Kontextsensitive Empfehlungssyteme
Ein Graph-Basierter Ansatz

Context-Aware Recommender Systems
A Graph-Based Approach

Michael Thomas

Bachelorarbeit an der
Freien Universität Berlin,
Institut für Informatik

Matrikelnummer: *4282523*
Betreuer: *Prof. Dr. Adrian Paschke*

Berlin, Juli 2013

Zusammenfassung

In einem Zeitalter rasant wachsender Informationsmassen, wird es für Nutzer des digitalen Raums zunehmend schwieriger, relevante Informationen zu finden. Man ist dabei auf Werkzeuge angewiesen, die diese Suche vereinfachen oder gar erst ermöglichen. Neben gezielt einsetzbaren Suchmaschinen, spielen auch Empfehlungssysteme eine wichtige Rolle beim Auffinden von Wissen. Sie ermöglichen eine Einschränkung des Suchraums anhand bekannter Präferenzen des Nutzers, um ihm schneller relevante Informationen präsentieren zu können.
Meist arbeiten Empfehlungssysteme zweidimensional. Sie versuchen für einen Menge von Nutzern und Objekten eine Vorhersage für die Relevanz zu treffen. Beobachtungen haben gezeigt, dass die Relevanz für bestimmte Objekte nicht nur von dem Nutzer und dessen Bewertungen abhängt, sondern von weiteren Faktoren beeinflusst wird. Diese Faktoren lassen sich zusammenfassend als *Kontext* bezeichnen. Mögliche Beispiele für solche Kontextvariablen sind Zeitpunkt und Ort. Zur Empfehlung von Kinofilmen oder Restaurants, könnte darüber hinaus die Begleitung von Bedeutung sein. Bei Musik spielen möglicherweise die Stimmung, das Wetter und die zuletzt gehörten Musikstücke eine Rolle.
Im Rahmen dieser Arbeit sollen verschiedene Ansätze zur Implementierung von kontextsensitiven Empfehlungssystemen untersucht werden, mit speziellem Fokus auf die Entwicklung eines flexiblen, graphbasierten Empfehlungssystems für Musik.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Michael Thomas, 23. Juli 2013

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die heutige Gesellschaft befindet sich in einem Informationszeitalter. Ähnlich früherer Perioden der Menschheitsgeschichte, wie etwa die Stein- oder Bronzezeit, in der die Gewinnung und Verarbeitung dieser Rohstoffe die Gesellschaft und Kultur prägten, so sind es heute die immer rasanter wachsenden Massen an Informationen und digitalen Daten, die unsere Gesellschaft bestimmen. Informationen werden überall und zeitgleich produziert, sei es die Information über den Termin eines Konzertes, Nachrichten über die Wahl eines Präsidenten oder die Existenz eines Films. Durch die Möglichkeiten der modernen Kommunikationstechnik werden all diese Informationen für einen großen Teil der Menschheit zugänglich, zu jeder Zeit und an fast jedem Ort. Zugang zu Informationen erhält jedoch nur, wer effizientes Navigieren beherrscht, wer Daten suchen und finden kann. Die Menge an Informationen, die von alltäglicher Bedeutung sind, überschreiten die Kapazität unseres Gedächtnis bei weitem. Der Erfolg innerhalb der Gesellschaft hängt jedoch immer mehr von dem gekonnten Umgang im Auffinden und Verarbeiten von Wissen ab. So sind die Menschen auf Techniken angewiesen, die diesen Prozess vereinfachen. Neben den Suchmaschinen, welche das gezielte Suchen nach Informationen, etwa durch Schlüsselwörter ermöglichen, bilden Empfehlungssysteme ein weiteres wichtiges Instrument, um die Vielfalt an Wissen und Möglichkeiten zu durchdringen.
Im Gegensatz zu den aktiven Suchmaschinen, sind Empfehlungssysteme in der Regel passiver Art. Sie versuchen aus dem gesamten, komplexen Raum von Objekten oder Informationen, eine kleinere Menge von nützlichen Objekten herauszufiltern, meist unter der Berücksichtigung der persönlichen Präferenzen des Nutzers.
Empfehlungssysteme finden ihren Einsatz in einem breiten Spektrum von Anwendungen. Neben der klassischen Anwendung bei E-Commerce Plattformen wie *Amazon*[1], finden sie großen Zuspruch beim Empfehlen von Filmen (*MoviLens*[2]), Musik (*LastFM*[3]) und wissenschaftlichen Artikeln (*CiteULike*[4]). Auch ein Einsatz in *eLearning* Plattformen, etwa zum automatischen Vorschlagen von weiterführenden Lernmaterialien, ist denkbar.
Die Anforderungen sowie die erhältlichen Informationen variieren häufig zwischen verschiedenen Domänen. Die Menge der Daten, die in webbasierten und mobilen Applikationen produziert werden, enthalten zahlreiche latente Informationen die es zu nutzen gilt.
Die Leistung und Qualität von Empfehlungssystemen hängt an einer Vielzahl von Faktoren. Die Verwendung von inhaltlichem und relationalem Wissen über die Objekte kann eben so wichtig sein, wie die Modellierung des Benutzerprofils durch Analyse der Interaktion mit dem System auf lange und kurze Sicht.
Herkömmliche Empfehlungssysteme wie *MovieLens* oder *Netflix*[5] betrachten dabei lediglich zwei Dimensionen: Nutzer und Objekte. Basierend auf den Bewertungen für bekannte Objekte, werden Bewertungen für unbekannte Objekte vorhergesagt, wobei dem Nutzer die am besten bewerteten Objekte vorgeschlagen werden. Es hat sich jedoch gezeigt, dass es weit mehr Faktoren gibt, die über die Nützlichkeit eines Objektes für den Nutzer entschei-

[1] `amazon.com` (letzter Zugriff: 01.05.2013)
[2] `www.movielens.org` (letzter Zugriff: 01.05.2013)
[3] `last.fm` (letzter Zugriff: 01.05.2013)
[4] `www.citeulike.org` (letzter Zugriff: 01.05.2013)
[5] `netflix.com` (letzter Zugriff: 01.05.2013)

den. Womöglich möchte man mit seiner Freundin in einen anderen Film gehen, als mit seinen Freunden, hört am Montag morgen ruhigere Musik, als an einem Freitagabend. Für einen Studenten, der gerade für eine Mathematik Klausur lernt, sind Artikel über die Geschichte der französischen Revolution nicht sinnvoll, obwohl er sich sonst gerne mit diesem Thema befasst. Auch sind die Bewertungen eines Nutzers oft unterschiedlich zu interpretieren. Ein Nutzer könnte einen Film als gut bewerten, da ihm das Thema und die Umsetzung gefällt oder etwa weil Regisseur oder darstellende Schauspieler zu seinen Favoriten gehören.

1.2 Fragestellung

Es ist also offensichtlich, dass Entscheidungen und Präferenzen abhängig sind von dem *Kontext* in dem gehandelt wird. Dieser Kontext wird in klassischen Systemen nicht berücksichtigt, kann aber aber in vielen Fällen zu einer Verbesserung der Ergebnisse beitragen. Aus welchen Informationen sich der Kontext genau zusammensetzt und wieviel Einfluss Kontext und andere, semantische Informationen jeweils auf die Qualität der Empfehlungen haben, ist meist nicht genau bekannt und oft abhängig von der Domäne. Auch hängt die Anzahl und Art der zur Verfügung stehenden Metainformationen von der Anwendung ab.
Flexibilität ist somit eine wichtige Anforderung an moderne Empfehlungssysteme, gerade bei der immer stärker wachsenden Menge an Informationen und Informationsquellen. Im Rahmen dieser Arbeit soll der Frage nachgegangen werden, wie sich ein flexibles Empfehlungssystem entwickeln lässt, das die Nutzung verschiedener Kontextinformationen ermöglicht. Dazu sollen verschiedene Quellen von Informationen und ihr Einfluss auf die Qualität der Empfehlungen untersucht werden, um die Leistung des Systems zu verbessern.

1.3 Gliederung

Zunächst wird eine Einführung in den Bereich der klassischer Empfehlungssysteme gegeben. Die grundlegenden Ansätze, ihre Implementierung sowie verbreitete Probleme und Einschränkungen werden vorgestellt. Im zweiten Kapitel wird das Konzept des Kontextes eingeführt. Sowohl eine allgemeine Charakterisierung von Kontext, als auch seine Bedeutung im Bezug auf Empfehlungssysteme wird erklärt. Ferner werden verschiedene Paradigmen zur Aufnahme von Kontextinformationen in Empfehlungssytemen sowie mögliche Ansätze zur Implementierung dargelegt. Ein generisches, graphbasiertes Modell wird anschließend vorgestellt. Es vereint verschiedene Ansätze von Empfehlungssystemen.
Eine Realisierung dieses Modells erfolgt durch den konkreten Anwendungsfall eines Empfehlungssystems für Musik. Das letzte Kapitel dient der Implementierung. Anhand geeigneter Daten erfolgt schließlich eine Evaluierung, sowohl zum Vergleich der Leistung des graphbasierten Systems zu den klassischen Methoden, als auch zur Feststellung der Relevanz verschiedener Kontext- und Metainformationen.

2 Einführung in Empfehlungssysteme

In diesem Kapitel soll zunächst eine Übersicht zu Empfehlungssystemen gegeben werden. Wie lassen sich die System durch verschiedene Ansätze und Anforderungen unterscheiden? Welche typischen Probleme tauchen auf und wie lassen sie sich lösen?
Die Entwicklung von Empfehlungssystemen geht auf ganz unterschiedliche Bereiche der Forschung zurück, so etwa Kognitionswissenschaften und *Information Retrieval* [4]. Erst Anfang der 1990er Jahre ist daraus ein eigener Forschungsschwerpunkt entstanden, als sich zunehmend darauf konzentriert wurde, Empfehlungen durch explizite Bewertungen der Nutzer zu generieren.
Allgemein lassen sich traditionelle Empfehlungssysteme als das Problem beschreiben, eine Nützlichkeitsfunktion (*utility function*) zu definieren [4]. Sei I der Raum der Objekte (*items*) und U der Raum der Nutzer in dem System, dann wird eine Funktion u gesucht, welche für jeden Nutzer eine Nützlichkeit r für alle Objekte angibt $f_u : U \times I \rightarrow R$. R ist hierbei die geordnete Menge aller Nützlichkeitswerte r und besteht aus einem nicht negativen, reellen oder natürlichen Zahlenbereich, z. B. $R = [0, 1]$. In der Regel ist der Raum der Objekte und Benutzer sehr groß, typischerweise mehrere Hunderttausend bis Millionen. Aufgabe des Systems ist es nun, die Werte dieser Funktion zu schätzen und dem Nutzer die Menge derjenigen Objekte $s_R \in I$ vorzuschlagen, welche diese maximieren. Formal lässt sich dies wie folgt beschreiben:

$$s_R = argmax_{s \in I}(f_u(s, u))$$

In einer konkreten Anwendung ist es üblich eine Menge von N Objekten zu generieren. Das *recommendation set* $RS_N(u)$ sei definiert durch eine geordnete Menge der Länge N, bestehend aus den Objeken, die $f_u(s, u)$ maximieren.
Die Nützlichkeit kann eine beliebige Funktion sein, die beschreibt wie sinnvoll das Objekt für den Nutzer ist. In vielen Anwendungen wird sie durch eine explizite Bewertung von Objekten durch Nutzer beschrieben. Nutzer können in dem System Bewertungen abgeben, z. B. gibt Nutzer *U* dem Film *Matrix* die Bewertung *5* und dem Film *Fluch der Karibik* eine Bewertung von *1*. Nun kann – basierend auf den bekannten Bewertungen – eine Bewertung für unbekannte Filme prädiziert werden.
Wie und mit Hilfe welchen Informationen diese Bewertungsfunktion geschätzt wird, unterscheidet die verschiedenen Ansätze für Empfehlungssysteme.

2.1 Taxonomie der Empfehlungssysteme

Es wurde gezeigt, wie sich ein Empfehlungssystem formal beschreiben lässt. Nun sollen grundlegende Ansätze genauer betrachtet werden. Im Allgemeinen lassen sich diese durch die Beantwortung verschiedene Fragen klassifizieren. Die erste Frage ist, *welche* Informationen verwendet werden, um Bewertung vorherzusagen und *woher* dieses Wissen kommt. Auf der einen Seite kann diese Wissen durch den Nutzer selbst gewonnen werden, etwa durch Bewertungen und Analyse seiner Interaktion sowie Wissen *über* den Nutzer, etwa demographische Angaben, wie *Alter, Geschlecht* und *Wohnort*. Auf der anderen Seite kann Wissen über die Objekte selbst verwendet werden, also Informationen, die die Objekte charakterisieren. Eine weitere Möglichkeit besteht darin, assoziatives Wissen über die Relation zwischen Objekten auszunutzen, etwa durch Interferenz oder mit Hilfe von

Ontologien.
Eine grobe Einteilung nach genutzter Informationsquelle sieht wie folgt aus [11]:

- *Collaborative Filtering*: Dieser Ansatz basiert auf der Intuition, das Nutzer, die verschiedene Objekten ähnlich bewerten, über einen vergleichbaren Geschmack verfügen. Ähnliche Benutzer haben somit das Potential, nützliche Empfehlungen zu machen. Gegeben des aktiven Nutzers, versucht das System ihm ähnliche Nutzer zu finden. Es vergleicht die bekannten Bewertungen, um schließlich die Objekte auszugeben, die ein ähnlicher Benutzer als gut bewertet hat, ihm jedoch nicht bekannt sind.

- *Content-based Filtering*: Hier werden die charakteristischen Eigenschaften der Objekte selbst sowie die Bewertungen der Benutzer verwendet. Dem aktiven Nutzer werden Objekte vorgeschlagen, die ähnliche Eigenschaften zu jenen haben, die er als gut bewertet.

- *Semantic Filtering*: Bei diesem Ansatz, auch *knowledge-based filtering* genannt, werden Empfehlungen durch die Interferenz von domainspezifischen Wissen über die Nützlichkeit der Objekte und Nutzer generiert. Dieses Wissen erscheint meistens in Form von vordefinierten Ontologien.

Die Verwendung von demographischen Angaben über den Benutzer wird in der Literatur häufig als eine eigenständige Klasse betrachtet, die *demographic filtering* genannt wird [11]. In einer allgemeineren Betrachtung kann sie jedoch zu den *content-based* Methoden gezählt werdem, da es sich um weitere inhaltliche Merkmale handelt, die in das Benutzerprofil mit aufgenommen werden können.
Wird die Nützlichkeit der Objekte anhand von Bewertungen bestimmt, kann eine weitere Unterscheidung vorgenommen werden, nämlich danach *wie* diese Bewertungen ermittelt werden:

- *explizit*: Die Bewertung wird direkt durch den Nutzer vorgenommen, etwa durch ein ein 5-Sterne Bewertungssystem

- *implizit*: Die Bewertung wird aufgrund des Benutzerverhaltens geschätzt, etwa durch den Verlauf von Aktionen. In der Musik Domäne kann z. B. angenommen werden, dass Musikstücke, die der Nutzer öfter hört, einen höheren Wert für den Nutzer haben.

2.1.1 Content-based Filtering

Wie schon erwähnt, werden in der Methode des *content-based filtering* ähnliche Objekte zu den vom Nutzer bevorzugten, angezeigt. Objekte werden dazu anhand charakteristischer Features unterschieden. All diese Features und ihre Ausprägungen ergeben zusammen ein Objektprofil.
Soll zum Beispiel Musik empfohlen werden, kann sich das System etwa auf Genres und Interpret beziehen, um dem Nutzer unbekannte Musikstücke seiner beliebtesten Interpreten oder Genres zu empfehlen.
Weiter wird anhand des Verhaltens des Benutzers (z. B. durch Bewertungen oder gekaufte

Produkte) ein Präferenzprofil erzeugt, das verzeichnet, welche Vorlieben der Nutzer für bestimmte Objekteigenschaften hat. Ein solches Profil des Objektes besteht beispielsweise aus einer Menge von *keywords* sowie den zugehörigen Gewichten, welche die Ausprägung des betreffenden *keywords* im Bezug auf das Objekt angibt. Weiter wird ein Benutzerprofil dadurch erstellt, dass der Geschmack des Nutzers durch Interaktion mit dem System (z. B. Bewertungen) anhand seiner Präferenzen für bestimmte *keywords* ermittelt wird.
Adomavicius gibt eine formale Definition eines *content-based* Systems [4]:
Sei $Content(s) = (w_{s1}, w_{s2}, \dots w_{sk})$ das Objektprofil des Objektes $s \in I$, wobei w_{sj} die Relevanz des *keywords* k_j beschreibt und $ContentBasedProfile(u) = (w_{u1}, w_{u2}, \dots w_{uk})$ das Benutzerprofil des Nutzers $u \in U$ wobei w_{uj} die Wichtigkeit des *keywords* k_j für den Nutzer u darstellt, dann ist die Nützlichkeitsfunktion wie folgt definiert:

$$f_u(u, s) = score(ContentBasedProfile(u, Content(s))$$

Die *score* Funktion ist in der Regel eine beliebige Ähnlichkeitsfunktion für Vektoren, z.B. die *cosine similarity*.

Probleme, Beschränkungen Die Voraussetzung für die Berechnung der Ähnlichkeit von Objekten ist, dass sie charakteristische Attribute besitzen, die von einer Maschine verwertbar sind. Die Verfügbarkeit und Signifikanz dieser Informationen hängt stark von der Domäne ab. In manchen Fällen können diese Attribute durch Anwendung von Methoden aus dem Bereich *Information Retrieval* extrahiert werden. [4]. Dies ist relativ einfach für unstrukturierten Text [44] (z. B. Webseiten, Nachrichten), jedoch schwer für Multimediainhalte, wie Bilder, Filme und Musik. Lässt sich eine Repräsentation nicht algorithmisch extrahieren, gibt es die Möglichkeit auf bekannte Informationen zurückzugreifen, welche die Objekte beschreiben. Für Filme können diese Eigenschaften z. B. Schauspieler, Regisseure oder Genres sein. Eine weitere vielversprechende Möglichkeit ist der Einsatz von Folksonomien, die sich durch die steigende Beliebtheit von *social tagging* Webseiten in den letzten Jahren zu einem interessanten Instrument zur Klassifizierung von unstrukturierten Daten entwickelt haben [13].
Weiter ist es nicht trivial Eigenschaften zu ermitteln, die subjektiver Natur sind. Es könnten z. B. zwei Filme als ähnlich gelten, weil sie das selbe Genre haben, dennoch würden unterschiedliche Betrachter den Film verschieden bewerten.

Overspecialization Auch kann es passieren, dass es einen isolierten thematischen Bereich gibt, der vom Nutzer nie gesehen werden kann, solange er noch keine Bewertung in diesem Bereich abgegeben hat. In der Domäne der Gastronomie könnte vorkommen, dass ein Kunde Gerichte der Indischen Küche mag, jedoch noch nie eine Thailändische Spezialität gekostet hat, obwohl diese ihm womöglich zusagen würden. Dieses Problem wird in der Literatur auch *Overspecialization* genannt [4].

New User Problem Zur die Generierung von passenden Empfehlungen muss das System erst ausreichend Informationen über die Präferenzen des Benutzers gesammelt haben, daher ist es nicht möglich, neuen Benutzern gute Empfehlungen zu geben.

2.1.2 Collaborative Filtering

Eine weitere Klasse von Empfehlungssystemen stellen die kollaborativen Methoden dar. Im Gegensatz zum *content-based filtering*, wird hier kein explizites Wissen über die Objekte an sich gebraucht. Die Empfehlung erfolgt vielmehr durch andere Nutzer, die einen Geschmack haben. Dies hat den weiteren Vorteil, dass subjektive Verbindungen zwischen Objekten besser angesprochen werden können.
Einer der ersten, erfolgreichen Anwendungen basierend auf *collaborative filtering* waren Tapestry [19], Ringo [48] und GroupLens[6]. Ein populäres Beispiel aus heutige Zeit ist Amazon[7]. Das Prinzip beruht auf der Annahme, dass eine Person, die sich etwa ein gutes Buch empfehlen lassen will, wahrscheinlich einen Freund fragen würde, von dem sie weiß, dass er den selben Geschmack in Bezug auf Bücher hat. In einem Onlinesystem lässt sich dies automatisieren, in dem eine *user-item* Matrix erstellt wird, welche die bekannten Bewertungen der Benutzer enthält, beispielsweise die Bewertungen für die Filme, die er bereits gesehen hat. Jede Zeile dieser Matrix bildet nun eine numerische Repräsentation des Nutzers, die es ermöglicht ihn mit anderen Nutzern zu vergleichen. Mithilfe eines Ähnlichkeits- oder Korrelationsmaß wird eine Nachbarschaft zu dem aktiven Nutzer bestimmt, also all jene Nutzer, die mit seinen Präferenzen am meisten korrelieren. Daraufhin werden die am besten bewerteten Objekte dieser Nachbarn als Empfehlungen ausgegeben. Auch die umgekehrte Art ist möglich, so dass anstelle der Benutzerzeilen, die Objektspalten verwendet werden, um ähnliche Objekte anhand der Bewertungen die sie von verschiedenen Benutzern erhalten haben, zu finden. Es wurde gezeigt, dass diese *item-based* Methode in vielen Fällen besser, sowohl in Bezug auf die Qualität als auch der Performance funktioniert, als die *user-based* Methode [46]. Die prädiktiven Algorithmen, also solche, die anhand bekannter Bewertungen, unbekannte Bewertungen vorhersagen, lassen sich grob in zwei Kategorien teilen, nämlich *memory-based* und *model-based* [10].

memory-based Diese Methoden operieren direkt auf der gesamten *user-item* Matrix. Durch eine Korrelationsfunktion wird für einen Benutzer u eine Nachbarschaft der Größe N bestimmt. Anschließend wird durch die Bildung des Mittelwertes der Benutzervektoren die fehlenden Bewertungen für Benutzer c vorhergesagt. Eine häufiges Verfahren ist die *adjusted weighted sum* Methode:

$$p_{u,s} = \overline{r}_u + \alpha \sum_{u' \in neigh(u,N)} sim(u,u') \times (r_{u,s} - \overline{r}_{u'})$$

Wobei $\alpha = \frac{1}{\sum_{u' \in neigh(u,N)} sim(u,u')}$ einen Normalisierungskoeefizienten darstellt, $neigh(u,N)$ die Nachbarschaft der Größe N um den Benutzer u, $sim(u,v)$ Korrelation- oder Distanzfunktion, $r_{u,s}$ die echte Bewertung des Nutzers u auf das Objekt s und $\overline{r_u}$ jeweils die mittlere Bewertungen des Nutzers. Der Grund, warum nicht über die tatsächlichen Bewertung, sondern nur über die mittlere Differenz summiert wird, ist der Versuch mögliche unterschiedliche Bewertungsmaßstäbe verschiedener Benutzer zu korrigieren [4].
Eine geläufige Ähnlichkeitsmetriken ist die einfache *cosine similarity*, welche ein Maß für den Winkel zweier Vektoren im euklidischen Raum darstellt:

[6] `www.grouplens.org` (letzter Zugriff: 01.05.2013)
[7] `amazon.com` (letzter Zugriff: 01.05.2013)

$$cos(u,v) = \frac{u \cdot v}{\|u\| \cdot \|v\|} = \frac{\sum_{s \in I_{uv}} r_{u,s} \cdot r_{v,i}}{\sqrt{\sum_{s \in I_{uv}} r_{u,s}^2} \sqrt{\sum_{s \in I_{uv}} r_{v,s}^2}}$$

wobei I_{uv} die Menge der gemeinsam bewerteten Objekte von Benutzer u und v sei.[4]

model-based Für sehr große Datenmengen von Millionen von Objekten und Benutzern ist es impraktikabel direkt auf der *user-item* Matrix zu arbeiten, da es sehr speicher- und rechenaufwändig ist. Ein alternativer Ansatz für *collaborative filtering* ist daher, aus dem gesehen Daten ein prädiktives Modell zu erlernen, mit Hilfe dessen Vorhersagen über die Bewertungen getroffen werden können.
Dafür existieren verschiedene Ansätze, die Methoden aus dem Bereich des maschinellen Lernens verwenden und sowohl auf Skalierbarkeit, als auch auf die Verbesserung der Genauigkeit abzielen.
Ein Beispiel hierfür ist das *Naive Bayes Clustering*, das versucht, eine feste Anzahl von latente Klassen für Stereotypen von Benutzern zu schätzen [10].
Weitere Ansätze, die sich als sehr erfolgreich herausgestellt haben, sind *latent factor models*. Hierbei wird versucht, anhand der Daten eine feste Anzahl von Faktoren zu bestimmen, welche bestimmte Eigenschaften von Benutzern und Objekten charakterisieren. Eine populärer Algorithmus hierfür ist die *matrix factorization* [31].

