CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Kontinuierliche Verarbeitung von Datenströmen zur Berechnung von Objektempfehlungen auf Basis von Datenstromqueries

Einsatz eines Datenstrommanagementsystems als Framework für Online-Recommender-Systeme

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften
der Carl von Ossietzky Universität Oldenburg
zur Erlangung des Grades und Titels eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

angenommene Dissertation von

Cornelius A. Ludmann

geboren am 5. November 1984 in Dortmund

Gutachterinnen und Gutachter:
Prof. Dr. Susanne Boll-Westermann
Prof. Dr. Daniela Nicklas
Prof. Dr. Dietmar Jannach

Tag der Disputation:
10. Januar 2020

Zusammenfassung

Empfehlungssysteme finden in verschiedenen Informationssystemen Anwendung. Mit ihnen sollen aus einer großen und unübersichtlichen Menge an Objekten (z. B. Nachrichtenartikel, Produkte, Musikstücke etc.) eine Auswahl für eine Benutzerin bzw. einen Benutzer als möglichst nützliche Empfehlung herausgefiltert werden. Die Berechnung von relevanten Objekten erfolgt in der Regel durch die Analyse von Verhalten der Personen, die mit dem Informationssystem interagieren (Seitenaufrufe, Käufe etc.). Dabei spielt in vielen Anwendungen nicht nur das über einen langen Zeitraum beobachtete Verhalten der Personen, sondern auch das aktuelle Verhalten in einem kurzen Zeitfenster eine Rolle. Sorgt man dafür, dass die jüngsten Interaktionen aller Personen in die Berechnung der Empfehlungen miteinfließen, so kann dies die Qualität der Empfehlungen verbessern.

Die kontinuierliche Berücksichtigung von Interaktionen im Prozess der Empfehlungsberechnung stellt Entwicklerinnen und Entwickler von Empfehlungssystemen vor neue Herausforderungen. Während die eingesetzten Methoden und Algorithmen bislang wahlfreien Zugriff auf einen statischen Datenbestand an Interaktionen von Personen hatten, muss nun ein Strom an Daten kontinuierlich verarbeitet werden, dem ständig neue Datenelemente hinzugefügt werden. Mit den Herausforderungen, die mit dieser Art der Datenverarbeitung einhergehen, beschäftigt sich die Forschung an Datenstrommanagementsystemen. Die Technologie der Datenstrommanagementsysteme bietet eine flexible Basis für die kontinuierliche Verarbeitung von Datenströmen.

In dieser Arbeit wird untersucht, wie die Technologie der Datenstrommanagementsysteme für die Umsetzung von Empfehlungssystemen eingesetzt werden kann, um Empfehlungen kontinuierlich auf Grundlage des aktuellsten Datenbestandes zu berechnen. Ergebnis sind Konzepte sowie ein Framework zur Umsetzung eigener Empfehlungssysteme auf Basis von Datenstrommanagementsystemen.

Abstract

Recommender systems are used in various information systems in which a small amount of objects (e.g. news articles, products, music, etc.) is selected from a large number of objects in order to recommend them to the users. Relevant objects are usually calculated by analyzing the behavior of the people interacting with the information system (page views, purchases, etc.). In many applications not only the behavior of the users observed over a long period of time plays a role, but also the current behavior in a short time window. If one ensures that the most recent interactions of all users are included in the calculation of the recommendations, this can improve the quality of the recommendations.

The continuous consideration of interactions in the recommendation process poses new challenges for developers of recommender systems. While the methods and algorithms used in the past had random access to a static database of user interactions, a stream of data must now be continuously processed where new data elements are constantly added. The research on data stream management systems addresses the challenges associated with this type of data processing. The technology of data stream management systems provides a flexible basis for the continuous processing of data streams.

This thesis examines how data stream management system technology can be used to implement recommender systems to continuously calculate recommendations based on the most up-to-date data. The results are concepts and a framework for the implementation of recommender systems on the basis of data stream management systems.

Inhaltsverzeichnis

1 Einführung

Das Internet ermöglicht Zugriff auf eine große und stetig wachsende Menge an Informationen. Kontinuierlich erscheinen auf Webseiten, digitalen Handelsplattformen und Social Networks neue Nachrichtenartikel, Blog-Einträge, Musikstücke, Videos und Statusnachrichten, die von Internetbenutzerinnen und -benutzern konsumiert werden können. Diese stehen zunehmend vor dem Problem, aus dieser Datenmenge die Objekte zu finden, für die sie sich zum gegenwärtigen Zeitpunkt interessieren. Fühlen sich Benutzerinnen und Benutzer aufgrund der Menge der verfügbaren Informationen überfordert, so spricht man von Informationsüberflutung (engl. *Information Overload*). Um dem entgegenzuwirken, filtern Softwaresysteme aus der unüberschaubaren Menge der Informationen eine überschaubare Auswahl heraus. Diese Art der Systeme lässt sich in zwei Gruppen unterteilen: Information-Retrieval- und Recommender-Systeme.

Beim Information Retrieval werden Objekte aufgrund von Schlüsselwörtern gefiltert, welche Benutzerinnen und Benutzer explizit angeben. Eine Benutzerin oder ein Benutzer gibt beispielsweise in einem Suchfeld eines Nachrichtenportals ein, welche Begriffe in einem Nachrichtentext vorkommen sollen, und das Information-Retrieval-System filtert die Objekte heraus, die diese Begriffe enthalten.

Eine Alternative zu Information-Retrieval-Systemen stellen Recommender-Systeme dar. Diese filtern Objekte ohne explizite Vorgabe durch Benutzerinnen oder Benutzer. Dazu wird in der Regel das Interesse der Benutzerinnen und Benutzer von den vorherigen Interaktionen mit dem System abgeleitet. Beispielsweise kann analysiert werden, welche Objekte diese in der Vergangenheit häufig genutzt haben. Alternativ wird Benutzerinnen und Benutzern die Möglichkeit gegeben, Objekte explizit zu bewerten (z. B. auf einer Skala von 1 bis 5 Sternen), um dadurch ihr Interesse an Objekten mitteilen zu können. Während bei Information-Retrieval-Systemen Benutzerinnen und Benutzer explizit vorgeben, welche Informationen angezeigt werden sollen (i. d. R. durch Suchwörter), präsentieren Recommender-Systeme von sich aus eine Menge an Empfehlungen, die zu den Nutzungsinteressen passen.

Betreiberinnen und Betreiber von Recommender-Systemen auf großen Plattformen, wie zum Beispiel Handels-, Nachrichten- oder Multimediaportalen, stehen vor der Herausforderung, eine große Menge an Daten zu analysieren, um den Benutzerinnen und Benutzern Empfehlungen geben zu können, die persönlich auf sie zugeschnitten sind. Jede Interaktion, die eine Benutzerin oder Benutzer mit dem System durchführt – wie zum Beispiel Aufruf oder Bewertung eines Objektes – gibt Aufschluss darüber,

wofür er oder sie sich interessiert. Dabei spielt nicht nur das grundsätzliche, sondern auch das *aktuelle* Interesse eine große Rolle: Ist die Benutzerin in diesem Moment an Actionfilmen oder romantischen Komödien interessiert? Sucht ein Benutzer ein Produkt für sich oder als Geschenk? Möchte eine Benutzerin ausgiebig zu Mittag essen oder nur einen Snack kaufen? Auch die Interaktionen anderer Benutzerinnen und Benutzer können einbezogen werden: Welche Objekte sind aktuell besonders beliebt? Oder: Welche Objekte werden aktuell von Benutzerinnen oder Benutzern mit ähnlichen Interessen aufgerufen oder positiv bewertet? Je schneller diese Informationen durch ein Recommender-System verarbeitet werden, desto eher kann die Berechnung der Empfehlungen diese Informationen einbeziehen und die Ergebnisse auf das aktuelle Interesse der Benutzerinnen und Benutzer anpassen.

