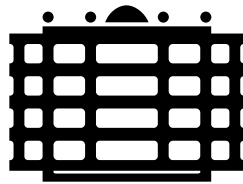




Technische Universität Chemnitz
Fakultät für Informatik
Professur Künstliche Intelligenz



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 19. November 2012

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

1	Einleitung	1
2	Grundlagen	2
2.1	Suchindexe	2
2.1.1	Indexbildung	3
2.1.2	Relevanzberechnung	4
2.1.3	Personalisierung	6
2.2	Recommendation Konzepte	7
2.2.1	Kollaboratives Filtern	8
2.2.2	Gruppen-basierte Empfehlungen	9
2.2.3	Demographisch gestützte Empfehlungen	10
2.2.4	Inhaltsbasierte Empfehlungen	10
2.2.5	Wissensbasierte Empfehlungen	11
2.2.6	Utility-basierte Empfehlungen	12
2.3	Filtermodelle	13
2.3.1	Ähnlichkeitsmaße	13
2.3.2	Nachbarschaftsmodelle	16
2.3.3	Matrixfaktorisierung	19
2.4	Schwierigkeiten von Recommendern	21
2.5	Skalierungsstrategien	23
2.5.1	Skalierung der Datenhaltung	24
2.5.2	Horizontalen Fragmentierung	26
2.5.3	MapReduce basierte Algorithmen	27
2.5.4	Skalierung der Filtermodelle	28
2.6	Qualitätsmaße	29
2.6.1	Mittlere Abweichung	29
2.6.2	Trefferquote und Genauigkeit	29
2.6.3	Empirische Messung	31
3	Entwurf	32
3.1	Anforderungen	32
3.1.1	Anwendungsfälle	32
3.1.2	Leistungsanforderungen	33
3.2	Systemarchitektur	33
3.3	Datenerhebung	33
4	Realisation	34
4.1	Apache Mahout	37
4.1.1	Bestandteile	37
4.1.2	Datenaufbereitung	37
4.1.3	Skalierung	37
4.2	Apache Solr	37
4.2.1	Indexierung	37
4.2.2	Score-Anpassung	37

4.3 Searchperience Integration	38
5 Evaluation	38
5.1 Ergebnisse	44
5.2 Diskussion	44
6 Zusammenfassung	44
Abkürzungsverzeichnis	48
Abbildungsverzeichnis	48
Literatur	49

*Liste der noch zu erledigenden Punkte


■ MAE dazu?	15
■ Quelle dass “Normalisierung” auch bei item-item relevant ist	17
■ Ergebnisse der Rechnung einfügen	17
■ Weiterführend: Regression vs. Classification aus HB04 evt. ergänzen	19
■ ggf. SlopeOne ergänzen	19
■ PureSVD erläutern?	19
■ Pseudocode aus Louppe10 ?	20
■ Temporale Effekte dazu ?	20
■ Weitere Ansätze ergänzen? zB. Graphen-Modelle? Boltzmann-Dings?	21
■ Formen von Attacken oder Maßnahmen könnten ergänzt werden	23
■ ggf. Langford09 Parallelisierungsstrategien dazu	23
■ Ggf. MapRed. Beispiel einfügen.	27
■ BSP	27

1 Einleitung


Hinweis dass “Elemente” und “Item” oft als Synonym verwendet werden, ähnlich wie auch “Rating” und “Bewertung” gleichzusetzen sind.

Goldberg92 -> Ursprung des Ausdrucks

2 Grundlagen

 In folgenden Abschnitten 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 Google¹ 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 -talen 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 das Wort “Fantasy” explizit darin vorkommt. [Manning u. a., 2008]

¹[!\[\]\(9409513254e1c623c5edc59502b38e15_img.jpg\)](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

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] [C

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-Formates² jede Zeile einer Datei als einzelnes Dokument gesehen werden. Auch die Wahl einer geeigneten Methode zur Wortstammbildung (engl. Stemming) und das Filtern 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 verschiedene Erweiterungen des invertierten Index, um noch effizienter in sehr

²siehe RFC 4155 - <http://tools.ietf.org/rfc/rfc4155.txt>

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 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.

$$\text{tf}(i, d) = \frac{q(i, d)}{\max_{z \in Z}(\text{freq}(z, d))} \quad (1)$$

$$\text{idf}(i) = \log \frac{N}{n(i)} \quad (2)$$

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


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 ($\text{freq}(i, j)$) zur maximalen Anzahl aller anderen Terme Z im Dokument. Um Wörtern, die nur in wenigen Dokumenten vorkommen, zusätzliches Gewicht zu geben, bzw. um solche, die in nahezu jedem Dokument vorkommen, abzuwerten, wird die *Termfrequenz* zudem mit Hilfe der *inversen Dokumentenfrequenz* gewichtet. Diese 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]

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

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. Dokumenten  auf die häufig verwiesen wird, besitzen demnach einen hohen PageRank und Verweise von populären Dokumenten üben einen größeren Effekt auf den PageRank der verknüpften Dokumente aus.

