertungen ein gewisser Informationsgehalt und damit das Potential zur Verbesserung der Empfehlungen gezeigt. Angewandt wird dies zum Beispiel in [Töscher u. a., 2008] durch die Kombination von Nachbarschafts- und Matrixfaktorisierungsmodellen.

Dünnbesetzte Matrizen Überträgt man die zu 6.3% gefüllte *User-Item* des 100K MovieLense Datensatzes[4] auf Anwendungsfälle mit einer Million Elementen, so wird schnell

[4] http://www.grouplens.org/node/73, Enthält Bewertungen von 943 Nutzer für 1682 Filme

klar, dass Annahmen über 60.000 Bewertungen pro Nutzer eher unrealistisch sind. Abhängig von der Größe des Systems wird es daher immer schwerer, ähnliche Nutzer ausschließlich über gemeinsam bewertete Elemente zu finden. Das daraus resultierende Problem der *Neighbor transitivity* bezeichnet den Fall in dem aufgrund der zu geringen Datenmenge Nutzer mit ähnlichen Interessen nicht gefunden werden können, weil für sie keine sich überschneidenden Bewertungen vorliegen.

Die in Abschnitt 2.3.3 vorgestellten Methoden der Matrixfaktorisierung bieten eine mögliche Lösung. Da man über die trainierten Modelle jeden Nutzer mit jedem anderen Nutzer und jedem Element in Relation setzen kann, wirken sich die fehlenden Daten nur noch auf die Genauigkeit des Modells aus, nicht aber auf dessen Fähigkeit überhaupt Empfehlungen zu generieren. Ein weiterer Ansatz ist die Kombination mit anderen nicht-kollaborativen Techniken. [Koren u. a., 2009; Claypool u. a., 1999]

Graue Schafe Ein weiteres Problem in kleinen und mittleren Systemen ist nach [Claypool u. a., 1999] das der "grauen Schafe". In diese Gruppe fallen demnach alle Nutzer die in keine Gruppe fallen und für die es folglich nicht möglich ist, ähnliche Nutzer zu finden. Folglich können diese Nutzer keinen oder nur sehr geringen Nutzen aus den Empfehlungen ziehen. In [Claypool u. a., 1999] wird ein hybrides System auf inhaltsbasierten und kollaborativen Modellen zur Verbesserung der Empfehlungen in diesen Fällen erfolgreich evaluiert. [Burke, 2002]

More-of-the-same Existieren von einem Element verschiedene Variationen in der *User-Item* Matrix, ist es dem System nicht möglich, diese indirekte Verbindung zu erkennen. Aus der Präferenz für die E-Book-Ausgabe eines Buches kann folglich nicht die Ähnlichkeit zu anderen Nutzern gefunden werden, die die gedruckte Form des Buches bewertet haben. Gleichzeitig sind sich die Variationen oft trotzdem zu ähnlich, so dass die Empfehlungen aus offensichtlichen Elementen zusammengesetzt sind und für den Nutzer wenig Überraschungen bzw. sehr geringen Nutzen bieten. Zur Verbesserung konnte zum Beispiel zusätzliche inhaltliche Diversifikation der generierten Empfehlungen genutzt werden. [Jannach u. a., 2010, Kap. 3]

Rich-gets-richer Dem Ziel, mit Hilfe von Recommender-Systemen die Diversität der vom Nutzer wahrgenommenen Elemente zu vergrößern, steht der sog. *Short-Head vs. Long-Tail* oder *Rich-gets-richer* Effekt gegenüber. Dieser tritt auf, wenn Bewertungen nicht gleichmäßig auf die Elemente verteilt sind. So beziehen sich zum Beispiel im MovieLense Datensatz[5] 33% der Bewertungen auf weniger als 10% der Filme. Erschwert wird das Problem durch seine geringe Sichtbarkeit in den populären Metriken zum Vergleich von Recommender-Modellen. Die von [Cremonesi u. a., 2010] durchgeführte Evaluation zeigt, dass alle Methoden anfällig für diese Effekte sind und das der Ausschluss der populärsten 2% der Elemente vom Training zur Verbesserung der Empfehlungen der restlichen 98% führen konnte. In [Yin u. a., 2012] wird eine graphenbasierte Methode zur Verbesserung der Diversität in den Empfehlungen vorgeschlagen.

Manipulation Die Fähigkeit eines Recommender-Systems das Verhalten der Nutzer zu adaptieren stellt zugleich eine große Angriffsfläche für gezielte Manipulationen dar. Diese haben entweder die gezielte Verstärkung (*product push*) oder Verminderung (*product nuke*) der Häufigkeit mit der ein Element empfohlen wird zum Ziel. Die Bandbreite der möglichen Attacken hängt dabei vom Grad der Kenntnis die der Angreifer über das System hat ab. Die verschiedenen Angriffsformen, deren mögliche Auswirkungen und Maßnahmen werden in [Burke u. a., 2011] beschrieben.

Formen von Attacken oder Maßnahmen könnten ergänzt werden

2.5 Skalierungsstrategien

Die in den vorangegangenen Abschnitten beschriebenen Techniken zum Aufbau von Recommender-Systemen entfalten vor allem bei einer möglichst großen Datenbasis ihren Nutzen. Der Umfang der zu diesem Zweck erhobenen Daten steigt schnell in Bereiche die nicht mehr sinnvoll von einzelnen Systemen verarbeitet werden können. Um diesem Problem zu begegnen, beschreibt dieser Abschnitt mögliche Lösungsstrategien zur Speicherung, Verteilung und Verarbeitung von Daten auf einer großen Zahl von Rechnern.

ggf. Langford09 Parallelisierungsstrategien dazu

[5] http://www.grouplens.org/node/73

2.5.1 Skalierung der Datenhaltung

Die Architektur eines Dateisystems, dass den Ansprüchen großer verteilter Systeme genügt, wird in [Ghemawat u. a., 2003] beschrieben. Das darin entworfene Google Filesystem (GFS) orientiert sich dabei an den Erfahrungen aus dem praktischen Anwendungen und stellt folgende Kernannahmen:

- Das Gesamtsystem wird aus vielen handelsüblichen und damit günstigen Komponenten aufgebaut. Da Ausfälle einzelner Komponenten keine Ausnahme darstellen, müssen diese automatisch erkannt und behoben bzw. tolerant ausgeglichen werden.
- Die typische Größe der verwalteten Dateien bewegt sich über dem 100 MB Bereich, mehrere Gigabyte große Dateien sind der Regelfall und sollten für das System kein Problem darstellen. Aus diesem Grund sollte das Dateisystem für die Verwaltung von großen Dateien optimiert werden.
- Dateien werden vorwiegend durch lange, kontinuierliche Lesezugriffe (Streaming) genutzt, welche auf große zusammenhängende Bereiche der Dateien zugreifen. Kleine wahlfreie Zugriffe auf Teile der Dateien können von den Anwendungen oft zu Leseoperationen von größeren Bereichen zusammengefasst werden.
- Schreibzugriffe erfolgen ebenfalls vorwiegend auf große zusammenhängenden Bereichen. Bereits geschriebene Bereiche werden zudem selten geändert, d.h. neue Daten werden in der Regel an das Ende einer Datei angehängt.
- Einzelne Dateien werden oft von mehreren Quellen gleichzeitig beschrieben. Deshalb muss das Dateisystem die Synchronisation dieser Quellen mit möglichst wenig Zusatzaufwand ermöglichen.
- Um möglichst effizient große Datenmengen "am Stück" verarbeiten zu können, hat die Ausnutzung der Schreib/Lesebandbreite hohe Priorität. Der Latenz einzelner Operationen wird geringeres Gewicht beigemessen.

Diese Annahmen begründen zentrale Eigenschaften des Systems. Ein GFS-Cluster wird aus einem *Master*- und vielen *Chunkservern* aufgebaut. Jede Datei wird in Blöcke (engl.

Chunks) mit einer festen Größe von 64 MB aufgeteilt. Die Speicherung der Blöcke erfolgt im lokalen Dateisystem der *Chunkserver*. Zur Gewährleistung der Verfügbarkeit wird jeder Block auf mehrere *Chunkserver* repliziert. Die Verwaltung der zum Betrieb notwendigen Metadaten wird von *Masterserver* übernommen. D.h. jeder *Chunkserver* kennt nur die von ihm verwalteten Blöcke, welche durch ein eindeutiges 64 Bit *Handle* identifiziert werden. Die Zuordnung der Blöcke zu Dateien, die Verwaltung der Dateistrukturen, die Rechteverwaltung und die Verteilung der Blöcke zwischen den *Chunkservern* obliegt dem *Masterserver*.

Um zu verhindern, dass Leistungsengpässe entstehen, werden Daten weitestgehend ohne den *Masterserver* gelesen und geschrieben. Einzig um den Speicherort eines Blockes zu finden oder um strukturelle Operationen durchzuführen (Erzeugen, Umbenennen, Löschen, ...) wird er von den Anwendungen benötigt. Alle weiteren Operationen geschehen direkt zwischen den *Chunkservern* und der Anwendung. Die Größe der Blöcke minimiert zudem den Kommunikationsaufwand zwischen Anwendung und *Masterserver*, da mit jeder Anfrage ein sehr großer Datenbereich abgedeckt werden kann. Um dennoch konsistente Ergebnisse beim Schreiben neuer Daten zu erhalten, wird zwischen der Anwendung und den *Chunkservern* ein fest definiertes Kommunikationsprotokoll verfolgt, welches sicherstellt, dass Änderungen tatsächlich bei allen Servern gleich geschrieben wurden bevor die entsprechende Operation erfolgreich abgeschlossen wird (vgl. [Ghemawat u. a., 2003, Kap. 3]).

Alle Operationen die vom *Masterserver* ausgeführt werden, werden zudem in einem *Operation Log* gesichert. Fällt er aus, kann er leicht mit Hilfe der gesicherten Logs durch einen neuen *Masterserver* ersetzt werden. Wie bei den Blockoperationen, werden auch Schreiboperationen in das Log, nur dann als erfolgreich abgeschlossen, wenn sie auch auf allen vorhandenen Realisationen erfolgreich ausgeführt wurden. Das "Wissen" der *Chunkserver* über die auf ihnen gespeicherten Blöcke minimiert zudem die im *Operation Log* gehaltenen Informationen.

Die Integrität der Daten wird von den *Chunkservern* sichergestellt. Innerhalb der 64 MB Blöcke werden Prüfsummen für jeden 64 KB Abschnitt geschrieben. Stellt der *Chunkserver*

beim Lesen fest, dass die Daten eines Blockes beschädigt sind, wird vom *Master* veranlasst, dass der Block von einem anderen *Chunkserver* ausgeliefert wird und dass er zu einem weiteren repliziert wird. Ist dies geschehen, wird der beschädigte Block entfernt.

Wie in [Ghemawat u. a., 2003] gezeigt wird, kann ein so aufgebautes System auf der Basis von handelsüblichen Hardware effizient betrieben werden, falls die zuvor beschriebenen Annahmen für die damit betriebenen Anwendungen zutreffen. [Ghemawat u. a., 2003]

2.5.2 Horizontalen Fragmentierung

Will man große Datenmengen innerhalb eines Datenspeichers halten ohne an die Grenzen einzelner Speicherknoten zu stoßen, ist die *horizontale Fragmentierung* (engl. *Sharding*) eine weitere mögliche Strategie. Zur Verarbeitung einer Anfrage sind hierfür zwei Knotentypen bzw. Servertypen notwendig. Die *Backend-Knoten* halten jeweils einen Teil der Daten und verarbeiten alle Anfragen dafür. Die *Frontend-Knoten* bilden die Schnittstelle zu den Applikationen. Sie verteilen die Anfragen an die richtigen Backend-Knoten und fassen Anfrageergebnisse verschiedener Backend-Knoten zusammen. In der Regel werden die Daten möglich gleichmäßig mit Hilfe eines, über einen Hashing-Algorithmus generierten, Schlüssels auf die Backend-Knoten verteilt. [Michael u. a., 2007]

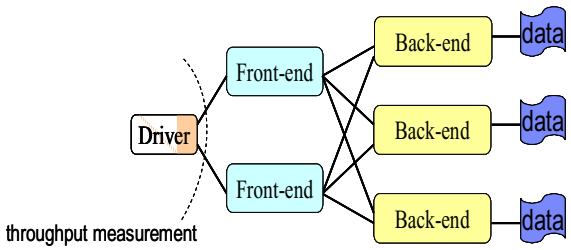


Abbildung 1: Abfragestruktur horizontal fragmentierter Systeme [6]

Der Nachteil einer derartigen Verteilung ist, der erhebliche Zusatzaufwand den Transaktionen atomarer Operationen nach sich ziehen. In Bereichen wo diese nicht notwendig sind, etwa bei der Verwaltung von Suchindexen, kann horizontale Fragmentierung allerdings zu signifikanten Leistungssteigerungen führen. [Michael u. a., 2007]

[6] Quelle: [Michael u. a., 2007]

2.5.3 MapReduce basierte Algorithmen

Liegen die Daten in einem verteilten System vor, muss auch die Verarbeitung über viele Systeme verteilt werden. Mit *MapReduce* wird in [Dean u. Ghemawat, 2004] ein dafür geeignetes Programmier-Paradigma vorgestellt.

Die Eingabe- und Ausgabedaten der Verarbeitung bestehen aus einfach strukturierten Parameter-Wert-Paaren (engl. Key-Value Pairs), welche in den drei festen Phasen *Map*, *Shuffle* und *Reduce* verarbeitet werden. In der *Map*-Phase wird jeder Datensatz einzeln und unabhängig von anderen durch die Anwendung verarbeitet. Die Unabhängigkeit ermöglicht es, diesen Schritt auf viele parallele Prozesse zu verteilen und die Verarbeitung sehr "nah" bei den Daten — also über das gesamte verteilte System hinweg — durchzuführen. Die Ergebnisse der Verarbeitung — wiederum ein Parameter-Wert-Paar — werden in der darauffolgenden *Shuffle*-Phase entsprechend der Parameter sortiert. In der abschließenden *Reduce*-Phase werden alle zu einem Parameter gehörenden Werte gemeinsam verarbeitet. Die Verarbeitung der Gruppen erfolgt wieder unabhängig von anderen, so dass auch dieser Schritt auf viele parallele Prozesse und Rechenknoten verteilt werden kann. [Dean u. Ghemawat, 2004]

Ggf. MapRed. Beispiel einfügen.