Im optimalen Fall berücksichtigt ein Recommender-System bei der Berechnung von Empfehlungen neue Informationen ab dem Moment, wo sie zur Verfügung stehen. Gängige Recommender-System-Methoden sind dafür allerdings nicht ausgelegt. Sie analysieren große Datenmengen, um das Interesse von Benutzerinnen und Benutzern zu lernen und wenden anschließend das Gelernte an, um personalisierte Empfehlungen zu berechnen. Da die Analyse der Daten aufwändig ist und viel Zeit in Anspruch nimmt, wird das Lernen der Interessen in Produktivsystemen normalerweise in regelmäßigen Abständen wiederholt – neue Informationen bleiben allerdings bis zur nächsten Iteration ungenutzt.

In den letzten Jahren wurden zunehmend Algorithmen veröffentlicht, die Recommender-System-Modelle inkrementell lernen, um neue Informationen schneller in Modellen zu berücksichtigen. Das ermöglicht die Umsetzung von Recommender-Systemen mit einem neuen Verarbeitungsparadigma, den sogenannten Online-Recommender-Systemen. Im Gegensatz zu offlinelernenden Recommender-Systemen aktualisieren diese ihre Modelle kontinuierlich, sobald neue Informationen bereitstehen.

Diese Arbeit stellt einen Ansatz vor, wie Online-Recommender-Systeme als Bestandteil von sogenannten Datenstrommanagementsystemen (DSMS) umgesetzt werden können. Ähnlich den weit verbreiteten Daten*bank*managementsystemen (DBMS), stellt ein Datenstrommanagementsystem ein universelles Softwaresystem zur Datenverarbeitung dar. Während Datenbankmanagementsysteme auf Basis einmaliger Anfragen (Query) eine statische und endliche Menge an Daten verarbeitet, verarbeiten Datenstrommanagementsysteme kontinuierlich einen Strom an Daten. Die Integration von Online-Recommender-System-Algorithmen in Datenstrommanagementsystemen erleichtert die Entwicklung und den Betrieb eines Online-Recommender-Systems, da auf eine Vielzahl an Funktionen eines Datenstrommanagementsystems zurückgegriffen werden kann.

1.1 Motivation und Problemstellung

Im Fokus der Forschung an Recommender-Systemen stand lange Zeit das Problem, auf Grundlage eines statischen Datenbestands mit gespeicherten Interaktionen von Benutzerinnen und Benutzern (z. B. den Aufruf eines Videos oder die Bewertung eines Produkts) abzuschätzen, für welche Objekte (z. B. Videos oder Produkte) sich diese zusätzlich interessieren, um diese zu empfehlen. Die entstandenen Methoden reichen von der Empfehlung von häufig genutzten/favorisierten Objekten über die Empfehlung von Objekten, die den bereits genutzten Objekten ähneln, bis hin zu Methoden, die aus den Daten ein Modell erstellen, mit dem man abschätzen kann, wie sehr sich eine Benutzerin oder ein Benutzer für ein bestimmtes Objekt interessiert.

Die Nutzung eines statischen Datenbestands als Datengrundlage für Recommender-Systeme spiegelt den realen Einsatz eines Recommender-Systemen allerdings nur unvollständig wider. In einem System, in dem kontinuierlich neue Objekte hinzugefügt werden und eine große Anzahl an Benutzerinnen und Benutzern fortwährend Objekte aufruft und durch eine Bewertung ihr Interesse an den Objekten mitteilt, ist die Datengrundlage für ein Recommender-System nicht statisch und endlich, sondern entspricht eher einer unendlichen Sequenz an Ereignissen. Diese Art der kontinuierlich erzeugten Daten bezeichnet man auch als *Datenstrom*. Um Methoden, die für statische und endliche Datensätze mit wahlfreiem Zugriff auf die Daten entwickelt wurden, in einem realen Szenario einzusetzen, müssen die Empfehlungen auf Grundlage einer Momentaufnahme der Daten bestimmt werden. Insbesondere bei Methoden, die aufwändig zu berechnende Modelle erstellen, erfolgt die Aktualisierung der Modelle aufgrund der Berechnungskomplexität in großen Abständen, sodass aktuelle Trends und aktuelle Interessen von Benutzerinnen und Benutzern in den Empfehlungen nicht oder erst verzögert berücksichtigt werden. Des Weiteren werden *neue* Benutzerinnen und Benutzer sowie *neue* Objekte erst mit der Verarbeitung der nächsten Momentaufnahme einbezogen.

Die datengetriebene Verarbeitung von kontinuierlich und sequentiell erzeugten Daten bezeichnet man als *Datenstrom-* oder *Online-Verarbeitung*. Unter den Publikationen zu Recommender-Systemen findet man zu den Stichwörtern *online*, *stream-based*, *incremental*, *scalable* oder *large-scale* eine Vielzahl an Veröffentlichungen, in denen Algorithmen vorgestellt werden, um neue Informationen in vorhandene Modelle zu integrieren (z. B. [Tak+09]). Frameworks, wie zum Beispiel *Massive Online Analysis* (MOA) [Bif+10] bieten den Rahmen, neue Algorithmen zu evaluieren.

Ein Recommender-System (wie jedes Machine-Learning-System) besteht allerdings aus mehr als nur einem Algorithmus. Viele Verarbeitungsschritte, die in Forschungs-

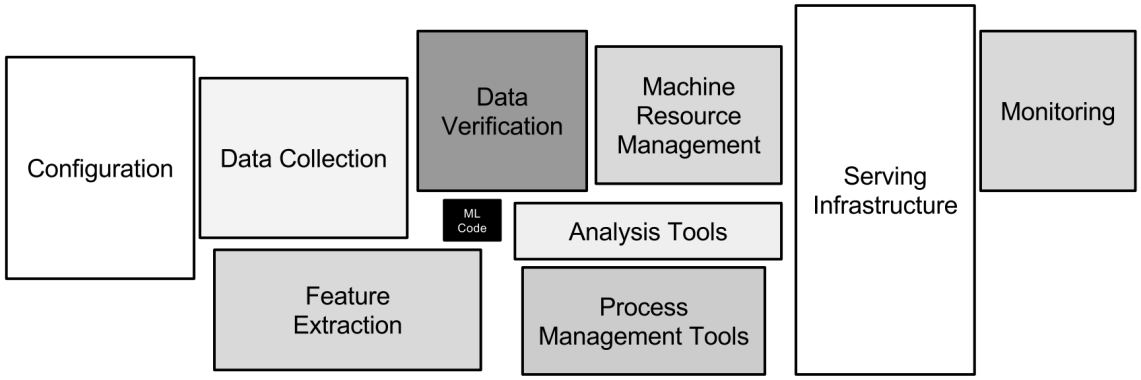


Abbildung 1.1 Der eigentliche Maschine-Learning-Algorithmus ist in einem Produktivsystemen nur einer kleiner Baustein (hier als schwarze Box in der Mitte dargestellt). (Quelle: [Scu+15])

settings häufig ausgeklammert werden, sind für den erfolgreichen Einsatz im Produktivbetrieb von großer Bedeutung (vgl. [Scu+15]). Abbildung 1.1 stellt die Bedeutung des Lernalgorithmus im Verhältnis zu anderen Systemkomponenten grafisch dar.