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

Berechnet wird der PageRank R einer Seite u mit Hilfe der Formel (5). Der Faktor $c < 1$ dient dabei zur ktion 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])

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

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, 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 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 “feature erte” PageRank-Werte pro Dokument berechnet. Während der Anfrage werden diese vorberechneten Werte entsprechend des Nutzerprofils gewichtet. [Haveliwala u. a., 2003]

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* (, 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  in 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  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 Elementen, 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 community-basierten 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 themenspezifische Suchmaschine, nur von einer bestimmten Nutzergruppe (zB. Forscher) genutzt wird, sind die kol-

laborativ 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 entsprechend 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 Inhaltbasierte Empfehlungen




Bei der inhaltsbasierten 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 und so kann es 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  einer der Vorteile. Wichtiger noch ist die Tatsache, dass in sbasierte 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  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 schw  ute 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 Frequen  z mit der Nutzer ein Element brauchen oder konsumier  eh 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 Wis  selches 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. Die Information darüber das keine Empfehlungen für eine Anfrage gefunden werden konnten, werden Nutzer schneller akzeptieren wenn das System zudem eine Reihe von Vorschlägen unterbreiten kann welche der Regeln ausgelassen werden könnten um neue Empfehlungen zu generieren. 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.

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 des Vertrauens in die generierten Empfehlungen führen. [Ricci u. a., 2010, Kap. 1] [Burke, 2002; Victor u. a., 2011]

	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

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 dann 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.

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

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

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]

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

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]

$$sim(a, b) = \frac{\sum_{p \in P} (r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P} (r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P} (r_{b,p} - \bar{r}_b)^2}} \quad (10)$$

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 Rich-

tung zeigen.

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

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-Koeffizient vom *Tanimoto*- und vom *Dice-Koeffizienten* (vgl. [Bogers u. van den Bosch, 2009]). [Jannach u. a., 2010, Kap. 3.1] [Segaran, 2007]

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

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 Pearson-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:

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

Wie [Herlocker u. a., 2002] zeigt, 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 \bar{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]

$$pred(u, i) = \bar{r}_u + \frac{\sum_{v \in \mathcal{N}_i(U)} sim(u, v) * (r_{v,i} - \bar{r}_v)}{\sum_{v \in \mathcal{N}_i(U)} sim(u, v)} \quad (14)$$

$$pred(u, i) = \bar{r}_u + \sigma_u \frac{\sum_{v \in \mathcal{N}_i(U)} sim(u, v) * \frac{r_{v,i} - \bar{r}_v}{\sigma_v}}{\sum_{v \in \mathcal{N}_i(U)} sim(u, v)} \quad (15)$$

	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 Ähnlichkeit bzw. Nachbarschaften der Elemente $\mathcal{N}_u(I)$ zu anderen kann dann, analog zu Formel (13), wie folgt zur Voraussage der Bewertungen genutzt werden:

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

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).

Für die Abwägung zwischen nutzer- und elementbasierten Methoden gibt [Desrosiers u. Karypis, 2011] die folgenden (zusammengefassten) 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 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 das 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.

Quelle
dass
"Nor-
malisie-
rung"
auch bei
item-
item re-
levant
ist

Ergebnisse
der
Rech-
nung
einfügen

- *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 Das zum Vergleich mit den im Folgenden beschriebenen 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

2.3.3 Matrixfaktorisierung

Merkmal-basierte Empfehlungen Bei den Methoden der Matrixfaktorisierung 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]).

ggf. SlopeOne ergänzen

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:

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

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) durch die Minimierung der mittleren quadratischen Abweichung zwischen existierenden und vorausberechneten Bewertungen:

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

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 können zwei 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. Das Training wird entsprechend der Anzahl der zu ermittelnden Merkmale f -mal, jeweils bis zur Konvergenz, durchgeführt. [Funk, 2006; Langford u. a., 2009; Koren u. Bell, 2011]

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

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

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

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

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

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 großen Datenmengen aber durch eine bessere Parallelisierbarkeit ausgeglichen. [Bell u. Koren, 2007; Koren u. Bell, 2011]