Trotz des sehr einfachen Verarbeitungsschemas ist es möglich, komplexe Algorithmen mit *MapReduce* auszuführen. In [Chu u. a., 2006] werden für zahlreiche Algorithmen des maschinellen Lernens und die dafür notwendigen Umformungen beschrieben und analysiert. Grundlage der Umformungen ist das in [Kearns, 1998] beschriebene *Statistical Query Model*. Dieses ermöglicht es, statistische Konzepte für Testdatensätze zu untersuchen und aus diesen darin vermutete statistische Verteilungen abzuleiten. Auf dieser Basis beschreibt [Chu u. a., 2006] wie spezielle Algorithmen umgeformt werden können, um diese durch die Bildung von Teil-Approximationen und nachgelagerte Zusammenfassung in eine verteilbare *MapReduce* Form zu überführen. Die so umgeformten Algorithmen skalieren nahezu linear mit der Anzahl der verfügbaren Rechenknoten bzw. Prozessorkernen. Dass diese Umformungen auch für die in Abschnitt 2.3 beschriebenen Algorithmen des kollaborativen Lernens möglich sind, wird in [Jiang u. a., 2011] gezeigt. [Chu u. a., 2006]

BSP

2.5.4 Skalierung der Filtermodelle

Die zuvor beschriebenen allgemeineren Methoden zur Verteilung und Verarbeitung großer Datenmengen sind im Zusammenhang mit Filtermodellen (vgl. Abschnitt 2.3) vor allem im Bereich der Vorverarbeitung bzw. des Trainings relevant.

Geht man davon aus, dass zur Verarbeitung eines einzelnen Bewertungseintrages 48 Byte im Speicher belegt werden, so stößt man bei "üblichen" 16 GB Hauptspeicher nach ca. 300 Millionen Einträgen an die Grenzen eines einzelnen Knotens. Auch wenn effizientere Speichernutzung — etwa durch die Reduktion von Objektreferenzen — den Speicherbedarf pro Eintrag auf 28 Byte senkt (vgl. [Owen u. a., 2011]), ist es dennoch ratsam erheblich früher verteilte Algorithmen zu nutzen. Wie [Chu u. a., 2006] und [Jiang u. a., 2011] zeigen, profitieren Systeme mit mehreren Prozessorkernen ebenfalls signifikant von der Parallelisierung durch *MapReduce*.

Da die Laufzeit aller in Abschnitt 2.3 beschriebenen Modelle von der Anzahl der Nutzer und Elemente anhängig ist, machen es große Datenmengen zudem sehr schwer Empfehlungen "online" zu berechnen. Die Reduktion der verarbeiteten Informationen (zB. durch Cluster-Mechanismen) stellt einen möglichen Ausweg dar. Da dadurch allerdings auch Informationen verloren gehen, sind die resultierenden Ergebnisse auch ungenauer. Einen möglichen Kompromiss zur Erzeugung von schnellen und präzisen Ergebnisse beschreibt [Linden u. a., 2003]. Alle rechenaufwendigen Verarbeitungsschritte, wie zum Beispiel die Berechnung von Element- oder Nutzerähnlichkeiten (vgl. Abschnitt 2.3.2), werden in eine Vorverarbeitungsphase verschoben und nur die eigentliche Generierung der Empfehlungen geschieht "online". Als Ergebnis der Vorverarbeitung erhält man entsprechend eine Matrix, die Nutzer zu allen anderen Nutzern bzw. Elemente zu allen anderen Elementen korreliert. Der Rechenaufwand zur Generierung der Empfehlungen ist dann nur noch abhängig von der Anzahl der vom Nutzer gekauften bzw. bewerteten Elemente. Ein wichtiger Nachteil ist, dass neue Daten erst verspätet in das abgeleitete Filtermodell einfließen können, dies kann abhängig von der "Stabilität" der Datenbasis, zum Beispiel wenn Elemente nur kurze Zeit "verfügbar" sind (vgl. [Cornelis u. a., 2007]), ein Ausschlussfaktor für Offline-Berechnungen sein. [Linden u. a., 2003]

2.6 Qualitätsmaße

Im folgenden Abschnitt werden verschiedene Wege zur Bewertungen von Empfehlungsdiensten beschrieben. Sie dienen zum qualitativen Vergleich von Filtermodellen und als Kontrollwerkzeug vor und während der Inbetriebnahme des Dienstes.

2.6.1 Mittlere Abweichung

Eine naheliegende Methode zum Vergleich von Filtermodellen ist es, deren Fehlerquote in Bezug zu vorliegenden historischen Nutzerdaten zu bestimmen, bzw. zu messen wie stark die ermittelten Werte von den tatsächlichen abweichen. Hierfür wird oft der *mittlere absolute Fehler* (engl. Mean Average Error (MAE)) oder die *Wurzel aus dem mittleren quadratischen Fehler* (engl. Root Mean Squared Error (RMSE)) genutzt.

Auf der Basis einer Testmenge $\mathcal{T}$ wird für alle darin befindlichen Nutzer-Element-Paare (u, i) der berechnete Ratingwert $\hat{r}_{ui}$ mit dem tatsächlichen r_{ui} vergleichen. Die beiden Fehlermaße werden dann wie folgt gebildet:

$$MAE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} |\hat{r}_{ui} - r_{ui}|} \tag{24}$$

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i)\in\mathcal{T}} (\hat{r}_{ui} - r_{ui})^2} \tag{25}$$

Der Unterschied zwischen den Methoden liegt in der Bewertung großer Fehler. Wie in Tabelle 4 gezeigt wird, kann dies durchaus ausschlaggebend sein. Im gezeigten Beispiel würde das Modell $\hat{r}_{(1)}$ vom MAE bevorzugt, weil ein Fehler durch die drei richtigen Werte stärker ausgeglichen wird. Beim Vergleich mit dem RMSE würde Modell $\hat{r}_{(2)}$ bevorzugt weil die vier "kleinen" Fehler in $\hat{r}_{(2)}$ weniger ins Gewicht fallen als der größere Fehler in $\hat{r}_{(1)}$. [Shani u. Gunawardana, 2011]

2.6.2 Trefferquote und Genauigkeit

Wird eine Liste von Empfehlungen erzeugt, kann neben der möglichen Bewertung und deren Genauigkeit auch gemessen werden, ob und wieviele der Empfehlungen korrekt waren.

	r	$\hat{r}_{(1)}$	$\hat{r}_{(2)}$
Item1	3	3	$\underline{2}$
Item2	4	$\underline{1}$	$\underline{2}$
Item3	2	2	$\underline{3}$
Item4	3	3	$\underline{2}$
MAE		**0.75**	1.25
RMSE		1.5	**1.32**

Tabelle 4: Beispielrechnung zum Vergleich von MAE und RMSE - gegenübergestellt sind tatsächliche Bewertungen r und von zwei verschiedenen Quellen erzeugte angenommene Bewertungen $\hat{r}_{(1)}$ und $\hat{r}_{(2)}$. Abweichende Bewertungen sind unterstrichen.

Bei der Evaluierung mit historischen Daten werden hierzu aus einem Nutzerprofil eine Reihe von Elementen entfernt und für das abgeleitete Profil eine Liste von Empfehlungen erzeugt. Durch den Vergleich der entfernten Elemente mit den Empfehlungen können dann *Genauigkeit* P (engl. Precision) und *Trefferquote* R (engl. Recall) wie folgt gemessen werden:

$$P_u = \frac{|hits_u|}{|recset_u|} \tag{26}$$

$$R_u = \frac{|hits_u|}{|testset_u|} \tag{27}$$

Die Anzahl der korrekt empfohlenen Elemente $hits_u$ wird zur Anzahl aller empfohlenen Elemente $recset_u$ bzw. zur Anzahl der möglichen richtigen Empfehlungen $testset_u$ ins Verhältnis gesetzt. Die Verbesserung beider Werte steht dabei oft im Widerspruch zueinander. So kann durch die Vergrößerung der Anzahl von empfohlenen Elementen auch die Wahrscheinlichkeit von Treffern und damit die Trefferquote gesteigert werden. Allerdings hätte die insgesamt gestiegene Zahl von Empfehlungen auch einen negativen Einfluss auf die Genauigkeit. Zur Kombination beider Werte wird daher oft die F_1 Metrik genutzt. Sie gewichtet beide Werte mit einer Neigung zum schlechteren.

$$F_1 = \frac{2 * P * R}{P + R} \tag{28}$$

Die ebenfalls mögliche Bestimmung der Ausfallquote (engl. Fallout), also der Anzahl der nicht korrekt empfohlenen Elemente im Verhältnis zur Menge aller Elemente, wird seltener zur Bewertung von Ergebnissen herangezogen. [Shani u. Gunawardana, 2011; Jannach u. a., 2010]

2.6.3 Empirische Messung

Ergänzend zur Messung der Approximationsgüte des erzeugten Modells muss bei einem praktischen Einsatz auch der tatsächliche Erfolg der eingesetzten Methode gemessen werden. Dafür bieten sich die aus dem Marketing stammenden *Klickraten* und *Conversion-Raten* Messung an, des weiteren kann in Verbindung mit Suchindexen auch die Bestimmung der *Successful Session* Quote genutzt werden.

Klickrate Beschreibt die Anzahl der Klicks auf ein Element, zum Beispiel einen Werbebanner, im Verhältnis zur Häufigkeit mit der es angezeigt wurde.

Conversion-Rate Mit der Conversion-Rate wird bestimmt welcher Anteil der Besucher ein bestimmtes Ziel, die sog. *Conversion*, erreicht haben. Mögliche Ziele sind zum Beispiel erfolgreiche Kaufabschlüsse oder Kontaktaufnahmen der Kunden. Die Ziele werden abhängig vom Kontext so gewählt, dass diese aktiv zur Wertschöpfung beitragen. Die Berechnung der Conversion-Rate entspricht gem. Formel (29) dem Verhältnis der gemessenen Anzahl an Conversions *conv* und der Anzahl der Besucher $visitors$ die im gegebenen Zeitraum eine oder mehrere Seiten aufgerufen haben. Maschinelle Seitenzugriffe, zum Beispiel durch Suchmaschinen-Spider, fließen nicht in Messung der Nutzerzahlen ein. Ebenso werden wiederholte Conversions des gleichen Besuchers nicht mehrfach gezählt. [Krüger, 2011]

$$CR = \frac{|conv|}{|visitors|} * 100 \tag{29}$$

Successful Session Da die Nutzeroberfläche von Suchmaschinen in der Regel kein explizites Feedback vom Nutzer zur Qualität der gefundenen Ergebnisse zulässt, muss die Relevanz einer Ergebnisliste über implizite Verfahren gemessen werden. In [Smyth u. a., 2005] wird die Methode der *Successful Session* für diesen Zweck beschrieben. Die Ergebnisliste einer Suche wird demnach immer dann als relevant bewertet, wenn der Nutzer mindestens eines der Ergebnisse ausgewählt hat. Wird keines gewählt, wird die Liste ent-

sprechend als irrelevant bewertet. Eine weitere Abstufung bzw. Steigerung der Relevanz, zum Beispiel bei wiederholter Ergebniswahl, wird nicht getroffen. [Smyth u. a., 2005]

HB Kap. 10.4.2 dazu?

3 Entwurf

Aufbauend auf den im vorangegangen Abschnitt beschriebenen Grundlagen wurde im Rahmen der Diplomarbeit eine Beispielapplikation implementiert. Die an diese Applikation gestellten Anforderungen, die zu lösenden Problemstellungen bei der Integration, der gewählte Ansatz zur Bildung von Empfehlungen und der konzeptionelle Aufbau der Gesamtapplikation werden im Folgenden beschrieben.

3.1 Anforderungen

Da die Skalierbarkeit der Anwendung einen hohen Stellenwert hat, werden im vorliegenden Abschnitt neben funktionellen Anforderungen auch Anforderungen an die Leistungsfähigkeit beschrieben.

3.1.1 Anwendungsfälle

Aus der angestrebten Kombination von Empfehlungsdienst und Suchindexen ergeben sich zahlreiche Anwendungsfälle. Für jeden der vorab definierten Fälle beschriebt der Abschnitt, welcher Mehrwert sich ergibt, wie dieser geprüft werden kann und welche Aufgaben die beiden Bestandteile haben. Die Anwendungsfälle werden zudem durch Beispiele ergänzt, welche an den Verkauf von Konzertkarten angelehnt sind.

Ggf. Beispielbasis in der gesamten Arbeit so definieren

Personalisierte Suche Wie in Abschnitt 2.1.3 beschrieben, kann die Relevanzberechnung beim Sortieren einer Dokumentenliste auch entsprechend eines Nutzerprofils angepasst werden. In der zu einer Anfrage gefundenen Dokumentenliste sollen dadurch die für den Nutzer relevanten Dokumente bevorzugt werden. Ob die erzielten Ergebnisse tat-

sächlich höheren Nutzen bieten, kann zum Beispiel durch eine Steigerung des "Successful Session" Maßes (vgl. Abschnitt 2.6.3) gemessen werden.

Die Aufgabe der "Profilbildung" soll durch die Mittel der Empfehlungsdienste (vgl. Abschnitt 2.3) integriert werden. Der Suchindex verarbeitet die Anfrage des Nutzers, findet alle relevanten Elemente und sortiert diese entsprechend der Relevanz (vgl. Abschnitt 4.1.3).

Sucht ein Nutzer zum Beispiel nach "Musicals Hamburg" so wäre es die Aufgabe des Suchindexes "Disneys König der Löwen" und "Caveman" als relevante Ergebnisse zu finden. Da der eigentliche Suchausdruck hier keinen Schluss auf die Präferenz des Nutzers zulässt, sollte die Sortierung der Ergebnisse durch den Empfehlungsdienst entsprechend angepasst sein.

Kontextbasierte Empfehlungen Ohne eine konkrete (Such-)Anfrage des Nutzers kann man auf Basis des vorliegenden Nutzerprofils Empfehlungen generieren. Die Kombination mit einem Suchindex ermöglicht es zudem inhaltliche Aspekte einfließen zu lassen. Bei der Empfehlung von Veranstaltungen wäre es dadurch zum Beispiel möglich, nur Veranstaltungen zu empfehlen die noch nicht ausgebucht sind oder die in einem bestimmten Zeitraum liegen.

Der Empfehlungsdienst ist in diesem Szenario für die Zusammenstellung der möglichen Kanidaten-Elemente zuständig, dem Suchindex fällt die Aufgabe der Anreicherung und Filterung dieser Elemente zu. Die Kontrolle der Ergebnisse kann zum Beispiel durch den Vergleich von Klickraten oder Conversion-Raten erfolgen.

Komplementär-Suche Als Ergänzung zu einer bestehenden Liste von Elementen, zum Beispiel in einer Merkliste oder einem Warenkorb, ist es möglich für den Nutzer relevante komplementäre Elemente zu finden. Wie bei den kontextbasierten Empfehlungen werden mögliche komplementäre Elemente vom Empfehlungsdienst gefunden und durch die Informationen des Suchindex ergänzt oder gefiltert.