In einem realitätsnahen Setting, wie zum Beispiel im Rahmen des News Recommendation Evaluation Lab *CLEF NewsREEL*[1] [Hop+14] zur Empfehlung von Nachrichtenartikeln, wird deutlich, dass die Analyse von Datenströmen für Online-Recommender-Systeme ein System erfordert, das neben der Implementierung eines Algorithmus weitere Aufgaben erfüllen muss. Dazu gehören beispielsweise:

Zugriff auf Datenströme

Die Daten können auf verschiedene Weise (z. B. als HTTP-Requests, über eine TCP-Socket-Verbindung etc.) und in verschiedenen Formaten (z. B. JSON, XML, CSV etc.) übertragen werden. Um einen neuen datenstrombasierten Ansatz für Recommender-System zu testen, muss eine Infrastruktur aufgebaut werden, die die Daten entgegennimmt und entsprechend des Formats zerlegt sowie die ausgehenden Daten im richtigen Format überträgt.

Speicherung und Verwaltung von Daten

Methoden für Recommender-Systeme benötigen in der Regel neben dem jüngsten Datenstromelement auch historische Daten. Da ein Datenstrom weder einen expliziten Anfang noch ein Ende hat, sondern potenziell unendlich Daten liefert, kann lediglich ein endlicher Ausschnitt der Daten im Speicher gehalten werden. Die Organisation der temporär gespeicherten Daten sowie das Verwerfen nicht mehr benötigter Daten spielen für Online-Recommender-Systeme eine zentrale Rolle.

[1] `http://www.clef-newsreel.org/`

Vor- und Nachverarbeitung von Daten

Zur Nutzung der Daten müssen diese häufig vor- und nachverarbeitet werden. Dazu gehört beispielsweise das Zusammenfügen von Daten aus verschiedenen Quellen sowie die Aggregation, Normalisierung, Auswahl und Konvertierung von Daten. Bei diesen Verarbeitungen handelt es sich in der Regel um Standardoperationen, die für den Einsatz von Online-Recommender-System-Algorithmen nötig sind und auf den jeweiligen Anwendungsfall angepasst werden müssen.

Bilden Datensätze die Grundlage für Recommender-Systeme, so bieten Datenbankmanagementsysteme (DBMS) und Programmbibliotheken Methoden für den Zugriff, die Speicherung und die Vor- und Nachverarbeitung der Daten. Bei der Entwicklung von Online-Recommender-Systemen unterstützen Programmierframeworks wie beispielsweise Apache Storm (vgl. [AJT11]) Softwareentwicklerinnen und Softwareentwickler bei der Implementierung und verteilten Ausführung von datenstrombasierten Algorithmen. Die Flexibilität durch die Formulierung von Datenbankanfragen, eine operatorbasierte Verarbeitung der Daten, wie man sie mit der relationalen Algebra von Datenbankmanagementsystemen kennt, sowie eine Abstraktion der Datenhaltung ist dort nicht vorgesehen. Um die Konzepte von relationalen Datenbankmanagementsysteme auf die Verarbeitung von Datenströmen zu übertragen, sind in der Datenbankforschung Konzepte und Implementierungen von sog. Datenstrommanagementsystemen (DSMS; z. B.: [Bab+02; ABW06; KS09; App+12]) entstanden. Diese Technologie unterstützt bei der Bewältigung der o. g. Aufgaben und bietet eine Abstraktion auf den Datenzugriff, wie man es von Datenbankmanagementsystemen gewohnt ist.

Datenstrommanagementsysteme sind bislang eine von der Recommender-System-Forschung kaum beachtete Technologie. Einen ersten Ansatz, wie man Recommender-Systeme mit einem Datenstrommanagementsystem umsetzen kann, zeigen Chandramouli et al. [Cha+11]. Eine systematische Untersuchung der Möglichkeiten und Grenzen der Datenstrommanagementsystem-Technologie als Basis für Online-Recommender-Systeme gibt es bislang nicht. Diese Lücke soll diese Arbeit schließen.

1.2 Zielsetzung und Abgrenzung

Das Ziel dieser Arbeit ist die Entwicklung von Konzepten zur Umsetzung von Online-Recommender-Systemen auf Basis der Datenstrommanagementsystem-Technologie. Durch die Nutzung von Operatoren und Ausführungsplänen soll eine anpassbare Datenstromverarbeitung für Online-Recommender-Systeme ermöglicht werden, um

neue Recommender-System-Methoden in realen Szenarien umsetzen und evaluieren zu können.

Grundlage dieser Arbeit stellt das Datenstrommanagementsystem-Framework *Odysseus* [App+12] dar. Odysseus ist als Open-Source-Projekt frei verfügbar und bietet eine Plattform, um neue Datenstromkonzepte zu entwickeln und zu erforschen. Die Datenstromverarbeitung basiert in Analogie zu Datenbankmanagementsystemen auf Operatoren, die jeweils eine festgelegte Operation auf den Daten ausführen. Die Verarbeitung der Datenstromelemente wird durch einen aus Operatoren bestehenden Ausführungsplan festgelegt. Dazu stellt die DSMS-Benutzerin oder der DSMS-Benutzer mit Hilfe einer Anfragesprache eine kontinuierlich laufende Anfrage an das Datenstrommanagementsystem, die in einen Ausführungsplan überführt wird. Odysseus wurde als Grundlage ausgewählt, da bereits viele häufig benötigte Funktionen eines Datenstrommanagementsystems als Module vorhanden sind (z. B. Operatoren der relationalen Algebra, temporäre Datenhaltung im Hauptspeicher etc.) und es auf Erweiterbarkeit ausgelegt ist. Die Konzepte in dieser Arbeit lassen sich auch auf Basis anderer, vergleichbarer Datenstrommanagementsysteme umsetzen.

Die in dieser Arbeit entwickelten Konzepte lassen sich grundsätzlich zur Empfehlung von verschiedenen Objekten einsetzen. Die Arbeit ist explizit nicht auf ein bestimmtes Szenario festgelegt. Zur Veranschaulichung wird in dieser Arbeit in der Regel der Anwendungsfall eines Recommender-Systems zur Empfehlung von Nachrichtenartikeln auf Online-Portalen herangezogen. Anhand dieses Beispiels lassen sich verschiedene Arten von Recommender-Systemen umsetzen. Die dynamischen Aspekte wie Trends und temporäre Interessen machen diesen Anwendungsfall für den Einsatz von Datenstromverarbeitung besonders geeignet.

In dieser Arbeit liegt der Fokus auf der Fragestellung, wie ein Datenstrommanagementsystem eingesetzt werden kann, um Online-Recommender-Systeme umzusetzen. Dabei steht die Forschung an Datenstrommanagementsystemen und der Datenstromverarbeitung im Vordergrund. Die Recommender-System-Technologie spielt die Rolle der Anwendung für ein Datenstrommanagementsystem.

Mit der Veröffentlichung *StreamRec* von Chandramouli et al. [Cha+11] gibt es bislang lediglich eine eher knappe Veröffentlichung, die sich mit dem Thema Datenstrommanagementtechnologie für Recommender-Systeme beschäftigt. In dieser Veröffentlichung wird das Datenstrommanagementsystem *Microsoft StreamInsight* eingesetzt. Im Gegensatz zu der vorliegenden Arbeit stellt die Veröffentlichung *StreamRec* ein konkretes und sehr begrenztes Szenario vor, in dem ausschließlich mit Basisoperatoren eines Datenstrommanagementsystems Objektähnlichkeiten zur Bestimmung der Empfehlungsmenge berechnet werden. Das Konzept in dieser Arbeit

verfolgt einen generischeren Ansatz, der z. B. auch die Integration von modellbasierten Lernalgorithmen erlaubt.