Probleme, Beschränkungen

Cold-Start Wie bei die *content-based* Methoden, tritt auch in den *collaborative* Methoden das *cold-start* Problem auf. Ein neuer Benutzer im System muss erst ausreichend Bewertungen abgeben, um gute Empfehlungen zu erhalten.
Zusätzlich stellt auch das Hinzufügen von neuen Objekten ein Problem da. Da Empfehlungen nur basierend auf den Benutzerbewertungen generiert werden, jedoch keine Informationen über das Objekt selbst vorhanden sind, müssen erst ausreichend viele Bewertungen für das Objekt abgegeben werden, bis es empfohlen werden kann.

Sparsity Eine weitere Schwierigkeit besteht darin, dass Benutzer in der Regel nur eine sehr kleine Menge von Objekten bewerten. Benutzer, die einen ungewöhnlichen Geschmack haben, werden wenige ähnliche Benutzer finden und daher schlechtere Empfehlungen erhalten. Gleiches gilt auch für Objekte, die weniger Bewertungen erhalten haben und daher seltener empfohlen werden, als populäre Objekte.

2.1.3 Anforderungen und Schwierigkeiten

Beide der vorgestellten Systeme haben ihre Stärken und Schwächen. Alle Systeme sind anfällig für das *cold-start Problem* in in unterschiedlichen Ausprägungen. *Content-based* Methoden haben jedoch den Vorteil, dass dies nur für neue Benutzer zutrifft, nicht jedoch für neue Objekte. Mit der Verwendung von demograpischen Daten lässt sich dieses Problem teilweise beheben, da hier schon *a priori* Wissen über den Nutzer zur Verfügung steht.

Kollaborative Methoden haben dagegen die Stärke, subjektive Relationen zwischen Objekten besser zu erkennen, die nicht rational durch den Vergleich von Attributen erfassbar sind. Ihre Schwäche ist jedoch, dass eine hinreichende Menge Bewertungen für ein Objekt nötig ist, um dieses überhaupt erst empfehlen zu können. Da in realen Anwendungen die Mehrheit der Objekte nur wenige Bewertungen erhalten (*long tail*), besteht die Gefahr, dass diese durch das Empfehlungssystem nicht zugänglich sind [42].
Umgekehrt kann es passieren, dass ein System zu sehr auf dominante Vorlieben des Benutzers konditioniert ist und ihm daher keine neuartigen Empfehlungen mehr geben kann. In der Literatur wird dieses Phänomen als *stability vs. plasticity* Problem bezeichnet [11]. Es spiegelt das bekannte Problem des *overfitting* aus dem Bereich des maschinellen Lernens wider [49]. Es ist durchaus üblich, dass der Nutzer im Laufe der Zeit seine Präferenzen ändert, das System dies jedoch erst langsam erkennt. Die Kontroverse ergibt sich daraus, dass sich Kurzzeitinteressen (*short-term preferences*) mehr oder weniger stark von den Langzeitinteressen des Nutzers unterscheiden können, jedoch Langzeitinteressen auch nicht außer Acht gelassen werden dürfen. So hat ein Benutzer einer Nachrichtenplattform beispielsweise großes Interesse daran, die den Wahlkampf in den USA zu verfolgen. Da dieses Ereignis jedoch nur alle vier Jahre auftritt, könnte es sein, dass das System zur nächsten Wahl vergessen hat, dass der Benutzer dafür gerne detaillierte Nachrichten erhalten möchte.
In vielen Anwendungen ist es wichtig, dass nicht nur zu unterschiedliche Objekte aus der Liste der Empfehlungen heraus gefiltert werden, sondern auch Objekte, die *zu* ähnlich sind. Beispielsweise möchte der Benutzer der Nachrichtenplattform nicht alle Artikel, die zu dem gleichen Thema veröffentlicht wurden, erhalten. Dieses Problem wird auch als *diversity dilemma* bezeichnet. Eine Lösung mit Hilfe von graphbasierten Verfahrens ist in [51, 37] beschrieben. Auf die graphbasierten Verfahren soll später noch genauer eingegangen werden, sie bilden den Schwerpunkt dieser Arbeit.

2.2 Hybride Systeme

Da verschiedene Systeme unterschiedliche Stärken haben, die teilweise komplementär sind, liegt es auf der Hand, mehrere Verfahren zu verbinden, um den Schwächen jeweils entgegenzuwirken.
Es existieren viele verschiedene Möglichkeiten, die vorhandenen Verfahren zu kombinieren. Die Ergebnisse mehrere Methoden können zusammengefasst und unterschiedlich gewichtet werden (*Weighted*), oder es können unterschiedliche Methoden für verschiedene Anforderungen ausgewählt werden (*Switching*). Eine integratives Verfahren ist *Feature Combination*, bei welchem ein einzelner Algorithmus verwendet, Features jedoch aus unterschiedlichen Quellen gewählt werden. Auf dieser Idee bauen die später beschriebenen graphbasierten Methoden auf. Neben der Verbindung von *collaborative* und *content-based* Methoden, besteht auch die Möglichkeit des Einsatzes von *knowledge-based* Verfahren, die explizit semantischen Wissen über Objekte und Nutzer verwenden und somit dem *cold-start* Problem entgegen wirken können [4]. In einem Empfehlungssystem für Restaurants z. B. könnte in der Wissensdatenbank der Zusammenhang zwischen *veganer* und *vegatarischer* Küche vermerkt sein, somit würden dem vegetarischen Nutzer auch Restaurants mit veganer Küche vorgeschlagen. Ein Beispiel für ein solches System ist Entré [11].

Eine Schwierigkeit bei solchen wissensbasierten Systemen stellt jedoch der Zugang zu eben diesem Wissen dar. Das Wissen wird in der Regel durch maschinenlesbaren Ontologien repräsentiert, die oft manuell von einem Experten der Domäne konstruiert werden müssen [4].
Burke gibt in [11] eine umfangreiche Übersicht mit empirischen Vergleichen der möglichen Kombinationen zur Entwicklung hybrider Empfehlungssysteme.

2.3 Kritik an Empfehlungssystemen

Wie bei allen Personalisierungssystemen, besteht bei Empfehlungssystemen die Gefahr einer sogenannten „*filter bubble*“ [41]. Dem Nutzer wird nur eine sehr eingeschränkte Sicht auf alle möglichen Informationen gewährt – jene Informationen, die das System relevant für ihn hält [38]. Alle weiteren Informationen werden herausgefiltert. Das Bevorzugen einer Art von Informationen, könnte in der Zukunft zum Entfernen anderer Informationen führen. So würde einem Nutzer der Zugang zu wichtigen Nachrichten verwehrt bleiben, sofern das System glaubt, er habe kein Interesse daran. Das erwähnte Problem der Überanpassung (*overspecialization*), verstärkt dieses Phänomen. Folgt der Nutzer den präsentierten Informationen, verfestigen sich die gelernten Präferenzen (*plasticity problem* [11]).
Durch die zunehmende Abhängigkeit unserer Gesellschaft von Informationssystemen, kann dieses Problem weitreichende soziale Auswirkungen haben. Die eigentliche Aufgabe der Empfehlungssysteme – das Filtern relevanter Informationen – kann zur Gefahr werden. Entwickler von Empfehlungssystemen sollten sich dieser Problematik bewusst sein. Möglichkeiten, dem entgegenzuwirken, liegen sowohl im Design der Algorithmen, als auch der Benutzerschnittstelle. Wichtige Kriterien, welche diese Problem angehen, sind *diversity* [37, 51] und *serendepity* [38]. Letzteres beschreibt das Konzept, glückliche und unerwartete Entdeckungen zu machen – ein Phänomen, dass durch übermäßige Personalisierung verhindert wird. Ebenso ist die Gestaltung einer transparenten Benutzerschnittstelle wichtig. Dem Nutzer sollte ersichtlich sein, welche Informationen nach welchen Kriterien gefiltert werden. Ferner sollte er Einfluss auf diese haben können. [41] Die Flexibilität von Empfehlungssystemen ist eine wichtige Vorraussetzung dafür.
Ein System kann auch im negativen Sinne die Beeinflussung von Nutzern indendieren [25]. Zur Maximierung des Profits, könnten E-Commerce Plattformen den Nutzer dazu bewegen wollen, Produkte zu kaufen, die er möglicherweise nicht zu kaufen beabsichtigt hatte. Auch dieses Vorgehen ist kritisch zu beurteilen. Obwohl das *long tail* Problem [18] den Einsatz von Empfehlungssystemen unverzichtbar macht[8], ist auch hierbei die Beachtung der oben genannten Eigenschaften wünschenswert.

In diesem Kapitel wurde eine allgemeine Einführung in die verschiedenen Ansätze und Technologien der Empfehlungssysteme gegeben. Die zwei groben Kategorien ergeben sich aus Art und Herkunft von Information die für die Generierung der Empfehlungen verwendet wird: inhaltliche Features der Objekte und kollaborative Informationen durch die Bewertung anderer Benutzer, bzw. der Ermittlung von *peers*[9]. Um den Schwächen einzel-

[8] In Onlineshops gibt es in der Regel weit mehr Produkte, als in realen Läden
[9] *peers* bezeichnet mehrere Benutzer mit ähnlichem Geschmack

ner Ansätze entgegenzuwirken, gibt es die Möglichkeit mehrere Methoden zu kombinieren, was zu der Entwicklung von *hybriden Systemen* führte. Darüber hinaus existieren viele Erweiterungen, die oft auf der Verbesserung der Modelle und Algorithmen beruhen [4]. Mit der Verwendung des *Kontextes* wird im nächsten Kapitel eine weitere Möglichkeit zur Verbesserung der Qualität von Empfehlungen vorgestellt.

3 Kontextsensitive Systeme

Die Mehrheit der populären Methoden arbeitet auf dem *user-item* Modell, benutzt also nur zwei Dimensionen zur Generierung von Empfehlungen. Es wurde mehrfach gezeigt, dass es weitere Faktoren gibt, welche die Präferenz eines Objektes für eine Person in einer bestimmten Situation beeinflusst, nämlich der *Kontext* in dem sich der Benutzer befindet [3]. Damit wird die Anforderung an das Empfehlungssystem dahingehend erweitert, dass es nicht nur eine statische Liste von Empfehlungen ausgibt, sondern vielmehr Empfehlungen, die in dem aktuellen Kontext von Bedeutung sind. Diese Empfehlungen können sich für den selben Nutzer je nach Kontext unterscheiden. Die Verwendung von *Kontextinformationen* bietet also nicht nur eine Verbesserung der Qualität, sondern macht einige neue Anwendungen überhaupt erst möglich. Gerade im mobilen Bereich sind die Stärken offensichtlich: Bei einem mobilem Empfehlungsdienst für Gastronomie ist es wenig sinnvoll, wenn dem Benutzer die besten Indischen Restaurant von New York angezeigt werden, wenn er sich gerade in Berlin befindet. Ein intelligenteres System etwa würde dem Benutzer optional auch noch die Restaurants in der unmittelbaren Nähe anzeigen, sofern ihm die GPS Daten der aktuellen Position zur Verfügung stehen. In mobilen Anwendungen ist der *Ort* also ein wichtiger Kontextfaktor.
Auch die Zeit spielt in vielen Anwendung eine bedeutende Rolle. Menschen tendieren dazu, bestimmten Tätigkeiten zu bestimmten Zeiten nachzugehen. Beispielsweise möge ein Benutzer eines Nachrichtenportals morgens nur die wichtigsten Nachrichten in kompakter Form bekommen, beschäftigt sich jedoch an einem Sonntag Vormittag gerne mit längeren, tiefer gehenden Artikeln. Auch bei Gewohnheiten zum Hören von Musik lassen sich individuelle zeitliche Muster feststellen [26].
Bei Kinogängern ist darüber hinaus entscheidend, mit *wem* sie ins Kino gehen. Mit dem Freund oder der Freundin, oder mit der Familie? [3]
Eine weitere Art von Kontext, die z.B. beim Kauf von Produkten entscheidend ist, ist die Intention des Nutzers. Sucht eine Person ein Buch für ihr Mathematikstudium, oder einen Roman als Geschenk an eine Freundin? Auch hinter dem Musikhörer mag sich eine Intention verbergen – *anregende* Musik zum wach werden oder *ruhige* Musik zur Entspannung. Man könnte hier noch weiter gehen und so etwas, wie die aktuelle *Stimmung* des Hörers als Kontext betrachten (z. B. *fröhlich, traurig, entspannt*).
Wie wichtig der Kontext bei menschlichen Entscheidungsprozessen und Gewohnheiten ist, begründet bereits die Verhaltensforschung [43].

3.1 Was bedeutet Kontext

Doch was genau ist Kontext, welche Arten von Kontext gibt es, worauf bezieht er sich? Wie lässt er sich ermitteln und anwenden? Welcher Kontext ist relevant? Diese Fragen sollen in den folgenden Kapiteln diskutiert werden.

3.1.1 Definition

Trotz der Verwendung und Erforschung des Begriffes *Kontext* in einem breiten Spektrum von Forschungsbereichen – in der Informatik gleichermaßen, wie in den Kognitionswissenschaften – scheint es schwierig, eine einheitliche Definition zu finden.

Nach Bazire et al. [8] lautet die häufigste, universale Definition im Bereich der Kognitionswissenschaften wie folgt: „*Kontext ist die Menge der Umstände, die ein Ereignis oder Objekt umgeben*“
Diese sehr allgemeine Definition hilft jedoch wenig. Bazire et al. gibt keine genaue Definition, stellt jedoch eine Reihe interessanter Fragen, untere deren Gesichtspunkten eine Menge von 150 verschiedene Definitionen untersucht werden:

> Ist es möglich, Kontext a priori zu definieren, oder sind nur dessen Effekte a posteriori sichtbar? Ist es etwas statisches oder dynamisches? [...] Welcher Kontext ist relevant für uns? Der Kontext der Person? Der Kontext der Aufgabe? Der Kontext der Interaktion? Wo beginnt Kontext und wo hört er auf? Worin besteht der echte Zusammenhang zwischen Kontext und Kognition?

Bei genauerer Betrachtung der oben genannten Beispiele für die Verwendung von Kontext, fällt auf, dass es verschiedene Arten von Kontext geben muss. Zeit und Ort einer Aktivität sind greifbare, konkrete Konzepte, die in definierbaren Einheiten (*Tag, Jahreszeit, Stadt, Koordinaten*) angegeben werden können. Die aktuelle *Stimmung* einer Person als Kontext einer Interaktion betrachtet, ist vielmehr abstrakter Natur. Es fällt schwer sie durch exakte Klassen zu beschreiben. Auch kann sie als dynamisch charakterisiert werden – veränderbar im Laufe der Interaktionen. Wird ein neues Musikstück gespielt, etwa aufgrund eines Vorschlags der Anwendung, mag dies die Stimmer des Hörers verändern und so seine zukünftigen Entscheidung für die Wahl der nächsten Stücke beeinflussen.
Dourish beschreibt diesen Unterschied durch zwei verschiedene Betrachtungsweisen von Kontext [16]. Die eine Sicht nennt der Autor *representational view* – die gegenständliche Sicht. Für diese wird angenommen, dass sie eine Art Information ist: beschreibbar, unveränderlich und unabhängig von der Aktivität. In dieser Sichtweise besteht der Kontext aus einer Menge von Attributen welche die Umgebung einer Aktion charakterisieren, die für jede Anwendung genau bekannt sind und dessen Struktur explizit definiert werden kann. Sie sind konstant zwischen verschiedenen Aktionen innerhalb einer Anwendung, unterliegen somit keiner zeitlichen Veränderung.
Im Gegensatz dazu ist die *interactional view* eine Ansicht, in der sich die Beschaffenheit der Kontextmerkmale dynamisch ergibt und nicht fest vorgegeben werden kann. In dieser Sichtweise ist Kontext als Eigenschaft einer Relation zwischen Objekten oder Aktivitäten zu verstehen. Er ist bestimmt für jedes Ereignis an sich, kann sich jedoch in seiner Form zwischen verschiedenen Ereignissen unterscheiden. Seine Merkmale lassen sich nicht vorher definieren, sie entstehen vielmehr aus der Interaktion – Aktivität und Kontext beeinflussen sich gegenseitig.
Bedeutender als die Frage der genauen Definition von Kontext ist, wie sich diese allgemeinen Sichtweisen auf die konkrete Anwendung von Kontext in Empfehlungssystemen abbilden lässt. Welche Bedeutung haben sie genau, wie kann diese Erkenntnis genutzt werden, um bessere, nützlichere Empfehlungen zu generieren?

3.1.2 Kontext in Empfehlungssystemen

Im klassischen Sinne wird der Kontext, in welcher Aktionen getätigt und Empfehlungen bereitgestellt werden durch eine Menge von Kontextfaktoren repräsentiert, z. B. Zeit, Stimmung und Intention.
Als Struktur der Kontextfaktoren lässt sich häufig eine Liste von Kategorien definieren,

die ein Faktor annehmen kann. So könnte der Kontext Ort etwa durch die Kategorien *„zu Hause“*, *„im Büro“* oder *„in der Bahn“* dargestellt werden. Dem selben Kontext könnten jedoch auch ganz andere Kategorien zu Grunde liegen, etwa den Namen der Stadt oder absolute Koordinaten. An Stelle einer linearen Struktur, kann Kontext auch durch eine Hierarchie repräsentiert werden. So kann im Beispiel des Ortes der Maßstab berücksichtigt werden, so dass der Kontext als *Land, Region* und *Stadt* angegeben wird. Es wird deutlich, dass sich die Struktur auf sehr unterschiedliche Arten definieren lässt.
Eine Möglichkeit der Klassifikation von Kontext im Rahmen von Empfehlungssystem ist die der Entität, auf die sich der Kontextfaktor bezieht. Hier sind vor allem Kontext der Umgebung, Kontext des Nutzers und Kontext des Objektes zu nennen.
Der Kontext der Umgebung charakterisiert die aktuelle Umgebung des Nutzers, z. B. Ort, Zeit, Wetter und Temperatur.
Der Kontext des Nutzers kann aus statischen Informationen, wie etwa *demographische Angaben* oder dynamische Informationen, wie *Stimmung* oder *Intention*, bestehen. In der Literatur werden teilweise auch inhaltliche Features der Objekte (*content profiles*) als Kontext gewertet [27]. Es stellt sich heraus, dass mit dieser Definition, inhaltliche Features, wie sie bei den *content-based* Methoden verwendet werden, auch als statischer Kontext interpretiert werden können.
Eine andere Unterscheidung nach Adomavicius et al. untersucht die Beschaffenheit von Kontext im Rahmen von Empfehlungssystemen nach den den zwei folgenden Aspekten. Zum einen das Wissen, welches das System über den Kontext besitzt und zum anderen, wie sich dieses über die Zeit verändert. Zu dem Wissen gehören die genauen Kontextfaktoren, ihre Relevanz und ihre Struktur. Er unterscheidet drei Fälle:

1. *observable:* Dem System ist alles bekannt: die Menge der Faktoren, ihre Struktur und Relevanz sowie die Werte ihrer Ausprägungen.

2. *partially observable:* Dem System ist nur ein Teil der Informationen bekannt. So möge es Kenntnis darüber haben, welche Kontextfaktoren eine Rolle spielen, jedoch nicht über deren Struktur. Ein Musikempfehlungssystem weiß beispielsweise, dass Zeit und Stimmung relevant für Präferenzen des Nutzers sind, jedoch ist die Struktur des Faktors *Stimmung* nicht genau bekannt, da sie sich nicht durch feste Kategorien beschreiben lässt.

3. *unobservable:* Das System weiß nichts über die Kontextfaktoren.

Die zweite Dimension wird danach unterschieden, wie sich die Kontextfaktoren über die Zeit ändern:

- *static:* Die Kontextfaktoren und ihre Relevanz sind unveränderlich. So lässt sich die Struktur durch eine Liste von Kategorien, wie etwa *morgens, mittags, abends* festlegen.

- *dynamic:* Die Kontextfaktoren und ihre Relevanz sind veränderlich. In unterschiedlichen Situation kann sich der Einfluss verschiedener Faktoren ändern, auch können neue Kategorien entstehen, oder ein Kontextfaktor verliert im Laufe der Zeit an Bedeutung.

Zwischen Dimensionen lassen sich verschiedene Kombinationen in unterschiedlichen Abstufungen finden. Die zwei Grenzen entsprechen den Definitionen von Dourish [16]. Die *representational view* wäre ein System, das alle Faktoren kennt und in dem sich diese nicht verändern: *static* und *observable*. Die *interactional view* ist ein System, das nichts über die relevanten Kontextfaktoren und ihre Struktur weiß. Nach dieser Klassifikation wäre es sowohl *dynamic*, als auch *unobservable*. [2] Es stellen sich mehrere Herausforderungen für kontextsensitive Empfehlungssysteme. Zum einen muss das Wissen über den Kontext gewonnen werden, zum anderen müssen die sich daraus ergebenden Daten in einer Form repräsentiert werden, die sie für die Anwendung nutzbar macht.
Wie bereits dargestellt, teilt sich das Wissen, welches für die Verwendung von Kontext benötigt wird in verschiedene Bereiche. Zuerst müssen die Kontextfaktoren definiert werden, die eine Rolle spielen. Daraufhin muss eine Struktur gefunden werden, in welcher die Faktoren repräsentiert werden können und schließlich müssen die Kontextinformationen erhoben werden.
Die Festsetzung der Kontextfaktoren ist keineswegs trivial, da sie meist von der Domäne abhängen. Sie werden in der Regel von einem Domänen-Experten anhand von Heuristiken festgelegt [2]. Auch die Struktur ist nicht immer wohldefiniert und kann – je nach Anwendung – unterschiedlich festgelegt werden.

3.1.3 Gewinnung von Kontextinformationen

Die Gewinnung von Kontextinformationen kann auf verschiedene Arten erfolgen. Insbesondere wird unterschieden, ob die Informationen implizit, explizit oder durch Interferenz ermittelt werden [5].

Explizit In diesem Fall wird der Nutzer explizit durch Angaben dazu aufgefordert, einen Kontext zu definieren. So könnte auf einer Filmplattform die Benutzerin die Möglichkeit haben, auszuwählen, ob sie den Film mit ihrem Partner oder ihren Freundinnen ansehen will, ob an einem Wochenende oder unter der Woche, etc.
Bei der Erhebung expliziter, kontextuelle Bewertungen muss dieser Kontext ebenfalls angegeben werden.