Im Gegensatz zum vorangegangenen Anwendungsfall ist der Bezug zu der vom Nutzer

getroffenen Auswahl maßgeblich. Hat sich der Nutzer zum Beispiel für ein Musical in Hamburg entschieden, scheint es sinnvoll ihm noch eine passende Hotelreservierung oder passende Restaurants zu empfehlen. Das Zusammenspiel von Suchindex und Empfehlungsdienst ist in diesem Zusammenhang vielversprechend. Der Suchindex kann den "inhaltlichen" Bezug sicherstellen und so zum Beispiel nur Restaurants in der richtigen Stadt und zum richtigen Zeitpunkt zu finden, der Empfehlungsdienst kann auf dieser Basis dann entsprechend der Präferenzen des Nutzers die richtige Vorauswahl treffen. Die Messung der Conversion-Raten kann auch hier zur Kontrolle genutzt werden.

Cross-Selling Nutzt man die in Abschnitt 2.3.2 beschriebenen elementbasierten Nachbarschaftsmodelle innerhalb des Empfehlungsdienstes so ist es möglich, für jedes Element weitere ähnliche Elemente zu finden. Die Ähnlichkeit der Elemente bezieht sich dann je nach Maß (vgl. Abschnitt 2.3.1) vor allem darauf wie oft diese gemeinsam ähnlich bewertet wurden. Die daraus getroffenen Empfehlungen können für das sog. *Cross-Selling* genutzt werden.

Sucht ein Nutzer nach "Musicals Köln" kann dadurch zum Beispiel die Empfehlung ausgesprochen werden, dass der Besuch des Kölner Doms oder eine Altstadtführung von vielen anderen Nutzern hinzu gebucht wurde. Im Gegensatz zu den komplementären Suchen, bei denen die Art der Komplementbildung durch entsprechende Vorkonfiguration eingeschränkt ist, bestimmen hier einzig die Resultate der Nachbarschaftsmodelle.

Der Empfehlungsdienst nimmt hierbei zwei Aufgaben ein, er findet über die Nachbarschaftsmodelle relevante Kanidatenelemente und kann die daraus resultierende Liste in einem zweiten Schritt entsprechend des Nutzerprofils personalisiert sortieren bzw. filtern. Der Suchindex kann wie bei den kontextbasierten Empfehlungen zur Filterung und inhaltlichen Anreicherung genutzt werden.

3.1.2 Leistungsanforderungen

Der Bestimmung der Leistungsanforderungen wurde die mit Google Analytics gemessene[7] Seitennutzung vom 01. September 2012 und bis 30. November 2012 zugrunde gelegt . In diesem Zeitraum wurden ca. 40 Millionen Seitenaufrufe von ca. 1,2 Millionen unterschiedlichen Nutzern verursacht. Die durchschnittliche Zahl von 500.000 Seitenaufrufen pro Tag war zudem nicht gleichbleibend konstant, sondern enthielt zwei Zugriffsspitzen mit bis zu 1 Million Zugriffen am Tag. Der Anteil der Seitensuche am Gesamtaufkommen beträgt ca. 10%. Die Verteilung der Seitenaufrufe am Tag ist abhängig von der Tageszeit. Im Zeitraum von 8 Uhr bis 18 Uhr werden ca. 80% aller Aufrufe getätigt. Im gesamten Zeitraum gab es ca. 13.000 Kaufabschlüsse bzw. Conversions. Der Anteil wiederkehrender Nutzer beträgt im Mittel 70%, dies entspricht dem 15-fachen der in diesem Zeitraum registrierten Nutzer.

GA Screen ergänzen

Datenerhebung Nimmt man an, dass zudem jede der besuchten Seiten für die Datenerhebung relevant ist und dass im Mittel zwei Events mit je 150 Byte pro Seitenaufruf aufgezeichnet werden (vgl. Abschnitt 4.4.1), ergibt sich pro Tag ein Datenaufkommen zwischen 150 MB und 300 MB. Zieht man zudem die zeitliche Verengung in Betracht, so muss die Datenerhebung bis zu 2000 Anfragen pro Minute bzw. ca. 35 Anfragen pro Sekunde in den Lastzeiten verarbeiten. Um sicherzustellen dass der Dienst auch unter sehr hoher Last stabil und schnell arbeitet, werden diese Anforderung um das 5-fache gesteigert. Dadurch ergibt sich die Anforderung dass die Datenerhebung bis zu 10.000 Anfragen pro Minute verarbeiten können muss.

Personalisierung Die im vorangegangenen Abschnitt beschriebenen Anwendungsfälle zur Personalisierung sind für ca. 50% aller Seiten relevant. Beachtet man zudem die Zahl der wiederkehrenden Nutzer, so resultiert dies in ca. 125.000 Personalisierungsanfragen an durchschnittlichen Tagen und ca. 300.000 Anfragen an Tagen mit Zugriffsspitzen. Beachtet man auch hier die zeitliche Verengung der Anfragen, muss mit 500 Anfragen pro Minute bzw. 10 Anfragen pro Sekunde in den Lastzeiten gerechnet werden. Wie bei der

[7] http://www.google.com/analytics/

Datenerhebung resultiert daraus die Anforderung, dass die Personalisierung bis zu 2.500 Anfragen pro Minute verarbeiten können muss.

Lidl-Usecase ergänzen?

3.2 Problemstellungen

Bei der Kombination von Suchindexen (vgl. Abschnitt 2.1) und kollaborativen Empfehlungsdiensten (vgl. Abschnitt 2.3) treffen zwei Techniken mit verschiedenen Kernaufgaben aufeinander. Den möglichen Nutzen der daraus entstehenden Resultate zeigen die im vorangegangenen Abschnitt beschriebenen Anwendungsfälle, die dafür zu lösenden Schwierigkeiten und mögliche Lösungsansätze werden im Folgenden beschrieben.

3.2.1 Disjunkte Kanidatenlisten

Da beide Bestandteile des Systems zur Erstellung der beschriebenen Anwendungsfälle eigene Problembereiche abdecken, sind die für eine Anfrage generierten Ergebnislisten erwartungsgemäß auch unterschiedlich. Würde man beide Systeme vollständig unabhängig voneinander betreiben und abfragen, wäre die Gefahr groß, dass die Ergebnismengen keine oder nur eine sehr kleine Schnittmenge zur weiteren Verarbeitung generieren würde. Im Anwendungsfall der personalisierten Suche hätte das zur Folge, dass nur wenige Ergebnisse wirklich entsprechend der Präferenzen des Nutzes bewertet würden und alle Weiteren als nicht relevant herabgestuft würden. Anschaulich wird dies zum Beispiel, wenn bei einer Suche nach “Musicals Hamburg”, wegen der fehlenden inhaltlichen Bindung, vom Empfehlungsdienst auf Basis des Nutzerprofiles nur Präferenzen zu anderen Städten oder anderen Veranstaltungstypen gegeben würden. Die Bewertung der vom Suchindex gelieferten Ergebnisse wäre damit nicht möglich. Je spezieller die Suche bzw. Filterung des Nutzers ist, desto kleiner ist die vom Suchindex gefundene Ergebnismenge und umso geringer wäre dann die Wahrscheinlichkeit eine Schnittmenge zu finden.

Bei den Anwendungsfällen der kontextbasierten Empfehlungen und der Komplementär-Suche besteht das Problem, dass die Größe der möglichen Schnittmengen durch die vordefinierte Filterung des Suchindexes bereits stark eingeschränkt und damit die Wahrschein-

lichkeit eine ausreichend große Schnittmenge zwischen beiden Ergebnissmengen zu erhalten von Beginn an sehr gering ist. Auch dieser Fall kann durch das im vorangegangene Beispiel geschilderte Szenerio leicht erzielt werden, die vordefinierten Filter der Suche entspricht dann einer gezielten bzw. speziellen Suche mit den bereits beschriebenen Auswirkungen. Einzig der Anwendungsfall des Cross-Selling ist hiervon nicht betroffen, weil hier der Suchindex nur zur "Anreicherung" der vom Empfehlungsdienst gefundenen Ergebnisse genutzt wird. Hierfür muss nur sichergestellt werden, dass beide Systeme auf der gleichen Elementmenge operieren damit für jede Empfehlungen auch tatsächlich inhaltliche Anreicherungen durchgeführt werden können.

Ein möglicher Ansatz um den resultierenden Problemen vorzubeugen ist es, für die beschriebenen Anwendungsfälle jeweils ein führendes System zu wählen. Anstatt eine Anfrage gleichzeitig in beiden Systemen zu verarbeiten, soll die eigentliche Kandidatenliste immer von einem System erstellt werden um in einem darauf folgenden Schritt vom Zweiten weiterverarbeitet zu werden. Für den Anwendungsfall der personalisierten Suche und die kontextbasierten Empfehlungen bedeutet dies, dass der Suchindex als führendes System die Kandidatenliste aufbaut die dann im weiteren bewertet und umsortiert wird. Bei Komplementär-Suche und Cross-Selling ist der Emfehlungsdienst das führende System dessen Ergebnisse durch den Suchindex inhaltlich ergänzt oder gefiltert werden. Können keine oder zuwenige Empfehlungen gefunden werden, ist es in beiden Fällen möglich und akzeptabel Standardelemente zu ergänzen oder die Filter des Suchindexes zu lockern.

Ein weiterer Lösungsansatz ist die Veränderung bzw. Erweiterung der Suchanfragen durch den Empfehlungsdienst (vgl. Abschnitt 2.1.3). Ähnlich dem von [Boughareb u. Farah, 2011] gewählten Ansatz, könnte die Anfrage durch die n wichtigsten Terme der gefundenen Empfehlungen ergänzt werden. In allen Anwendungsfällen bei denen die Suche als führendes System genutzt wird, würde damit die Anfrage auch direkt zum Mittel der Personalisierung. Da nur der Suchindex eine Ergebnisliste zusammenstellen muss, welche durch die zusätzlichen Terme anders gewichtet wäre, wird das Problem disjunkter Kandidatenlisten vermieden. Die Anwendungsfälle bei denen der Empfehlungsdienst als führendes System dienen muss, könnten ebenfalls über diesen Ansatz gelöst werden, indem die Suchanfrage

direkt vom Empfehlungsdienst mit Hilfe der gefundenen Elemente “formuliert” wird.

3.2.2 Lern-Laufzeiten

Eine wichtige Eigenschaft aller in Abschnitt 2.3 beschriebenen Methoden ist, dass die vorliegende *User-Item* Matrix genutzt wird um Modelle der Ähnlichkeiten zwischen Nutzern oder Elementen abzuleiten. Auch wenn die vollständige Neugenerierung der Modelle durch iterative Methoden verhindert werden kann, so müssen neu gewonnene Daten dennoch einen zusätzlichen Schritt durchlaufen bevor diese Einfluss auf für den Nutzer generierte Empfehlungen haben. Je nach Dauer dieses zusätzlichen Schrittes, kann dies den Nutzen des Systems erheblich beeinflussen.

Iterative Updates bei User/Item based Recommendation ergänzen

Aus den notwendigen Aktualisierungen durch neue Daten ergibt sich ein weiterer Nachteil, der vor allem in Lastphasen kritisch ist. Würde man mit jeden Datensatz das Modell (inkrementell) erweitern resultiert daraus, dass immer dann viele Aktualisierungen notwendig würden wenn der Dienst ohnedies durch viele Nutzern unter Last gesetzt wird. Eine Entlastung, etwa durch das Zwischenspeichern von Ergebnissen, wäre in einer solchen Phase wegen des ständig aktualisierten Modells kaum möglich.

In [Linden u. a., 2003] beschreibt eine mögliche Strategie die Auswirkungen des Problems zu verhindern. Da nutzerbasierte Modelle wegen der direkten Bindung an das Verhalten des einzelnen Nutzers stärkeren Schwankungen unterworfen sind, wird die Nutzung der stabileren elementbasierte Modelle vorgeschlagen. Die Stabilität begründet sich vor allem darauf, dass im beschriebenen Fall die Einstellungsänderung einer großen Nutzermenge notwendig ist um die Ähnlichkeit zwischen Elementen zu beeinflussen. Durch die Nutzung der stabileren Modelle verringert sich die Notwendigkeit Modelle mit jedem neuen Datensatz neu berechnen zu müssen. Abhängig von Anwendungsfall ergibt zudem die Möglichkeit große Teile der Vorberechnung auf zusätzlich vorgehaltener Infrastruktur *offline* zu verrichten ohne die Ressourcen des Empfehlungsdienstes in Anspruch nehmen zu müssen.

Auch der Verzicht auf ein explizites Filtermodell und die Anreicherung der Suchanfrage aus [Boughareb u. Farah, 2011] stellt eine mögliche Alternative dar. Um dies auf die be-

schriebenen Anwendungsfälle zu übertragen, könnten die aus den Vorberechnungen des Empfehlungsdienstes gewonnenen Informationen zur Ähnlichkeit zwischen Elementen genutzt werden um die entsprechend dieser Elemente zu gruppieren. Die Personalisierung könnte dann abhängig von der Präferenz des Nutzers für die gefundenen Gruppen stattfinden. Die Zwischenergebnisse der Matrixfaktorisierung, vor alle die Werte der q_i Vektoren, könnten in ähnlicher Weise genutzt werden (vgl. Abschnitt 2.3.3 und 4.4.5). Die Lern-Laufzeiten sind dann bei stabilen Modellen wie in der vorangegangen Lösungsstrategie kein anwendungskritischer Faktor mehr.

3.2.3 Skalierbarkeit

Beim Aufbau eines Gesamtsystems, welches den beschriebenen Leistungsanforderungen genügt, muss jeder Bestandteil Möglichkeiten zur Skalierung bieten. D.h. nicht nur die Komponenten der Datenhaltung und Datenverarbeitung sollten diesbzgl. betrachtet werden, sondern auch die Kommunikationswege zwischen ihnen. Wie die in Abschnitt 2.5 zusammengefassten Grundlagen zeigen, existieren verschiedene Ansätze zur Skalierung einzelner Bestandteile. Die Ableitung einer einheitlichen Architektur ist deshalb ein weiteres zu lösendes Problem.

Ein möglicher Lösungsansatz zum Aufbau eines skalierbaren Systems zur allgemeinen Datenanalyse wird in [Lin u. Kolcz, 2012] beschrieben. Auf Basis von einfach zu beschreibenden *MapRecude*-Programmen werden in einem GFS gehaltene Daten direkt verarbeitet und analysiert. Die Architektur eines skalierbaren und für den Nutzer möglichst unsichtbaren Logging-Systems, sowie die daran gestellten Anforderungen beschreibt [Sadekar, 2012] am Beispiel der Netflix-Plattform[8]. In beiden Fällen geschieht die Verarbeitung der Daten asynchron. Vom Nutzer erzeugte Daten bewirken nicht sofort eine Aktualisierung des Gesamtsystems, vielmehr werden die Daten in beiden Fällen zunächst gesammelt und nachgelagert verarbeitet. Anhand der Google-News[9] Architekturbeschreibung zeigt [Das u. a., 2007], dass dies auch in sehr kurzen Zeitintervallen für eine sehr große Nutzerzahl

[8] http://netflix.com

[9] http://news.google.com/

möglich ist. Neben der asynchronen Datenverarbeitung wird hier zudem gezeigt, wie die Methoden der horizontalen Skalierung (vgl. Abschnitt 2.5.2) einfließen können.