Einige Veröffentlichungen nutzen Frameworks zur Implementierung eines Recommender-Systems, die bei der Entwicklung von datenstrombasierten Systemen unterstützen. Zum Beispiel setzen Ali et al. [AJT11] *Apache Storm* zur Implementierung eines Recommender-System-Algorithmus ein. Das datenstrombasierte Data-Mining-Framework *Massive Online Analysis* (MOA) [Bif+10] ermöglicht die Evaluation unter anderem von datenstrombasierten Recommender-System-Algorithmen. Jugovac et al. stellen in [JJK18] ein Framework vor, um datenstrombasierte Recommender-System-Algorithmen für Nachrichtenartikel zu evaluieren. Im Gegensatz zu dem Konzept in dieser Arbeit bieten diese Arbeiten keinen anwendungsunabhängigen, erweiterbaren und flexiblen Ansatz, der auf die Komposition von Datenstromoperatoren setzt. Der hier vorgestellte Ansatz kann außerdem von diversen Datenstrommanagementsystem-Funktionen profitieren.

Für modellbasierte Recommender-System-Methoden zeigen verschiedene Veröffentlichungen der Recommender-System-Forschung über *inkrementelle* oder *online* Algorithmen für das kollaborative Filtern (z. B. [LXZ12; Tak+09]), wie neue Lerndaten in Modelle integriert werden können, ohne dass das Modell komplett neu gelernt werden muss. Diese Algorithmen bieten die Basis für eine modell- und datenstrombasierte Verarbeitung von Bewertungsdaten für Recommender-Systeme. Die Arbeit von Diaz-Aviles et al. [Dia+12] stellt eine Methode zur datenstrombasierten Verarbeitung von Bewertungsdaten zum Lernen eines Matrixfaktorisierungsmodells vor. Dazu wird das Modell mit Daten aus einem Reservoir aktualisiert, welches eine Stichprobe des Datenstroms vorhält. Im Gegensatz zu diesen Arbeiten wird in der vorliegenden Arbeit kein neuer Algorithmus vorgestellt, sondern der Fokus auf die Komposition eines flexiblen Recommender-Systems mithilfe von Datenstromoperatoren gesetzt, in denen diese Algorithmen als Operatoren integriert werden können.

Darüber hinaus gibt es unter dem Begriff *Sequence-Aware Recommender Systems* innerhalb der Recommender-System-Forschung ein Forschungsfeld, welches Überschneidungen mit der Datenstromverarbeitung und somit auch mit dieser Arbeit hat. Einen guten Überblick bietet das Survey von Quadrana et al. [QCJ18]. Bei den Sequence-Aware Recommender Systems steht die Entwicklung von Algorithmen im Fokus, die Empfehlungen aufgrund von Lerndaten berechnen, die sequentiell geordnet sind. Die Nutzung von Datenstrommanagementsystemen zur Umsetzung von Sequence-Aware Recommender Systems wird bislang nicht diskutiert.

Die in dieser Arbeit vorgestellten Ergebnisse wurden vom Autor in verschiedenen Veröffentlichungen publiziert. Dazu gehören Veröffentlichungen auf der deutschsprachigen Konferenz für Datenbankforschung *Datenbanksysteme für Business, Technologie*

und Web (BTW) [Lud+15; Lud17c], den internationalen Konferenzen *User Modeling, Adaptation, and Personalization (UMAP)* [LGA15], *Conference and Labs of the Evaluation Forum (CLEF)* [Lud17a], *ACM Conference on Recommender Systems (ACM RecSys)* [Lud15; Lud16] und im Journal *Datenbank-Spektrum* [Lud17b].

1.3 Vorgehen und Aufbau der Arbeit

Die in dieser Arbeit angewandte Forschungsmethodik, und somit auch das Vorgehen und der Aufbau der Arbeit, orientiert sich an dem Prozessmodell der Design Science Research Methodology (DSRM) nach Peffers et al. [Pef+07]. Die DSRM ist auf die Entwicklung und Evaluation von Artefakten in der Forschung an Informationssystemen ausgelegt. Das Prozessmodell (Abbildung 1.2) besteht aus sechs Schritten, die durch einen Eintrittspunkt initiiert werden. Die Initiierung kann problemzentriert, zielzentriert, konzeptions- und implementierungszentriert oder durch Kunden bzw. Anwendungskontexte erfolgen. Eine Forschungsarbeit nach der DSRM beginnt mit der Problemidentifikation und Motivation, woraus eine Zielsetzung gefolgert wird. Anschließend wird ein Artefakt entworfen und entwickelt und dieses zur Demonstration des Artefakts zur Problemlösung in einem geeigneten Kontext eingesetzt. Abschließend wird das Artefakt evaluiert und die Ergebnisse veröffentlicht.

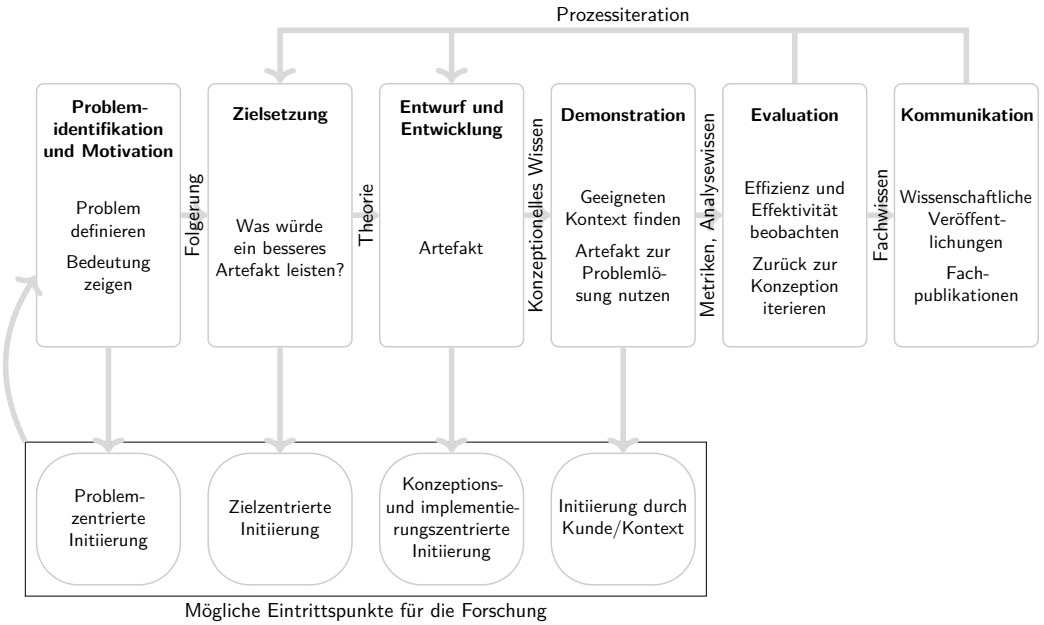


Abbildung 1.2 Prozessmodell für die DSRM nach [Pef+07]

Abbildung 1.3 zeigt die Instanziierung des Prozessmodells für diese Arbeit. Die Initiierung erfolgt durch die in Abschnitt 1.2 definierte Zielsetzung. Nach der Moti-

vation und Abgrenzung in diesem Kapitel werden zunächst die Grundlagen in den Themenbereichen *Recommender-Systeme* und *Datenstromverarbeitung* in Kapitel 2 (Grundlagen) zusammengefasst.