Implizit Eine weitere Möglichkeit besteht in der impliziten Ableitung der Informationen, ohne direkte Befragung des Nutzers. Hierzu wird der Verlauf der Interaktionen des Nutzers mit dem System verwendet (z. B. besuchte Webseiten, betrachtete Produkte). Viele physikalische Kontextinformationen können durch Sensoren gesammelt werden, so etwa die Temperatur oder der Ort. Dies ist vor allem in mobilen Anwendungen interessant, da moderne, mobile Endgeräte zunehmend über die Technik zur Ermittlung jener Daten verfügen [5]. Ferner kann der temporale Kontext durch den Zeitpunkt der Interaktion abgeleitet werden. Der Zeitpunkt einer expliziten Bewertung hingegen, muss nicht zwingend mit der Nutzung übereinstimmen und kann daher nicht verwendet werden [36].
Eine andere implizite Informationsquelle für Kontextdaten besteht in der Nutzung von sozialen Netzwerken. Durch die wachsende Popularität von sozialen Netzwerken bietet es sich an, Information über die soziale Umgebung des Nutzers als zusätzliche Kontextinformation zu verwerten. Es wurde gezeigt, dass es dadurch möglich ist dem *cold-start*

Problem für neue Nutzer entgegen zu wirken, da über den sozialen Kontext schon a priori Wissen über den Nutzer existiert [47, 24].

Interferenz Für Kontextfaktoren, die nicht direkt sichtbar (*unobservable*) sind, besteht die Möglichkeit diese durch Interferenz abzuleiten. Ist die Struktur dieser Faktoren bekannt, können sie als latente Variablen im System modelliert werden, deren Werte durch die sichtbaren Interaktionen des Nutzers gelernt werden [2]. Dazu können verschiedene *machine learning* Algorithmen einsetzen werden. Beispielsweise illustriert Palmisano et al. die Verwendung von Bayeschen Netzwerken, um durch Interferenz des Nutzerverhaltens beim Betrachten von Produkten in einem Onlineshop, die Intention des Kaufs vorherzusagen. [40].
Ein weiteres Beispiel ist die Erkennung der emotionalen Verfassung des Nutzers in einem Musikempfehlungssystem [27]. Selbst wenn die Faktoren nicht sichtbar sind und sich durch ihre dynamische Struktur keine genauen Kategorien definieren lassen, so ist es doch in vielen Fällen möglich, einen Kontext abzuleiten [5]. In einem Empfehlungssystem für Musik etwa, kann angenommen werden, dass sich der aktuelle Kontext der Benutzerin durch die Musikstücke und Genres, die sie kürzlich gehört hat, ergibt. Hierbei ist nicht genau bekannt, ob es sich dabei um Stimmung oder Aktivität handelt. Es wird jedoch angenommen, dass es einen latenten Faktor gibt, der die aktuellen Präferenzen des Hörers beeinflusst. Hariri et al. schlägt ein Verfahren zur Erkennung sequenzieller Muster in der *playlist* der Benutzer vor, mit dessen Hilfe eine bessere Vorhersagen für passende Musikstücke getroffen werden kann. Der Autor repräsentiert dabei jedes Musikstück durch eine Menge von Themen, die durch die Aufbereitung von *social tags* gewonnen werden. Weiter analysiert er eine Menge durch Nutzer erstellte *playlists* nach sequentiellen Mustern. Mit diesen Informationen wird schließlich ein *hidden markov model* trainiert, das wahrscheinliche Übergänge zwischen Themen erlernt [22].
In diesem Fall ist der Kontext latent, also „verborgen“ hinter der Sequenz von Musikstücken. Eine allgemeinere Interpretation dieser Idee, ist das Kurzzeitinteresse (*short-term preference*) des Benutzers. Ein Benutzer möge in seiner aktuellen Session eine besondere Intention oder Interesse haben, die sich seinen unmittelbaren Aktionen widerspiegelt. Die *short-term preferences* eines Benutzers können von seinen Langzeitinteressen (*long-term* preferences) abweichen [50]. Da sich *short-term preferences* über die Zeit verändern, können diese als eine Form des dynamischen Kontextes betrachtet werden.

3.1.4 Repräsentation von Kontext

Die Repräsentation von Kontext im Falle des statischen, sichtbaren Kontextes, also der *representational view*, geschieht durch eine Menge expliziter Variablen, welche die relevanten Kontextfaktoren modellieren (z. B. Zeit, Ort, Wetter). Die einzelnen Kontextfaktoren haben eine wohldefinierte Struktur, können also u.a. als eine Liste von Kategorien repräsentiert werden (z. B. morgens, mittags, abends oder kalt, warm, Regen).
Wie bereits erwähnt, ist es in manchen Fällen sinnvoll die Struktur als Hierarchie zu modellieren, um die Relation zwischen verschiedenen Sichtweisen, etwa von *allgemein* zu *spezifisch* oder die Granularität, semantisch abzubilden. Der Kontext *Ort* könnte folgende Hierarchie haben [2]: *Ort:* Koordinaten → Stadt → Region → Land.

Formal kann die konkrete Ausprägung eines Kontextes, durch ein Tupel verschiedener Kategorien repräsentiert werden, die den aktuellen Kontext beschreiben [5, 33]:

$$\begin{aligned} c &= (c_1, c_2, \ldots c_k) \\ &= (morgens, winter, kalt) \end{aligned}$$

Kontextabstraktion Viele Daten, die als Kontextinformation dienen, liegen in einer kontinuierlichen, numerischen Form vor. Dies ist vor allem der Fall, wenn diese Daten implizit durch Sensoren oder *timestamps* der Aktivitäten gewonnen werden. Für eine Anwendung in einem *context-aware* Empfehlungssystem wird jedoch in der Regel eine diskrete, kategorische Form benötigt [33]. Der Kontext Zeit als numerischer Wert interpretiert, ist wenig hilfreich, da ein Ereignis, wie etwa das Ansehen eines Films zu einem exakten, absoluten Zeitpunkt nur einmal auftritt. Vielmehr wird eine Einteilung in zeitliche Kategorien benötigt, um ein periodisches Nutzungsverhalten zu identifizieren. Diese Kategorien werden bereits gut durch die natürliche Sprache abgebildet, z.B. durch die Phrasen *„morgens“* oder *„am Wochenende“*. Ein periodisches Verhalten könnte sich aus dem folgenden Satz ableiten: *„Abends höre ich gerne ruhige Musik zur Entspannung“*. In diesem Satz befinden sich zwei semantische Informationen, die als Kontext betrachtet werden können: das Adverb „abends“, welches eine zeitliche Kategorie bildet und das Adjektiv „ruhig“, welches ein inhaltliches Merkmal der Musik darstellt.
Für Wetter und Temperatur gilt ähnliches. Zwischen den numerischen Angaben *27˚C* und *28˚C* wird kein subjektiver Unterschied feststellbar sein, jedoch zwischen *4˚C* und *27˚C*. Hier lassen sich ebenfalls nominale Kategorien finden (etwa *heiß, warm, kalt*), welche subjektive die Semantik der der Zahlen reflektieren.
Diese Diskretisierung wird in der Literatur als *context abstraction* bezeichnet [33]. Die einfachste Möglichkeit ist dabei, feste Grenzen zu definieren. Die Bereiche zwischen diesen Grenzen bilden jeweils die Kategorien.
Zur Verwendung der Tageszeit als Kontextdimension, würde der Tag in verschiedene zeitliche Kontextkategorien aufgeteilt, beispielsweise durch die Grenzen 0, 6, 12 und 18 Uhr. Die daraus entstehenden nominalen Kategorien könnten als *morgens, mittags, abends, nachts* bezeichnet werden. Der Zeitraum *6:00 - 12:00* erhält das Label *morgens*, *18:00 - 00:00* das Label *abends*, usw. Abb. 1 zeigt eine mögliche Aufteilung von Tag und Woche in feste Kategorien.

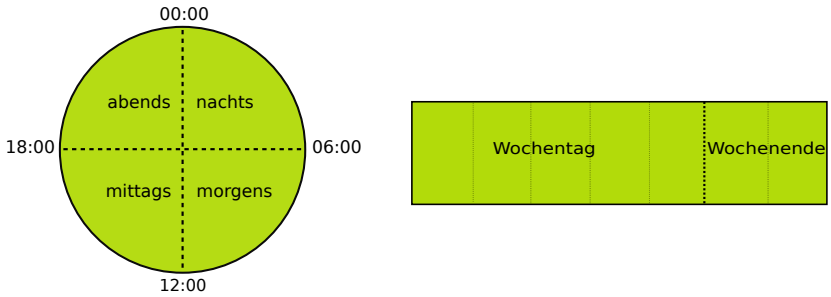


Abbildung 1: Beispiel für die Unterteilung von Tag und Woche in Kontextkategorien

***Fuzzy* Kontext** Das Bestimmen genauer Grenzen zur Aufteilung ist nicht immer möglich, da diese entweder verschwimmen (etwa bei Jahreszeiten) oder subjektiv unterschiedlich interpretiert werden (so bei Tageszeiten oder Temperaturen). Eine Lösung dafür stellt Lee et al. mit dem Konzept des *fuzzy* Kontextes vor [33]. Hierzu werden einzelne Kontextkonzepte mit einer Gewichtung versehen. Sei $C = c_1, \ldots c_n$, dann kann der Kontext c als eine Menge von Tupeln definiert werden, welche aus der Kategorie c_k sowie einem Gewicht w_k, das den Grad der Zugehörigkeit zur Kategorie c_k angibt:

$$c = \{(c_k, w_k) \,|\, c_k \in C \wedge w_k \in [0, 1]\}$$

3.2 Taxonomie kontextsensitiver Empfehlungssysteme

Die Kontextinformationen können als zusätzliche Variablen in dem System verstanden werden, so dass sich der Raum, über den die Nützlichkeitsfunktion (*utility function*) definiert wird, auf mehrere Dimensionen erstreckt. Daher wird dieser Ansatz auch *multi-dimensional recommendations* genannt.

Formal wird die ursprüngliche Definition $f_u : U \times I \rightarrow R$ um den Kontext erweitert $f_u : U \times I \times C \rightarrow R$
Analog zu traditionellen Empfehlungssystemen wird hierbei die Nützlichkeit der Objekte für die Benutzer ermittelt, etwa die Bewertung die eine Benutzerin dem Film X geben würde.
Das System schätzt aufgrund bekannter Informationen (z. B. Bewertungen, Präferenzen), Bewertungen für unbekannte Objekte. Es kann also allgemein als eine Funktion aufgefasst werden, die zu einem *input*, etwa bekannte Präferenzen des Benutzers, einen *output*, die Liste von Empfehlungen liefert.
Im Unterschied zu traditionellen Systemen wird nicht allein die Präferenz des Benutzers, sondern die Präferenz des Benutzers in einem bestimmten *Kontext* verwendet. Das bedeutet, dass die Eingabe in das System aus einem Tupel $(User, Item, Context, Rating)$ besteht, welches als die Bewertung r eines Nutzers u des Objektes i im Kontext c zu interpretieren ist [5]. Ein Beispiel für einen Bewertungsdatensatz mit Kontextinformation ist in Tabelle 1 dargestellt.

Benutzer	**Film**	**Kontext** (Begleitung)	**Bewertung**
Alice	Twilight	Freundinnen	3
Alice	Twilight	Partner	1
Alice	Matrix	Egal	5
Bob	Twilight	Egal	1
Bob	Grindhouse	Freunde	3
Bob	Grindhouse	Familie	0
Bob	Ratatouille	Familie	4

Tabelle 1: Beispiel für kontextuelle Bewertungen

Um eine Liste von Empfehlungen zu generieren, wird der Funktion neben dem Nutzer der Kontext angegeben, in dem er sich befindet oder in dem die Empfehlungen relevant sein sollen. Die Funktion RS_N, welche das *recommendation set* erzeugt wird damit um einen Kontextparameter erweitert:

$$RS_N(u, c) = \{s | s \in I \wedge f_u(u, s, c)\ maximal\};\ \ |RS_N(u, c)| = N$$

wobei I die Menge der Objekte und N die Anzahl der Top-N Empfehlungen ist. [5]
Nach der formalen Definition von kontextbasierten Empfehlungssystemen stellt sich die Frage, wie ein solches System, welches den Kontext integriert, zu implementieren ist. Grundsätzlich bestehen zwei Möglichkeiten. Einmal können traditionelle 2D Empfehlungssysteme verwendet werden, um die zusätzliche Information des Kontextes an einer Stelle im System als einen Filter einzufügen. Der Kontext kann hierbei entweder *vor* der oder *nach* der Generierung von Empfehlungen in den Prozess einfließen.
Alternativ kann ein neues System zu entwickelt werden, welches den Kontext direkt mit berücksichtigt.
Insgesamt lassen sich also die drei folgenden Fälle unterscheiden:

- *pre-filtering* Die Eingabedaten, bestehend aus den Bewertungstupeln, werden gefiltert. Dabei werden nur jene Tupel verwendet, welche dem aktuellen Kontext entsprechen. Diese werden einem herkömmlichen 2D System übergeben.

- *post-filtering* Das 2D System generiert eine Liste von Empfehlungen unter Verwendung des gesamten Datensatzes. Diese wird anschließend unter Berücksichtigung des Kontextes weiter gefiltert. Es werden nur die Objekte ausgegeben werden, die im dem aktuellen Kontext nützlich waren.

- *contextual modeling* Es wird ein neues Systems entwickelt, welches die Kontextinformation direkt in die Implementierung der Empfehlungsfunktion mit einfließen lässt.

3.2.1 Pre-Filtering

Mit diesem Ansatz werden also die Eingabedaten selbst gefiltert, bevor sie der Empfehlungsfunktion übergeben werden. Soll in Beispiel aus Tabelle 1 etwa eine Empfehlung für den Kontext $c = Familie$ gegeben werden, würden all jene Bewertungen verwendet,

die ein Benutzer für den Kontext *„gesehen mit der Familie“* angegeben hat. Der Vorteil besteht offensichtlich darin, dass hier klassische 2D Empfehlungssysteme verwendet werden können, die bereits gut erforscht sind.
Für einen einfachen *memory-based collaborative filtering* Ansatz sieht das wie folgt aus. Die *user-item* Matrix, wie sie bisher verwendet wurde, wird um ein oder mehrere Kontextdimension erweitert, etwa Zeit, Ort oder Begleitung. Aus der Matrix wird somit ein mehrdimensionaler *Würfel*. Diese Datenstruktur ist vergleichbar mit OLAP[10], ein multidimensionales Datenmodell, das im Bereich *data warehouses* breite Verwendung findet. In diesem Model wird jeder Kontextfaktor als eine zusätzliche Dimension eingeführt. Es ergibt sich also ein Raum der Dimension n als das kartesischen Produkt der einzelnen Dimensionen $R = D_1 \times D_2 \times \cdots \times D_n$.
Für ein konkretes Beispiel mit den Kontextfaktoren Ort und Zeit, wäre die Datenstruktur ein Tensor $m \in R$ der Dimension $n = 4$, wobei $R = User \times Item \times Time \times Location$. [5]
Die Implementierung nach der Definition des klassischen *memory-based collaborative filtering* ist der *reduction-based* Ansatz nach Adomavicius et al. [3]. Sie schätzt die Präferenz eines Benutzers u für ein Objekt i in dem Kontext c:

$$p_{u,s,c} = \sum_{u' \in \mathcal{N}_u} sim_c(u, u') \times (r_{u,s,c} - \overline{r}_{u'})$$

wobei $r_{u,s,c}$ die bekannten Bewertungen des Nutzers u im jeweiligen Kontext und $\mathcal{N}_u$ die Nachbarschaft des Nutzers darstellen. Die Funktion sim_c ist eine Ähnlichkeitsfunktion zwischen zwei Nutzern, welche den aktuellen Kontext reflektiert. Dazu werden, dem Paradigma des *pre-filtering* folgend, nur Bewertungen hinzugezogen die dem aktiven Kontext entsprechen. Eine häufig verwendete Metrik ist die *cosine similarity*. Diese kann analog zur nicht kontextuellen Variante definiert werden [3, 33]:

$$sim_c(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|} = \frac{\sum_{s \in I_{uv}} r_{u,s,c} \cdot r_{v,s,c}}{\sqrt{\sum_{s \in I_{uv}} r_{u,s}^2} \sqrt{\sum_{s \in I_{uv}} r_{v,s}^2}}$$

wobei I_{uv} die Menge der gemeinsam bewerteten Objekte von Benutzer u und v sei.
Als weitere Variante kann, anstelle der kontextabhängigen Ähnlichkeit, eine globale Ähnlichkeit verwendet werden, welche alle Bewertungen einbezieht um gleichgesinnte Benutzer zu finden [33]. Dazu wird über alle Kontextwerte aggregiert. Der Tensor wird somit wieder auf die zweidimensionale *user-item* Matrix projiziert:

$$sim(u, v) = \frac{\sum_{c \in C} \sum_{s \in I_{uv}} r_{u,s,c} \cdot r_{v,s,c}}{\sqrt{\sum_{c \in C} \sum_{s \in I_{uv}} r_{u,s,c}^2} \sqrt{\sum_{c \in C} \sum_{s \in s_{uv}} r_{v,s,c}^2}}$$

wobei C der Raum aller möglichen Kontextwerte darstellt.
Der Ansatz des *pre-filtering* hat jedoch einen großen Nachteil: Da Bewertungen herausgefiltert werden, verringert sich die Menge der Observationen, auf deren Basis die Bewertungen gemacht werden und führt zu dem Problem der *sparsity* [36]. In dem Beispiel der Filmbewertungen aus Tabelle 1 hat Alice bisher keinen Film mit Begleitung der Familie bewertet. Daher würden ihre Bewertung nicht beachtet werden, wenn nach dem Kontext $c = Familie$ gefragt ist. Das Problem macht sich insbesondere dann bemerkbar,

[10] OLAP steht für: *OnLine Analytical Processing* [5]

je präziser der Kontext definiert wird. Es wird unwahrscheinlicher, bekannte Bewertungen zu finden, die mit diesen Kriterien gemacht wurden. Will sich Bob Empfehlungen für den Kontext $c = (Freundin, Mittwoch, Abends, Kreuzberg)$ geben lassen, könnte es sein, dass nur sehr wenige Bewertungen existieren, welche diesem entsprechen. Bob würde schlechte oder keine Empfehlungen erhalten.
In dem Versuch, diesem Problem entgegenzuwirken, schlägt Adomavicius et al. die Idee der *context-generalization* vor. Hierbei wird der Kontext verallgemeinert, sollten nicht ausreichend Bewertungen vorhanden sein. Das dazu verwendete Modell repräsentiert den Kontext als eine einfache Hierarchie. Konzepte, die weiter oben in der Hierarchie liegen, beinhalten die der unteren Hierarchiestufen. Für das gegebene Beispiel, könnte die Hierarchie wie folgt aussehen: *Ort:* Alle → Stadt → Stadtteil, *Zeit:* Alle → Wochenzeit → Wochentag. Der ein generischerer Kontext ergibt sich durch das jeweilige Aufsteigen um eine Hierarchiestufe, aus c würde $c' = (Freundin, Wochentag, Abends, Berlin)$.

3.2.2 Post-Filtering

In diesem Ansatz wird eine Liste von Empfehlungen auf die klassische Art und Weise ohne Berücksichtigung des Kontextes generiert. Anschließend wird die Kontextinformation genutzt, um die Liste zu verändern. Dabei gibt es mehrere Möglichkeiten. Entweder werden Objekte, die im aktuellen Kontext nicht relevant sind, entfernt oder die Liste wird so umsortiert, dass relevantere Objekte (bezüglich des Kontextes) weiter oben erscheinen.

- *Filtering* Nicht relevante Objekte werden entfernt (bzgl. des Kontextes)
- *Ranking* Objekte werden neu gewichtet basierend auf der Relevanz im gegebenen Kontext

Eine weitere Unterscheidung besteht darin, wie das *post-filtering* umgesetzt wird: durch eine *Heuristik* oder *model-based*. Eine Heuristik kann z. B. sein, dass Objekte, die öfter in diesem Kontext auftauchen, eine höhere Gewichtung erhalten.
Im *model-based* Ansatz, wird dagegen ein prädiktives Modell gelernt, welches die Relevanz eines Objektes in einem gegebenen Kontext berechnet. Diese Wahrscheinlichkeit kann wiederum verwendet werden, um die Liste zu filtern oder das Ranking zu verändern. Das Problem der *sparsity* aus dem *pre-filtering* Ansatz tritt hier nicht auf, da zu Beginn alle Bewertungen verwendet werden.
Es wurde empirisch gezeigt, dass der *post-filtering* Ansatz durch Ranking bessere Ergebnisse liefert als der *pre-filtering* Ansatz ohne *context-generalization*. [5]

3.2.3 Contextual Modeling

Anstatt der Anwendung von klassischen, zweidimensionalen Algorithmen, versucht der Ansatz des *contextual modeling* Methoden zu entwickeln, welche die Kontextinformation direkt in der Modellierung verwenden. Einige Implementierungen sind darauf ausgerichtet, bestehende 2D Systeme auf den n-dimensionalen Raum zu erweitern und direkt auf diesem zu operieren.
Heuristische Algorithmen, wie etwa die *nearest-neighborhood* Methode für das *collaborative filtering* verwenden geeignete Ähnlichkeitsfunktionen zum Aufspüren ähnlicher Benutzer, um aus der gewichteten Summe ihrere Bewertungen neue Bewertungen für den

Zielbenutzer zu schätzen. Diese Gewichtung kann als Distanz der Bewertungspunkte im $User \times Item$ Raum interpretiert werden. Anstelle des *pre-filtering* von Bewertungen ließe sich auch eine mehrdimensionale Distanzfunktion auf dem $User \times Item \times Contex$ Raum verwenden [5].
Für modellbasierte Methoden existieren bereits einige Ansätze der multidimensionalen Erweiterung. Zu den Besten gehören die *latent factor models* Ansätze, wie die *matrix factorization* Methode, welche bereits im *Netflix Prize* Wettbewerb [11] besondere Stärke demonstrierten [31]. Eine Generalisierung dieser auf mehrdimensionale Räume ist möglich, wobei Matrizen zu Tensoren werden. Eine erste erfolgreiche Anwendung der *tensor factorization* zur Erzeugung von kontextbasierten Empfehlungen zeigte Karatzoglou et al. [28], eine Verbesserung bezüglich der Skalierbarkeit wurde von Rendle et al. vorgestellt [45].