Die Möglichkeiten der Vorberechung von Empfehlungs- oder Ähnlichkeits-Modellen werden in [Linden u. a., 2003] und [Das u. a., 2007] beschrieben. Die Skalierbarkeit und Leistungsfähigkeit der Systeme ist vor allem dadurch möglich, dass komplexe Berechnungen ebenfalls asynchron, d.h. vor der eigentliche Anfrage des Nutzers erfolgen (vgl Abschnitt 2.5.4).

3.3 Systemarchitektur

Aus den zu lösenden Anwendungsfällen und Problemen sowie den verschiedenen bekannten Lösungsansätzen lässt sich die in Abbildung 2 skizzierte Systemarchitektur ableiten. Die darin gezeigte Struktur und die aufgezeigten Datenaustausch-Pfade beschreiben dabei den Datenfluss zwischen den Komponenten[10]. Sie baut sich aus den folgenden Bestandteilen auf:

- Der **Nutzer** steht an Beginn und Ende der Interaktionswege. Durch seine Interaktion mit der Webseite wird ein entsprechendes Profil erstellt, welches es ermöglichen soll Suchergebnisse und Webseiteninhalte zu Personalisieren (vgl. Abschnitt)
- Die **Webseite** mit der eingebetteten **Suche** stellt das Interaktionsmedium für den Nutzer dar. Zudem sollen mit Hilfe der eingebetteten Tracking-Mittel die Interaktionen des Nutzers aufgezeichnet werden.
- Der **Tracker** dient der Aufzeichnung der Interaktionen des Nutzers mit der Webseite. Er stellt sicher, dass die aufgezeichneten Daten verlustfrei in den Datenspeicher übertragen werden. Er muss den in Abschnitt 3.1.2 beschriebenen Leistungsanforderungen bei der Datenerhebung genügen.
- Die **Datenhaltung** erfolgt auf den in Abschnitt 2.5.1 beschriebenen Grundlagen.

[10] Auf die Verwendung eines exakteren, formalen Datenflussdiagrammes wurde aus Gründen der Übersichtlichkeit verzichtet.

Sie soll sicherstellen, dass Daten zwischen mehreren Servern effizient verteilt werden können und ermöglich die Ausführung von *MapReduce*-Programmen (vgl. Abschnitt 2.5.3) für diese Daten.

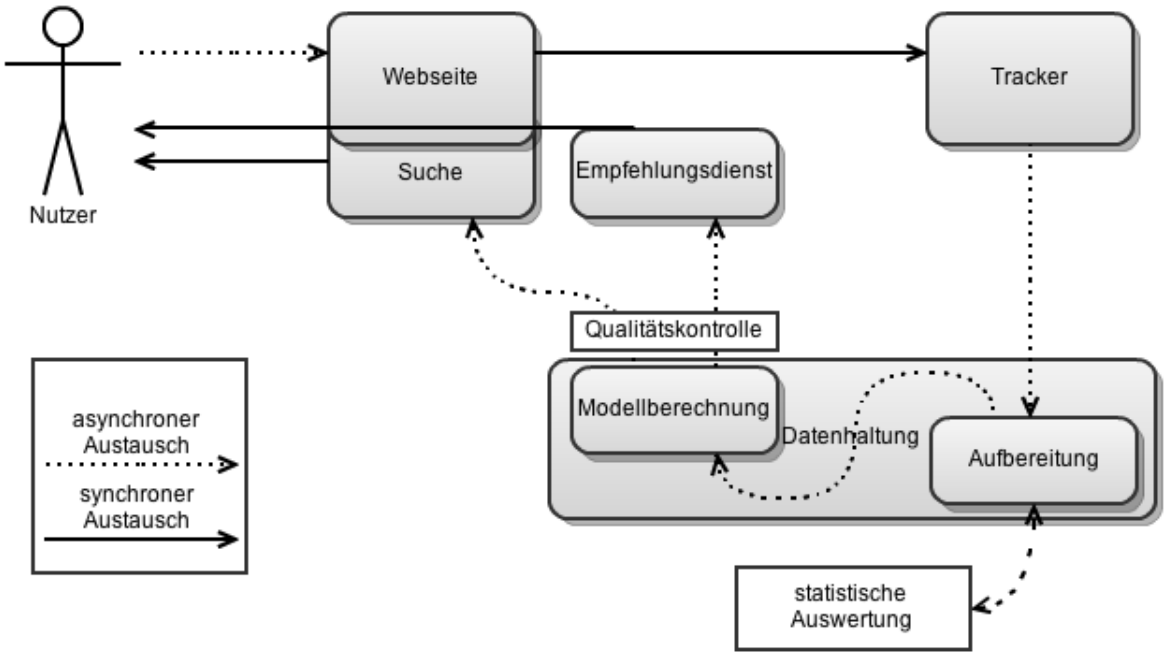


Abbildung 2: Systemarchitektur-Entwurf der alle notwendigen Komponenten und den Interaktionsfluss umreißt

- Die **Datenaufbereitung** analysiert die vom Tracker geschriebenen Interaktionsprotokolle aller Nutzer und führt notwendige Vorverarbeitungsschritte durch. Das Ergebnis der Vorverarbeitung sind zum einen kompakte Nutzerprofile die der in Abschnitt 2.2.1 beschriebenen *User-Item* Matrix entsprechen. Ein weiteres Ergebnis welches sich bei der Datenaufbereitung gewinnen lässt, sind verschiedene Maße zur Qualitätskontrolle bzw. zur statistischen Auswertung (vgl. Abschnitt 2.6.3).
- Bei der **Modellberechnung** werden mit Hilfe der in Abschnitt 2.3.2 und 2.3.3 beschriebenen Methoden alle notwendigen Berechnungen zur Bildung der Ähnlichkeitsmodelle bzw. zur Matrixfaktorisierung durchgeführt. Ebenso wie die Datenaufbereitung erfolgt sie verteilt auf den Datenhaltungs-Servern, zum Beispiel durch die

Ausführung vom *MapReduce*-Programmen. Vor der anschließenden Nutzung, müssen sie einer Qualitätskontrolle unterzogen werden. Dies kann durch die Überwachung der in Abschnitt 2.6 beschriebenen Maße erfolgen.

- Der **Empfehlungsdienst** nutzt das Ergebnis der Modellberechnung um Empfehlungen für die Anfragen einzelner Nutzer zu generieren. Durch die direkte Nutzung des Empfehlungsdienstes wird die Umsetzung der Anwendungsfälle zu kontextbasierten Empfehlungen, Komplementär-Suchen und Cross-Selling ermöglicht. Er muss den in Abschnitt 3.1.2 beschrieben Leistungsanforderungen zur Personalisierung genügen.

- Die Ergebnisse der **Suche** werden durch die Ausgabe des Empfehlungsdienstes den Präferenzen des Nutzers entsprechend priorisiert. Mit dem in Abschnitt 4.4.5 beschriebenen Ansatz, ist es zudem möglich, die Ergebnisse der Modellberechnung der Matrixfaktorisierung direkt in der Suche zu nutzen und damit bei der Personalisierung auf einen separaten Empfehlungsdienst zu verzichten. Wie der Empfehlungsdienst muss auch die Suche den in Abschnitt 3.1.2 beschrieben Leistungsanforderungen zur Personalisierung genügen.

Wie in der Abbildung aufgezeigt ist, soll der Datenaustausch zwischen einigen Komponenten asynchron verlaufen. Dies entspricht den in Abschnitt 3.2.3 beschriebenen Lösungsstrategien zur Skalierbarkeit. Es ermöglicht ohne zusätzliche Anpassung der Kommunikationswege, dass jede der Komponenten bei Bedarf auch auf mehrere Server verteilt werden kann.

Da die Verarbeitung der Daten und die Erstellung neuer Modellberechnungen im produktiven Betrieb möglichst regelmäßig durchgeführt werden muss, wird zudem ein zentraler Dienst zur Steuerung bzw. Überwachung des Ablaufs benötigt. Da dieser keinen besonderen Anforderungen entsprechen muss und die Bandbreite der möglichen Lösungswege den Rahmen der Arbeit sprengen würde, kann in der weiteren Betrachtung stets von einfachen zeitgesteuerten Prozessen ausgegangen werden.

Ggf Zookeeper oder MQ Konzepttelink ergänzen / Kapitel verlinken

3.4 Bildung von Empfehlungen

Da die verfügbaren Suchlösungen (vgl. Abschnitt 4.1) die Relevanzberechnung nicht ausschließlich *TF-IDF*-basiert durchführen (vgl. Abschnitt 4.1.3) und auch andere Faktoren einbeziehen können, wird auch die direkte Integration der Berechnungen zur Personalisierung möglich. Die dafür notwendigen Anpassungen und Annahmen werden im Folgenden beschrieben und sollen bei der Umsetzung evaluiert werden.

Aufbauend auf den möglichen Maßnahmen zur Personalisierung der Relevanzberechnungen aus Abschnitt 2.1.3 können auch kollaborative Methoden einbezogen werden. Die *TF-IDF*-basierte Relevanzberechnung aus Formel (4) wird zu diesem Zweck wie in Formel (6) um ein Nutzerprofil u ergänzt. Entsprechend des Profils kann dann die Berechnung der Empfehlungen einbezogen werden (Formel (30)). Der Faktor α wird zur "Normalisierung" des Empfehlungswertes genutzt.

Die Zwischenergebnisse der Matrixfaktorisierung bieten sich dafür besonders an, da sie im Gegensatz zu den auf Nutzer- oder Elementähnlichkeiten basierenden Methoden Element- und Nutzerfeatures in Form von Vektoren getrennt repräsentieren. Die Featurevektoren p_u repräsentieren das Nutzerprofil, welches über ein Skalarprodukt mit den Vektoren q_i der einzelnen Dokumente direkt zur Relevanzberechnung genutzt werden kann. Formel (31) erweitert die ursprüngliche Relevanzberechnung aus Formel (4) um die Empfehlungsberechnung aus Formel (17). Zur Wahrung der Lesbarkeit beschreiben $v(d)$ und $v(u)$ die dem Dokumenten bzw. Nutzer zugehörigen Featurevektoren q_i bzw p_u.

$$\text{score}(q, d, u) = \text{score}_{\text{tf-idf}}(q, d) * \alpha * \text{pred}(d, u) \tag{30}$$

$$\text{score}_{svd}(q, d, u) = \text{score}_{\text{tf-idf}}(q, d) * \alpha * (\mu + b_d + b_u + v(d)^T v(u)) \tag{31}$$

Das mit Hilfe der Matrixfaktorisierung gefundene Modell wird dadurch auf verschiedene Bestandteile des Systems verteilt. Durch die Erweiterung der im Suchindex gehaltenen Dokumente und die Ergänzung der Suchanfrage um die jeweiligen Featurevektoren wird es möglich, auch ohne einen zusätzlichen Empfehlungsdienst die Ergebnislisten zu personalisieren. Neben möglichen Leistungsverbesserungen sollen damit die Probleme disjunkter Kandidatenlisten (vgl. Abschnitt 3.2.1) in getrennten Systemen vermieden werden.

Um zudem die in Abschnitt 2.4 beschriebenen *Cold-Start* Probleme zu vermeiden und neu gewonnene Daten direkt einbeziehen zu können, wird die Berechnung wie in [Paterek, 2007] und [Koren u. a., 2009] um weitere Informationsquellen ergänzt. Das Nutzerprofil kann dadurch um weitere Bewertungen J_u ergänzt werden, ohne dass diese vollständig in das Modell einbezogen wurden:

$$v(u) \approx p_u + (|J_u| + 1)^{-0.5} * \sum_{j \in J_u} w_j \tag{32}$$

$$\tag{33}$$

Wurde für einen Nutzer noch kein Nutzervektor p_u berechnet, kann dieser ausgelassen werden, ohne die Möglichkeit zu verlieren die Relevanzberechnung zu personalisieren.

$$v(u) \approx (|J_u| + 1)^{-0.5} * \sum_{j \in J_u} w_j \tag{34}$$

Die hierfür genutzten Featurevektoren w_j sind ebenfalls dokumentenspezifisch. Sie werden gemeinsam mit dem gesamten Modell mit den Methoden des *stochastischen Gradientenverfahrens* oder den *Alternating Least Squares* gelernt (vgl. Abschnitt 2.3.3). [Paterek, 2007; Koren u. a., 2009]

4 Realisation

Mit der im vorangegangenen Abschnitt beschriebenen Architektur und den aufgestellten Leistungsanforderungen, wird im folgenden Abschnitt die Umsetzung der Beispielapplikation, die dafür genutzten Werkzeuge und notwendige Anpassungen beschrieben. Der Fokus liegt dabei auf den Bestandteilen die direkt bei der Personallisierung der Suche genutzt werden.

Ausgangspunkt bei der Umsetzung war die bereits vorhandene *Searchperience* Suchlösung welche entsprechend der Anforderungen angepasst werden sollte. Da *Searchperience* das Java basierte Apache Solr zum Aufbau und zur Abfrage der Suchindexe nutzt, wurden für die weiteren Bestandteile ebenfalls Java-Bibliotheken genutzt.

4.1 Apache Solr

Apache Solr (Solr) ist ein "Enterprise Search Server" welcher unter der Apache 2.0 Lizenz frei zur Verfügung steht. Aufbauend auf der Apache Lucene Suchbibliothek ermöglicht Solr den Aufbau einer sehr vielfältig konfigurierbaren Volltextsuche. Zu den wichtigsten Konfigurations- und Erweiterungseigenschaften zählen facettierte Suchen, automatische Anfragevervollständigung, dynamische Relevanz-Anpassungen (Boosting), Indexreplikation und Indexsharding (vgl. Abschnitt 2.5.2). Genutzt werden kann Apache Solr über eine "RESTful HTTP" Schnittstelle, welche sowohl XML als auch JSON zum Datenaustausch nutzten kann.

Konfiguration Die zwei wichtigsten Quellen zur Konfiguration des Dienstes sind die *solrconfig.xml* und die *schema.xml*. Die *solrconfig.xml* dient zur Integration aller Komponenten und integriert die Bestandteile und der Schnittstellen. Sie bestimmt zum Beispiel in welcher Reihenfolge und mit welchen Voreinstellungen die einzelnen Komponenten eine Suchanfrage bearbeiten und ermöglicht daneben auch die Integration eigener Komponenten. Mit Hilfe der *schema.xml* werden die vom Server verwalteten Dokumente bzw. Dokumentenbestandteile beschrieben und notwendige Vorverarbeitungsschritte konfiguriert.