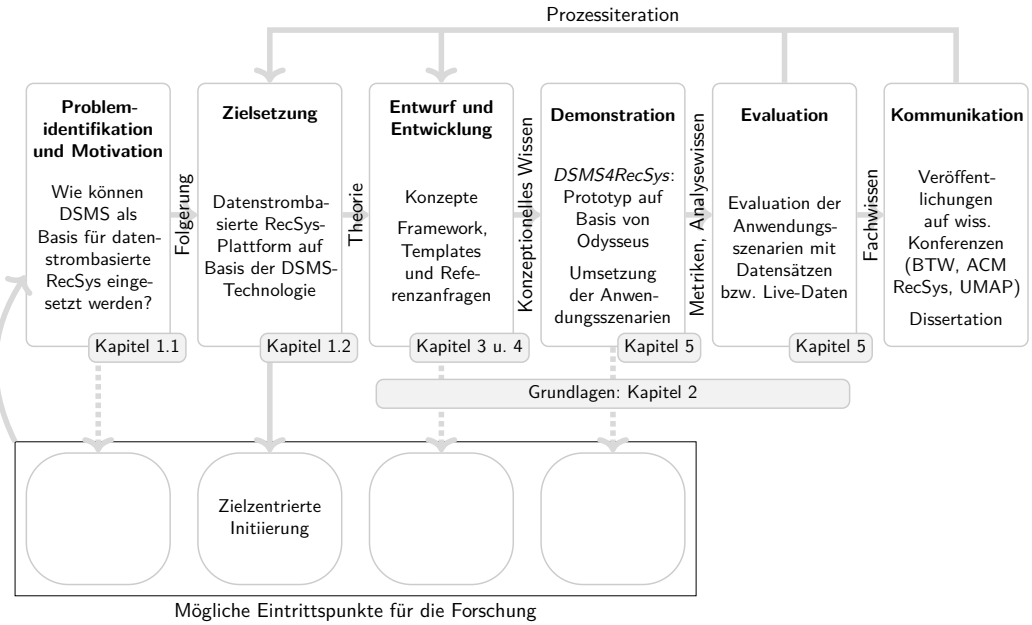


Abbildung 1.3 Instanziierung des Prozessmodells nach [Pef+07] für diese Arbeit

Im Rahmen des Prozessschritts *Entwurf und Entwicklung* werden in Kapitel 3 (Konzepte) verschiedene Konzepte für die Umsetzung des Vorhabens erarbeitet und vorgestellt. Dazu gehören die Diskussion von Architekturentscheidungen und -alternativen, die Entwicklung eines Datenmodells sowie die Diskussion und Entwicklung von Implementierungsmustern für verschiedene Arten von Empfehlungsmethoden.

Darauf aufbauend wurden Templates für die Umsetzung von Recommender-Systemen mit einem Datenstrommanagementsystem entwickelt, in denen die Konzepte angewendet wurden und die es ermöglichen, konkrete Recommender-Systeme flexibel mit Hilfe eines Datenstrommanagementsystems umzusetzen. Kapitel 4 (Framework) stellt die Templates vor und beschreibt die festen und variablen Teile. Letztere ermöglichen die Anpassung an spezifische Anforderungen.

Um die Machbarkeit der erarbeiteten Konzepte und Templates zu zeigen, wurden diese auf Basis des Datenstrommanagementsystems *Odysseus* implementiert und durch die Umsetzung von beispielhaften Recommender-Systemen evaluiert (Kapitel 5). Dazu wurde zum einen ein Recommender-System umgesetzt, welches die Methode der Matrixfaktorisierung einsetzt und anhand eines Beispieldatensatzes evaluiert. Die Ergebnisse wurden mit der Implementierung des selben Algorithmus in dem Evaluationsframework *MOA* verglichen. Diese Evaluation zeigt die Korrektheit der Implementierung und die Umsetzbarkeit der Methode mit den erarbeiteten Konzep-

ten und Templates. Der zweite Teil der Evaluation zeigt die Machbarkeit in einem realistischen Szenario. Dazu wurden die hier vorgestellten Konzepte im Rahmen der *CLEF NewsREEL Challenge* eingesetzt. Im Rahmen dieser Challenge müssen echte Benutzerdaten in Echtzeit verarbeitet werden, um Empfehlungen zu berechnen. Die Evaluationsmetrik bezieht die Verarbeitung hoher Datenraten sowie die Anzahl der Klicks auf die Empfehlungen ein (Click-Through-Rate). Die mit diesen Konzepten umgesetzte Lösung erreichte den ersten Platz. Diese Evaluation zeigt, dass das vorgestellte Konzept den Anforderungen im realen Einsatz gewachsen ist und der Ansatz einen Mehrwert gegenüber den Ansätzen der anderen Teilnehmenden bietet.

1.4 Begriffe und Notationen

Für die Entitätstypen eines Recommender-Systems sind in der Literatur unterschiedliche Begriffe gebräuchlich. Die benutzten Begriffe in dieser Arbeit sind möglichst allgemein gehalten, sodass sie auf unterschiedliche Arten von Recommender-Systemen zutreffen. Wenn es eindeutige und allgemein gebräuchliche deutsche Begriffe gibt, so werden diese verwendet. Andernfalls werden die englischen Begriffe genutzt.

In Formeln oder formalen Definitionen bezeichnen wir Entitäten mit kleinen, lateinischen Buchstaben, z. B. i für ein Objekt (engl. *item*), welches einer Benutzerin bzw. einem Benutzer empfohlen werden kann (z. B. ein Nachrichtenartikel). Hat eine Entität Attribute, so werden die Attribute bzw. deren Werte mit der Punktnotation referenziert: $i.y$ meint somit das Attribut y der Entität i.

Große, lateinische Buchstaben haben je nach Kontext unterschiedliche Bedeutungen. Zum einen wird mit ihnen in dieser Arbeit eine Entitätenmenge bezeichnet. Wenn nicht anders angegeben, sind in der Regel alle möglichen Ausprägungen eines Entitätentyps gemeint. So ist mit $i \in I$ ein Objekt i aus der Menge aller vorhanden Objekte I gemeint. I ist somit der Wertebereich von i. Zum anderen werden Konstanten mit großen, lateinischen Buchstaben bezeichnet. So wird beispielsweise K für verschiedene Konstanten genutzt (z. B. die Anzahl der Empfehlungen, die ein Recommender-System für eine Benutzerin bzw. Benutzer ausgibt).

Ein Datenstrom eines bestimmten Datentyps wird in dieser Arbeit mit einem großen, kalligrafischen Buchstaben bezeichnet. Beispielsweise wird der Datenstrom aller Objekte mit $\mathcal{I}$ bezeichnet. Mit $\mathcal{I}_t$ bezeichnen wir die Menge aller Datenstromelemente des Datenstroms $\mathcal{I}$, die zum Zeitpunkt t gültig sind (siehe dazu auch das Grundlagenkapitel 2.2 zur Datenstromverarbeitung).

Die Objekte, die Benutzerinnen und Benutzer empfohlen bekommen können (z. B. Nachrichtenartikel), bezeichnen wir als *Empfehlungsobjekte* oder kurz als *Objekte*. i bezeichnet ein konkretes Objekt, i' ein zweites, von i verschiedenes Objekt ($i \neq i'$).

Wenn in dieser Arbeit von *Benutzerinnen und Benutzern* oder von *Personen* die Rede ist, sind in der Regel die Personen gemeint, für die das Recommender-System Empfehlungen berechnet bzw. zu denen historische Daten über deren Interessen und Verhalten vorliegen (u. a. auch Profilinformationen). Sie werden mit $u \in U$ bezeichnet. Eine aktive Benutzerin bzw. ein aktiver Benutzer, also die Person, für die in diesem Moment eine Empfehlung berechnet werden soll, wird mit u' bezeichnet, um diese von den Personen in den historischen Daten zu unterscheiden.