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. Mit zahlreichen Erweiterungen, die zum Beispiel temporale Effekte oder nicht-lineare Zusammenhänge in den Daten auszunutzen, konnten in verschiedenen Anwendungsfällen zusätzlich verbesserte Ergebnisse erzielt werden. [Koren u. Bell, 2011; Vozalis u. Margaritis, 2007]

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

2.4 Schwierigkeiten von Recommendern

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

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 MovieLens Datensatzes³ auf Anwendungsfälle mit einer Million Elementen, so wird schnell

³<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 Datensatz⁴ 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 dass 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 werden 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 Größen die nichtmehr 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

⁴<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 eher 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 große zusammenhängende Bereiche der Dateien nutzen. 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 das Flaschenhalse 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 das 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]

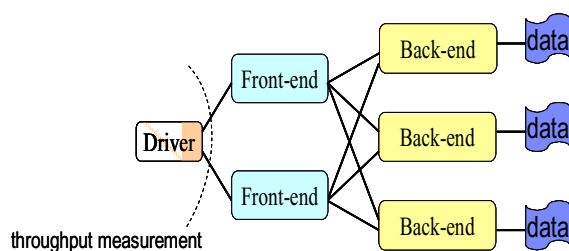


Abbildung 1: Abfragestruktur horizontal fragmentierter Systeme ⁵

Der Nachteil einer derartigen Verteilung ist, dass Transaktionen um atomare Operationen zu gewährleisten erheblichen Zusatzaufwand 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]

⁵Quelle: [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. Ma-
pRed.
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 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) ist eine möglicher Ausweg. 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} verglichen. Die beiden Fehlermaße werden dann wie folgt gebildet:

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

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

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 der eine 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	<u>2</u>
Item2	4	<u>1</u>	<u>2</u>
Item3	2	2	<u>3</u>
Item4	3	3	<u>2</u>
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:

$$P_u = \frac{|hits_u|}{|recset_u|} \quad (26)$$

$$R_u = \frac{|hits_u|}{|testset_u|} \quad (27)$$

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.

$$F_1 = \frac{2 * P * R}{P + R} \quad (28)$$

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 eine Werbeanzeige, 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 Conversions *conv* und der Anzahl der Besucher *visitors* die im gegebenen Zeitraum eine oder mehrere Seiten aufgerufen haben. Maschinelle Seitenzugriffe, zum Beispiel Suchmaschinen-Spider, fließen nicht in Messung der Nutzerzahlen ein. Ebenso werden wiederholte Conversions des gleichen Besuchers nicht mehrfach gezählt. [Krüger, 2011]

$$CR = \frac{|conv|}{|visitors|} * 100 \quad (29)$$

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

3 Entwurf

Aufbauend auf den im vorangegangenen Abschnitt beschriebenen Grundlagen wurde im Rahmen der Diplomarbeit eine Beispielapplikation implementiert. Die an diese Applikation gestellten Anforderungen, der konzeptionelle Aufbau und die Maßnahmen zur Datenerhebung werden im Folgenden beschrieben.

3.1 Anforderungen

3.1.1 Anwendungsfälle

Potentielle Use-Cases:

A) Personalisierte Suche: Wenn man nach allgemeinen Begriffen wie "GeschenkWein" oder "Kleidstück", bekommt man bereits durch andere Nutzer gelernte passende Empfehlungen in der Suche höher angezeigt. (Nutzer schaut sich Rotweine an und bekommt bei Suche nach Wein passende Rotweine angezeigt. Im Vergleich dazu ein Nutzer der sich Weißweine angeschaut hat)

B) Personalisiertes-Recommendation Widget: - In der rechten Spalte werden (passend zu den letzten besuchten Items oder durchgeführten Suchen) persönliche Empfehlungen gegeben

C) Context-Recommendation Widget: 1) Alla "Nutzer die x gekauft haben haben auch y gekauft" oder simple "Find most similar items to an item" Hier macht die Kombination von Suchindex und Recommendation auch Sinn weil: - Der Suchindex alle zur Ausgabe relevanten Daten eines Dokumentes hat - (Vorbereitung?) - Es einfach möglich ist die Items über die die Recommendation gemacht werden sollen über inhaltliche (regelbasierte) Suchqueries weiter einzuschränken oder zu boosten (e.g. Items die höheren Preis haben und in gleicher Kategorie sind höher boosten)

2) Crossselling Widget im Warenkorb (Empfehlungen zu den Items im Warenkorb)

3.1.2 Leistungsanforderungen

Traffic / Nutzerzahlen - hochgerechnete Bandbreiten / Memorybedarf / Speicherbedarf