Listing 1: Instanzierung von Apache Solr zur horizontalen Skalierung

```
# Server 0
java -DzkRun -Dbootstrap_confdir=solr/conf \
  -DnumShards=5 -DreplicationFactor=3 -DmaxShardsPerNode=3 \
  -jar start.jar


# Server 1..N
java -DzkHost=server0:9983 -jar start.jar
```

Kommentierte Beispiel- bzw. Basiskonfigurationen der beiden Konfigurationsdateien[11,12], sowie die Dokumentation der Anfragesyntax[13,14] sind ebenfalls Bestandteil der offenen Quelltexte und werden hier zur Wahrung des Umfangs nur kurz beschrieben.

4.1.1 Datenhaltung

Da Dokumente selten aus fortlaufendem unstrukturierten Text bestehen, oftmals ergänzende Informationen zu einem Dokument im Suchindex verfügbar gemacht werden sollen und da die Datenhalten möglichst effizient geschehen soll, ist es notwendig die mögliche Struktur der Daten mit Hilfe der *schema.xml* zu beschrieben. Um Redundanz innerhalb der Strukturbeschreibung zu vermeiden ist diese zweigeteilt. Zunächst werden alle Datentypen (*types*) konfiguriert – d.h. aus der Liste der möglichen Standardtypen werden die für den Anwendungsfall notwendigen ausgewählt und ggf. mit angepasster Basiskonfiguration verfügbar gemacht. Eine detailliertere Unterscheidung der Typen ist notwendig um sicherzustellen, dass die Daten möglichst effizient für den konkreten Anwendungsfall gespeichert werden. Im zweiten Schritt werden dann die eigentlichen Dokumentenbestandteile (auch Felder oder *fields*) den vorher konfigurierten Typen zugeordnet. In einer auf einen einzelnen Typen und ein Feld reduzierten Form zeigt Listing 4 dies beispielhaft. Anhand der Konfiguration werden die vorgehaltenen Dokumente mit Hilfe der geeigneten invertierten Indexe (vgl. Abschnitt 2.1.1) organisiert.

Die horizontale Skalierung (vgl. Abschnitt 2.5.2) der in Solr vorgehaltenen Daten kann

[11] http://svn.apache.org/viewvc/lucene/dev/trunk/solr/example/solr/collection1/conf/solrconfig.xml?view=markup

[12] http://svn.apache.org/viewvc/lucene/dev/trunk/solr/example/solr/collection1/conf/schema.xml?view=markup

[13] Anfragesyntax: http://wiki.apache.org/solr/ExtendedDisMax

[14] Anfrage-Funktionen: http://wiki.apache.org/solr/FunctionQuery

über beliebig viele *Backend-Knoten* erfolgen. Dabei werden, wie in Listing 1, die einzelnen Solr-Instanzen mit einer Referenz auf eine zentrale Konfigurationsinstanz gestartet (im Beispiel über Apache Zookeeper). Die Wahl der Server-Rollen (Frontend- o. Backend) erfolgt je nach Verfügbarkeit und wird automatisch erneutert wenn Knoten ausfallen oder ergänzt werden. Mögliche Redundanz in der Datenhaltung, und damit die Steigerung der Leistungsfähigkeit von Leseoperationen, kann durch du die Konfiguration der insgesamt genutzten Datenblöcke (auch *Shards*), den Replikations-Faktor und die Datenblöcke pro Knoten erzielt werden.

4.1.2 Suchanfragen

Innerhalb von Solr können sog. *RequestHandler* genutzt werden um Daten aus dem Index zu lesen, bzw. um Suchen innerhalb der Dokumentensammlung durchzuführen. Jeder *RequestHandler* ist wiederum aus verschiedenen Komponenten aufgebaut welche innerhalb der *solrconfig.xml* konfiguriert bzw. ergänzt werden können.

Die Komponenten innerhalb der Konfiguration spiegeln auch die Verarbeitungsreihenfolge der Suchanfragen wieder. Die Standardkomponenten des *SearchHandler* sind `solr.QueryComponent` (Query), `solr.FacetComponent` (Facet), `solr.MoreLikeThisComponent` (Mlt), `solr.HighlightComponent` (Highlight) und `solr.HighlightComponent` (Stats). Innerhalb einer Suchanfrage wird in `Query` die eigentliche Ergebnisliste gebildet welche dann im folgenden gefiltert (`Facet`), ergänzt (`Mlt`) oder weiterverarbeitet (`Highlight`) wird.

Die eigentliche Verarbeitung der Suchanfrage und die Bildung der Ergebnislisten geschieht gemäß einer erweitreen Tf-Idf Berechnung (vgl. Abschnitt 2.1.2.). Die Anpassung dieser Berechnung wird in [Muir u. a., 2011] umfassend beschrieben und hier zusammenfassend erläutert. Die Notwendigkeit zu dieser Anpassung ergibt sich aus dem Bedarf die Gewichtung einzelner Feldern und Terme anzupassen.

$$\text{score}(q,d) = \text{coord}(q,d) * \text{queryNorm}(q) * \sum_{t \in q} \text{tf-idf}(t,d) * \text{boost(t)} * \text{norm}(t,d) \qquad (35)$$

Listing 2: Einfache Solr-Anfrage mit angepassten Feld-Boosts

```
?q=Theater Hamburg Freitag
&qf=title^10 tag^3 stadt^3 beschreibung
```

Die Ausgangsberechnung aus Formel (4) wird durch die folgenden zusätzlichen Bestandteile zu Formel (35) ergänzt.

- **coord(q,d)** - Bewertet wieviele der Terme der gesamten Suchanfrage im Dokument gefunden wurden und hebt bzw. senkt die Relevant entsprechend.
- **queryNorm(q)** - Normalisiert alle gefundenen Relevanzwerte einer Anfrage mit dem gleichen Faktor. Dies dient vor allem der Vergleichbarkeit von Relevanzwerten zweier Anfragen.
- **boost(t)** - Ermöglicht die Anpassung der Relevant einzelner Terme einer Anfrage.
- **norm(q)** - Normierung des Relevanzwertes auf Feldebene, ermöglicht verschiedene Gewichtung von verschiedenen Feldern und Positionsabhängige Relevanzbeeinflussung.

Basierend darauf könnte die Suche nach "Theater Hamburg Freitag" in einer Dokumentensammlung zum Beispiel wie in Listing 2 aussehen. Die Gewichtung zwischen den vier genutzten Feldern würde Treffer im Feld "Titel" als relevanter bewerten als Treffer mit gleichen Tf-idf Wert im Feld "Beschreibung". [Muir u. a., 2011]

4.1.3 Relevanz-Anpassung

Die Nutzerprofil bezogene Anpassung der Relevanzwerte eines Dokumentes für eine Suchanfrage kann durch zwei Methoden innerhalb von Solr erreicht werden. Zum einen ist es möglich die vorhandenen Komponenten eines *RequestHandlers* durch eigene zu ergänzen, innerhalb dieser wäre dann die Erweiterung der Suchanfrage selbst, die Umsortierung der Ergebnislisten oder die Veränderung der Relevanzberechnung möglich. Des weiteren ermöglicht die *Query* Komponente auch die Integration von komplexeren Relevanzerechnungen über native Funktionen. Letzteres wird vor allem gebraucht um Berechnungen wie

Listing 3: Integration der Recommenderkomponenten in der solrconfig.xml

```
<config>
        <lib dir="Search/target/dependency" regex=".*\.jar" />
        <lib path="Search/target/Search.jar" />
        <query>
                <searchComponent name="booleanBoost"
                        class="com.aoe.search.SpringAwareSearchComponent">
                        <str name="searchDelegate">booleanBoostComponent</str>
                </searchComponent>
        </query>

        <requestHandler name="/select" class="solr.SearchHandler">
                <arr name="last-components"><str>booleanBoost</str></arr>
        </requestHandler>
</config>
```

Geographe-Distanzbestimmungen (näher ist relevanter), Produktpreise (Aufwertung von Produkten innerhalb eines Bereiches) und Datumsberechnungen ebenfalls als Faktoren bei der Sortierung der Ergebnislisten berücksichtigen zu können.

Komponentenintegration Die Nutzung einer eigenen Komponente innerhalb des *RequestHandlers* wurde für die Einbindung von extern gebildeten Empfehlungen verwendet. Die Integration in die *solrconfig.xml* geschieht wie in Listing 3 gezeigt. Alle vom Empfehlungsdienst gefundenen Dokumente werden so mit Hilfe der Komponente höher gewichtet. Des Sequenzdiagramm in Abbildung 3 zeigt den Ablauf bzw. die Verarbeitung einer Suchanfrage und das Zusammenspiel der zusätzlichen Komponente. Da die Komponente nur die Anfragen eines Nutzers erweitert und in den Dokumentendaten selbst keine Veränderung vorgenommen werden muss, kann diese Komponente leicht integriert werden. Problematisch ist, dass die Personalisierung nicht durchgeführt werden kann, wenn die Menge der empfohlenen Dokumente und die Menge der vom Suchindex gefundenen Dokumente disjunkt sind.

Integrierte-Empfehlungsberechnung Integriert man die mittels SVD berechneten Features in den gespeicherten Dokumente, kann man die Berechnung der kollaborativ gebildeten Empfehlungen mit den in Solr vorhandenen Berechnungsmöglichkeiten umsetzen (vgl. Abschnitt 4.4.5.) Voraussetzung dafür ist, dass die Featurevektoren typgerecht als Vektor im Index gespeichert werden. Dies kann durch eine entsprechende Konfiguration

Listing 4: Anpassungen der schema.xml für 4-dimensionalen Featurevektor

```
<schema name="core" version="1.1">
        <types>
                <fieldType name="featureVector" class="solr.PointType" dimension="4" subFieldSuffix="_f"/>
        </types>
        <fields>
                <field name="features" type="featureVector" />
        </fields>
</schema>
```

Listing 5: Solr-Anfrage um Vektor-Distanz einzubeziehen

```
?qq = Theater Hamburg Freitag
&qf=title^10 tag^3 stadt^3 beschreibung
&b = div(1, dist(2,$v, features))^5
&v = vector(0.57,-1.01,0.66,0.11)
&q = {!boost b=$b defType=dismax v=$qq}
```

innerhalb der *schema.xml* wie in Listing 4 sichergestellt werden.

Einbezogen werden die so definierten Featurevektoren über eine explizit definierte *Boosting*-Funktion. Beispielhaft wird dies für die Anwendungsfälle der personalisierten Suche (Listing 5) und zur Auflistung von Komplementen (Listing 6) gezeigt. Für die personalisierte Suche werte die *Boosting*-Funktion (b) Dokumente auf, wenn deren Featurevektoren (features) einen geringeren euklidischen Abstand zum Nutzervektor (v) haben. Im gezeigten Beispiel ist der Nutzervektor (v) explizit angegeben, in realen Anwendungsfällen würde dieser innerhalb einer ergänzenden Solr-Komponente gebildet.

Zur Bildung von Komplementen wird ebenfalls die Distanz zwischen Featurevektoren gemessen um die Relevanz der Dokumente zu bestimmen. Hierbei wird allerdings ausschließlich die distanzbasierte Relevanz genutzt und auf die Integration von Tf-idf-basierte Werten verzichtet. Ebenfalls wird der Nutzervektor durch den Featurevektor eines oder mehrerer Dokumente — im Beispiel das Dokument 49040 — ersetzt, um zum entsprechenden Dokument die Komplementäre zu finden.

Listing 6: Solr-Anfrage zur featurebasierten Dokumentenähnlichkeit

```
?q={!boost b=div(1,sqedist($features,features)) v=id:*}
&fq={!frange l=1 u=1}if(termfreq(id,$id),0,1)
&similarityField=features
&id=49040
```

4.2 Apache Mahout

Apache Mahout (Mahout) ist eine Programmbibliothek, die vorwiegend Werkzeuge des *maschinellen Lernens* bzw. der *kollektiven Intelligenz* zur Verfügung stellt. Die drei Schwerpunkte der darin implementierten Algorithmen umfassen das *Clustern* von Daten, die Daten-*Klassifikation* und das *kollaborative Filtern* (vgl. Abschnitt 2.2). Die Skalierbarkeit der entsprechenden Algorithmen steht dabei im Zentrum des Projektes, weshalb ein Großteil der Implementierungen auch in Form von Apache Hadoop kompatiblen MapReduce-Implementierungen vorliegt (vgl. Abschnitt 2.5.3). Apache Mahout wird ebenso wie Solr frei unter der Apache 2.0 Lizenz zur Verfügung gestellt. [Owen u. a., 2011]

Da für die vorliegende Arbeit ausschließlich die Werkzeuge des kollaborativen Filterns genutzt wurden, werden im weiteren Abschnitt nur die dafür notwendigen Aspekte von Apache Mahout betrachtet. Umfassendere Beschreibungen aller Bestandteile und deren praktischer Anwendung werden in "Mahout in Action" [Owen u. a., 2011] beschrieben.

4.2.1 Bestandteile

Teilausschnitt Komponentendiagramm einfügen

Mit Hilfe von Apache Mahout werden drei Bestandteile der in Abbildung 2 gezeigten bzw. in Abschnitt 3.3 beschriebenen Systemarchitektur realisiert.

wenn dieDatenaufbereiten mit rein kommt dann vier

Modellberechnung Die Basisdistribution[15] von Mahout ermöglicht es, verschiedene Modelltypen zu generieren. Diese umfassen die nutzer-bzw. elementbasierten Modelle wie sie in Abschnitt 2.3.2 beschrieben werden und die merkmalbasierten Modelle aus Abschnitt 2.3. Die Eingabedaten zur Modellbildung werden im CSV-Format abgelegt und bestehen aus Tripeln die jeweils einem Nutzer für ein Element einen Präferenzwert zuordnet. Die Verarbeitung geschieht wie in Listing 7 und Listing 8 beispielhaft gezeigt, mit Hilfe von lokale ausgeführten Java-Jobs oder mit Hilfe von in Apache Hadoop verteilten MapReduce Jobs.