Der *Kontext* von Benutzerinnen und Benutzern beschreibt die Situation, in dem sich diese befinden (z. B. der Ort und oder die Tageszeit). Die Angabe der Benutzerin bzw. des Benutzers mit evtl. vorhandenen Profilinformationen (dazu können demografische Angaben oder Vorlieben gehören) bilden zusammen mit evtl. vorhandenen Kontextinformationen die Daten, die eine *Anfrage* an ein Recommender-System beinhalten kann. Diese Anfrage ist eine Aufforderung an das Recommender-System, eine auf die Attribute in der Anfrage zugeschnittene Empfehlung zu liefern. Da der deutsche Begriff *Anfrage* sowohl in einem Recommender-System (engl. *Request for Recommendations*) als auch in einem Datenstrommanagementsystem (engl. *Query*) eine Rolle spielt und Unterschiedliches bezeichnet, werden in dieser Arbeit durchgängig die englischen Begriffe *Request* resp. *Query* verwendet, um Verwechslungen auszuschließen. Einen Request bezeichnen wir mit q, die Menge aller möglichen Requests mit Q. Der Request-Datenstrom wird mit $\mathcal{Q}$ bezeichnet[2].

Objekte, die für eine Empfehlung infrage kommen, bezeichnen wir als *Empfehlungskandidaten* oder kurz *Kandidaten*. Einen Kandidaten bezeichnen wir mit $c \in C \subseteq I$.

Kandidaten werden vom Recommender-System sortiert, um die K Kandidaten an der Spitze der Liste zu empfehlen (geordnete Top-K-Menge). Die in dieser Arbeit betrachteten Verfahren berechnen für jeden Kandidaten einen numerischen Score, der zur Sortierung herangezogen wird (Punktschätzer). Dieser Score wird in dieser Arbeit verallgemeinert als *geschätzte Bewertung* $\hat{r}$ bezeichnet, unabhängig davon, ob das Verfahren tatsächliche Benutzerbewertungen ($r \in R$) zur Berechnung heranzieht (vgl. Abschnitt 2.1).

Alle Informationen, die für die Berechnung von Empfehlungen von Interesse sein könnten (z. B. Bewertungen zu Objekten, Daten zu Interaktionen mit Objekten wie

[2] Da r bereits für die Bewertung (engl. Rating) belegt ist, wird für Requests auf den Buchstaben q ausgewichen. Für die Entität *Query* bedarf es keine Notation, da diese in dem Datenmodell dieser Arbeit keine Rolle spielt.

bspw. Clicks und Views etc.) werden in dieser Arbeit allgemein als Feedbackdaten F bezeichnet. Alle Daten, die von einem Recommender-System herangezogen werden, um die Empfehlungen für einen Request zu berechnen, werden in dieser Arbeit als *Lerndaten* L bezeichnet (im kollaborativen Filtern klassischerweise die Bewertungen von allen Benutzerinnen und Benutzern). Diesen Begriff nutzen wir auch, wenn kein Lernen im eigentlichen Sinne stattfindet (z. B. bei speicherbasierten Verfahren, vgl. Abschnitt 2.1.3). Lerndaten sind in der Regel eine Untermenge der Feedbackdaten.

Eine Empfehlung, die ein Recommender-System als Antwort auf einen Request ausgibt, bezeichnen wir mit einem e. e stellt somit eine sortierte Liste von K Objekten dar, die der aktiven Benutzerin bzw. dem aktiven Benutzer u' als Empfehlung angezeigt werden soll. Die Menge aller möglichen Empfehlungen bezeichnen wir mit E. E enthält somit alle möglichen K-elementigen, sortierten Listen, die mit den Empfehlungskandidaten gebildet werden können. Ein Datenstrom, über den Empfehlungslisten e übertragen werden, wird als $\mathcal{E}$ dargestellt.

2 Grundlagen

Um den Einsatz eines Datenstrommanagementsystems für die Umsetzung eines Recommender-Systems zu untersuchen, müssen zunächst die Grundlagen in diesen beiden Gebieten betrachtet werden. Dazu führt dieses Kapitel zunächst in Abschnitt 2.1 in die Grundlagen der Recommender-Systeme und anschließend in Abschnitt 2.2 in die der Datenstrommanagementsysteme ein.

2.1 Recommender-Systeme

Recommender-Systeme findet man in vielen Informationssystemen. Sie haben das Ziel, einer Benutzerin bzw. einem Benutzer aus einer großen Menge an Objekten (engl. *item*; z. B. Filme/Videos, Musikstücke, Künstler, Dokumente, Point of Interests etc.) einige wenige zu empfehlen, für die er/sie sich interessiert. Als erste wissenschaftliche Veröffentlichungen zu dem Thema gelten der Journal-Beitrag zum Tapestry-System [Gol+92] sowie der Proceedings-Beitrag zum GroupLens-System [Res+94]. Das Tapestry-System adressiert das automatische Filtern von E-Mails, das GroupLens-System fokussiert die Empfehlung von Nachrichtenartikeln.

Die Menge der Objekte, die einer Benutzerin oder einem Benutzer empfohlen werden, wird in dieser Arbeit als Empfehlungen oder Empfehlungsmenge e bezeichnet. Diese Empfehlungsmenge ist in der Regel eine sortierte Liste der K Objekte, die vom Recommender-System zur Empfehlung ermittelt wurden. Ein Recommender-System ist dafür zuständig, auf einen Request for Recommendations $q \in Q$ eine solche Menge $e \in E$ zurückzugeben. Ein Recommender-System kann somit als folgende Funktion betrachtet werden:

$$f_{\text{RecSys}}(q) \mapsto e \tag{2.1}$$

Eine gängige Methode ist die des kollaborativen Filterns (*CF*, engl. *Collaborative Filtering*), welche Informationen über genutzte Objekte mehrerer Benutzerinnen und Benutzer verarbeitet, um Empfehlungen für eine bestimmte Benutzerin bzw. einen bestimmten Benutzer zu ermitteln. Dahinter steckt folgende Annahme: Wenn die Menge der Objekte, die eine Benutzerin bzw. ein Benutzer konsumiert hat, eine große Überschneidung mit der Menge der Objekte hat, die eine andere Benutzerin

bzw. ein anderer Benutzer konsumiert hat, so sind auch die Objekte für den jeweils anderen interessant, die nur einer von beiden konsumiert hat.

Neben dem kollaborativen Filtern sind einige weitere Methoden für Recommender-Systeme entstanden. Weit verbreitet ist das inhaltsbasierte Filtern (engl. *content-based filtering*), welches der Benutzerin bzw. dem Benutzer Objekte empfiehlt, welche inhaltlich ähnlich zu seinen bzw. ihren bisher konsumierten Objekten sind. Weitere Methoden integrieren zusätzliche Informationen in die Empfehlungsberechnung, wie zum Beispiel Kontextdaten (*kontextsensitive Recommender-Systeme*), formales Wissen (*knowledge-based Recommender-Systeme*) oder demografische Daten der Benutzer (*demografische Recommender-Systeme*) [Bur02].

Die Herausforderung, der sich die Recommender-System-Forschung stellt, besteht aus dem Finden einer Bewertungsfunktion (engl. *Rating Function*), die eine Vorhersage ermittelt, wie groß das Interesse einer Benutzerin bzw. eines Benutzers an einem Objekt ist. Um eine Bewertungsfunktion zu ermitteln, wird i. d. R. das bekannte Interesse an Objekten verarbeitet, die die Benutzerinnen und Benutzer bereits genutzt haben. Dieses Interesse wird durch eine Bewertung (engl. *Rating*) quantifiziert. Die Bewertung von Objekten, die die Benutzerin bzw. der Benutzer bereits kennt, kann explizit durch die Benutzerin bzw. den Benutzer angegeben (die Benutzerin bzw. der Benutzer wird dazu aufgefordert, ein bestimmtes Objekt zu bewerten) oder implizit vom Verhalten der Benutzerin bzw. des Benutzers abgeleitet werden (im einfachsten Fall durch eine binäre Bewertung: Objekt genutzt vs. nicht genutzt).