In diesem Abschnitt wurde ein neuer Bereich der Empfehlungssysteme erschlossen, indem gezeigt wurde, dass es weit mehr Informationen gibt, als in traditionellen Systemen verwendet werden. Zusammengefasst unter dem Stichwort Kontext, haben diese das Potential, die Qualität der Empfehlungen zu verbessern. Einige wichtige Ansätze zur Verwendung dieser zusätzlichen Kontextinformationen wurden vorgestellt. Es stellte sich heraus, dass die Verwendung von bestehenden Systemen mit Erweiterung einige Probleme aufwirft. Die Forschung bewegt sich weiter in Richtung integrativen Lösungen. So vielfältig die Forschung ist, so sind auch die Möglichkeiten der Anwendung. Neben bekannten Anwendungsbereichen wie E-Commerce[12] und Filmempfehlungen[13], lassen sich Empfehlungssysteme in verschiedensten Domänen wie Musik, Informationssuche, Nachrichten, Reise und Tourismus, Gastronomie etc. einsetzen [5]. Die dabei relevanten Informationen sind unterschiedlicher Art. Sie können je nach Anwendung variieren, was auch die Möglichkeiten ihrer Gewinnung unterschiedlich gestaltet. Viele klassische Systeme beruhen auf expliziten Bewertungen, doch wurde gezeigt, dass auch wertvolle Informationen aus impliziten Daten, beispielsweise den Verlauf der Benutzerinteraktionen (*implicit feedback*), extrahiert werden können [33]. Zu den Informationen gehören, neben den Bewertungen verschiedene Kontextinformationen (z. B. Zeit, Ort, Wetter), auch inhaltliche Informationen über die Objekte (Annotationen, Tags) und die Benutzer (demographische Angaben, soziale Verbindungen). Darüber hinaus können semantische Daten, wie etwa Ontologien, verwendet werden, um zusätzliches Wissen über Relationen zwischen Objekten, Nützlichkeit und Kontexten einzubringen.
Ziel der Empfehlungssysteme soll es nun sein, mit Hilfe dieser Informationen aus der oft großen Anzahl von Objekten, diejenigen herauszufinden, die für den Benutzer in der gegebenen Situation nützlich sind.
Zunächst ist es wichtig, diese Informationen auf geeignete Weise zu repräsentieren. Hier liegt die Verwendung von Graphen nahe.
Graphen haben sich in vielen Bereichen als eine gute und flexible Repräsentation von Wissen bewährt [15]. Dies macht sie somit insbesondere für Empfehlungssysteme interessant. In dem folgenden Kapitel soll untersucht werden, wie sich Graphen verwenden lassen,

[11] `www.netflixprize.com`(letzer Zugriff: 01.06.2009)
[12] Amazon (`www.amazon.com` (letzter Zugriff: 01.05.2013)), eBay (`www.ebay.com` (letzter Zugriff: 01.05.2013))
[13] Netflix (`netflix.com` (letzter Zugriff: 01.05.2013))

um mit einfachen Mitteln leistungsstarke und flexible Empfehlungssysteme zu entwickeln, welche verschiedene Ideen aus den bisher vorgestellten Paradigmen integrieren.

4 Graphbasierte Systeme

Viele Informationen, die für die Verwendung in Empfehlungssystemen interessant sind, weisen eine inherente Graphstruktur auf, die ausgenutzt werden kann. Der anschaulichste Fall ist die Linkstruktur von Webseiten, insbesondere Weblogs, da diese oft in einer thematischen Relation zueinander stehen [1].
Einzelne Blogs können als Knoten betrachtet werden, die Links von einem Blog zum anderen als gerichtete Kanten. Diesen Kanten kann eine gewisse Semantik zugeschrieben werden, etwa eine inhaltliche Verwandtschaft der Blogs.
Weiter kann durch Gewichte der Kanten eine Maß für die Verwandtschaft modelliert werden – eine einfache und effektive Möglichkeit, Objekte und die Assoziationen zwischen ihnen zu beschreiben. Nun ist es möglich, nicht nur eine Art von Objekten, wie Blogs in dem obigen Beispiel, sondern verschiedene Objekte durch verschiedene Knotentypen zu repräsentieren. Für den einfachen Fall des *collaborative filtering* ließen sich Benutzer und Objekte als zwei unterschiedliche Knotentypen beschreiben. Die Präferenzen der Benutzer für Objekte werden als gewichtete Kanten dargestellt. Zur Bestimmung der Relevanz der Objekte für einen Benutzer lassen sich nun verschiedene Metriken aus der Graphentheorie verwenden[17]. In der Regel berechnen diese eine Distanz zwischen zwei oder mehreren Knoten, welche ein Maß für die Ähnlichkeit gibt. Hierzu existieren verschiedene Ansätze. Die am häufigsten verwendete Klasse von Algorithmen basiert auf dem *random-walk* Modell, wobei sich vor allem der *PageRank* Algorithmus in verschiedene Variationen bewährt hat [35, 36, 34, 9, 20]. Weitere Ansätze sind *spectral clustering* [1] und *heatconduction* Modelle [37], wobei eine gewisse Ähnlichkeit zu den auf *random-walks* basierenden Methoden besteht [17].

Das Kapitel ist wie folgt gegliedert: Zunächst sollen verschiedene Möglichkeiten zur Modellierung der relevanten Informationen durch Graphen vorgestellt werden. Anschließend soll untersucht werden, wie sich aus den Graphen Empfehlungen generieren lassen. Im letzten Abschnitt wird anhand eines konkreten Beispiels eine mögliche Konstruktion des Graphen im Detail beschrieben.

4.1 Graphenmodelle

Es gibt viele Möglichkeiten aus den zur Verfügung stehenden Daten einen Graphen zu konstruieren.
Sie unterscheiden sich durch die darin verwendeten Knotentypen und ob Kanten gerichtet sind oder nicht. Ferner existiert die Möglichkeit, auch unterschiedliche Kantentypen zu benutzen.
Die einfachste Variante besteht darin, nur eine einzige Art von Knoten zu verwenden, nämlich genau einen für jedes Objekt in dem System. Die gewichteten Kanten repräsentieren hierbei die Ähnlichkeit zwischen Objekten. Offensichtlich muss diese Relation vorher berechnet werden, etwa aus den kollaborativen Bewertungen oder inhaltlichen Features. Der *ItemRank* Algorithmus verfolgt diese Methode [20].
Wesentlich flexibler ist es jedoch, die gesamte Information direkt im Graphen zu modellieren, wobei unterschiedliche Knotentypen zum Einsatz kommen[34].

Obgleich die meisten Ansätze, die in der Forschung zu finden sind, auf *homogenen Graphen* basieren, also Graphen die nur einen Typ von Kanten besitzen, besteht auch die Möglichkeit verschiedene Kanten zu nutzen (*heterogene Graphen*), um unterschiedliche Arten von Relationen gezielt auszudrücken. Von Vorteil ist außerdem, dass semantische Informationsquellen, wie etwa Ontologien, direkt übernommen werden können. Lee et al. stellt dazu den *PathRank* Algorithmus vor, in welchem durch Anfrage unterschiedlicher *Pfade* im heterogenen Graphen, verschiedene Arten von Empfehlungssystemen abgebildet werden können [34].
In dieser Arbeit liegt der Schwerpunkt jedoch auf der Verwendung *homogener Graphen*.
Die grundlegende Idee besteht darin, jedes Konzept durch einen eigenen Knotentyp im Graphen zu modellieren. Die Relation zwischen diesen Konzepten werden durch gewichtete Kanten ausgedrückt, wobei im homogenen Fall die Bedeutung der Kante implizit ist und von dem jeweiligen Knotentyp abhängt.
Die essentiellen Konzepte sind Benutzer und Objekte. Zusätzliche Informationen können durch weitere Knotentypen oder Kanten ausgedrückt werden.
Graphen mit mehreren Knotentypen können weiter dadurch charakterisiert werden, ob disjunkte Teilmengen von Knoten existieren, innerhalb derer *keine* Verbindungen bestehen. Beispielsweise stellt Lee et al. einen Ansatz mit *bipartite Graphen* vor [36]. Jedes Konzept wird durch einen eigenen Knotentyp dargestellt, allerdings wird zwischen *feature* Knoten und *target* Knoten unterschieden. Die Menge der zu empfehlenden Objekte bilden die *target* Knoten, alle übrigen sind *feature* Knoten (z. B. Benutzer und Kontextinformationen). Kanten existieren nur zwischen *feature* und *target* Knoten, nicht jedoch zwischen zwei *feature* Knoten. Diese Entscheidung wird nicht begründet.
Im Unterschied dazu propagiert Bogers die Verwendung von Kanten auch zwischen allen Knotentypen, um zusätzliche Information aufzunehmen [9].

Es wurden verschiedene Konzepte von Informationen vorgestellt, die in Empfehlungssystemen relevant sind. An dieser Stelle werden diese noch einmal hinsichtlich ihrer Bedeutung in der Modellierung des Graphen zusammengefasst:

Benutzerprofil: Angaben über den Benutzer, z. B. Alter, Wohnort, Job. Dies sind Attribute des Benutzers und können jeweils als Knoten modelliert werden. Der Besitz des Attributs durch einen Nutzer wird durch eine Kante zwischen diesen ausgedrückt. Somit sind Benutzer, welche beispielsweise den selben Wohnort haben, indirekt miteinander verbunden.

Objektprofil: Angaben über die Objekte, z. B. Tags, Interpret, Regisseur. Analog zum Benutzerprofil, können diese Informationen als Attribute der Objekte in Form von Knoten eingefügt werden, die verwandte Objekte miteinander verbinden.

Präferenzen: Angaben, welche die Vorlieben der Benutzer zu anderen Entitäten, insbesondere Objekten ausdrücken, z. B. Bewertungen. Diese werden durch Kanten zwischen Nutzern und Objekten dargestellt. Es können jedoch auch Präferenzen zu bestimmten Objektattributen, z. B. Genres modelliert werden.

soziale Verbindungen: Informationen aus sozialen Netzwerken können durch Kanten zwischen zwei Benutzern repräsentiert werden. Zu beachten ist, dass diese

Kanten auch gerichtet sein können, etwa bei asymmetrischen *follower networks*, wie z. B. *Twitter*[14].

statischer Kontext: Kontextinformationen mit fester, kategorischer Struktur, wie Zeit, Ort, Begleitung. Diese können als Attribute einer Interaktion aufgefasst werden. Eine Möglichkeit der Repräsentation im Graphen ist es, für jede Kombination der an einer Interaktion beteiligten Entitäten einen neuen Knoten einzufügen [36]. Beispielsweise würde die Präferenz von *Bob* zum Film *Matrix* im Kontext *Wochenende* ausgedrückt, in dem ein neuer Knoten $(Bob, Wochenende)$ eingeführt wird, der durch eine Kante mit dem Objektknoten $Matrix$ verbunden ist.

zusätzliches Wissen: Weitere Informationensquellen, z. B. Wissensdatenbanken wie *WordNet*, sind denkbar, um den Graph zu erweiterten [9]. So könnten z.B. Ähnlichkeiten zwischen Tags oder Genres durch weitere Kanten zwischen den jeweiligen Objektattributen modelliert werden.

Information muss nicht immer durch neue Knoten modelliert werden. Sie kann von ihrer ursprünglichen Semantik abstrahiert und direkt durch eine Kante ausgedrückt werden. Beispielsweise ließe sich die Information über den Interpreten durch einen neuen Knoten formulieren, welcher mit den jeweiligen Tracks[15] verbunden ist. Im anderen Fall würden Tracks des selben Interpreten direkt miteinander verbunden werden. Die Semantik des Interpreten geht verloren, ihre Bedeutung, nämlich Tracks in eine Relation zu setzen, bleibt jedoch erhalten.

4.2 Generierung der Empfehlungen

Ist der Graph erstellt, können Knoten nach ihrer Relevanz bezüglich einer Anfrage geordnet werden.
Wie bereits erwähnt, wird dazu das *random walk* Modell verwendet.

4.2.1 Random Walk

Der Kern aller Empfehlungssysteme besteht im wesentlichen darin, Ähnlichkeiten zu bestimmen, etwa ähnliche Objekte zu den Präferenzen des Nutzers. Im Graphen werden bekannte Relationen zwischen Knoten durch gewichtete Kanten ausgedrückt. Die Idee eines *random walk* ist es, an einen bestimmten Knoten zu starten und über den Graphen zu laufen. Die Gewichte der ausgehenden Kanten geben dabei eine Wahrscheinlichkeit an, mit der diese Kante im nächsten Schritt genommen wird. Das Modell ist angelehnt an den Benutzer eines *Internetbrowsers*, der über die Links von Seite zu Seite navigiert. Beispielsweise fängt er auf der Seite eines Film an, der ihm gefällt. Auf dieser Seite ist das Genre und der Regisseur verlinkt. Er klickt weiter auf den Regisseur und gelangt schließlich zu einem anderen Film, der ihm zusagt und beendet die Suche.
Diese Idee wird in dem Modell dadurch simuliert, dass die Übergangswahrscheinlichkeit zwischen Knoten durch verschiedene Informationen, wie Präferenzen der Benutzer oder Relevanz von Tags, geschätzt werden [9].

[14] `twitter.com` (letzter Zugriff: 01.05.2013)
[15] Es wird die englische Bezeichnung *Track* für Musikstück verwendet

Neben Algorithmen, welche von nur einem Knoten ausgehend, die Distanz zu anderen Knoten bestimmen [17], besteht die Möglichkeit an mehreren Knoten simultan zu starten und so ähnliche Knoten zu einer Menge von Startknoten zu ermitteln. Diese Methode besticht durch besondere Flexibilität, da sich hierdurch eine Vielfalt von Anfragen an das System stellen lassen. So kann der Einfluss einzelner Entitäten auf Erstellung der Empfehlungen durch das gezielte „Aktivieren" der jeweiligen Knoten, erhöht werden. Es lassen sich beispielsweise Empfehlungen an eine Gruppe von Benutzern generieren, indem mehrere Benutzerknoten als Start gewählt werden [47].
Ein bewährtes Verfahren zum Ranking von Knoten basierend auf *random walks*, ist der *personalized PageRank* Algorithmus. Dieser wird im nächsten Abschnitt genauer beschrieben.
Der Vorteil des *random walk* Modells liegt offensichtlich darin, dass indirekte Relationen zwischen Objekten ausgenutzt werden können [35]. Der Schlüssel für die Leistung des Systems liegt vor allem in der geeigneten Modellierung des Graphen [47].

Gegeben eines oder mehrerer Startknoten, wählt ein Läufer zufällig eine der ausgehenden Kanten aus. Die Auswahl erfolgt nach einer Wahrscheinlichkeitsverteilung, die durch die Gewichte der Kanten bestimmt werden kann. Das Ranking ergibt sich aus der Wahrscheinlichkeit, dass der Läufer nach einer endlichen Anzahl von Schritten den Knoten erreicht.
Ein *random walk* im gewichteten Graphen lässt sich formal durch einen Markov Prozess erster Ordnung mit den Zufallsvariablen $X_0, X_1, \ldots, X_t, \ldots$ beschreiben [47]. Der Zustand einer Variable X_{t+1} hängt nur vom aktuellen Zustand X_t ab. Die Wahl des Zustands im nächsten Schritt erfolgt bezüglich einer Wahrscheinlichkeitsverteilung, welche durch eine Übergangsmatrix P repräsentiert wird. Ihre Elemente p_{ij} seien wie folgt definiert [47, 36]:

$$p_{ij} := P(X_{t+1} = j | X_t = i) = \begin{cases} \frac{w_i}{\sum_{k \in \mathcal{N}_i} w_{ik}} & \text{falls } \mathcal{N}_i \neq \emptyset \\ 0 & \text{sonst.} \end{cases}$$

wobei $\mathcal{N}_i$ die Nachbarschaft des Knotens v_i bezeichnet. Sie ist definiert durch seine ausgehenden Verbindungen: $\mathcal{N}_i := \{v_j | (v_i, v_j) \in E\}$. Da jede Spalte der Übergangsmatrix P eine Wahrscheinlichkeitsverteilung über die ausgehenden Kanten gibt, muss diese Matrix spaltenstochastisch sein, d.h. ihre nicht negativen Einträge summieren sich zu 1: $\forall 0 \leq j < n : \sum_{i=0}^{n} p_{ij} = 1$, mit $n = |P|$ [6]. Die Gewichte der ausgehenden Kanten werden daher jeweils durch die Summe aller ausgehenden Kantengewichte geteilt.
Eine Verletzung dieser Eigenschaft möge dann auftreten, wenn ein Knoten keine ausgehenden Kanten besitzt, da die Summe der entsprechenden Spalte 0 ist. Im *PageRank* Algorithmus, der im Folgenden genauer beschrieben wird, ist dieses Problem durch eine *reset probability* gelöst, die den *random walker* zu einem zufälligen Knoten teleportiert [36].

4.2.2 PageRank

Als eine Erfindung von Google[16], liegt sein Ursprung des *PageRank* Algorithmus im Ranking von Webseiten für Suchmaschinen [32]. Er besitzt die zwei wichtigen Eigenschaften der *propagation* und *attenuation* [36, 20]. Die erste besagt, dass sich relevante Information über mehrere Knoten hinweg ausbreitet, so dass auch Beziehungen zwischen Knoten hergestellt werden können, die keine direkte Kante besitzen. Diese Eigenschaft hilft dem Problem der *sparsity* entgegenzuwirken, da die latente Information hinter den transitiven Verbindungen ausgenutzt werden kann [36]. Die Eigenschaft der *attenuation* bedeutet, dass sich der Einfluss eines Knotens abschwächt, je weiter sich von diesem entfernt wird. In seiner ursprünglichen Form erstellt der *PageRank* Algorithmus ein globales Ranking für allen Knoten im Graphen, den *PageRank score*. Dieser *score* π ist durch eine rekursive Form definiert:

$$\pi = \alpha P\pi + (1-\alpha)\theta$$

wobei α ein *damping factor*[17], P die stochastische Übergangsmatrix und θ die Wahrscheinlichkeitsverteilung über die Startknoten bezeichnet. Für den normalen *PageRank* Algorithmus ist dies eine uniforme Verteilung über alle Knoten $\theta = \frac{1}{|P|}\mathbf{1}$ mit $\mathbf{1} = (1, 1, \ldots, 1)^T$.
Der *PageRank* folgt dabei dem oben beschrieben Prinzip des *random walk*: Der Läufer startet zufällig an einem Knoten, gegeben der Verteilung in θ. Mit einer Wahrscheinlichkeit von α wählt er eine ausgehende Kante. Weiter wird angenommen, dass der Läufer nach einiger Zeit stoppt und wieder an einen zufälligen Knoten bezüglich der Startverteilung beginnt. Dies erfolgt mit der Wahrscheinlichkeit von $1-\alpha$, welche deshalb diese *reset probability* genannt wird. [36] Für Empfehlungssysteme wird der *personalized PageRank* Algorithmus verwendet, eine Variante des *PageRank* [47, 36, 34, 20]. An Stelle der uniformen Startverteilung über alle Knoten, wird beim *personalized PageRank* Algorithmus die Startverteilung so gewählt, dass Knoten, welche das Nutzerprofil repräsentieren, eine hohe Wahrscheinlichkeit und alle anderen Knoten eine Wahrscheinlichkeit von Null erhalten. Der *personalized PageRank* Algorithmus berechnet nun das Ranking der Knoten, jedoch mit einer Ausrichtung an Interessen des Nutzers. Knoten, die ähnlich sind zu den ausgewählten Startknoten, erhalten ein höheres Ranking. Dazu wird der Vektor der Startverteilung θ durch einen Personalisierungsvektor p ersetzt. Gegeben einer *query* Menge $Q = e_1, e_2, \ldots e_k$ welche die Entitäten des Nutzerprofils enthält, wird p wie folgt definiert:

$$p_i = \begin{cases} w_{e_{v_i}} & \text{falls } e_{v_i} \in Q \\ 0 & \text{sonst.} \end{cases}$$

wobei $k = |Q|$ die Anzahl der Personalisierungsentitäten sei und e_{v_i} die Entität, welche dem Knoten v_i entspricht. Die Gewichte $w_{e_{v_i}}$ steuern den Einfluss der Entität e_{v_i} auf die Anfrage. Da der Vektor p eine Wahrscheinlichkeitsverteilung über die Startknoten angibt, muss dieser stochastisch sein, daher gilt: $\sum_{e_j \in Q} w_j = 1$. Der *personalized PageRank* ist somit definiert durch [36]: $\pi = \alpha P\pi + (1-\alpha)$
Die Menge Q enthält im einfachsten Fall den aktiven Nutzer: $Q = \{u_a\}$. Für die kontextsensitiven Empfehlungen werden zusätzlich die Entitäten hinzugefügt, die dem Nutzer

[16] `google.com` (letzter Zugriff: 01.05.2013)
[17] Ein gebräuchlicher Wert ist $\alpha = 0,85$ [36]

im aktuellen Kontext entsprechen: $Q = \{u_a, e_{u_a c_a}\}$.
Durch die Wahl der Startknoten ist eine große Flexibilität gegeben. Beispielsweise ließe sich die Empfehlung an eine Gruppe von Nutzern dadurch realisieren, dass gleich mehrere Startknoten „aktiviert“ werden.

Short-Term Preferences Es wurde gezeigt, dass sich hinter den unmittelbaren Aktionen eines Benutzers ein latenter, dynamischer Kontext verbirgt, welcher sein aktuelles Interesse bestimmt (siehe Kapitel 3.1.3). Nach der Idee von Xiang et al. [50], die *short-term preferences* eines Benutzers direkt in einem Graphen zu modellieren, wird in dieser Arbeit ein neuer Ansatz verfolgt: Die Objekte, die der Benutzer unmittelbar vor dem Zeitpunkt der Anfrage genutzt hat, werden in die *query* Menge mit aufgenommen. Die Intuition dahinter liegt darin, den Objekten, die von aktuellem Interesse sind, einen größeren Einfluss auf die Generierung der Empfehlungen zu geben. Dieser Einfluss lässt sich über die Gewichtung der Knoten im Personalisierungsvektor p kontrollieren.
Zur Beachtung der k letzten Objekte des aktiven Nutzers u_a zu einem Zeitpunkt t einer Anfrage, wird die *query* Menge Q wie folgt definiert:

$$Q = \{u_a, s_{t-1}, s_{t-2}, \ldots, s_{t-k}\}$$

Auswahl der Ergebnisse Der *PageRank* Algorithmus produziert einen *score* Vektor, welcher die Relevanz aller Knoten bezüglich der Startknoten enthält. Zum Zweck der Empfehlungen werden jedoch nur die Knoten benötigt, welche der Zielentität entsprechen. Daher werden nur diese als Empfehlungen ausgegeben. Das *recommendation set* sei als eine Menge von Tupeln definiert:

$$RS = \{(e_{v_i}, r_i) \,|\, e_{v_i} \in D_T,\ r_i = \pi_i\}$$

wobei D_T die Zielentität, e_{v_i} die Entität entsprechend des Knotens v_i und r_i den *PageRank score* für e_{v_i}, bezeichnet. Die Zielentität ist in diesem Fall die Menge der Tracks S, jedoch besteht die Möglichkeit, andere Zielentitäten zu wählen, um beispielsweise statt Tracks, andere Personen oder Genres vorzuschlagen.
Um die *Top-N recommendations* zu erhalten, wird die Menge RS absteigend nach dem *score* r_i sortiert. Anschließend werden die ersten N Elemente ausgegeben.

Berechnung Die Bestimmung des *PageRank score* Vektors kann auf die Berechnung des Eigenvektors zum Eigenwert 1 einer Matrix M zurückgeführt werden:

$$\pi = (\alpha P + (1-\alpha)E)\pi := M\pi$$

mit $E = (p, p, \ldots, p) = p\mathbf{1}^T$, $\mathbf{1} = (1, 1, \ldots, 1)^T$.
Die Berechnung erfolgt iterativ, wobei π mit einer uniformen Verteilung initialisiert wird. Der Pseudocode für die Berechnung des PageRank ist in Algorithmus 1 dargestellt.

Algorithm 1 Iterative Berechnung des *PageRank Score*

$\pi_0 \leftarrow \frac{1}{|P|}\mathbf{1}$
$\pi_1 \leftarrow \alpha P\pi_0 + (1-\alpha)p$
$k = 1$
while $|\pi_k - \pi_{k-1}| < \epsilon$ **do**
    $k \leftarrow k + 1$
    $\pi_k \leftarrow \alpha P\pi_{k-1} + (1-\alpha)p$
end while

Es wurde gezeigt, dass der Algorithmus bereits nach wenigen Iterationen konvergiert [32, 20].