[15] Bezogen auf die Version 0.7 der offiziellen Apache Mahoutdistribution

Listing 7: Generierung eines elementbasierten Ähnlichkeitsmodelles mit Apache Mahout

```
bin/mahout itemsimilarity \
    --input inputFolder \
    --output outputFolder \
    --booleanData false \
    --similarityClassname SIMILARITY_EUCLIDEAN_DISTANCE \
    --tempDir outputFolder/tmp
```

Listing 8: Modellberechnung wie in 7 innerhalb eines Apache Hadoop Systems

```
hadoop jar mahout-core-0.7-job.jar org.apache.mahout.driver.MahoutDriver \
    itemsimilarity \
    --input inputFolder \
    --output outputFolder \
    --booleanData false \
    --similarityClassname SIMILARITY_EUCLIDEAN_DISTANCE \
    --tempDir outputFolder/tmp
```

Die Ergebnisse der Berechnung werden ebenfalls im CSV-Format abgelegt. Bei den Nachbarschaftsmodellen enthalten sie jeweils Tripel bestehend aus zwei Elementen und deren Ähnlichkeit welche mit Hilfe des im Aufruf gewählten Ähnlichkeitsmaßes bestimmt wird. Die Wahl eines geeigneten Ähnlichkeitsmaßes muss, wie in Abschnitt 2.3.1 beschrieben, vorher getroffen werden bzw. durch entsprechende Tests evaluiert werden. Innerhalb von Apache Mahout stehen neben anderen auch die in Abschnitt 2.3.1 erläuterten Maße zur Verfügung.

Generiert man merkmalbasierte Modelle im Vorverarbeitungsschritt, werden diese ebenfalls in textheller Form abgelegt. Hierbei wird jedoch Zeilenweise jeweils ein Element und der entsprechende Vektor gespeichert.

Modellevaluation Zur qualitativen Bewertung der generierten Modelle bietet Apache Mahout gut integrierte Testwerkzeuge. Als Testmaß (vgl. Abschnitt 2.6) für die Modelle wird vorwiegend der mittlere quadratische Fehler genutzt. Um diese bestimmen zu können, ist es notwendig, dass das Modell nur mit einem Teil der Daten trainiert wird und danach gegen den zweiten Teil evaluiert wird. Da bei großen Datenmengen selbst die zufällige Zuteilung der Daten in eine der beiden Gruppen zum Problem wird, bietet Apache Mahout auch dafür einen in MapReduce implementierten Verarbeitungsschritt (vgl. Listing 9).

Wurde das Modell entsprechend trainiert, kann für die verbliebenen Testdaten der tatsächliche mit dem "berechneten" Wert verglichen werden um so entsprechend den mittle-

Listing 9: Trennung von Test- und Trainingsdaten mit Apache Mahout

```
bin/mahout splitDataset \
    --input inputFolder \
    --output outputFolder \
    --trainingPercentage 0.9 \
    --probePercentage 0.1 \
    --tempDir outputFolder/tmp
```

Listing 10: Evaluation eines trainierten Modelles

```
bin/mahout com.aoe.cf.recommender.SimilarityEvaluator \
    --input inputFolder \
    --model modelFolder \
    --output outputFolder \
    --tempDir outputFolder/tmp
```

ren quadratischen Fehler abzuleiten (vgl. Abschnitt 2.6.1). Da die Implementierung bzw. Parametrisierung des eigentlichen Recommendersystems zu verschieden ist, gibt es keinen einheitlichen Weg die Modelle zu evaluieren. Innerhalb der Klassenbibliothek von Apache Mahout existieren allerdings alle Bausteine um einen entsprechenden "Evaluation" zuverlässig und nach den Prinzipien der verteilten Datenverarbeitung zu erstellen. Für die im Rahmen der Diplomarbeit genutzten Modelle kann die Evaluation wie in Listing 10 erfolgen. Die Ausgabe entspricht dann dem RMSE des entsprechenden Modells. [Owen u. a., 2011]

use the actual code

Empfehlungsdienste Die Nutzung der generierten Modelle und die Berechnung von Empfehlungen für Nutzer kann innerhalb von Apache Mahout ebenfalls über entsprechende Kommandozeilenaufrufe erfolgen (vgl. Listing 11). Da im gegebenen Anwendungsfall vorgerechnete Empfehlungen allerdings wenig Nutzen bieten und die dafür notwendige Vorlaufzeit nicht den Erwartungen des Nutzers entspricht, wurden die innerhalb der Klassenbibliotheken vorhandenen Konzepte kombiniert und in Form eines Webservices mit den angebundenen Systemen integriert. Die vorgerechneten und geprüften Modelle werden darin mit Hilfe der sog. *GenericItemSimilarity* geladen und innerhalb eines *GenericItemBasedRecommender* zur Generierung der Empfehlungen genutzt. Neben den reinen element- bzw. nutzerbasierten Empfehlungen unterstützt Apache Mahout auch die Berechnung von Empfehlungen die Abweichungen der mittleren Bewertung pro Element und Nutzer in

Listing 11: Berechnung von Empfehlungen basierend auf Elementähnlichkeitsmodellen

```
bin/mahout recommenditembased \
    --input inputFolder \
    --output outputFolder \
    --numRecommendations 5 \
    --maxRating 5.0
```

Listing 12: Durchführung der Matrixfaktorierung

```
$JAVA_HOME/bin/java -classpath core.jar \
   com.aoe.cf.recommender.Evaluator \
   -Dcf.home=inputFolder
```

Betracht ziehen (siehe zB. *BiasedItemBasedRecommender*). Empfehlungen für anonyme Nutzer zur Minimierung des *Cold-Start* Problems (vgl. Abschnitt 2.4) werden ebenfalls in den integrierten Datenmodellen unterstützt.

Hinweis auf konkrete Implementierung einbauen

Matrixfaktorisierung Neben den zuvor beschriebenen Berechnungswegen implementiert Mahout auch SVD-basierte Empfehlungsdienste. Bei der Berechnung wird zwischen der inkrementiellen Berechnung basierend auf [Funk, 2006] und mittels *Alternating Least Squares* [Bell u. Koren, 2007; **?**] erstellten Modellen unterschieden (vgl Abschnitt 2.3.3).

Ergänzend zu den in Mahout enthaltenen Methoden, wurde innerhalb der Arbeit die vorhandene inkrementelle Modellberechnung (siehe *RatingSGDFactorizer*) so erweitert, dass diese auch ausschließlich elementspezifischen Featurevektoren verarbeitet (siehe *NSVD-Factorizer*, vgl. Abschnitt 4.4.5). Da innerhalb der Mahout-Distribution vorwiegend binäre Austauschformate genutzt werden, wurde die Persistenzschicht der vorhandenen Modelle zudem um einen CSV-Backend erweitert um die spätere Indexierung in Apache Solr zu vereinfachen. Die Ausführung der Matrixfaktorisierung erfolg damit innerhalb einer eigenen Klasse wie in Listing 12.

4.2.2 Erweiterungen

Da Apache Mahout generell nur Basisalgorithmen implementiert, ist es notwendig gewesen diese für bestimmte Anwendungsfälle zu erweitern. Alle Erweiterungen wurden innerhalb

eines Java-Paketes zusammengefasst und können gemeinsam mit der Basisdistribution von Mahout genutzt werden. Die Erweiterungen der Mahout-Komponenten sind hierbei innerhalb des *Core* Moduls im *com.aoe.cf.recommender* Namespace versammelt.

Pakete "verlinken"

Die Erweiterungen umfassen:

- Implementierung der Empfehlungsdienste für anonyme Nutzer (*PlusAnonymousRecommender*)
- Implementierung der NSVD-Faktorisierung zur Erzeugung von ausschließlich elementspezifischen Featurevektoren (*NSVDFactorizer*)
- Integration eines *MapReduce*-Jobs zur Transformation der Ähnlichkeitsmatrix (vgl. Listing 7) in paarweise Ähnlichkeitswerte (*PairSimilarityJob*)
- Integration eines CSV-Persistenzbackend für die Matrixfaktorisierung (*CsvPersistenceStrategy*)
- Implementierung von Evaluationsklassen für beide Methoden (*Evaluation*, *FactorEvaluation*)

4.3 Integration

Zur Integration der Empfehlungsdienste in Apache Solr wurden zwei Komponenten entwickelt welche im Folgenden beschrieben werden. Beide können wie in Listing 3 eingebunden werden.

4.3.1 Externe Empfehlungsdienste

Zur Anbindung von des ausgelagerten Empfehlungsdienstes über eine HTTP-Schnittstelle wurde die *BooleanBoost*-Komponente implementiert.

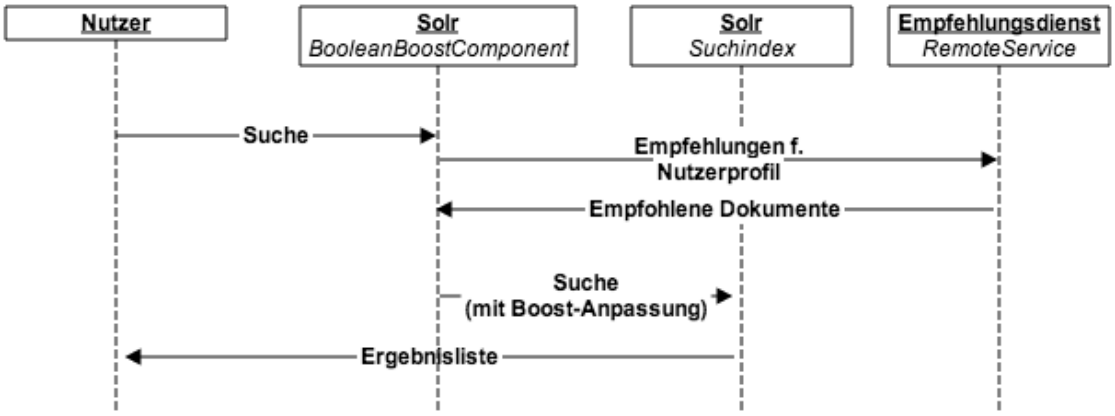


Abbildung 3: Flussdiagramm: Personalisierung von Suchanfragen über einen externen Empfehlungsdienst.

Wie in 3 dargestellt, wird die ursprüngliche Suchanfrage des Nutzers innerhalb der Komponente zunächst unterbrochen, um für das entsprechende Nutzerprofil die Empfehlungen vom externen Dienst berechnen zu lassen. Enthält die Antwort des Dienstes eine Liste von Empfehlungen, so wird diese innerhalb der Komponente in eine Solr-Anfrage, in diesem Fall einen *BooleanQuery*, umgewandelt. Innerhalb des *BooleanQuery* wird über *TermQuery*-Elemente jeweils die Empfehlung für ein einzelnes Dokument der entsprechenden Solr-DokumentenID zugeordnet. Die so generierte Liste wird mit der ursprünglichen Anfrage verknüpft und entsprechend weiterverarbeitet. Die Kombination beider Anfragen hat dann zur Folge, dass Dokumente aus den Empfehlungen über die Faktoren des *TermQuery* entsprechend "besser" bewertet werden wenn sie in der Ergebnisliste der ursprünglichen Suchanfrage enthalten sind. Sind die empfohlenen Dokumente nicht Teil der Ergebnisliste, kann keine Personalisierung vorgenommen werden und die Gewichtung bzw. Sortierung der Ergebnisse erfolgt rein TF-IDF-basiert.

Zur Konfiguration der Komponente stehen folgende Parameter zur Verfügung:

- *boostBase* - Normalisierungswert bei der Dokumentenboost-Berechnung. (α aus Formel (30)).
- *boostAmount* - Menge der zu bildenden Empfehlungen

- *boostInline* - Schaltet die Integration in die ursprüngliche Anfrage ab und zeigt nur die Empfehlungen.

Die Verarbeitung kann hierbei komplett in der Vorverarbeitungsphase von Apache Solr erfolgen, so dass die eigentlichen Suchanfragen auch von innerhalb von Solr in Zwischenspeichern gehalten werden können und wiederholte Anfragen erbeblich schneller verarbeitet werden können.

4.3.2 Integrierte Empfehlungsdienste

Die Implementierung von Empfehlungen mit Hilfe von Element-Featurevekoren, wurde innerhalb der *SimilarityBoost*-Komponente implementiert. Die Verarbeitung einer Suchanfrage wird in Abbildung 4 dargestellt.

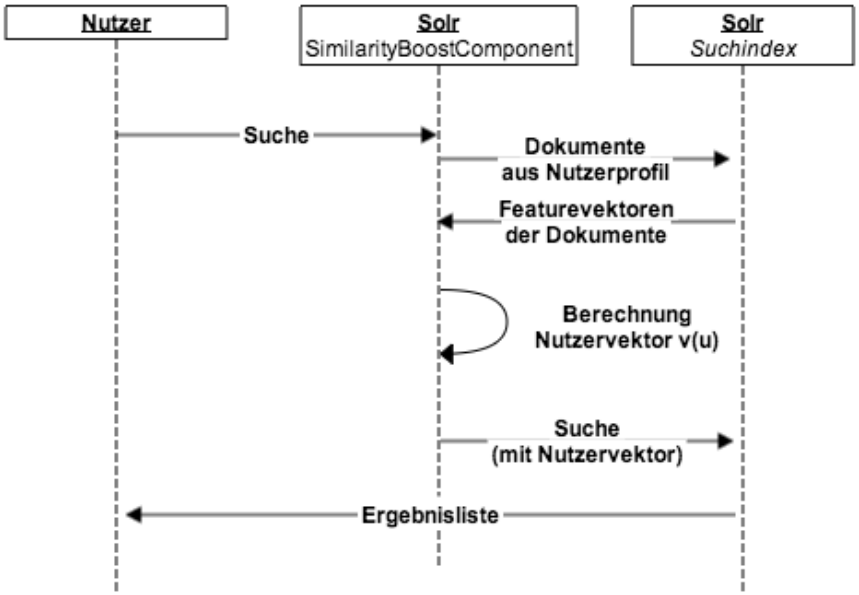


Abbildung 4: Flussdiagramm: Personalisierung von Suchanfragen über Featurevektoren im Suchindex.

Nachdem die Komponente die Anfrage des Nutzers erhalten hat, muss zunächst über eine zusätzliche Anfrage, die Liste der Dokumente des Nutzerprofils aus dem Suchindex gelesen werden. Mit Hilfe dieser Liste kann dann, der Nutzervektor $v(u)$ wie in Formel 34 gebildet werden. Danach kann die ursprüngliche Anfrage weiterverarbeitet werden.

Da die Konfiguration der Komponente (vgl. Listing 6) die weitere Verarbeitung von $v(u)$ (im Listing als *$features* bezeichnet) bestimmt, kann die Personalisierung bzw. Empfehlungsbildung unabhängig von der *SimilarityField*-Komponente erfolgen. Das in Listing 6 gezeigte Beispiel nutzt die Distanz zwischen den Dokumentenvektoren $w(i)$ und dem Nutzervektor $v(u)$ um für den Nutzer relevante Dokumente zu bevorzugen.

Da der größte Teil der Empfehlungsbildung innerhalb der normalen Apache Solr Komponenten erfolgt, beschränkt sich die Konfiguration der *SimilarityField*-Komponente auf die Benennung des Featurevektoren-Feldes über den *similarityField* Parameter. Alle weiteren Verarbeitungsschritte und Parameter werden innerhalb der *solrconfig.xml* direkt angegeben.

4.4 Systemaufbau

Für die in Abbildung 2 skizzierten Systembestandteile wird im Folgenden die gewählte Technologie erläutert und mit möglichen Alternativen verglichen.