In der klassischen Recommender-System-Forschung hat ein Recommender-System als Entitäten eine Menge Benutzerinnen und Benutzer U sowie eine Menge Objekte I und greift auf einen Datensatz L mit bekannten Bewertungen zurück, die aus Benutzer-Objekt-Bewertung-Tripeln bestehen. Die Aufgabe eines Recommender-Systems ist es, einer aktiven Benutzerin bzw. einem aktiven Benutzer $u' \in U$ eine Menge an Objekten $e \subset I$ zu empfehlen, für die diese bzw. dieser sich interessiert. Dazu bildet es als Erstes die Menge der Empfehlungskandidaten $C \subseteq I$. Dies sind üblicherweise alle Objekte, zu denen keine Bewertungen der Benutzerin bzw. des Benutzers vorliegen und somit für den diese bzw. diesen unbekannt sind. Das Recommender-System schätzt für jedes infrage kommende Objekt $c \in C$ eine Bewertung $\hat{r} \in R$. Der Wertebereich R der Bewertungen ist beispielsweise $[0, 1]$, $[1, 5]$, oder $\mathbb{R}$. Je höher der Wert ist, desto stärker das (geschätzte) Interesse. Die Empfehlungsmenge e wird durch die K Objekte der Empfehlungskandidaten gebildet, die die höchsten Bewertungen haben (Top-K-Menge). Die gesuchte Bewertungsfunktion zur Schätzung der Bewertungen ist eine Annäherung $\hat{f}_R$ an eine wahre aber unbekannte Bewertungsfunktion

$$f_R : U \times I \to R \tag{2.2}$$

Diese Bewertungsfunktion kann zu jedem Benutzer-Objekt-Paar eine Bewertung $\hat{r}$ berechnen [Ado+05].

2.1.1 Abgrenzung von Recommender-Systemen zu Information Retrieval

Information Retrieval ist laut [BR+11] ein Teilgebiet der Informatik, welches primär zum Ziel hat, Benutzerinnen und Benutzer einen einfachen Zugang zu Informationen zu verschaffen, die für diese von Interesse sind.

Die Forschungsbereiche des Information Retrieval und der Recommender-Systeme teilen sich eine gemeinsame Herausforderung: Wie kann ein Softwaresystem ermitteln, welche Objekte für die Benutzerin bzw. den Benutzer interessant sind? IR-Systeme haben die Aufgabe, einer Benutzerin bzw. einem Benutzer zu einer Suchanfrage (engl. *search query*; in der Regel in Form von Suchwörtern) diejenigen Objekte anzuzeigen, die entsprechend der Suchanfrage relevant sind.

Bei ursprünglichen IR-Systemen spielen die Benutzerin bzw. der Benutzer und seine bzw. ihre generellen Interessen keine Rolle. Allein die Suchanfrage bestimmt die Objekte, die vom System als relevant erachtet werden. Bei Recommender-Systemen wiederum gibt es im klassischen Sinne keine Suchanfrage an das System. Es wird aufgrund der Benutzerbewertungen darauf geschlossen, welche weiteren Elemente für die Benutzerin bzw. den Benutzer interessant sind.

In beiden Fällen wählt das System aus der Menge aller infrage kommenden Objekte (Kandidaten) eine Teilmenge aus, die der Benutzerin bzw. dem Benutzer präsentiert wird. Dazu wird eine Rangfolge der Kandidaten bestimmt (engl. *ranking*). Die relevantesten Elemente erhalten einen hohen Rang, weniger relevante Elemente einen niedrigen. Das Bestimmen einer Rangfolge spielt im Information Retrieval eine große Rolle (z. B. in [CSS99; Joa02; BRL06; Cha07]). Viele IR-Verfahren lassen sich auf Recommender-Systeme übertragen.

2.1.2 Recommender-System-Methoden: inhaltsbasiertes vs. kollaboratives Filtern

Eine grundsätzliche Einteilung von Recommender-System-Methoden basiert auf der Frage, welche Daten zur Bestimmung der Empfehlungen genutzt werden. Das inhaltsbasierte Filtern (engl. *content-based filtering*) nutzt als Datenbasis das von einer Benutzerin bzw. einem Benutzer aktuell genutzte Objekt und/oder in der

Vergangenheit von der Benutzerin bzw. vom Benutzer genutzte Objekte um die Objekte zu empfehlen, die den genutzten Objekten am ähnlichsten sind. Zusätzlich können auch Bewertungen zu den Objekten einbezogen werden, um die Objekte zu empfehlen, die den höchstbewerteten am ähnlichsten sind. Basierend auf einer gegebenen Ähnlichkeitsfunktion $f_{\text{sim}} : I \times I \to [0, 1]$, kann eine Bewertungsfunktion für das inhaltsbasierte Filtern wie folgt definiert werden:

$$\hat{f}_R(u, i) = \frac{1}{|I_u|} \sum_{i' \in I_u} f_{\text{sim}}(i', i) \cdot r_{u,i'} \tag{2.3}$$

Diese Funktion berechnet das arithmetische Mittel der Bewertungen $r_{u,i'}$ aller bereits bewerteten Objekte der Benutzerin bzw. des Benutzers (I_u) jeweils gewichtet mit der Ähnlichkeit des zur Bewertung gehörenden Objektes i' mit dem zu bewertenden Objekt i [Bur02].

Das inhaltsbasierte Filtern hat zwei Nachteile. Erstens muss eine Ähnlichkeitsfunktion für die Objekte existieren. Während dies für textbasierte Objekte keine große Herausforderung darstellt (man nehme z. B. die *Term Frequency – Inverse Document Frequency*, TF-IDF [Jon72]), so ist es für andere Objekte wie z. B. Videos schwieriger, geeignete Merkmale zum inhaltlichen Vergleich der Objekte zu extrahieren. Der zweite Nachteil ist die starke Fokussierung auf inhaltliche Nähe. Objekte, die auch dem Interesse der Benutzerin bzw. des Benutzers entsprechen, aber keine inhaltliche Nähe zu bereits benutzten Objekten aufweisen, kommen bei dem Ansatz zur Empfehlung nicht infrage.

Einen anderen Ansatz bietet das kollaborative Filtern. Hierbei werden der Benutzerin bzw. dem Benutzer Objekte empfohlen, die andere Benutzerinnen bzw. Benutzer mit ähnlichem Bewertungsverhalten hoch bewerten. Das führt dazu, dass nicht die Objekte verglichen werden müssen, sondern lediglich die Benutzerinnen und Benutzer aufgrund der bisherigen Bewertungen. Gegeben der Annahme, dass Personen mit ähnlichem Bewertungsverhalten ähnliche Interessen haben, werden auch die hochbewerteten Objekte ähnlicher Personen empfohlen, die nicht zwangsläufig ähnlich zu den bereits bewerteten Objekte sind.

Ein einfaches Beispiel eines Ansatzes für das kollaborative Filtern stellt das nachbarschaftsbasierte Verfahren nach [BHK98] dar. Gegeben sei eine Funktion $f_{\text{sim}} : U \times U \to [0, 1]$ sowie der Wert $\bar{r}_u$. Die Funktion f_{sim} ist eine Ähnlichkeitsfunktion, die zwei Benutzerinnen bzw. Benutzer aufgrund des Bewertungsverhaltens vergleicht. Der Wert $\bar{r}_u$ ist das arithmetische Mittel aller bekannten Bewertungen der Benutzerin bzw. des Benutzers u. Mit f_{sim} und $\bar{r}_u$ kann eine einfache Bewertungsfunktion für

das kollaborative Filtern nach [BHK98] wie folgt definiert werden:

$$\hat{f}_R(u,i) = \bar{r}_u + \frac{1}{\sum\limits_{u' \in U \setminus u} |f_{\text{sim}}(u,u')|} \sum_{u' \in U \setminus u} f_{\text{sim}}(u,u')(r_{u',i} - \bar{r}_{u'}) \tag{2.4}$$

Diese Funktion berechnet für alle anderen Personen des Recommender-Systems die mit Benutzerähnlichkeit gewichtete Summe der Abweichung der Bewertungen von der durchschnittlichen Bewertung der Benutzerin bzw. des Benutzers.