In diesem Abschnitt wurden einige Überlegungen zur Aufnahme unterschiedlicher Informationen für die Modellierung des Graphen getroffen. Ferner wurde gezeigt, wie sich daraus Empfehlungen generieren lassen. Im nächsten Kapitel soll die Konstruktion eines Graphen im Detail beschrieben werden.

4.3 Erstellung eines Graphen am Beispiel

Kern dieser Arbeit ist die Entwicklung eines Empfehlungssystems für Musik, basierend auf einem Datensatz von LastFM[18]. Es handelt sich dabei um *implicit feedback* Daten, in Form eines *listen logs* [33]. Jeder Eintrag dieses Logs besteht aus einem Tupel $(User, Artist, Track, Timestamp)$. Ferner ist jeder *Track* durch eine Menge von Tags charakterisiert. Eine genaue Beschreibung des Datensatzes sowie der Gewinnung der Tags erfolgt im nächsten Kapitel.
Die Intuition bei der Verwendung von *implicit feedback* Daten liegt in der Annahme, dass die Präferenz des Nutzer für ein Objekt umso größer ist, je öfter er mit dem Objekt interagiert (z. B. in dem er einen Track anhört).

4.3.1 Beschreibung der Entitäten

Im Folgenden werden die verwendeten Informationen formal definiert.

Tracks: Die zu empfehlenden Objekte, auch als Zielentität bezeichnet. Sei S die Menge aller Tracks.

User: Die Benutzer des Systems. Sei U die Menge aller Benutzer.

Tags: Features der Tracks. Sei T die Menge aller Tags[19] und $T_s = \{(t, r_{st}) | t \in T\}$ eine Menge aus Tupeln, welche die relevanten Tags für einen Track $s \in S$ enthält, während $r_{st} \in [0, 1]$ die Relevanz des Tags t für Track s darstellt.

Artists: Tracks des selben Interpreten[20] stehen in Relation. Sei A die Menge aller Interpreten und $a_s \in A$ der Interpret von Track $s \in S$.

[18] `last.fm` (letzter Zugriff: 01.05.2013)
[19] Es wird die englische Bezeichnung *Tag* für Schlagwort verwendet
[20] Es wird die englische Bezeichnung *Artist* für Interpret verwendet

Context: Der Zeitpunkt einer Interaktion wird als statischer Kontext interpretiert. Eine kategorische Repräsentation kann auf verschiedene Arten erfolgen. Jahreszeit, Tageszeit und Wochenzeit sind einige Beispiele. Sei $C = \{C_1, \dots C_n\}$ die Obermenge aller temporalen Kontextdimensionen, $C_i = \{c_1, \dots, c_k\}$ die Menge aller Kategorien für Kontext C_i und $c = \{c_i \mid c_i \in \bigcup_{C_i \in C} C_i\}$ eine diskrete Repräsentation als eine Menge von Kategorien.

Zur Gewährung der Flexibilität des Systems, sollen einzelne Entitäten auswählbar sein und unterschiedliche Gewichtungen erhalten können, um ihren Einfluss auf die erzeugten Empfehlungen zu steuern. Zu diesem Zweck wird eine Menge aus Tupeln bestehenden Faktoren definiert: $F = \{(f_1, w_1), \dots, (f_{n_F}, w_{n_F})\}$. Jeder Faktor entspricht einer aktiven Entität, die nicht Ziel der Empfehlungen ist. Die Gewichtung des Faktors f_k sei gegeben durch w_k, wobei $\sum_{k=1}^{n_F} w_k = 1$. Beispielsweise drückt der Faktor $F = \{(f_1 = User, 0.5), (f_2 = Tag, 0.5)\}$ aus, dass Benutzerbewertungen und Tags verwendet werden und gleichermaßen Einfluss haben. [36]

4.3.2 Konstruktion des Graphen

Sei $G = \{V, E\}$ ein gerichteter, gewichteter Graph, bestehend aus der Menge der Knoten $V = V_S \cup V_{f_1} \cup \dots \cup V_{f_{n_F}}$, wobei V_S Knoten der Tracks S bezeichnet und V_{f_k} die Knoten des Faktors f_k. $E \subset V \times V$ sei die Menge aller Kanten. Ferner sei w_{ij} die Gewichtung der Kante $(v_i, v_j) \in E$. [47]
Die Modellierung von Kontextinformationen wird durch einen neuen Faktor realisiert, welcher einen Benutzer in dem jeweiligen Kontext repräsentiert. Dieser Faktor kombiniert die Entitäten *User* und *Context*. Es können Faktoren für jede einzelne Kontextdimension aber auch für Kombinationen dieser hinzugefügt werden [36]. Ein *User-Context* Faktor für die Kontextentität C_i sei gegeben durch die Menge der Kombinationen von Benutzern und Kontextkategorien: $f_{U,C_i} = \{\{u, c\} \mid u \in U, c \in C_i\}$.

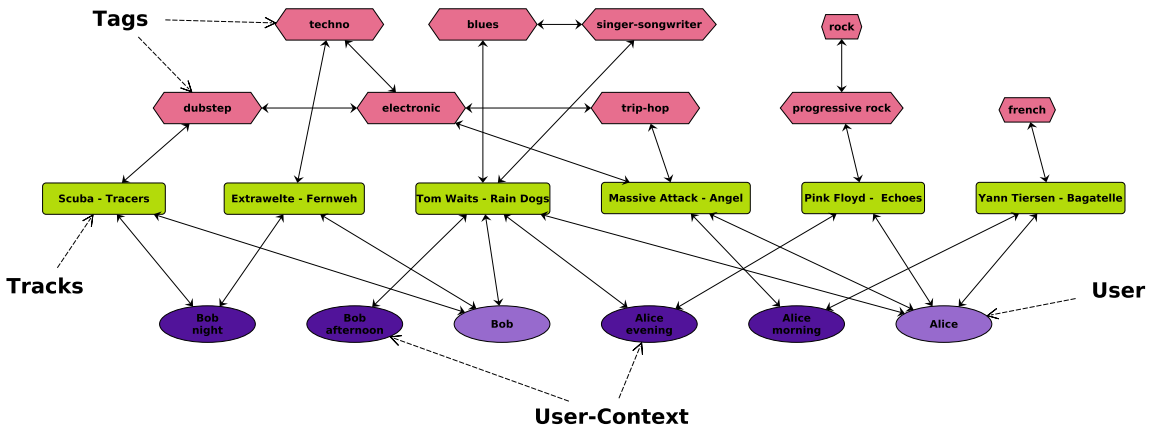


Abbildung 2: Beispiel für die Konstruktion eines Graphen mit *User, Track, Tag* und *User-Context* Knoten.

Kanten Die Kanten ergeben sich aus den Relationen zwischen den verwendeten Entitäten. Es möge Kanten zwischen gleichen und unterschiedlichen Entitäten geben. In Abb. 2 ist dargestellt, wie ein Graph mit den beschriebenen Entitäten aussehen könnte.
Eine Kante wird genau dann eingefügt, wenn die Gewichtung größer Null ist: $(v_i, v_j) \in E \Leftrightarrow w_{ij} > 0$. Der Track wurde also mindestens einmal gehört. Um die Vergleichbarkeit der Relationen verschiedener Entitäten zu gewährleisten, werden die Kantengewichte normiert, so dass $0 \leq w_{ij} \leq 1$.
Der Graph G sei repräsentiert durch seine Adjazenzmatrix M, deren Elemente w_{ij} die Gewichtung der Kanten (v_i, v_j) ausdrücken.
Die Matrix M unterteilt sich in mehrere Submatrizen, welche jeweils die Kanten zwischen den einzelnen Entitäten repräsentieren. Eine mögliche Zusammensetzung von M ist in Tabelle 2 illustriert.

	User	Tracks	Tags	User-Context
User	**0**	**SU**	**0**	**0**
Tracks	**US**	**SS**	**TS**	$\mathbf{U_C S}$
Tags	**0**	**ST**	**TT**	**0**
User-Context	**0**	$\mathbf{SU_C}$	**0**	$\mathbf{U_C U_C}$

Tabelle 2: Adjazenzmatrix M, aufgeteilt in Submatrizen

Die Submatrizen seien wie folgt definiert:

Users, Tracks: Sei $L = l_1, \ldots, l_{n_l}$ die Menge der *listen events* aus der Logtabelle, wobei die Einträge $l_i = (u, s, a, c)$ Tupel mit Benutzer, Track, Interpret und Kontext darstellen.
Die Gewichte der Kanten zwischen V_S und V_U lassen sich durch das Zählen der *co-occurences* von Tracks und Benutzern bestimmen[21]

$$w_{us} = \begin{cases} \frac{n_{track}}{n_{total\_tracks}}, & \text{falls } n_{total} > 0 \\ 0, & \text{sonst.} \end{cases}$$

$$w_{su} = \begin{cases} \frac{n_{track}}{n_{total\_users}} \cdot w_U, & \text{falls } n_{total} > 0 \\ 0, & \text{sonst.} \end{cases}$$

$n_{track} = |\{l_i \in L \,|\, u \in l_i \wedge s \in l_i\}|$ bezeichnet, wie oft Nutzer u den Track s gehört hat, $n_{total\_tracks} = |\{l_i \in L \,|\, u \in l\}|$ gibt die Anzahl aller *listen events* des Nutzers an und $n_{total\_tracks} = |\{l_i \in L \,|\, s \in l\}|$ die Anzahl der Nutzer, die Track s gehört haben. Weiter sei w_U der Einfluss des Faktors *(Users)*.

Eine Kante wird genau dann eingefügt wenn die Gewichtung größer Null ist: $(v_u, v_s) \in E \Leftrightarrow w_{us} > 0$. Der Track wurde also mindestens einmal gehört.

[21] *co-occurence* bezeichnet das gemeinsame Auftreten zweier Entitäten

Tracks, Tags: Ein Track erhält eine Kante zu jedem Tag, mit dem er assoziiert ist. Die Gewichtung der Kante ist durch die normierte Relevanz des Tags für den entsprechende Track gegeben:

$$w_{ts} = w_{st} = \begin{cases} r_{st} \cdot w_T & \text{falls } (t, r_{st}) \in T_s \\ 0 & \text{sonst.} \end{cases}$$

T_s ist die Menge der gewichteten Tags für Track s wie oben definiert und r_{st} die Relevanz. Der Einfluss des Faktors *(Tags)* ist durch w_T gegeben.

Tags, Tags: Zwischen zwei unterschiedlichen Tags lassen sich ebenfalls Kanten einfügen, um ihre Relation zueinander auszudrücken:

$$w_{tt'} = \begin{cases} sim(t, t') & \text{falls } sim(t, t') > \theta \\ 0 & \text{sonst.} \end{cases}$$

wobei $sim(t, t')$ eine Ähnlichkeitsfunktion zwischen zwei Tags definiert. Die Relation zwischen Tags kann asymmetrisch sein, um beispielsweise eine Hierarchie auszudrücken.
Die Ähnlichkeit zweier Tags lässt sich durch die Häufigkeit des gemeinsamen Auftretens in dem selben Dokument (hier: Tracks) bestimmen (*co-occurences*). Diese wird in der Regel normiert, um den starken Einfluss populärer Objekte zu vermindern.
In dieser Implementierung wird die *Jaccard distance* verwendet. Diese ist ein gebräuchliches Ähnlichkeitsmaß für Mengen, das im Kontext von *social tagging* Anwendung findet [39]:

$$sim(t, t') = sim_{Jaccard}(x, y) = \frac{|X \cap Y|}{|X \cup Y|}$$

Y und Y bezeichnen die Mengen der Objekte, mit denen die Tags x bzw. y assoziiert sind.
Ein Beispiel des sich daraus ergebenden Graphen ist in Abb. 3 dargestellt.

Tracks, Tracks: Tracks können untereinander in Relation stehen. In diesem Fall wird eine binäre Relation zwischen zwei unterschiedlichen Tracks definiert, wenn diese den selben Interpret haben:

$$w_{ss'} = w_{s's} = \begin{cases} 1 & \text{falls } a_s = a_{s'} \\ 0 & \text{sonst.} \end{cases}$$

wobei $a_s, a_{s'} \in A$ den Interpret von Track s bzw. s' bezeichnet.

Tracks, User-Context: Kontextabhängige Präferenzen werden durch Kanten zwischen den Knoten der Objekte und des *User-Context* Faktors beschrieben. Die Gewichtung lässt sich durch zählen der *co-occurences* von Track, Benutzer und Kontext in der Logtabelle ermitteln [36]:

$$w_{uc,s} = \begin{cases} \frac{n_{track}}{n_{total\_tracks}}, & \text{falls } n_{total} > 0 \\ 0, & \text{sonst.} \end{cases}$$

$$w_{s,uc} = \begin{cases} \frac{n_{track}}{n_{total\_users}} \cdot w_{UC}, & \text{falls } n_{total} > 0 \\ 0, & \text{sonst.} \end{cases}$$

$n_{track} = |\{l_i \in L \,|\, u \in l_i \wedge s \in l_i \wedge c \in l_i\}|$ bezeichnet, wie oft Nutzer u den Track s im Kontext $c \in C$ gehört hat, $n_{total\_tracks} = |\{l_i \in L \,|\, u \in l_i \wedge c \in l_i\}|$ gibt die die Anzahl aller *listen events* des Nutzers im Kontext c an und $n_{total\_users} = |\{l_i \in L \,|\, s \in l_i \wedge c \in l_i\}|$ die Anzahl der Nutzer die Track s in Kontext c gehört haben. Der Einfluss des Faktors *(User,Context)* sei bestimmt durch w_{UC}.

User-Context, User-Context: Es können Kanten zwischen verschiedenen *User-Context* Knoten existieren, um kontextabhängige Präferenzen anderer Benutzer einfließen zu lasse. Zwei *User-Context* Knoten besitzen genau dann eine Kante, wenn sie der selben Kontextkategorie angehören:

$$w_{uc_iuc_j} = w_{uc_iuc_j} = \begin{cases} 1 & \text{falls } c_i = c_j \\ 0 & \text{sonst.} \end{cases}$$

Es wird angenommen, dass der zeitliche Kontext ein vorwiegend lokaler Effekt ist und sich auf die individuellen Präferenzen des Nutzers beschränkt [50]. Andere Kontextfaktoren, wie z.B. der Ort, mögen jedoch durchaus über die Grenzen einzelner Benutzer hinweg relevant sein.

Alle wichtigen Methoden zur Entwicklung eines, auf Graphen basierenden, kontextsensitiven Empfehlungssystems wurden formal beschrieben. Im nächsten Kapitel soll dieses Wissen angewandt werden, um ein konkretes System zu implementieren und anhand realer Daten zu evaluieren.

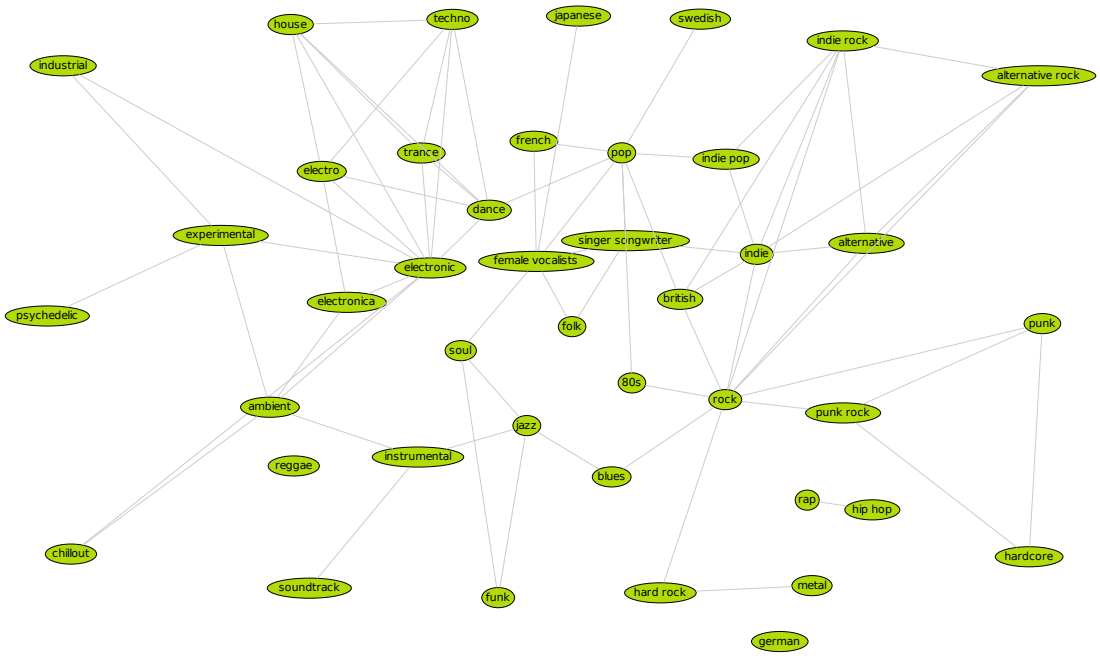


Abbildung 3: Korrelationsgraph der 42 häufigsten Tags des LastFM Datensatzes bei $\theta = 0.3$.

5 Implementierung

Dieser Kapitel ist in mehrere Abschnitte untergliedert. Zunächst soll der verwendete Datensatz genauer beschrieben werden. Weiter werden die eingesetzten Technologien sowie das entwickelte Framework vorgestellt. Im letzten Teil sollen verschiedene Ansätze mit unterschiedlichen Einstellungen evaluiert werden, um herauszufinden ob und in welcher Weise die Kontextinformationen Empfehlungen verbessern können.
Aufgrund des verwendeten Datensatzes, liegt der Fokus auf der Implementierung eines Empfehlungssystems für Musik. Diese ist jedoch allgemein gehalten, um auch in anderen Domänen Verwendung zu finden.

5.1 Datensatz

Das Anwendungsspektrum für Empfehlungssysteme ist breit, jedoch werden kontextsensitive Systeme bisher noch wenig eingesetzt, weshalb bisher kaum öffentlich verfügbare Datensätze mit Angaben zum Kontext erhältlich sind. Ein Grund dafür liegt vermutlich in der Schwierigkeit Kontextinformationen zu erheben. Es wurde die Möglichkeit genannt, diese explizit vom Benutzer zu erfragen. Das hat jedoch den Nachteil, dass es der Benutzerin lästig sein mag, diese Angaben jedesmal zu definieren, wenn sie eine Bewertung abgibt. Die weitere Möglichkeit ist die Gewinnung aus impliziten Daten, beispielsweise den Logs der Benutzerinteraktionen. Die Arbeit verwendet einen Datensatz von LastFM [22]. Musik stellt eine interessante Domäne für kontextsensitive Empfehlungssysteme dar. Es wurde gezeigt, dass Menschen zu temporalen Hörgewohnheiten neigen [26], was für die Verwendung von zeitlichen Kontextinformationen spricht. Weiter besteht eine starke Vermutung, dass sich hinter den kürzlich abgespielten Tracks ein latenter dynamischer Kontext verbirgt, der ausgenutzt werden kann, um Empfehlungen für die folgenden Tracks zu geben [22].
Bei dem Datensatz handelt es sich um den *lastfm-1K* Datensatz [23]. Er enthält die *listen logs* von ca. 1000 Benutzern, welche jedes Abspielen eines Tracks mitsamt eines *timestamps* verzeichnen.
Zusätzlich werden mithilfe der LastFM API Tags für jeden Interpreten ermittelt, welche als inhaltliche Features der Tracks Verwendung finden.

5.1.1 Beschreibung der Daten

Der Datensatz besteht aus zwei *TSV*[24] Dateien. Die erste enthält die *listen logs*, wobei jede Zeile ein *listen event* in Form eines Tupels <*User, Timestamp, Artist, Track*> repräsentiert.
Das exakte Format einer Zeile ist:

```
userid \t timestamp \t artistid \t artist-name
\t trackid \t trackname
```

[22] `last.fm` (letzter Zugriff: 01.05.2013)
[23] Die Daten werden mithilfe der LastFM API unter Benutzung der `user.getRecentTracks` Methode gesammelt. Der komplette Datensatz ist hier erhältlich:
`http://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html` (letzter Zugriff: 01.05.2013) [14]
[24] *Tab-Separated Values*

Der *timestamp* ist jeweils im UTC Format[25] gegeben. Die Felder *artistid* und *trackid* enthalten, falls vorhanden, jeweils eine eindeutige Identifikation des Interpreten bzw. Tracks durch die *musicbrainz* Datenbank[26] (MBID).
Die zweite Datei enthält Metadaten zu den Benutzern im folgenden Format:

```
userid \t gender ('m'|'f'|empty) \t age (int|empty)
\t country (str|empty) \t signup (date|empty)
```

Ein Beispiel für den Inhalt des Datensatzes ist in Listing 1 aufgeführt.

Listing 1: Ausschnitt aus dem Datensatz

```
userid-timestamp-artid-artname-traid-traname.tsv:

  user_000639 \t 2009-04-08T01:57:47Z \t MBID \t The Dogs D'
     Amour \t MBID \t Fall in Love Again?
  user_000639 \t 2009-04-08T01:53:56Z \t MBID \t The Dogs D'
     Amour \t MBID \t Wait Until I'm Dead
      ...

userid-profile.tsv:

  user_000639 \t m \t Mexico \t Apr 27, 2005
  ...
```

Eine statistische Übersicht des Datensatzes ist in Tabelle 3 gegeben.

Anzahl der Zeilen:	19,150,868
Anzahl der Benutzer:	992
Artists mit MBID:	107,528
Artists ohne MBDID:	69,420

Tabelle 3: Statistik des Datensatzes

5.1.2 Import der Daten

Um den Zugriff auf die Daten zu vereinfachen, werden die Rohdaten in eine SQLite[27] importiert.
Es werden nur Einträge mit gültiger MBID der Interpreten verwendet, da diese später zur Ermittlung der Tags benötigt wird. Das Datenbankschema ist in Abb. 4 dargestellt.

[25] *Coordinated Universal Time*
[26] `musicbrainz.org` (letzter Zugriff: 01.05.2013)
[27] Eine einfache, dateibasierte SQL Datenbank `www.sqlite.org` (letzter Zugriff: 01.05.2013)

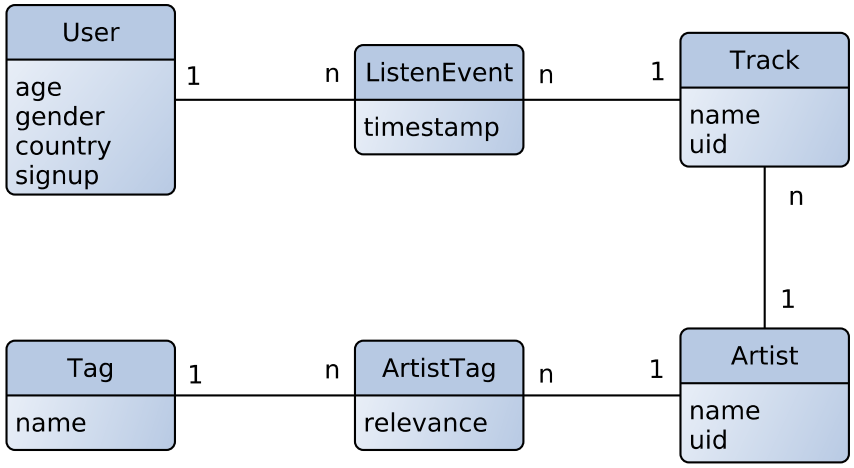


Abbildung 4: Datenbankschema

Lokalisierung der *Timestamps* Die *timestamps* der einzelnen Logeinträge sind normalisiert auf UTC. Um eine Vergleichbarkeit der Tageszeit von Nutzern unterschiedlicher Zeitzonen zu gewährleisten, müssen die *timestamps* auf eine lokale Zeit umgestellt werden. Hierzu werden die Zeitverschiebungen zur UTC der jeweiligen Herkunftsländer ermittelt, falls diese im Datensatz vermerkt sind. Für Länder mit mehreren Zeitzonen[28], wird die mittlere Zeitverschiebung aller Zeitzonen bestimmt. Die *timestamps* aller Logeinträge werden entsprechend angepasst.