3.2 Systemarchitektur

-Wenn man von großen Mengen (großer Raum) von Items ausgeht stellt sich das Problem, das die Teilmenge der Items die der Recommender und die Suche zu einer Anfrage zurück geben können disjunkt sein oder nur eine unerhebliche Schnittmenge haben. Teilprobleme wären - "Boosting in der Suchmenge": Recommender bekommt Solr-Menge und macht nur darüber Recommendations (wenn überhaupt möglich - vielleicht kommt man ja in die erste Schleife der user-based algorithmen "that u has no preference for yet") - "Boosting in der Recommender Menge": Recommender bestimmt Menge (OR query + Optionales query) und Solr boosted nur noch darin. (Use-Case widget) - "Selective Recommendation": Recommender wählt eine zum Query passende Datenmodell / Datengrundlage aus, die möglichst viele potentielle Schnittmengen mit dem Query hat - Hier könnte man Jobs haben die regelmäßig die User-Item Datensätze nur für Items lernen die von der Suche zu einem bestimmtem Query zurückkommen (e.g. die Top-Querys) - "Anreicherungsansatz" Der Recommender könnte zu Ergebnissen in der Suchmenge weitere Ergebnisse einstreuen (vgl "More like these" handler). - "Precomputation and Delegation to Solr" Da finde ich auch Interessant. Man kann sicherlich durch vorclustern ähnlicher Items oder ähnliche Items pro Usercluster alles direkt in einer Abfrage abwickeln (Durch anreichern der Daten im Index)

- Recommender empfehlen nur neue Items, man will aber ggf auch das bereits geklickte Items in der Suche höher gewichtet werden (so macht es Google ja auch) - Recommendation Algorithmen und Datengrundlage sind sehr verschieden. Wie kann man verschiedene Implementierungen nutzen/ansprechen/auswählen.

3.3 Datenerhebung

4 Realisation

Verfeinerung des Entwurfs, Schilderung bei der Umsetzung aus dem vorangegangenen Kapitel

4.1 Apache Mahout

4.1.1 Bestandteile

4.1.2 Datenaufbereitung

4.1.3 Skalierung

4.2 Apache Solr

4.2.1 Indexierung

4.2.2 Score-Anpassung

Listing 1: solrconfig.xml Anpassungen

```
<config>
  <lib dir="Search/target/dependency" regex=".*\.jar" />
  <lib path="Search/target/Search.jar" />
  <query>
    <searchComponent
      name="personalize"
      class="de.tolleiv.cf.search.SpringAwareSearchComponent">
        <str name="searchDelegate">booleanBoostRecommenderComponent</str>
      </searchComponent>
    </query>
  </config>
```

Listing 2: schema.xml Anpassungen

```
<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 3: Solr-Anfrage um Vektor-Distanz einzubeziehen

```
qq = *: *
b = div(1, dist(2,$v, features))
v = vector(0.57,-1.01,0.66,0.11)
q = {'!boost b=$b defType=dismax v=$qq}
```

4.3 Searchperience Integration

5 Evaluation

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

5.1 Ergebnisse

5.2 Diskussion

Cremonesi10 - Abwägung RSME / Precision / 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 dass sich die Präferenzen der Nutzer über die Zeit nicht ändert ist ggf. falsch
<-> Temporaler Kontext aus Boughareb11

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

Abkürzungsverzeichnis

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

Abbildungsverzeichnis

1	Horizontale Fragmentierung	26
---	--------------------------------------	----

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
- [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
- [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, 12, 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 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
- [Dean u. Ghemawat 2004] DEAN, Jeffrey ; GHEMAWAT, Sanjay: MapReduce: simplified data processing on large clusters. In: *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6*. Berkeley, CA, USA : USENIX Association, 2004 (OSDI'04), S. 10–10 27
- [Desrosiers u. Karypis 2011] DESROSIER, 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
- [Duraõ 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

- [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
- [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
- [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
- [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
- [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 ; SCHATZ, 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
- [Owen u. a. 2011] OWEN, Sean ; ANIL, Robin ; DUNNING, Ted ; FRIEDMAN, Ellen: *Mahout in Action*. 1. Manning Publications, 2011. – ISBN 1935182684 28
- [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. Version: 1998. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.1768>. 1998. – Forschungsbericht 5, 7
- [Ricci u. a. 2010] RICCI, F. ; ROKACH, L. ; SHAPIRA, B. ; KANTOR, P.B.: *Recommender Systems Handbook*. Springer, 2010. – ISBN 9780387858197 7, 9, 12
- [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
- [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, Oisín: 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
- [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, 10, 12
- [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