4.4.1 Tracker

Die Aufzeichnung der Interaktionen von Nutzer und Webseite, wurde mit Hilfe eines eigenen Java Servlets umgesetzt. Die Funktionalität des Trackers beschränkt sich auf das Erfassen von Interaktionen eines Nutzers innerhalb einer JavaScript-Bibliothek und die Weitergabe dieser innerhalb des Servlets. Da die Daten dem Anwendungsfall entsprechend gespeichert werden können, ist der Aufwand bei der weiteren Verarbeitung sehr gering.

Da ein "Datensatz" einer Komponente der in Tabelle 1 gezeigten Matrix entspricht, enthält dieser auch ähnliche Daten. Für den Nutzer wird jeweils eine ID für die aktuelle Session, eine sessionübergreifende ID und (falls bekannt) den Account im System aufgezeichnet. Für das Element bzw. Produkt wird die SKU-ID aufgezeichnet und ggf. eine Produktgruppen-ID. Die Information über die Art der Interaktion wird in textheller Form gespeichert. In der Summe besteht ein einzelner Datensatz so aus ca. 114-150 Byte. Die Weitergabe dieser

Daten erfolg über die im Abschnitt 4.4.3 beschriebene Transporttechniken. Der Quellcode der Komponente ist unter verfügbar.

Reg zum Tracker-Repo

Wichtigste Alternative zur Implementierung eines eigenen Dienstes wäre die Nutzung bzw. Auswertung schon vorhandener Daten aus Serverlogs und von externen Trackingdiensten (zB. Google Analytics) gewesen. Da dies allerdings mit sehr hohen Aufwand bei der Datenaufbereitung verbunden wäre und da nicht in jedem Anwendungsfall auf entsprechende Logs zurückgegriffen werden kann, wurde auf diese Art der Datengewinnung zunächst verzichtet.

4.4.2 Datenhaltung

Für die dauerhafte Speicherung der aufgezeichneten Daten wurde zunächst auf die Verwendung eines verteilten Dateisystems (vgl. Abschnitt 2.5.1) verzichtet. Da der Betrieb eines entsprechenden Serverclusters hohe Wartungskosten erzeugt hätte, wurde die lokale Speicherung der Daten und die bedarfsgemäße Nutzung von Amazon Web Services gewählt. Dies ermöglicht die MapReduce-basierte Verarbeitung der Daten (vgl. Abschnitt 2.5.3) ohne dedizierte Hardware vorhalten zu müssen.

Der Kostenunterschied kann mit Hilfe der folgenden Rechnung veranschaulicht werden. Geht man von 50GB an aufgezeichneten Nutzdaten pro Jahr aus, erzeugt die Speicherung dieser Datenmenge zur Verarbeitung innerhalb des *Amazon Simple Storage Services* (S3) Kosten von ca. €6 pro Transfer (Upload+Download). Würde man zur weiteren Verarbeitung drei *Amazon Elastik Compute Cloud m2.xlarge* Instanzen für einen Tag nutzen, entstünden Kosten von ca. €100 pro Tag [16]. Die Kosten von wöchentlichen Neuberechnungen innerhalb dieser Infrastruktur entsprechen damit ca. der Hälfte der anfallenden Kosten zur Anmietung ähnlicher Ressourcen bei normalen Hosting-Anbietern.

[16] Als Vergleichswert wurde die unter http://calculator.s3.amazonaws.com/calc5.html am 01.08.2013 gezeigten Tagespreise genutzt

4.4.3 Datentransport

Zum sicheren Transport der Daten zwischen Tracker und der Datenhaltung wird *Apache Flume* genutzt. Es implementiert ein einfaches *Message Queue* System welches sicherstellt, dass alle Daten einer Quelle (Tracker) auch auf allen Senken (Datenhaltung) geschrieben wurden bevor diese das System verlassen. Im Vergleich zur direkten Speicherung der Daten auf einer entsprechenden Festplatte, bietet Apache Flume die Möglichkeit mit mehrere Quellen oder mehrere Senken zu arbeiten und Datenströme zu verteilen bzw. zusammenzufassen. Neben dem lokalen Dateisystem unterstützt Apache Flume zudem auch HDFS-Senken und Amazon S3-Senken. Dank dieser Eigenschaften wird sowohl die horizontale Skalierung des Trackers auf mehrere Knoten vereinfacht, als auch die Anbindung anderer, dem Anwendungsfall entsprechender, Datenhaltungssysteme.

Alternative Message Queue Systeme wie etwas Rabbit MQ oder Active MQ wurden nicht betrachtet.

4.4.4 Datenaufbereitung

Die Vorverarbeitung der Daten wurde mit *Apache Pig* umgesetzt. Es stellt eine dem Anwendungsfall angepasste Datenflusssprache, das sog. *Pig Latin*, zur Verfügung (Beispiel siehe Listing 13). Die darin beschriebenen Operationen werden zur Verarbeitung in *Apache Hadoop* kompatible *MapReduce* Operationen kompiliert und sind so ohne weitere Anpassungen verteilt ausführbar. Damit ermöglichht es, die Anwendung der in Abschnitt 2.5.3 beschriebenen Methoden um Skalierbarkeit zu gewährleisten. Die in *Pig Latin* beschriebenen Datenverarbeitungsprogramme lassen sich über *Amazon Elastic MapReduce* auch für Daten nutzen welche in *Amazon S3* gehalten werden. [Lin u. Kolcz, 2012]

Eine umfassende Dokumentation der Möglichkeiten von *Apache Pig* zur Datenverarbeitung bietet [Gates, 2011]. Die Integration von Aufgaben des maschinellen Lernens mit *Apache Pig* wird in [Lin u. Kolcz, 2012] beschrieben.

Alternative *MapReduce* Implementierungen in Java oder *Apache Hive* wurden nicht betrachtet.

Listing 13: Apache Pig Beispielscript zur Kombination von Nutzerprofiledaten

```
logs = LOAD '$source' USING PigStorage(';') AS (u: chararray, us: chararray);
knownusers = FILTER logs BY u IS NOT NULL;
knownusers_d = DISTINCT knownusers;

sessions = JOIN logs BY (us) left outer, knownusers BY (us);
userlist = FOREACH sessions GENERATE (knownusers::u is null ? logs::us : knownusers::u) AS u, logs::us AS us;
userlist_d = DISTINCT userlist;

STORE userlist_d INTO '$target';
```

4.4.5 Empfehlungsdienst

Wie im vorangegangen Abschnitt 4.2 beschrieben, wurde die Integration der Empfehlungs- und Modellberechnung mit *Apache Mahout* vorgenommen.

Die Vorberechnung der Ähnlichkeitsmodelle (vgl. Abschnitt 2.5.4) wird als verteilte Berechnung wie in Listing 7 ausgeführt. Um die darin berechneten Modelle direkt im Empfehlungsdienst nutzen zu können, wird die Ähnlichkeitsmatrix in einem Nachverarbeitungsschritt (vgl. *PairSimilarityJob*) in ein textuelles Austauschformat überführt. Dieses Austauschformat wird dann über das in Mahout vorhandene *FileDataModel* in den Empfehlungsdienst geladen. Dieser wird als Webservice über eine HTTP-Schnittstelle (vgl. *com.aoe.cf.recommender.RemoteService*) zur Verfügung gestellt. Die Matrixfaktorisierung wird wie in Listing 12 gezeigt, mit den in Abschnitt beschriebenen Methoden durchgeführt.

Die Evaluation der Ergebnisse beider Vorberechnungsmethoden erfolgt über die ergänzend bereitgestellten Evaluationsprogramme. In beiden Fälle werden die Empfehlungsergebnisse für zuvor gespeicherten Testdaten (vgl. Listing 9) mit den tatsächlich bekannten Daten vergleichen um die in Abschnitt 2.6 beschriebenen Metriken zu generieren.

4.4.6 Suche

Die Integration mit der Suche erfolgt wie in Abschnitt 4.1.3 beschrieben. Die Empfehlungsdienste auf Basis des Webservices werden wie in Listing 3 über die *booleanBoost*-Komponente eingebunden und konfiguriert. Zur Integration der featurebasierten Empfehlungen wird die *similarityField* Komponente eingebunden und über die erweiterte Boost-

Konfiguration wie in Listing 6 zur Personalisierung benutzt.

Zur Erfüllung der in Abschnitt 3.3 beschriebenen Anwendungsfälle können beide Komponenten auch so integriert bzw. konfiguriert werden, dass Empfehlungen direkt "ausgegeben" werden oder dass Empfehlungen nicht für reale Nutzerprofile und stattdessen für anonyme Produktlisten generiert werden. Da beide Komponenten direkt in den Verarbeitungsablauf von *Apache Solr* integriert sind, obliegt die Ausgestaltung der konkreten Anwendungsfälle der konkreten Applikation und wird zur Wahrung des Umfangs nicht näher betrachtet.

5 Evaluation

Wie wird gemessen, welche Ergebnisse waren zu erwarten, was wurde erreicht. Warum gibt es Abweichungen, welche Probleme enthält die Messmethode. raus

Im folgenden Abschnitt wird die implementierte Lösung hinsichtlich der Leistungs- und Qualitätsanforderungen (vgl. Abschnitt 3.1) evaluiert. Da zum Zeitpunkt der Fertigstellung der Arbeit noch kein ausreichend umfangreicher realer Datensatz zur Verfügung stand, wurde der frei verfügbare MovieLens 1M [Shyong Lam, 2006] Datensatz genutzt. Dieser besteht aus ca. 1 Million Nutzerbewertungen von 3.900 Nutzern für 6040 Kinofilme. Er wurde gewählt, da er bereits in zahlreichen Publikationen zur Ergebnisevaluation genutzt wird und so eine gute Vergleichbarkeit bei den Ergebnissen erzielt werden kann.

5.1 Ergebnisse

5.1.1 Leistung

- Performance Tracker (req/s) - (concurrent con)
- Performance Solr Personalisierung 1 single core (req/s) - (concurrent con)
- Performance Solr Personalisierung 1 multi core / single recommender (req/s) - (concurrent con)

- Performance Solr Personalisierung 1 multi core / multi recommender (req/s) - (concurrent con)
- Performance Solr Personalisierung 2 single core (req/s) - (concurrent con)
- Performance Solr Personalisierung 2 multi core (req/s) - (concurrent con)

5.1.2 Qualität

- Qualität Personalisierung 1 - Metrik RMSE (Euclid, Pearson, Kosinus, Jaccard)
- Qualität Personalisierung 1 - Nachbarschaftsgröße (20,30,40,50)
- Qualität Personalisierung 2 - Metrik RMSE

5.2 Diskussion

Cremonesi10 - Abwägung RSME / Pression / Recall Messung bei Top-N Recommendern
Howe08 - Abwägung der versch. Distanzmaße zwischen Datensätzen

Skalierbarkeit evt. mit weiterem Datensatz "testen":
`http://aws.amazon.com/datasets/6468931156960467` -> (Subset: `http://labrosa.ee.columbia.edu/millionsong/tasteprofile`)

Signifikanz: `http://www.mitp.de/imperia/md/content/vmi/1634/1634_kapitel_20.pdf`
Joachim05 -> Wilcoxon test

6 Zusammenfassung

Abriss der Arbeit, was wurde erreicht bzw. gelernt. An welcher Stellen kann weitergearbeitet werden.

Annahme das sich die Präferenzen der Nutzer über die Zeit nicht änder ist ggf. falsch <-> Temporaler Kontext aus Boughareb11

Ggf. “Lesezeit” als Maß hinterfragen (siehe Joachims05 - Abschnitt 2)

Auswirkungen auf die Verkaufsdiversität Fleder09

Ausblick “Similarity Search in High Dimensions via Hashing” (LSH) als mögliche Erweiterung Ausblick “Content boosted matrix factorization” with in Forbes11

Glossar

AJAX ist ein Akronym for “Asynchronous JavaScript and XML”. Es wird zur Datenübertragung zwischen Browser und Server genutzt. Die Übertragung kann auch sowohl synchron als auch asynchron erfolgen und die ausgetauschten Daten sind in der Regel per XML oder JSON serialisiert.

ID bezeichnet eine Nummer die genutzt wird um Entitäten eindeutige zu referenzieren.

JSON ist ein Akronym für “JavaScript Object Notation”. Es ist ein für Menschen und Maschinen lesbares Dateiformat zum Datenaustausch und soll aus gültigen JavaScript bestehen.

Recommender sind automatisierte Empfehlungstechnologien. Recommender filtern Informationen auf der Basis statistischer Daten, um damit Elemente (Produkte) aus einer Menge von Alternativen zu bestimmen, die entweder zum Nutzer oder zu einem anderen Element passen (vgl. Abschnitt 2.2).

REST ist ein Akronym für “Representational State Transfer”. Es beschreibt die Nutzung von HTTP-basierten zustandslosen Diensten.

Servlet beschreibt eine Java - Technologie um dynamische Webinhalte zu erstellen. Servlets laufen auf einem Webserver, um auf HTTP Anfragen dynamische Antworten zu erstellen.

SKU-ID ist ein Akronym für “Stock keeping unit”. Es beschreibt eine ID, die die eindeutige Zuordnung zu einer bestimmten Bestandseinheit (Stückgut) und somit auch ihre Wiedererkennung ermöglicht.

Tracking beschreibt das Erstellen eines Protokolls über das Nutzerverhalten auf einer Webseite.

Webservice bezeichnet eine Technik zur Anwendungskopplung in heterogenen Systemen über standardisierte Protokolle. Der Datenaustausch erfolgt in der Regel über HTTP und die ausgetauschten Daten werden üblicherweise mittels XML oder JSON serialisiert.