5.1.3 Gewinnung der Tags

Die *artist tags* werden über die LastFM API mit der Methode `artist.getTopTags`[29] bezogen. Die erhält eine Artist MBID und liefert eine Liste der am meisten verwendeten Tags für diesen Interpreten zurück, absteigend geordnet nach der Relevanz. Ein weiteres Feld enthält die Relevanz als einen Wert zwischen 0 und 100. Es werden jeweils die ersten 5 Tags verwendet. Ein Beispiel einer Antwort ist in der Tabelle 4 dargestellt.

Name	**Relevanz**
alternative	100
alternative rock	79
rock	71
indie	58
electronic	47

Tabelle 4: Beispiel einer API Antwort der `artist.getTopTags` Methode für den Interpreten „Radiohead“

[28] z.B. USA, Russland

[29] `http://www.last.fm/api/show/artist.getTopTags` (letzter Zugriff: 01.05.2013)

Normalisierung Da unterschiedliche Konventionen in der Benennung der Tags vorliegen, werden diese im nächsten Schritt durch einfache Heuristiken normalisiert. Folgende Anpassungen werden gemacht:

Kleinschreibung: Alle Namen werden kleingeschrieben. Beispiel: *Rock* → *rock*

Trennzeichen: Alle Bindestriche („-“) werden durch Leerzeichen ersetz. Beispiel: *hip-hop* → *hip hop*

Trennung: Für Tags, die aus mehreren Wörtern bestehen, wird geprüft ob auch eine zusammengeschriebene Form existiert. Falls ja, wird diese ebenfalls getrennt. Beispiel: Gibt es neben *hip hop* auch *hiphop*, so wird letztere getrennt.

Anschließend werden alle Tags entfernt, deren Häufigkeit unter einer bestimmten Schwelle f_{min} gewählt liegt.
Alle übrigen Tags werden in der Datenbank gespeichert, zusammen mit der Relevanz für den jeweiligen Interpreten (siehe Abb. 4).

Statistik Im verwendeten Datensatz befinden sich insgesamt 83982 Interpreten. Für 73774 dieser Interpreten wurden Tags über die LastFM API gefunden, wobei jeweils die 5 relevantesten ausgewählt wurden.
Insgesamt wurden 21296 verschiedene Tags gefunden. Nach der Normalisierung bleiben davon 7110 Tags übrig. Von diesen werden schließlich die 400 häufigsten Tags ausgewählt[30]. Die durchschnittliche Anzahl an Tags pro Interpret beträgt $3,76$.

Korrelationsmatrix Über das gemeinsame Auftreten eines Tags beim gleichen Interpret lässt sich eine Relation zwischen zwei Tags ermitteln. Die Berechnung der Korrelationsmatrix erfolgt nach dem Verfahren wie in Kapitel 4.3.2 beschrieben.

5.1.4 Probleme, Entscheidungen

Zum Import des kompletten *listen log* müssen ca. 20Mio *records* in die Datenbank eingefügt werden. Das Einfügen einzelner *records* ist zeitintensiv[31]. Wesentlich effizienter ist es, mehrere *records* in einer Transaktion zusammenzufassen. Auch das Laden der gesamten Datei von 2.4GB mit einem Mal würde zu viel Arbeitsspeicher verbrauchen. So werden jeweils Blöcke von 100.000 Zeilen aus der Datei gelesen und in einer Transaktion in die die jeweiligen Tabellen der Datenbank geschrieben. Die dafür benötigten Schlüssel werden bereits vorher berechnet, um Leseoperationen zu vermeiden.

Die Auswahl einer Teilmenge der Daten zur Evaluierung, wird in Kapitel 5.4.1 beschrieben.

[30] Für die Schwelle wird $f_{min} = 100$ verwendet
[31] `sqlite.org/speed.html` (letzter Zugriff: 01.05.2013)

5.2 Technologien

In diesem Abschnitt soll eine kurze Übersicht über die Technologien, die in der Implementierung zum Einsatz kommen, gegeben und deren Auswahl begründet werden.

5.2.1 Programmiersprache

Das gesamte Framework ist in der Programmiersprache Python geschrieben. Python ist eine einfache, jedoch sehr mächtige Programmiersprache mit klarer und verständlicher Syntax. Sie unterstützt mehrere Programmierparadigmen, wobei insbesondere objektorientierte- und funktionale Programmierung eine flexible Entwicklung ermöglichen. Zusammen mit einem leistungsstarken Modulsystem, das es erlaubt einzelne Funktionen aus verschiedenen Dateien zu importieren, ist es möglich, Programme sehr modular aufzubauen und den Quellcode gut zu strukturieren. Objektorientierung vereinfacht die Wiederverwendung von Code, was besonders bei der Entwicklung von verschiedenen Algorithmen, die eine ähnliche Schnittstelle aufweisen, von Vorteil ist.
Wie auch bei anderen interpretierten Programmiersprachen, fördert Python einen schnellen *workflow*, da nicht kompiliert werden muss. In Verbindung mit einer interaktiven Konsole, wie *ipython*[32], ist es möglich einzelne Funktionen manuell zu testen sowie Daten und Ergebnisse zu untersuchen.
Darüber hinaus bietet Python ein beachtliches Ökosystem mit vielen hochwertigen Erweiterungen, insbesondere im wissenschaftlichen Bereich.

5.2.2 Datenbank

Zum Speichern des Datensatzes kommt SQLite mit der Programmierschnittstelle SQLAlchemy zum Einsatz.

SQLite ist eine leichtgewichtige relationale Datenbank, die den SQL Standard implementiert. Alle Daten werden dabei in einer einzelnen Datei gespeichert, was die Verteilung auf mehrere Rechner erleichtert. Im Gegensatz zu anderen SQL Datenbanken wie PostgreSQL und MySQL[33] erfordert SQLite keinerlei Konfiguration. Des Weiteren ist SQLite sogar in vielen Fällen schneller als seine Konkurenten. Insbesondere bei mehreren `INSERT` Operationen innerhalb einer Transaktion ist SQLite bis Faktor 5 schneller im Vergleich zu PostgreSQL.[34] Da beim Importieren des kompletten Datensatzes ca. 20Mio *records* eingefügt werden müssen, ist dieser Vorteil signifikant.

SQLAlchemy ist ein populärer *object-relational mapper* für Python[35].

5.2.3 Weitere Bibliotheken

SciPy/NumPy SciPy ist eine umfassende Sammlung von Open Source Software für wissenschaftliche Programmierung.[36] Mit NumPy stellt sie eine exzellente Numerik Bib-

[32] `ipython.org` (letzter Zugriff: 01.05.2013)
[33] `postgresql.org`, `mysql.com` (letzter Zugriff: 01.05.2013)
[34] `sqlite.org/speed.html` (letzter Zugriff: 01.05.2013)
[35] `sqlalchemy.org` (letzter Zugriff: 01.05.2013)
[36] `scipy.org` (letzter Zugriff: 01.05.2013)

liothek bereit, die viele nützliche Funktionen zum effizienten Verarbeiten von großen Matrizen enthält. Dabei besitzt NumPy ein intuitives Interface, welches an das von MATLAB angelehnt ist.
Da alle verwendeten Algorithmen sehr große Matrizen verwenden, ist eine speichereffiziente Darstellung sowie eine schnelle Operation auf diesen, entscheidend. NumPy unterstützt die Verwendung von hochoptimierten Bibliotheken für lineare Algebra[37] und ist dadurch bis zu Faktor 50 schneller als äquivalente Implementierungen in puren Python Code.[38]
Ein weiterer Bestandteil des SciPy Frameworks ist Matplotlib, welches zum Zeichnen von Graphen und Diagrammen Verwendung findet.

NetworkX ist eine Python Bibliothek für das Erstellen, Verarbeiten und Plotten von Graphen.[39] Der Vorteil liegt bei dem einfachen Interface sowie der Möglichkeit, Attribute für Knoten zu speichern. Es lassen sich schnell und einfach Knoten und Kanten hinzufügen. Die resultierenden Graphen sind in Form von Adjazenzmatrizen als *NumPy* Objekte exportierbar, welche in der Implementierung der Algorithmen genutzt werden.

5.3 Framework

Im Folgenden soll das Framework vorgestellt werden, das im Rahmen der Arbeit implementiert wurde. Es wird eine Beschreibung der einzelnen Komponenten, sowie der gesamten Systemarchitektur gegeben. Jede Komponente wird durch ein Python Module repräsentiert. Ein Modul besteht aus einer Datei, die Klassen oder Sammlung von Funktionen enthalten kann. Mehrere zusammenhängende Module werden in einem Python Paket zusammengefasst. Nach Konvention werden Modulnamen stets klein und Klassenamen groß geschrieben. Im Folgenden wird eine Übersicht der verschiedenen Pakete und Komponenten gegeben, wobei eine Komponente entweder als eine Klasse oder ein Modul mit Funktionen repräsentiert sei. Der Zusammenhang der wichtigsten Komponenten ist in Abb. 5 verdeutlicht.

[37] z.B. LAPACK (www.netlib.org/lapack/) (letzter Zugriff: 01.05.2013)
[38] Ein Vergleich mehrerer Implementierungen ist hier zu finden: *wiki.scipy.org/PerformancePython* (letzter Zugriff: 01.05.2013)
[39] networkx.github.io

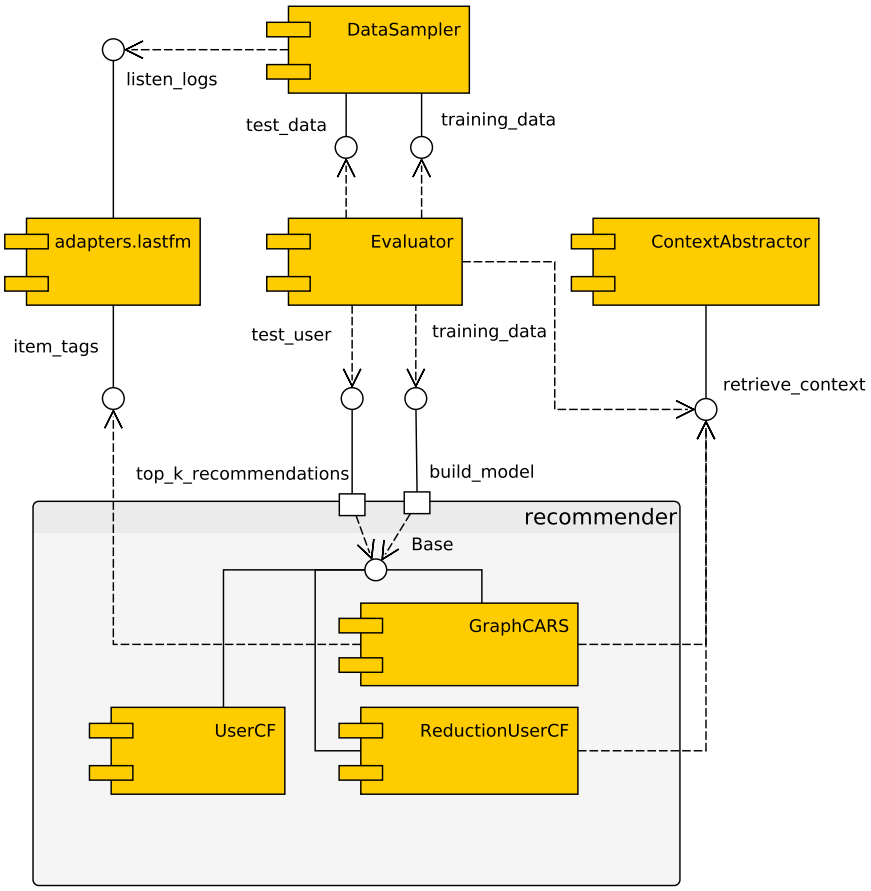


Abbildung 5: Systemarchitektur

adapters Um die implementierten Empfehlungssysteme von der Anwendungsdomäne zu entkoppeln, werden alle anwendungsspezifische Module in einem *adapters* Paket zusammengefasst. Dieses enthält vor allem Methoden zum Import und Aufbereiten der Daten, sowie die *ORM layer*.
Es wird ein domänenagnostisches Interface bereitgestellt, welches von der Semantik der Daten weitgehend abstrahiert. Beispielsweise werden Tracks als *items*[40] behandelt. Die Semantik des gemeinsamen Interpreten mehrerer Tracks wird durch eine *item-group* abgebildet, die mehrere in Beziehung stehende *items* zusammenfasst.
Da in dieser Arbeit mit LastFM nur eine Anwendung implementiert wird, enthält dieses Paket lediglich den *lastfm* Adapter. Die einzelnen Module sind:

load_data Dieses Modul enthält alle Funktionen zum Import und Aufbereiten der Daten.

models ORM Layer für den Datenbankzugriff.

api Client für die LastFM API.

crawler Laden der Tags über die LastFM API.

[40]Der englische Begriff *items* bezeichnet hier Objekte

item_adapter Zugriff auf die *items* und *item-goups*

tag_adapter Zugriff auf die *items tags*

recommender Dieses Paket enthält die Implementierungen der verschiedenen *recommendation* Algorithmen. Jeder Algorithmus ist durch eine eigene Klasse implementiert. Dabei wird jeweils von einer gemeinsamen Basisklasse geerbt, die ein einheitliches Interface zur Anfrage von Empfehlungen definiert. Folgende Klassen sind im *recommender* Paket enthalten:

Base Die Basisklasse für Recommenderalgorithmen. Sie definiert das Interface.

UserCF Diese Klasse enthält eine einfache Implementierung der *user-based collaborative filtering* Methode.

ReductionUserCF Enthält eine kontextsensitive Erweiterungen für das *user-based collaborative filtering* und erbt von der Klasse *UserCF*. Es wird das *reduction-based* Verfahren implementiert (siehe Kapitel 3.2.1).

GraphCars Implementiert das graphbasierte, kontextsensitive Verfahren (siehe Kapitel 4.3).

Weitere Komponenten

ContextAbstractor Die Klasse *ContextAbstractor* dient dazu, einen *timestamp* in feste Kategorien zu überführen. Es können zwei verschiedene Granularitätslevel eingestellt werden. Die erste gibt den Wochentag an (*day of week*), welcher den Zeitraum einer Woche unterteilt. In dieser Implementierung werden die zwei Kategorien Wochentag und Wochenende verwendet.
Ein feinere Einteilung erlaubt die Tageszeit (*time of day*). Als Grenzen sind die Stunde $dayhours = [2, 6, 9, 11, 13, 18, 22]$ vorgegeben. Andere Einstellungen sind möglich. Die Repräsentation der Kategorien erfolgt durch eine Nummerierung. Wird der Kontext *day of week* verwendet, entsprechen die Zahlen 0 und 1 den Kategorien Wochentag und Wochenende. Beide Kontextdimensionen können verbunden werden, wobei die einzelnen Kategorien dem kartesischen Produkt beider einzelnen Dimensionen entsprechen:

$$\begin{aligned} C &= C_{dow} \times C_{tod} \\ &= \{(Wochentag, morgens), \ldots, (Wochenende, nachts)\} \\ &\equiv \{0, 1, \ldots, |C_{dow}| \cdot |C_{tod}| - 1\}. \end{aligned}$$

Evaluator In dieser Klasse werden Funktionen zur Evaluierung der Algorithmen implementiert.

DataSampler Die Klasse dient der der Erstellung von Teilmengen des Datensatzes zur Evaluierung. Ein Teil der Daten wird aus der Datenbank geladen und nach verschiedenen Kriterien gefiltert, beispielsweise um nur Benutzer zu erhalten, die über eine mindest Anzahl von Logeinträgen verfügen.

random_walk Implementierung des *PageRank* Algorithmus.

numeric Numerische Funktionen, z.B. *cosine similarity*.

serialize Funktionen zum Import und Export der trainierten *recommender* Klassen.

tags Funktionen zur Aufarbeitung der Tags, wie Normalisierung, Entfernen von unrelevanten Einträgen und Berechnung der Korrelationsmatrix.

5.4 Evaluierung

In diesem Abschnitt soll das implementierte graphbasierte Verfahren (GraphCARS) evaluiert werden[41]. Zum Vergleich steh: Ein einfacher, auf Nachbarschaft von Benutzern basierender *collaborative filtering* Ansatz (UserCF) sowie dessen multidimensionale Erweiterung, das *reduction-based collaborative filtering* (ReductionUserCF). Wie gezeigt wurde, birgt das graphbasierte Verfahren eine hohe Flexibilität durch die Modellierung von Wissen aus unterschiedlichen Quellen. Durch Gewichtung kann der Einfluss verschiedene Entitäten auf die Empfehlungen verändert werden. Die verschieden Parameter ergeben viele Möglichkeiten für Experimente.
Zunächst soll getestet werden, ob das graphbasierte Verfahren in seiner einfachsten Form eine Verbesserung gegenüber des naiven UserCF Ansatzes bringt. Dies ist zu vermuten, da hier indirekte Relationen zwischen verschiedenen Benutzern und Objekten ausgenutzt werden können, was bei der UserCF Methode nicht möglich ist.
Im nächsten Schritt werden Stück für Stück verschiedene Informationen in die Modellierung des Graphen mit aufgenommen, um ihren Einfluss auf die Qualität der Empfehlungen zu testen.
Eine Erweiterung ist die Aufnahme von inhaltlichen Informationen (*content profiles*) in Form von Tags, welche die Objekte beschreiben. Ferner wurde eine Möglichkeit vorgestellt, die Relation zwischen diesen Tags zu ermitteln und als zusätzliches Wissen zu verwenden. Schließlich soll der temporale Aspekt der Benutzeraktionen ausgenutzt werden um treffendere Empfehlungen für den aktiven zeitlichen Kontext zu machen.
Für den ReductionUserCF Ansatz wurde bereits gezeigt, dass er mit realen Daten nicht gut funktioniert, da durch das *pre-filtering* Bewertungsdaten, die nicht dem aktiven Kontext entsprechen, herausgefiltert werden. Werden zu viele Daten entfernt, verschlechtert sich die Genauigkeit der Empfehlungen. Interessant ist, ob der graphbasierte Ansatz eine Verbesserung durch Nutzung des zeitlichen Kontextes bringt. Da kontextuelle Bewertungen (eine Teilmenge aller Bewertungen) als zusätzliche Information im Graphen modelliert werden, während die Information aus den nicht kontextuellen Bewertungen bestehen bleibt, ist eine Verbesserung möglich, jedoch keine Verschlechterung der Ergebnisse zu erwarten.

Dieser Abschnitt ist wie folgt gegliedert: Zu Beginn werden verschiedene Verfahren und Metriken zur Evaluierung vorgestellt und der Einsatz argumentativ begründet. Anschliessend werden eine Reihe Experimente und ihre Ergebnisse vorgestellt. Im darauf folgenden Abschnitt wird erörtert, wie diese zu interpretieren sind.

5.4.1 Verfahren

Ziel der Evaluierung ist es, Empfehlungssysteme in ihrer Leistung hinsichtlich der ihnen gestellten Anforderungen zu bewerten. In den meisten Fällen hat ein Empfehlungssystem die Aufgabe, einem Benutzer nützliche Objekte zu präsentieren. Verschiedene Algorithmen werden danach verglichen, wie gut sie dieser Aufgabe nachkommen. Es gibt zwei Wege dies zu testen: Naheliegend ist die Auswertung durch reale Benutzer, die mit dem System interagieren, auch Online Evaluation genannt. Deutlich häufiger in der Anwendung findet

[41] CARS steht für *Context Aware Recommender System*

sich jedoch die Methode der Offline Evaluation. Im Folgenden werden beide Verfahren genauer beschrieben. [21]

Online Evaluation Da Empfehlungssysteme bestrebt sind, gute und dem Benutzer zufriedenstellende Empfehlungen zu liefern, liegt es nahe, diese mit Hilfe der Benutzer zu bewerten. Ein Online Experiment könnte so aussehen, dass dem Benutzer Objekte vorgeschlagen werden und dieser explizit beurteilt, wie sinnvoll ihm diese Empfehlungen erscheinen.
Eine weitere Möglichkeit besteht darin, durch implizites Feedback der Benutzer auf die Nützlichkeit der Empfehlungen zu schließen, etwa ob ein Benutzer auf die vorgeschlagen Objekte reagiert.
Online Evaluation stellt damit ein gutes Mittel dar, um die Leistung realer Systeme zu prüfen. Ein Nachteil ist jedoch der hohe Aufwand, da für eine aussagekräftige Beurteilung viele Benutzer und Aktionen benötigt werden.

Offline Evaluation Eine günstigere Alternative zur Online Evaluation ist es, Experimente offline durchzuführen. Zu diesem Zweck werden Datensätze verwendet, von denen angenommen wird, dass sie dem angestrebten Einsatzbereich des zu testenden Systems gerecht werden. Zur Verringerung der Kosten der Evaluierung kann eine kleinere Teilmenge des Datensatzes ausgewählt werden, wobei auf eine gleichmäßige Verteilung von Benutzern und Präferenzen geachtet werden sollte, etwa durch zufälliges sampeln von Daten. Eine systematische Auswahl ist zu vermeiden, da dies Neigungen einbringen kann, die das Ergebnis verfälschen.
Zur Bestimmung der Genauigkeit eines Algorithmus, werden Vorhersagen über die Nützlichkeit oder Bewertung von Objekten gemacht und mit bekannten Werten verglichen. In der Regel wird dazu der Datensatz in zwei Hälften geteilt. Während die eine, welche das schon bekannte Wissen repräsentiert, zum Training des Algorithmus verwendet wird, dient die andere Hälfte zum Testen der Genauigkeit der Vorhersagen.
Nach Möglichkeit sollte bei den Offline Experimenten ein reales Online System simuliert werden. In vielen Anwendungsbereichen spielt der temporale Aspekt von Bewertungen und Interaktionen eine Rolle, da sich die Präferenzen eines Nutzers mit der Zeit verändern können [30]. Daher ist es sinnvoll, diesen auch in der Evaluation zu berücksichtigen. Beinhaltet der Datensatz den *timestamp* der jeweiligen Aktionen, lässt sich daraus der Online Prozess nachspielen. Idealerweise werden zu jeden Zeitpunkt Vorhersagen über die zukünftigen Aktionen gemacht, während alle bis dahin gesehen Daten zum Training des Algorithmus verwendet werden. Das System macht nun eine Vorhersagen, deren Präzision sich mit Hilfe der bekannten, zukünftigen Daten bestimmen lässt.
Dieses Vorgehen ist sehr rechenintensiv, da sich die Trainingsdaten für jeden Test verändern. Alternativ kann ein fester Zeitpunkt gewählt werden, anhand dessen die Daten aufgeteilt werden. Für jeden Nutzer werden alle Aktionen bis zu diesem Zeitpunkt als bekannt betrachtet, alle zukünftigen Aktionen werden zum Testen vorenthalten. Auch könnte ein zufälliger Zeitpunkt für jeden einzelnen Benutzer gewählt werden, unter der Annahme, dass es keine zeitliche Korrelation zwischen den Aktionen verschiedener Benutzer gibt, ihre Reihenfolge jedoch relevant ist. [21, 12]
Zur besseren Berücksichtigung der unmittelbar vergangenen Bewertung in temporalen Daten, eignet sich der *all but last* Ansatz. Bei dieser Methode wird für jeden Benutzer

die jeweils letzte Bewertung zum Testen benutzt, alle vorgehenden Daten werden zum Training verwendet [50, 12, 22]. Dies kommt der Online Evaluation recht nahe, hat jedoch den Nachteil, dass für jeden Benutzer nur ein Datum zum Testen zur Verfügung steht.
Um die Robustheit der zu testenden Algorithmen unter Variation der Trainingsdaten sicherzustellen, hat sich in der Forschung die Methode der *n-fold cross validation* bewährt. Bei diesem Verfahren wird der Datensatz in n Partitionen geteilt, wobei jeweils $n-1$ Teilmengen als Trainingsdaten benutzt werden. Der verbleibende Teil dient zum Testen. Das Experiment wird nun n Mal wiederholt, so dass jede Partition genau einmal als Testdatensatz fungiert. Dieses Verfahren ist jedoch in seiner klassischen Form nicht für die temporale Partition der Daten geeignet. Es widerspräche der Grundidee, lediglich Daten aus Vergangenheit zur Vorhersage der Zukunft zu verwenden, nicht jedoch umgekehrt. Eine Erweiterung der *cross-validation* für temporale Evaluation präsentiert Campos et al. Diese basiert auf der erneuten Unterteilung der Daten nach Benutzern. Die Evaluation wird mit jeweils $n-1$ Teilmengen von Benutzern gemacht wird, wobei Trainings- und Testdaten weiterhin zeitlich aufgeteilt werden. [12]
Gunawardana et al. argumentiert, dass die Methode der *cross-validation* für Empfehlungssysteme nicht notwendigerweise relevant sei, da das einfache Verfahren der Vorenthaltung einer Teilmenge ohne *cross-validation* bereits genügend Daten liefere, um verlässliche Schätzungen zu machen. [21].

Evaluation mit Kontext Zur Evaluation von kontextsensitiven Systemen (CARS) ist es zudem wichtig, den verwendeten Kontext auch in den Tests zu berücksichtigen. Mit Hilfe von CARS sollen Empfehlungen gemacht werden, die in einem gegebenen Kontext relevant sind. Im Rahmen einer Evaluation ist zu bewerten, ob und wie gut dieses Ziel erreicht wird. Dazu werden zunächst die Testdaten nach Kontextkategorien partitioniert. Für jede Kategorie wird eine Teilmenge erstellt, welche alle Aktionen oder Bewertungen des jeweiligen Benutzers in diesem Kontext enthält. Im Anschluss daran werden Empfehlung für jede dieser Kontextkategorien erstellt und mit den Testdaten des jeweiligen Kontextes verglichen, um die Genauigkeit der Vorhersagen abzuschätzen.

Grenzen der Offline Evaluation Alle Verfahren der Offline Evaluation haben den Nachteil, dass Bewertungen nicht für alle Objekte bekannt sind, da Benutzer in der Regel nur wenige Objekte bewerten, im Vergleich zur Gesamtzahl aller Objekte im System (*sparsity*). Es könnte weit mehr Objekte geben, die einem Benutzer gefallen, als dieser bereits bewertet hat. Die Genauigkeit der Vorhersagen für nicht bekannte Objekte ist folglich nicht zu ermitteln. Dies ist besonders dann problematisch, wenn das System darauf ausgerichtet ist, dem Nutzer neue, ihm noch nicht bekannte Objekte vorzustellen.
Eine weitere Schwäche der Offline Evaluation ist, dass nur geschätzt werden kann, ob Voraussagen zutreffend sind. Ein Maß für Zufriedenheit des Nutzers mit dem System ist daraus nicht unbedingt abzuleiten. Dies ist nur durch explizite Online Experimente möglich. [25]

Verfahren in dieser Implementierung Da der Schwerpunkt dieser Arbeit auf der Ausnutzung von Zeit als Kontextfaktor liegt, ist es plausibel, eine temporale Unterteilung der Daten für die Evaluierung vorzunehmen.
Aus dem gesamten Datensatz, der in Form zeitlich geordneter *listen events* vorliegt, wird

eine Teilmenge in dem Zeitraum von 17.04.2009 - 19.06.2009 zur Evaluation ausgewählt. Es wurde ein Zeitraum gegen Ende des Datensatzes gewählt, da hier relativ viele Benutzer aktiv waren. Weiter wurden alle Benutzer entfernt, die weniger als 9 *listen events* besitzen, so auch Tracks, die weniger als 9 mal gehört wurden. Die Hälfte der Benutzer (524) und 95% der Tracks (133403) wurden entfernt. Es verbleiben 524 Benutzer, 6419 Tracks und insgesamt 105717 *listen events*. Dies ist eine starke Verkleinerung zu Gunsten der Laufzeit. Die Einschränkung der Daten wird als kritisch beurteilt, da ungewollte Tendenzen entstehen können. Durch das Filtern werden unter anderem populäre Tracks bevorzugt [21]. Dieser Schritt wurde dennoch vorgenommen, da durch die Komplexität und den Speicherverbrauch der Algorithmen in dieser Implementierung eine Evaluation auf der zur Verfügung stehenden Hardware nicht durchführbar wäre.

Abbildung 6: Temporale Aufteilung der *listen events* eines Nutzers Traings- und Testdaten im Verhältnis 80:20

Zur Erstellung der Trainings- und Testdaten, werden die *listen logs* für jeden Benutzer zeitlich aufgeteilt. Hierfür wird kein fester Zeitpunkt für alle Benutzer gewählt. Die Aufteilung erfolgt nach Anzahl der Events im Verhältnis 80:20, um gleiche Anteile an Trainings- und Testdaten zu erhalten (*temporal split*). Eine schematische Darstellung der Aufteilung ist in Abb. 6 gezeigt.
Zur Evaluierung der *Short-term preference* Methode (Kapitel 4.2.2) wird eine Variation des *all but last* Verfahrens angewandt. Die Erstellung der Trainingsdaten geschieht weiterhin durch nach der *temporal split* Methode, die verbleibenden Events der Benutzer werde anschließend in Partitionen der Größe von $k+1$ Events aufgeteilt. Von diesen Teilmengen werden jeweils m zufällig ausgewählt[42]. Der letzte Eintrag einer $k+1$ Partition dient als Testdatum, die vorhergehenden k Einträge werden dem Empfehlungssystem als a priori Wissen der Anfrage übermittelt.

Das Verfahren sieht also wie folgt aus:

- Für jeden Benutzer werden Trainings- und Testdaten erstellt.
- Die Trainingsdaten aller Benutzer werden zur Konstruktion des Modells verwendet
- Es werden zufällig 50 Benutzer zur Evaluation ausgewählt.

Da das System dem Nutzer neue, ihm noch unbekannte Tracks vorschlagen soll, werden die bereits in den Trainingsdaten enthaltenen Tracks, aus den generierten Empfehlungen

[42] In diesem Verfahren wird $m = 10$ gewählt

entfernt.
Zur Gewährleistung der Vergleichbarkeit verschiedener Algorithmen in der Evaluierung, ist auf verschiedene Kriterien zu achten. Zum einen sind die gleichen Training- und Testdaten zu verwenden. Beim zufälligen sampeln von Daten, ist zu beachten, dass diese nicht für jeden Testlauf variieren[43]. Zum anderen ist eine einheitliche Metrik zur Bewertung der Qualität zu benutzt.

5.4.2 Metriken

In der Literatur werden verschiedene Metriken zur Bewertung von Empfehlungssystemen aufgeführt. Welche anzuwenden ist, hängt von der Art der Algorithmen und den zu evaluierenden Anforderungen an das System ab.
Die Evaluierung zielt in der Regel auf die Genauigkeit der Vorhersagen ab, jedoch können Systeme auch hinsichtlich spezieller Eigenschaften, wie z. B. die Diversität [37, 51, 52], die *serendepity* [38] und das Ranking [25] der Empfehlungen, untersucht werden.
Zur Evaluierung von klassischen *collaborative filtering* Methoden werden häufig Fehlermetriken, wie *Mean Absolute Error* (*MAE*) oder *Root Mean Squared Error* (*RMSE*) eingesetzt, welche die mittlere Abweichung zwischen Vorhersagen und bekannten Bewertungen misst [10, 25].
Der *MAE* ist wie folgt definiert:

$$E_{MAE} = \frac{\sum_{i=1}^{n} |r_i - p_i|}{n}$$

wobei p_i die prädizierte und r_i die echte Bewertung des Objekts s_i durch einen Benutzer bezeichnet. n gibt die Anzahl der zu testenden Objekte an.
Dieses Maß eignet sich jedoch nur für prädiktive Algorithmen, welche die tatsächlichen Bewertungen vorhersagen.
Für Top-N Algorithmen, die vor allem bei impliziten Daten eingesetzt werden, sind andere Metriken zu verwenden [29, 36].
Sehr populär im Bereich der *information retrieval* sind die Methoden *precision* und *recall* [1, 25, **?**]. *Precision* gibt einen Wert für die Wahrscheinlichkeit an, dass ein vorgeschlagenes Objekt relevant für den Nutzer ist. Sie ergibt sich aus der Zahl der relevanten Empfehlungen im Verhältnis zu allen Empfehlungen. Der *recall* hingegen stellt die Wahrscheinlichkeit dar, dass ein relevantes Objekt vorgeschlagen wird. Er ergibt sich aus der Zahl der relevanten Empfehlungen im Verhältnis zu der Anzahl aller relevanten Objekte.
Sei N die Anzahl der Top-N Empfehlungen, N_{rs} die Zahl der relevanten Empfehlungen und N_s die Zahl aller relevanten Objekte, dann sind *precision* und *recall* wie folgt definiert [25]:

$$precision = \frac{N_{rs}}{N}$$

$$recall = \frac{N_{rs}}{N_s}$$

[43] Dies ist zum Beispiel durch Verwendung des gleichen *seed* für den Zufallsgenerator zu erreichen.

Hit Ratio Da für Empfehlungssysteme die Relevanz in der Regel nicht für alle Objekte bekannt ist (*sparsity*), hat sich zu diesem Zweck eine Variation der ursprünglichen Definition des *recall* etabliert, welche diesen approximiert [25, 29, 47]. In der Literatur wird diese oft als *hit-ratio* oder auch *HR@N* aufgeführt aufgeführt, wobei N die Länge der Empfehlungsliste bezeichnet [36, 34, 35, 50]. Sei T die Menge der Testdaten, welche eine Teilmenge der für den Benutzer relevanten Objekte repräsentiert. Gegeben der Menge der Top-N Empfehlungen RS_N, misst *hit-ratio* die relative Anzahl der durch die Empfehlungen getroffenen Objekte in T:

$$hr(N) = \frac{|T \cap RS_N|}{|T|}$$

Für den kontextsensitiven Fall ist diese dahingehend zu erweitern, dass die Treffer in einem bestimmten Kontext, relativ zu allen relevanten Objekten in diesem Kontext bestimmt werden. Gegeben sei eine kontextuelle, nicht leeren Teilmenge $\mathcal{T} = \{T_c \subseteq T \,|\, c \in C \wedge T_c \neq \emptyset\}$, welche jeweils die in Kontext c relevanten Objekte enthält. Ferner sei $RS_N(c)$ die Menge der Top-N Empfehlungen in Kontext c. Die kontextsensitive *hit-ratio* lässt sich somit wie folgt definieren:

$$hr_{context}(N) = \frac{1}{|\mathcal{T}|} \sum_{T_c \in \mathcal{T}} \frac{|T_c \cap RS_N(c)|}{|T_c|}$$

Zur Evaluierung der *short-term preference* Methode, wird das oben beschriebene *all but last* Verfahren verwendet. Gegeben einer Sequenz der der $k+1$ kürzlich benutzten Objekte $S = \{s_1, \ldots, s_{k+1}\}$, wird gezählt, ob dieses durch die Empfehlungen getroffen wird. Sei $\mathcal{S} = \{S \subseteq T | S \neq \emptyset\}$ die Menge der nicht leeren Sequenzen S, dann ist die *hit-ratio* für die *short-term* Empfehlungen $RS_N(S)$ definiert durch:

$$hr_{short\_term}(N) = \frac{1}{|\mathcal{S}|} \sum_{S \in \mathcal{S}} \chi_{RS_N(S)}(s_{k+1})$$

wobei $\chi_T(\cdot)$ die Indikatorfunktion, s_{k+1} das zurückgehaltene Testobjekt der Sequenz S und k Anzahl der zu beachtenden Objekte (Fenstegröße) bezeichnet.

Die beschriebenen Metriken dienen zur Messung der Genauigkeit der Vorhersagen für einen einzelnen Benutzer. Das Experiment wird für alle zu testenden Benutzer wiederholt und anschließend der mittlere Wert bestimmt.

5.4.3 Experimente

Es wurden drei verschiedene Versuchsreihen durchgeführt. Ziel der erste Reihe ist es, die Leistung des graphbasierten Ansatzes mit den Baseline Methoden zu vergleichen, jeweils mit und ohne Kontextinformationen. Im zweiten Versuch liegt der Fokus auf der Analyse des Einflusses der Kontextinformationen und inhaltlichen Features im graphbasierten System. Es werden jeweils verschiedene Einstellung miteinander verglichen. Im letzten Experiment wird der Einfluss der *short-term preferences* genauer untersucht, insbesondere in Hinsicht auf die Optimierung der Fenstergröße.
Zum Vergleich verschiedener Ansätze, werden in den ersten beiden Versuchsreihen die

Experimente jeweils für die Werte $N \in \{5, 10, 15, 30\}$ der Top-N Empfehlungen wiederholt. Mit der Variation dieses Parameters lässt sich die Qualität des Rankings beurteilen, welches in der *hit-ratio* Metrik selbst nicht berücksichtigt wird [25, 36]. Vergleichsweise gute Ergebnisse für kleine N deuten darauf hin, dass der Algorithmus in der Lage ist, relevante Objekte an einer höheren Stelle in der Liste einzuordnen.

Übersicht der Algorithmen Im Folgenden werden eine Übersicht zur Benennung der verschiedenen Algorithmen und Einstellungen gegeben. Die verwendeten Kontextinformationen und Features werden jeweils an durch Abkürzungen der Namen des Algorithmus angehängt.

UserCF: Der einfache *k-nearest-neighbor colloaborative filtering* Ansatz.

ReductionUserCF_TOD: Kontextuelle Erweiterung des UserCF mit Kontextfaktor *time of day*.

ReductionUserCF_DOW: Verwendung des Kontextfaktors *day of week*.

GraphCARS: Der einfache graphbasierte Ansatz ohne Kontext.

GraphCARS_TOD: Der kontextuelle GraphCARS mit Kontextfaktor *time of day*.

GraphCARS_DOW: Verwendung des Kontextfaktors *day of week*.

GraphCARS_Tags: Aufnahme der Tags als Attribute der Tracks.

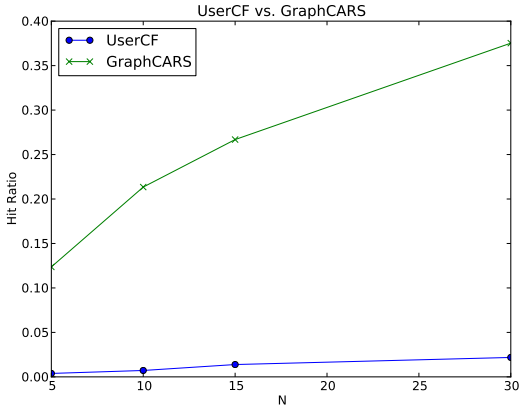
GraphCARS_Tags_rel: Direkte Relationen zwischen Tags aus der berechneten Korrelationsmatrix ($\theta = 0.3$).

GraphCARS_Artist: Aufnahme der Information über den Interpreten. Diese wird durch Kanten zwischen den Tracks eines Interpreten ausgedrückt.

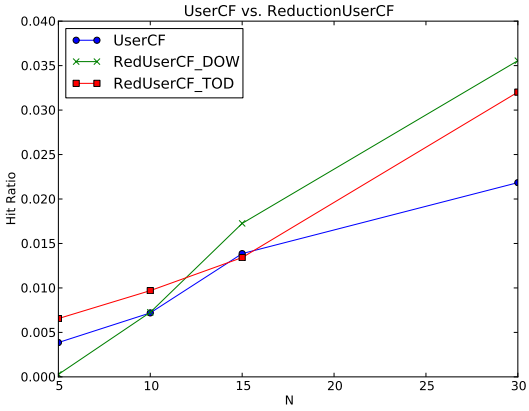
GraphCARS_STP: Verwendung der *short-term preferences*. Die Fenstergröße k gibt die Anzahl zuletzt gehörten Tracks des Benutzers an, die beachtet werden.

Der Kontextfaktor *time of day* ist unterteilt in 7 Tageszeiten, mit den Grenzen $dayhours = [2, 6, 9, 11, 13, 18, 22]$, während der Faktor *day of week* in die 2 Kategorien Wochenende und nicht Wochenende unterteilt ist.
Kombinationen mehrerer Einstellungen werden durch Aneinanderfügen der jeweiligen Suffixe gekennzeichnet. Die Kombination beider Kontextfaktoren wird durch eine einzelne Kontextdimension der Größe $|C| = |C_{dow} \times C_{tod}| = 2 \cdot 7 = 14$ ausgedrückt.

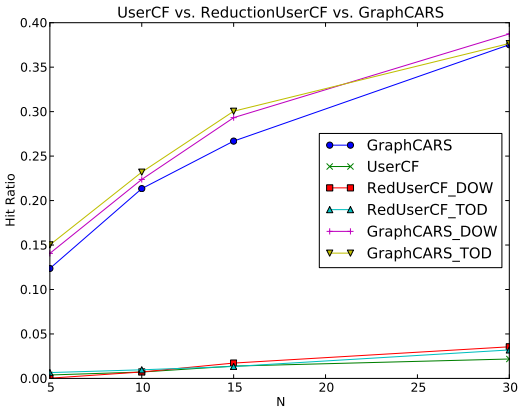
Vergleich mit Baseline Methoden Zunächst soll das graphbasierte Verfahren mit den *collaborative filtering* Methoden vergleichen werden. Für die Anzahl der nächsten Nachbarn wird in beiden Fällen der Wert $k = 5$ gewählt. Dieser wurde empirisch für den verwendeten Datensatz ermittelt. Für Gewichte der Faktoren für GraphCARS mit Kontext wird jeweils $F = \{(User, 0.5), (UserContext, 0.5)\}$ gewählt. Die Ergebnisse dieser Versuchsreihe sind in Abb. 7 dargestellt.



(a) GraphCARS und UserCF ohne Kontext



(b) Einfluss von Kontextfaktoren im *collaborative filtering*



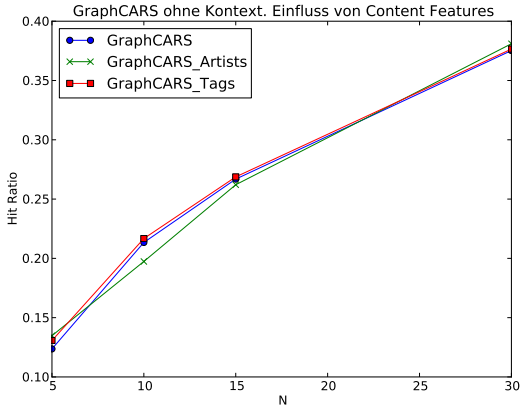
(c) Alle Methoden mit und ohne Kontext

Abbildung 7: Vergleich der *Hit Ratio*: Baseline Methoden und graphbasierten Ansatz

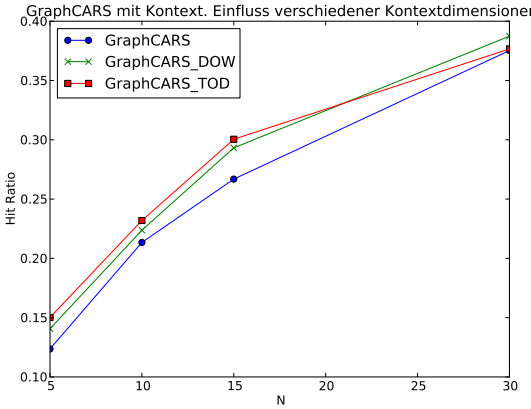
Vergleich unterschiedlicher Graphen In dieser Versuchsreihe werden dem Graphen verschiedene Informationen hinzugefügt, um jeweils ihren Einfluss auf die erzeugten Empfehlungen zu testen. Als Baseline Methode dient der einfache GraphCARS ohne zusätzliche Informationen. Die Faktoren werden gleich gewichtet:

$$F_1 = \{(User, 1.0)\}$$
$$F_2 = \{(User, 0.5), (UserContext, 0.5)\}$$
$$F_3 = \{(User, 0.5), (Tags, 0.5)\}$$
$$F_4 = \{(User, 0.33), (UserContext, 0.33), (Tags, 0.33)\}$$

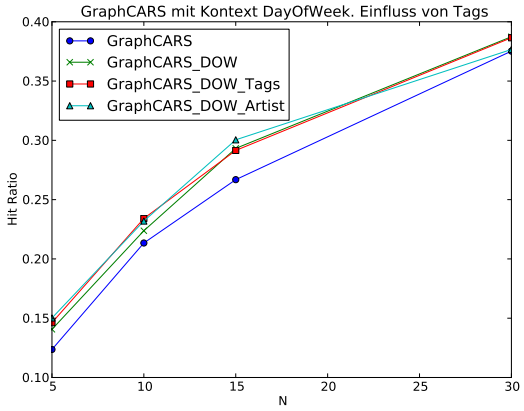
Abb. 8 zeigt die Ergebnisse der *Hit Ratio* für verschiedene Kombinationen. Eine detaillierte Übersicht aller getesteten Einstellung ist in Tabelle 5 dargestellt.