Abkürzungsverzeichnis

CF	Collaborative Filtering
GFS	Google Filesystem
IDF	Inverse Dokumentenfrequenz
IR	Information Retrieval
MAE	Mean Average Error
RMSE	Root Mean Squared Error
SVD	Singular Value Decomposition
TF	Termfrequenz

Abbildungsverzeichnis

Literatur

[Amatriain u. a. 2009] Amatriain, X. ; Jaimes, A. ; Oliver, N. ; Pujol, J. M.: Data Mining Methods for Recommender Systems. In: Kantor (Hrsg.) ; Ricci (Hrsg.) ; Rokach (Hrsg.) ; Shapira (Hrsg.): *Recommender Systems Handbook*, Springer, August 2009 14

[Bell u. Koren 2007] Bell, Robert M. ; Koren, Yehuda: Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In: *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*. Washington, DC, USA : IEEE Computer Society, 2007 (ICDM '07). – ISBN 0–7695–3018–4, S. 43–52. – 10.1109/ICDM.2007.90 20, 54

[Bogers u. van den Bosch 2009] Bogers, T. ; Bosch, A. van d.: Collaborative and Content-based Filtering for Item Recommendation on Social Bookmarking Websites. In: *Proceedings of the ACM RecSys'09 Workshop on Recommender Systems & the Social Web*. New-York, NY, USA, 2009, S. 9–16 15

[Boughareb u. Farah 2011] Boughareb, Djalila ; Farah, Nadir: Toward a Web Search Personalization Approach Based on Temporal Context. In: Cherifi, Hocine (Hrsg.) ; Zain, JasniMohamad (Hrsg.) ; El-Qawasmeh, Eyas (Hrsg.): *Digital Information and Communication Technology and Its Applications* Bd. 166. Springer Berlin Heidelberg, 2011. – ISBN 978–3–642–21983–2, S. 33–44. – 10.1007/978-3-642-21984-9_4 6, 37, 38

[Burke 2002] Burke, Robin: Hybrid Recommender Systems: Survey and Experiments. In: *User Modeling and User-Adapted Interaction* 12 (2002), November, Nr. 4, S. 331–370. – ISSN 0924–1868. – 10.1023/A:1021240730564 7, 9, 10, 13, 22

[Burke u. a. 2011] Burke, Robin ; O'Mahony, Michael P. ; Hurley, Neil J.: Robust Collaborative Recommendation. In: Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook*. Springer US, 2011. – ISBN 978–0–387–85820–3, S. 805–835. – 10.1007/978-0-387-85820-3_25 23

[Chu u. a. 2006] Chu, Cheng T. ; Kim, Sang K. ; Lin, Yi A. ; Yu, Yuanyuan ; Bradski, Gary R. ; Ng, Andrew Y. ; Olukotun, Kunle: Map-Reduce for Machine Learning on Multicore. In: Schölkopf, Bernhard (Hrsg.) ; Platt, John C. (Hrsg.) ; Hoffman, Thomas (Hrsg.): *NIPS*, MIT Press, 2006, S. 281–288 27, 28

[Claypool u. a. 1999] Claypool, Mark ; Gokhale, Anuja ; Miranda, Tim ; Murnikov, Pavel ; Netes, Dmitry ; Sartin, Matthew: *Combining Content-Based and Collaborative Filters in an Online Newspaper*. 1999. – 10.1.1.145.9794 21, 22

[Cornelis u. a. 2007] Cornelis, Chris ; Lu, Jie ; Guo, Xuetao ; Zhang, Guanquang: One-and-only item recommendation with fuzzy logic techniques. In: *Information Sciences* 177 (2007), Nr. 22, S. 4906 – 4921. – ISSN 0020–0255. – 10.1016/j.ins.2007.07.001 28

[Cremonesi u. a. 2010] Cremonesi, Paolo ; Koren, Yehuda ; Turrin, Roberto: Performance of recommender algorithms on top-n recommendation tasks. In: *Proceedings of the fourth ACM conference on Recommender systems*. New York, NY, USA : ACM, 2010 (RecSys '10). – ISBN 978–1–60558–906–0, S. 39–46. – 10.1145/1864708.1864721 23

[Das u. a. 2007] Das, Abhinandan S. ; Datar, Mayur ; Garg, Ashutosh ; Rajaram, Shyam: Google news personalization: scalable online collaborative filtering. In: *Proceedings of the 16th international conference on World Wide Web*. New York, NY, USA : ACM, 2007 (WWW '07). – ISBN 978–1–59593–654–7, S. 271–280. – 10.1145/1242572.1242610 39, 40

[Dean u. Ghemawat 2004] Dean, Jeffrey ; Ghemawat, Sanjay: MapReduce: simplified data processing on large clusters. In: *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*. Berkeley, CA, USA : USENIX Association, 2004 (OSDI'04), S. 10–10 27

[Desrosiers u. Karypis 2011] Desrosiers, Christian ; Karypis, George: A Comprehensive Survey of Neighborhood-based Recommendation Methods. In: Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook*. Springer US, 2011. – ISBN 978–0–387–85820–3, S. 107–144. – 10.1007/978-0-387-85820-3_4 16, 17, 19

[Durao u. a. 2012] Durao, Frederico ; Lage, Ricardo ; Dolog, Peter ; Coskun, Nilay: A Multi-factor Tag-Based Personalized Search. In: Filipe, Joaquim (Hrsg.) ; Cordeiro, José (Hrsg.): *Web Information Systems and Technologies* Bd. 101. Springer Berlin Heidelberg, 2012. – ISBN 978–3–642–28081–8, S. 192–206. – 10.1007/978-3-642-28082-5_14 6

[Funk 2006] Funk, Simon: *Netflix update: Try this at home, 2006.* `http://sifter.org/~simon/journal/20061211.html`. Version: Dezember 2006. – zuletzt abgerufen am 27.09.2012 20, 54

[Gates 2011] Gates, A.: *Programming Pig.* O'Reilly Media, 2011. – ISBN 9781449317683 60

[Ghemawat u. a. 2003] Ghemawat, Sanjay ; Gobioff, Howard ; Leung, Shun-Tak: The Google file system. In: *SIGOPS Oper. Syst. Rev.* 37 (2003), Oktober, Nr. 5, S. 29–43. – ISSN 0163–5980. – 10.1145/1165389.945450 24, 25, 26

[Golub u. Kahan 1965] Golub, G. ; Kahan, W.: Calculating the Singular Values and Pseudo-Inverse of a Matrix. (1965) 19

[Haveliwala u. a. 2003] Haveliwala, Taher ; Kamvar, Sepandar ; Jeh, Glen: An Analytical Comparison of Approaches to Personalizing PageRank / Stanford InfoLab. Stanford, June 2003 (2003-35). – Technical Report 7

[Herlocker u. a. 2002] Herlocker, Jon ; Konstan, Joseph A. ; Riedl, John: An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms. In: *Inf. Retr.* 5 (2002), Oktober, Nr. 4, S. 287–310. – ISSN 1386–4564. – 10.1023/A:1020443909834 16, 17, 18

[Herlocker u. a. 1999] Herlocker, Jonathan L. ; Konstan, Joseph A. ; Borchers, Al ; Riedl, John: An algorithmic framework for performing collaborative filtering. In: *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval.* New York, NY, USA : ACM, 1999 (SIGIR '99). – ISBN 1–58113–096–1, S. 230–237. – 10.1145/312624.312682 18

[Huete u. a. 2012] Huete, Juan F. ; Fernández-Luna, J. M. ; Campos, Luis M. ; Rueda-Morales, Miguel A.: Using past-prediction accuracy in recommender systems. In: *Inf. Sci.* 199 (2012), September, S. 78–92. – ISSN 0020–0255. – 10.1016/j.ins.2012.02.033 16, 17

[Jannach u. a. 2010] Jannach, D. ; Zanker, M. ; Felfernig, A. ; Friedrich, G.: *Recommender Systems: An Introduction.* Cambridge University Press, 2010. – ISBN 9780521493369 7, 8, 11, 12, 14, 15, 22, 30

[Jiang u. a. 2011] Jiang, Jing ; Lu, Jie ; Zhang, Guangquan ; Long, Guodong: Scaling-Up Item-Based Collaborative Filtering Recommendation Algorithm Based on Hadoop. In: *Services, IEEE Congress on* 0 (2011), S. 490–497. ISBN 978–0–7695–4461–8 27, 28

[Kearns 1998] Kearns, Michael: Efficient noise-tolerant learning from statistical queries. In: *J. ACM* 45 (1998), November, Nr. 6, S. 983–1006. – ISSN 0004–5411. – 10.1145/293347.293351 27

[Koren u. Bell 2011] Koren, Yehuda ; Bell, Robert: Advances in Collaborative Filtering. In: Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook.* Springer US, 2011. – ISBN 978–0–387–85820–3, S. 145–186. – 10.1007/978-0-387-85820-3_5 20, 21

[Koren u. a. 2009] Koren, Yehuda ; Bell, Robert ; Volinsky, Chris: Matrix Factorization Techniques for Recommender Systems. In: *Computer* 42 (2009), August, Nr. 8, S. 30–37. – ISSN 0018–9162. – 10.1109/MC.2009.263 19, 20, 22, 44

[Krüger 2011] Krüger, J.D.: *Conversion Boosting mit Website Testing.* mitp/bhv, 2011 (mitp Business). – ISBN 9783826690792 31

[Langford u. a. 2009] Langford, J. ; Smola, A. ; Zinkevich, M.: Slow Learners are Fast. (2009), November 20

[Lin u. Kolcz 2012] Lin, Jimmy ; Kolcz, Alek: Large-scale machine learning at twitter. In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data.* New York, NY, USA : ACM, 2012 (SIGMOD '12). – ISBN 978–1–4503–1247–9, S. 793–804. – 10.1145/2213836.2213958 39, 60

[Linden u. a. 2003] Linden, G. ; Smith, B. ; York, J.: Amazon.com recommendations: item-to-item collaborative filtering. In: *Internet Computing, IEEE* 7 (2003), Nr. 1, S. 76–80 17, 19, 28, 38, 40

[Lops u. a. 2011] Lops, Pasquale ; Gemmis, Marco ; Semeraro, Giovanni: Content-based Recommender Systems: State of the Art and Trends. In: Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook*. Springer US, 2011. – ISBN 978–0–387–85820–3, S. 73–105. – 10.1007/978-0-387-85820-3_3 11

[Machanavajjhala u. a. 2011] Machanavajjhala, Ashwin ; Korolova, Aleksandra ; Sarma, Atish D.: Personalized social recommendations: accurate or private. In: *Proceedings of the VLDB Endowment* 4 (2011), April, Nr. 7, S. 440–450. – ISSN 2150–8097 9

[Manning u. a. 2008] Manning, Christopher D. ; Raghavan, Prabhakar ; Schtze, Hinrich: *Introduction to Information Retrieval*. New York, NY, USA : Cambridge University Press, 2008. – ISBN 0521865719, 9780521865715 2, 3, 4, 5

[Michael u. a. 2007] Michael, Maged M. ; Moreira, José E. ; Shiloach, Doron ; Wisniewski, Robert W.: Scale-up x Scale-out: A Case Study using Nutch/Lucene. In: *IPDPS*, IEEE, 2007, S. 1–8 26

[Muir u. a. 2011] Muir, Robert u. a.: *Class TFIDFSimilarity*. `http://lucene.apache.org/core/4_3_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`. Version: Mai 2011. – zuletzt abgerufen am 08.06.2013 47, 48

[Owen u. a. 2011] Owen, Sean ; Anil, Robin ; Dunning, Ted ; Friedman, Ellen: *Mahout in Action*. 1. Manning Publications, 2011. – ISBN 1935182684 28, 51, 53

[Page u. a. 1998] Page, Lawrence ; Brin, Sergey ; Motwani, Rajeev ; Winograd, Terry: The PageRank Citation Ranking: Bringing Order to the Web / Stanford Digital Library Technologies Project. 1998. – Forschungsbericht 5, 7

[Paterek 2007] Paterek, Arkadiusz: Improving regularized singular value decomposition for collaborative filtering. In: *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, 2007, S. 39–42 44

[Ricci u. a. 2010] Ricci, F. ; Rokach, L. ; Shapira, B. ; Kantor, P.B.: *Recommender Systems Handbook*. Springer, 2010. – ISBN 9780387858197 7, 9, 13

[Sadekar 2012] Sadekar, Kedar: *Scalable Logging and Tracking*. `http://techblog.netflix.com/2012/06/scalable-logging-and-tracking.html`. Version: Dezember 2012. – zuletzt abgerufen am 21.12.2012 39

[Segaran 2007] Segaran, Toby: *Programming collective intelligence*. First. O'Reilly, 2007. – ISBN 9780596529321 14, 15

[Shani u. Gunawardana 2011] Shani, Guy ; Gunawardana, Asela: Evaluating Recommendation Systems. In: Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook*. Springer US, 2011. – ISBN 978–0–387–85820–3, S. 257–297. – 10.1007/978-0-387-85820-3_8 29, 30

[Shyong Lam 2006] Shyong Lam, Jon H.: *MovieLens 1M data set*. `http://www.grouplens.org/node/73`. Version: Dezember 2006. – zuletzt abgerufen am 26.08.2013 62

[Sinha u. Swearingen 2001] Sinha, Rashmi R. ; Swearingen, Kirsten: Comparing Recommendations Made by Online Systems and Friends. In: *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001 9

[Smyth u. a. 2005] Smyth, Barry ; Balfe, Evelyn ; Freyne, Jill ; Briggs, Peter ; Coyle, Maurice ; Boydell, Oisin: Exploiting Query Repetition and Regularity in an Adaptive Community-Based Web Search Engine. In: *User Modeling and User-Adapted Interaction* 14 (2005), Januar, Nr. 5, S. 383–423. – ISSN 0924–1868. – 10.1007/s11257-004-5270-4 6, 7, 9, 31, 32

[Steck 2010] Steck, Harald: Training and testing of recommender systems on data missing not at random. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2010 (KDD '10). – ISBN 978–1–4503–0055–1, S. 713–722. – 10.1145/1835804.1835895 21

[Tintarev u. Masthoff 2011] Tintarev, Nava ; Masthoff, Judith: Designing and Evaluating Explanations for Recommender Systems. In: Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook*. Springer US, 2011. – ISBN 978–0–387–85820–3, S. 479–510. – 10.1007/978-0-387-85820-3_15 18

[Töscher u. a. 2008] Töscher, Andreas ; Jahrer, Michael ; Legenstein, Robert: Improved neighborhood-based algorithms for large-scale recommender systems. In: *Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*. New York, NY, USA : ACM, 2008 (NETFLIX '08). – ISBN 978–1–60558–265–8, S. 4:1–4:6. – 10.1145/1722149.1722153 21

[Victor u. a. 2011] Victor, Patricia ; Cock, Martine ; Cornelis, Chris: Trust and Recommendations. In: Ricci, Francesco (Hrsg.) ; Rokach, Lior (Hrsg.) ; Shapira, Bracha (Hrsg.) ; Kantor, Paul B. (Hrsg.): *Recommender Systems Handbook*. Springer US, 2011. – ISBN 978–0–387–85820–3, S. 645–675. – 10.1007/978-0-387-85820-3_20 9, 11, 13

[Vozalis u. Margaritis 2007] Vozalis, M. G. ; Margaritis, K. G.: Using SVD and demographic data for the enhancement of generalized Collaborative Filtering. In: *Inf. Sci.* 177 (2007), August, Nr. 15, S. 3017–3037. – ISSN 0020–0255. – 10.1016/j.ins.2007.02.036 10, 21

[Yin u. a. 2012] Yin, Hongzhi ; Cui, Bin ; Li, Jing ; Yao, Junjie ; Chen, Chen: Challenging the long tail recommendation. In: *Proc. VLDB Endow.* 5 (2012), Mai, Nr. 9, S. 896–907. – ISSN 2150–8097 23