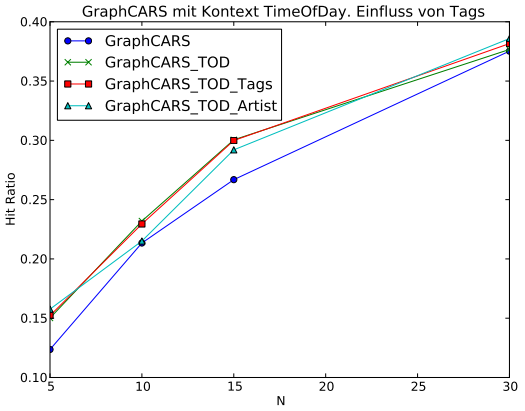
(a) Einfluss von Tags und Interpreten, ohne Kontext



(b) Einfluss der zeitlichen Kontextfaktoren



(c) Kontextfaktor *day of week* mit Tags

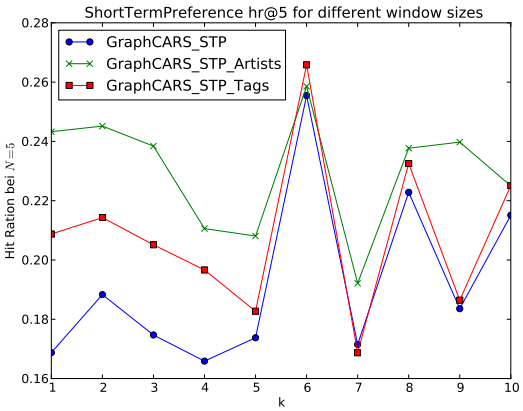


(d) Kontextfaktor *time of day* mit Tags

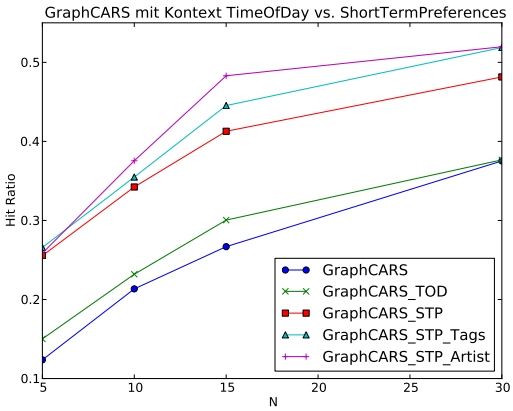
Abbildung 8: Vergleich der *Hit Ratio*: Verschiedene Graphen

Short-Term Preferences Die letzte Versuchsreihe widmet sich der Untersuchung der *short-term preferences*. Die *short-term preferences* sind durch eine Liste der zuletzt gehörten Tracks festgelegt (siehe Kapitel 4.2.2). Die Länge dieser Liste gibt an, wie weit ihr Einfluss zurückreicht. In dieser Implementierung wird eine feste Länge k nach Anzahl der Tracks gewählt (Fenstergröße). Die Länge des Zeitraums, in welchem diese liegen, wird nicht berücksichtigt. Auch wird in der Evaluierung nicht darauf geachtet, ob diese aus einer zusammenhängenden Session stammen.
Für die Evaluation ist zunächst interessant, wie sich die Fenstergröße k auf die Genauigkeit der Vorhersagen auswirkt. Zu diesem Zweck wird die *hit ratio* für ein festes $N = 5$ (HR@5) bei einer variablen Fenstergröße berechnet. Im nächsten Schritt werden zudem Tags und Interpreten mit aufgenommen. Die Ergebnisse sind in Abb. 9a dargestellt.
Im zweiten Test wird ein Vergleich des GraphCARS_STP Verfahrens zu der kontextfreien GraphCARS Methode und der besten kontextuellen Methode aus der zweiten Versuchsreihe (GraphCARS_TOD), angestellt. Für die Fenstergröße wird der aus dem ersten Versuch

empirisch ermittelte Wert $k = 6$ verwendet. Abb. 9b zeigt die Ergebnisse der *hit ratio* für $N \in [5, 10, 15, 30]$.



(a) Vergleich der HR@5 des GraphCARS_STP bei verschiedenen Fenstergrößen



(b) Vergleich von GraphCARS_STP zu GraphCARS mit und ohne Kontext. Fenstergröße: $k = 6$

Abbildung 9: Einfluss der *short-term preferences*

Algorithmus	HR@5	HR@10	HR@15	HR@30
UserCF	0.004	0.007	0.014	0.022
RedUserCF_DOW	0.000	0.007	0.017	0.036
RedUserCF_TOD	0.007	0.010	0.013	0.032
GraphCARS	0.124	0.213	0.267	0.375
GraphCARS_Artists	0.135	0.197	0.262	0.381
GraphCARS_Tags	0.131	0.217	0.269	0.377
GraphCARS_Tags_Artists	0.137	0.201	0.268	0.376
GraphCARS_Tags_rel	0.131	0.217	0.269	0.376
GraphCARS_DOW	0.141	0.224	0.293	0.387
GraphCARS_DOW_Tags	0.146	0.234	0.291	0.387
GraphCARS_DOW_Artist	0.150	0.232	0.300	0.377
GraphCARS_TOD	0.150	0.232	0.300	0.377
GraphCARS_TOD_Tags	0.152	0.229	0.300	0.382
GraphCARS_TOD_Artist	0.158	0.215	0.292	0.386
GraphCARS_DOW_TOD	0.141	0.234	0.301	0.385
GraphCARS_STP	0.255	0.342	0.413	0.482
GraphCARS_STP_Tags	**0.266**	0.355	0.445	0.519
GraphCARS_STP_Artist	0.259	**0.376**	0.483	0.520

Tabelle 5: Ergebnisse aller Algorithmen im Überblick für die *hit-ratio* HR@N

5.5 Diskussion der Ergebnisse

Im Wesentlichen entsprechen die Ergebnisse allen Erwartungen. Die erste Versuchsreihe zeigt eine deutliche Überlegenheit des graphbasierten Ansatzes. Das vergleichsweise schlechte Abschneiden der naiven UserCF Methoden, etwa um den Faktor 30 (Abb. 7a und 7c) erstaunt zunächst. Lee et al. gibt einen Verbesserung durch das graphbasierte Verfahren gegenüber der UserCF Methode mit einem Faktor von ca. 7 bei einem ähnlichen Datensatz[44] an. Für die UserCF Methode werden bessere, für das graphbasierte Verfahren etwas schlechtere Ergebnisse, als in dieser Implementierung genannt. [36] Die starke Divergenz ist vermutlich auf den Datensatz zurückzuführen. In dieser Implementierung werden vergleichsweise wenig Benutzer verwendet. Das Verhältnis der Benutzer zu Tracks ist in dieser Implementierung ca. 1:12, Lee et al. gibt ein Verhältnis von ca. 1:2 an. Eine weiterer Unterschied liegt in der Länge der zeitlichen Periode aus welcher die Daten stammen. In dieser Implementierung liegen die Bewertungen in einem Zeitraum von 2 Monaten, Lee et al. hingegen gibt einen Zeitraum von fast 12 Monaten an. Ob das verwendete Evaluierungsverfahren ebenfalls auf einer zeitlichen Aufteilung des Datensatzes basiert, ist nicht bekannt. Experimente haben gezeigt, dass eine zeitliche Vergrößerung des Datensatzes möglicherweise mit einer sinkenden Leistung des graphbasierten Verfahrens korreliert[45]. Dies könnte auf die Relevanz des temporalen Aspekts bei der Empfehlungen von Musik hindeuten: Bewertungen, die länger zurück liegen mögen sich von dem dem aktuellen Interesse unterscheiden und könnten daher einen negativen Einfluss auf die Empfehlungen haben.
Das vergleichsweise gute Abschneiden des GraphCARS spricht außerdem für ein großes Potenzial der transitiven Relationen über mehrere Benutzer hinweg.
Bei der Verwendung zeitlicher Kontextinformationen in der UserCF Methode (ReductionUserCF) ist ein positiver Einfluss auf die Genauigkeit der Vorhersagen zu erkennen. Für beide Faktoren liegt die *hit ratio* durchwegs über den Werten der nicht kontextuellen Methode (Abb. 7b). Der Kontext *time of day* bewirkt eine deutliche Verbesserung des Rankings, während durch den Kontext *day of week* mehr relevante Empfehlungen gemacht werden. Diese sind jedoch weiter hinten in den Top-N Empfehlungen gelistet.
Die Verbesserung durch die Aufnahme von der *content profiles* Tags, und Interpreten hingegen, ist marginal (Abb. 8a). Beide haben jeweils einen leichten, positiven Einfluss auf das Ranking, vermögen es jedoch nicht, mehr relevante Objekte zu entdecken. Bei der Benutzung von Tags und Interpreten zusammen, ist ebenso eine sehr leichte Verbesserung des Rankings gegenüber der Einzelnen zu sehen (Tabelle 5).
Die Modellierung direkter Relationen zwischen Tags hingegen, blieb ohne Auswirkung (Tabelle 5). Dies überrascht nicht – diese Information ist im Graphen bereits durch indirekte Verbindungen der Tags mit den Trackknoten vorhanden. Zusätzliche direkte Verbindungen bringen kein neues Wissen in das System.
Die Verwendung von Kontextinformationen im graphbasierten System bewirkt einen deutlichen Anstieg der Leistung, sowohl im Bezug auf das Ranking, als auch auf die generelle Fähigkeit, richtige Vorhersagen zu treffen (Abb. 8b). Auch hier verzeichnet der Faktors *time of day* einen stärkeren Einfluss gegenüber dem Faktor *day of week*. Während dessen Bedeutung in der ReductionUserCF Methode für längere Empfehlungslisten sinkt, vermag

[44] Der Datensatz stammt ebenfalls von LastFM

[45] Eine exakte Analyse konnte aufgrund der langen Laufzeit bei größeren Datenmengen leider nicht durchgeführt werden.

der GraphCARS_TOD auch für größere Werte von N relevante Empfehlungen zu finden. Die leicht verminderte Leistung des ReductionUserCF unter Verwendung des *time of day* Kontextes, ist mit der Abnahme der Bewertungsdichte (*sparsity*) bei steigender Anzahl der verschiedenen Kontextkategorien zu erklären. Für den GraphCARS Ansatz ist damit gezeigt, dass dieses Problem weniger stark ins Gewicht fällt.
Die Kombination beider Kontextfaktoren im Graphen (GraphCARS_DOW_TOD) bringt jedoch keine Verbesserung gegenüber der nicht kontextuellen Variante (siehe Tabelle 5). Die Anzahl der Kategorien ist auch hier zu groß, um eine hinreichende Menge an kontextuellen Bewertungen zu erhalten.

Im Vergleich zu dem Einfluss der statischen Kontextfaktoren und inhaltlichen Features, bewirkt die Berücksichtigung der dynamischen *short-term preferences* eine massive Verbesserung der Leistung. Ein interessantes Ergebnis zeigt die Betrachtung unterschiedlicher Fenstergrößen (Abb. 9a). Unter der Annahme, dass innerhalb einer Session ähnliche Tracks gehört werden, könnte der Graph die durchschnittliche Länge einer Session widerspiegeln. Ein erster Peak ist bei $k = 2$ zu beobachten, was damit zu erklären wäre, dass in vielen Fällen drei ähnliche Tracks aufeinander gehört werden. Die Berücksichtung der letzten zwei Tracks führt zu einer plausiblen Vorhersage des nächsten Tracks. Werden mehr Tracks hinzugenommen, sinkt die Genauigkeit wieder. Möglicherweise haben Tracks aus der vorhergehenden Session einen negativen Einfluss. Ein charakteristischer Wert für die Fenstergröße scheint $k = 6$ zu sein. Für diesen ist die größte Genauigkeit zu erzielen, während für $k = 7$ ein Minimum verzeichnet wird. Dies könnte auf eine durchschnittliche Session Länge von 7 Tracks hindeuten. Weitere Peaks bei $k = 8$ und $k = 10$ könnten ebenfalls mit einer relativen Häufung von Wiedergabelisten der Längen 9, bzw. 10 erklärt werden.
Ein interessantes Resultat zeigt auch die Auswirkung der *content profiles* auf die *short-term preferences*. Gerade für kleine Playlisten ist durch die Verwendung der Informationen über Tags und Interpreten eine markante Verbesserung der Vorhersagen zu erreichen, wobei der Einfluss des Interpreten deutlicher ausgeprägter ist. Dies ist darauf zurückzuführen, dass in der Regel mehrere Tracks des selben Interpreten oder Genre hintereinander gehört werden.
Für die optimale Fenstergröße $k = 6$ zeigt die Verwendung der *short-term preference* eine deutliche Überlegenheit gegenüber den vorherigen Verfahren. Durch die Aufnahme der Informationen über Tags und Interpreten ist eine weitere Steigerung der Leistung möglich (Abb. 9b).

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

In dieser Arbeit wurde ein Einblick in die aktuelle Forschung im Bereich der Empfehlungssysteme gegeben. Die wichtigsten Anforderungen an die Systeme sowie ihre Umsetzungen wurden vorgestellt. Ferner wurden Probleme und Defizite verschiedener Ansätze aufgezeigt und mögliche Lösungswege erörtert.
Mit dem Kontext wurde ein wichtiger Faktor präsentiert, der in klassischen Systemen oft wenig Beachtung findet. Es wurde argumentiert, dass die Nutzung von Kontextinformationen eine Verbesserung der Empfehlungen in bestimmten Situationen ermöglichen kann und somit zu größerer Nutzerzufriedenheit führt.
Verschiedene Möglichkeiten der Gewinnung der Kontextinformationen wurden vorgestellt, ebenso die unterschiedlichen Ansätze ihrer Verarbeitung in konkreten Anwendungen.
Auf den Ergebnissen der aktuellen Forschung basierend, wurde schließlich ein auf Graphen aufbauender Algorithmus entwickelt, welcher das Wissen aus verschiedenen Informationsquellen zu nutzen vermag. Ferner wurde der Ansatz des klassischen *collaborative filtering*, sowie eine kontextuelle Erweiterung implementiert. Die Evaluation erfolgt anhand eines geeigneten Datensatzes im Hinblick auf die Leistung, Flexibilität und Relevanz verschiedener kontextueller und inhaltlicher Information.
Der positiver Einfluss der zeitlichen Kontextfaktoren auf die Qualität der Empfehlungen hat sich in beiden kontextuellen Ansätzen erwiesen. Weiter wurde gezeigt, dass sich mithilfe des graphbasierten Systems nicht nur deutlich bessere Ergebnisse gegenüber der klassischen Methoden erzielen lassen, sondern durch die Möglichkeit der Aufnahme verschiedener Informationen auch eine hohe Flexibilität gegeben ist. Jede zusätzlich relevante Information, seien es Kontext oder Metadaten, ermöglicht eine Verbesserung der Ergebnisse. Ferner lassen sich die Stärken der einzelnen Faktoren durch ihre Kombination zusammenführen. Mit der Berücksichtigung der Kurzzeitinteressen eines Nutzers wurde eine besonders leistungsfähige Methode gefunden, treffende Empfehlungen zu machen.

6.2 Ausblick

Das entwickelte Verfahren hat sich als vielversprechend erwiesen. Es wurde ein generisches Modell präsentiert, das mit den getesteten Daten hervorragende Ergebnisse erzielt. Hierbei standen jedoch nur wenige Kontextinformationen zur Verfügung. Es wurden weitaus mehr Möglichkeiten solcher Informationen vorgestellt. Die Anwendung des Modells auf andere Domänen, sowie die Nutzung anderer Kontext- und Metainformation, wie zum Beispiel der Ort, Wetter und demographische Angaben, birgt großes Potential.
Die Flexibilität des Modells eröffnet viele Möglichkeiten, die es genauer zu erforschen gilt. Beispielsweise könnten durch die Wahl einer anderen Zielentität auch Personen, Interpreten oder Genres vorgeschlagen werden. Ferner stellt das System eine Vielzahl von Parametern bereit, die Kontrolle über den Einfluss verschiedener Faktoren ermöglichen. Eine genaue Analyse der Auswirkung dieser Einstellungen bleibt offen. Eine Evaluierung und Optimierung des Modells in Hinblick auf andere Anforderungen, beispielsweise die Diversität der Empfehlungen, ergibt eine interessante weitere Forschungsfrage. Ansätze dazu wurden bereits in [37, 51] vorgestellt.
Die Berücksichtigung der Kurzzeitinteresse demonstriert einen maßgeblichen Einfluss auf

die Genauigkeit der Vorhersagen. Variationen in der Gewichtung der zuletzt verwendeten Objekte könnten weitere Verbesserungen bringen. So wäre es denkbar, Objekte antiproportional zu der zurückliegenden Zeit zu gewichteten.
Eine Komplexitätsanalyse des Algorithmus bleibt aus. Es muss darauf hingewiesen werden, dass der Algorithmus in dieser Implementierung, sowohl im Speicherverbrauch, als auch in der Laufzeit, schlecht skaliert. Dabei ist anzumerken, dass dieses Verfahren im wesentlich auf dem *personalized PageRank* Algorithmus basiert. Dank seiner großen Popularität sind die Möglichkeiten zur Optimierung und Skalierung bereits sehr gut erforscht [7, 23].

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Abbassi, Z., and Mirrokni, V. S. A recommender system based on local random walks and spectral methods. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis* (New York, NY, USA, 2007), WebKDD/SNA-KDD '07, ACM, pp. 102–108.

[2] Adomavicius, G., Mobasher, B., Ricci, F., and Tuzhilin, A. Context-aware recommender systems. *AI Magazine 32*, 3 (2011), 67–80.

[3] Adomavicius, G., Sankaranarayanan, R., Sen, S., and Tuzhilin, A. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst. 23*, 1 (Jan. 2005), 103–145.

[4] Adomavicius, G., and Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING 17*, 6 (2005), 734–749.

[5] Adomavicius, G., and Tuzhilin, A. Context-aware recommender systems. In *Recommender Systems Handbook*. 2011, pp. 217–253.

[6] Augeri, C., and of Technology, A. F. I. *On Graph Isomorphism and the PageRank Algorithm*. Air Force Institute of Technology, 2008.

[7] Bahmani, B., Chowdhury, A., and Goel, A. Fast incremental and personalized pagerank. *Proc. VLDB Endow. 4*, 3 (Dec. 2010), 173–184.

[8] Bazire, M., and Brézillon, P. Understanding context before using it. In *CONTEXT* (2005), pp. 29–40.

[9] Bogers, T. Movie Recommendation using Random Walks over the Contextual Graph. In *Second Workshop on Context-Aware Recommender Systems* (2010).

[10] Breese, J. S., Heckerman, D., and Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence* (San Francisco, CA, USA, 1998), UAI'98, Morgan Kaufmann Publishers Inc., pp. 43–52.

[11] Burke, R. The adaptive web. Springer-Verlag, Berlin, Heidelberg, 2007, ch. Hybrid web recommender systems, pp. 377–408.

[12] Campos, P. G., Díez, F., and Sánchez-Montañés, M. Towards a more realistic evaluation: testing the ability to predict future tastes of matrix factorization-based recommenders. In *Proceedings of the fifth ACM conference on Recommender systems* (New York, NY, USA, 2011), RecSys '11', ACM, pp. 309–312.

[13] Cantador, I., Bellogín, A., and Vallet, D. Content-based recommendation in social tagging systems. In *RecSys* (2010), pp. 237–240.

[14] Celma, O. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[15] Chein, M., and Mugnier, M.-L. *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*, 1 ed. Springer Publishing Company, Incorporated, 2008.

[16] Dourish, P. What we talk about when we talk about context. *Personal and Ubiquitous Computing 8*, 1 (2004), 19–30.

[17] Fouss, F., Pirotte, A., Renders, J.-M., and Saerens, M. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Trans. Knowl. Data Eng. 19*, 3 (2007), 355–369.

[18] Gladwell, M. The tipping point: How little things can make a big difference, 2002.

[19] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. Using collaborative filtering to weave an information tapestry. *Commun. ACM 35*, 12 (Dec. 1992), 61–70.

[20] Gori, M., and Pucci, A. Itemrank: A random-walk based scoring algorithm for recommender engines. In *IJCAI* (2007), pp. 2766–2771.

[21] Gunawardana, A., and Shani, G. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research 10* (2009), 2935–2962.

[22] Hariri, N., Mobasher, B., and Burke, R. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems* (New York, NY, USA, 2012), RecSys '12', ACM, pp. 131–138.

[23] Haveliwala, T., Kamvar, A., and Jeh, G. An analytical comparison of approaches to personalizing pagerank. Tech. rep., 2003.

[24] He, J., and Chu, W. W. A social network-based recommender system (snrs). In *Data Mining for Social Network Data*. 2010, pp. 47–74.

[25] Herlocker, J. L., Konstan, J. A., Terveen, L. G., John, and Riedl, T. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems 22* (2004), 5–53.

[26] Herrera, P., Resa, Z., and Sordo, M. Rocking around the clock eight days a week: an exploration of temporal patterns of music listening. In *1st Workshop On Music Recommendation And Discovery (WOMRAD), ACM RecSys, 2010, Barcelona, Spain* (Barcelona, 26/09/2010 2010).

[27] Kaminskas, M., and Ricci, F. Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review 6*, 2-3 (2012), 89–119.

[28] Karatzoglou, A., Amatriain, X., Baltrunas, L., and Oliver, N. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems* (New York, NY, USA, 2010), RecSys '10', ACM, pp. 79–86.

[29] Karypis, G. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the tenth international conference on Information and knowledge management* (New York, NY, USA, 2001), CIKM '01', ACM, pp. 247–254.

[30] Koren, Y. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2009), KDD '09', ACM, pp. 447–456.

[31] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (Aug. 2009), 30–37.

[32] Lawrence, P., Sergey, B., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[33] Lee, D., Park, S. E., Kahng, M., Lee, S., and goo Lee, S. Exploiting contextual information from event logs for personalized recommendation. In *Computer and Information Science*. 2010, pp. 121–139.

[34] Lee, S. A generic graph-based multidimensional recommendation framework and its implementations. In *Proceedings of the 21st international conference companion on World Wide Web* (New York, NY, USA, 2012), WWW '12 Companion', ACM, pp. 161–166.

[35] Lee, S., Kahng, M., and Lee, S.-g. Flexible recommendation using random walks on implicit feedback graph. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication* (New York, NY, USA, 2011), ICUIMC '11', ACM, pp. 3:1–3:6.

[36] Lee, S., Song, S.-i., Kahng, M., Lee, D., and Lee, S.-g. Random walk based entity ranking on graph for multidimensional recommendation. In *Proceedings of the fifth ACM conference on Recommender systems* (New York, NY, USA, 2011), RecSys '11', ACM, pp. 93–100.

[37] Liu, J.-G., Shi, K., and Guo, Q. Solving the accuracy-diversity dilemma via directed random walks. *Phys. Rev. E 85* (Jan 2012), 016118.

[38] Maccatrozzo, V. Burst the filter bubble: Using semantic web to enable serendipity. In *The Semantic Web – ISWC 2012*, P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, Eds., vol. 7650 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012, pp. 391–398.

[39] Markines, B., Cattuto, C., Menczer, F., Benz, D., Hotho, A., and Stumme, G. Evaluating similarity measures for emergent semantics of social tagging. In *Proceedings of the 18th international conference on World wide web* (New York, NY, USA, 2009), WWW '09', ACM, pp. 641–650.

[40] Palmisano, C., Tuzhilin, A., and Gorgoglione, M. Using context to improve predictive modeling of customers in personalization applications. *IEEE Trans. Knowl. Data Eng. 20*, 11 (2008), 1535–1549.

[41] Pariser, E. *The filter bubble : what the Internet is hiding from you.* Penguin Press, New York, 2011.

[42] Park, Y.-J., and Tuzhilin, A. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems* (New York, NY, USA, 2008), RecSys '08', ACM, pp. 11–18.

[43] Payne, J., Bettman, J., and Johnson, J. *The adaptative decision making.* Cambridge University Press, 1993.

[44] Pazzani, M. J., and Billsus, D. Content-based recommendation systems. In *THE ADAPTIVE WEB: METHODS AND STRATEGIES OF WEB PERSONALIZATION. VOLUME 4321 OF LECTURE NOTES IN COMPUTER SCIENCE* (2007), Springer-Verlag, pp. 325–341.

[45] Rendle, S., Gantner, Z., Freudenthaler, C., and Schmidt-Thieme, L. Fast context-aware recommendations with factorization machines. In *SIGIR* (2011), pp. 635–644.

[46] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web* (New York, NY, USA, 2001), WWW '01', ACM, pp. 285–295.

[47] Shang, S., Kulkarni, S. R., Cuff, P. W., and Hui, P. A random walk based model incorporating social information for recommendations. *CoRR abs/1208.0787* (2012).

[48] Shardanand, U., and Maes, P. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1995), CHI '95', ACM Press/Addison-Wesley Publishing Co., pp. 210–217.

[49] Tetko, I. V., Livingstone, D. J., and Luik, A. I. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of chemical information and computer sciences 35*, 5 (1995), 826–833.

[50] Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., and Sun, J. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2010), KDD '10', ACM, pp. 723–732.

[51] Zhu, X., Goldberg, A. B., Van, J., and Andrzejewski, G. D. Improving diversity in ranking using absorbing random walks. In *Physics Laboratory – University of Washington* (2007), pp. 97–104.

[52] Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), WWW '05', ACM, pp. 22–32.