2.1.3 Recommender-System-Methoden: speicherbasierte vs. modellbasierte Methoden

Neben der Einteilung nach Datenbasis lassen sich Recommender-Systeme danach unterscheiden, ob sie speicherbasiert oder modellbasiert sind. Die im vorherigen Abschnitt vorgestellten Bewertungsfunktionen sind speicherbasiert. Das bedeutet, sie benötigen zur Schätzung einer Bewertung die kompletten Bewertungsdaten (oder Alternativ eine möglichst repräsentativ ausgewählte Teilmenge, die die Gesamtmenge abbildet). Dem gegenüber stehen modellbasierte Methoden, die aus dem Datensatz ein Modell berechnen, das zur Schätzung genutzt werden kann.

Im Gegensatz zu speicherbasierten Methoden bestehen modellbasierte Methoden aus zwei Phasen: In der ersten Phase wird das Modell gelernt, in der zweiten Phase wird das Modell zur Schätzung der Bewertung genutzt. Gegeben sei ein Datensatz $\mathcal{R}$ bestehend aus einer Menge Benutzer-Objekt-Bewertung-Tripeln (u,i,r), so berechnet ein Lernalgorithmus $l(\mathcal{R}) \mapsto \hat{f}_R$ ein Modell $\hat{f}_R$, welches eine Bewertungsfunktion realisiert. Im Gegensatz zu Bewertungsfunktionen bei speicherbasierten Methoden, bei denen über große Teile des Datensatzes iteriert werden muss, ist die Bewertungsschätzung mit dem gelernten Modell mit geringem Berechnungsaufwand möglich (i. d. R. durch einfache arithmetische Operationen wie die Multiplikation zweier Skalare). Somit können, sobald das Modell gelernt wurde, auch bei einer großen Menge an Bewertungsdaten die Bewertungen mit konstanter Laufzeit geschätzt werden. Lediglich die Laufzeit des Lernalgorithmus ist abhängig von der Menge der Bewertungsdaten.

Ein Nachteil der modellbasierten Methode ist, dass ein Modell nur die Bewertungsdaten berücksichtigt, die beim Lernen des Modells zur Verfügung standen. Bewertungen, die nach dem Lernen des Modells zum Datenbestand hinzugefügt werden, werden bei der Schätzung der Empfehlungen nicht berücksichtigt.

Gängige Methoden für modellbasierte Recommender-Systeme basieren auf der Matrixfaktorisierung (MF) [KBV09]. Dazu werden die Bewertungsdaten als dünn be-

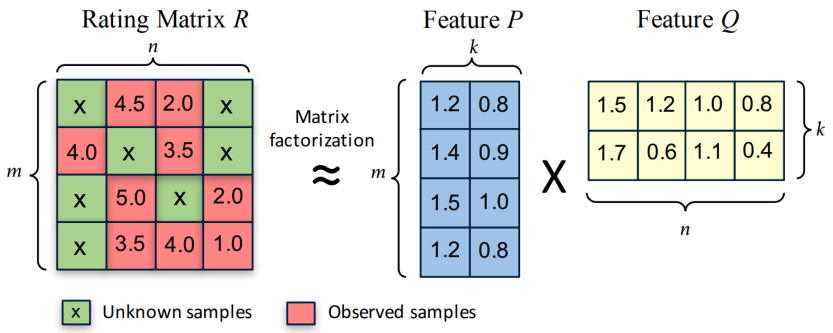


Abbildung 2.1 Matrixfaktorisierung (Quelle: [Xie+16])

setzte Bewertungsmatrix $\mathbf{R} \in R^{N \times M}$ (für N Benutzerinnen und Benutzer und M Objekte) aufgefasst, bei der die Zeilen den Benutzerinnen und Benutzern und die Spalten den Objekten entsprechen und die als Werte die Bewertungen enthält. Mithilfe der MF wird $\mathbf{R}$ in zwei Matrizen $\mathbf{P} \in \mathbb{R}^{N \times K}$ und $\mathbf{Q} \in \mathbb{R}^{K \times M}$ zerlegt, sodass die Matrix $\hat{\mathbf{R}} = \mathbf{P} \times \mathbf{Q}$ den Werten von $\mathbf{R}$ nahe kommt (vgl. Abbildung 2.1). $\mathbf{P}$ bezeichnet man auch als User-Feature-Matrix mit zeilenweisen User-Feature-Vektoren $\vec{p}_u$ und $\mathbf{Q}$ als Item-Feature-Matrix mit spaltenweisen Item-Feature-Vektoren $\vec{q}_i$. Der Parameter K bestimmt die Anzahl der Features je User-Feature-Vektor bzw. Item-Feature-Vektor. Ein Feature kann als eine Eigenschaft der Objekte interpretiert werden. Hat eine Benutzerin bzw. ein Benutzer beispielsweise im ersten Element von $\vec{p}_u$ einen hohen Wert, so bevorzugt dieser Objekte, die auch im ersten Element von $\vec{q}_i$ hohe Werte haben. Eine Schätzung für eine Bewertung wird mit Hilfe der Matrizen $\mathbf{P}$ und $\mathbf{Q}$ wie folgt berechnet:

$$\hat{f}_R(u,i) := \vec{p}_u \vec{q}_i \tag{2.5}$$

2.1.4 Nutzung zusätzlicher Informationen in Recommender-Systemen

Benutzerinnen und Benutzer können zu verschiedenen Zeitpunkten und in verschiedenen Situationen unterschiedliches Interesse an Objekten haben (*Concept Drift*). Um dies bei den Bewertungsschätzungen miteinzubeziehen, können Kontextdaten sowie der Zeitpunkt der Bewertungen berücksichtigt werden.

Kontextsensitive Recommender-Systeme Das Thema Context-Awareness wurde das erste Mal 1994 von Schilit und Theimer beschrieben [ST94]. Als *Context-Aware Computing* bezeichnen sie die Fähigkeit von (mobilen) Anwendungen, die Umgebung zu erkennen und auf deren Änderung zu reagieren. Um das zu erreichen, muss eine Anwendung Informationen über die Umgebung erheben und weiterverarbeiten. Die heute gängige Definition zu *Context* stammt von Abowd et al. [Abo+99]:

> Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Der Begriff *Kontext* wird in diesem Umfeld somit für jegliche Information benutzt, die die Situation einer Entität charakterisiert. Des Weiteren definieren sie *Context-Aware* wie folgt:

> A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

Ein System ist demnach *kontextsensitiv* (engl. *context-aware*), wenn es den Kontext benutzt, um der Benutzerin bzw. dem Benutzer *relevante* Informationen oder Dienste zur Verfügung zu stellen.

Ein Problem von Recommender-Systemen ohne Berücksichtigung des Kontexts ist, dass die Empfehlungen nicht auf die Situation der Benutzerin bzw. des Benutzers angepasst sind. Kauft man zum Beispiel in einem Online-Shop ein Geschenk für sein Kind, hat man an ganz anderen Produkten Interesse, als wenn man für sich selber einkauft. Relevante Kontextinformationen könnten die Tages- und Jahreszeit, das Wetter oder der Ort sein. Möchte man beispielsweise eine Veranstaltung für das Wochenende empfohlen bekommen, so hat man wahrscheinlich bei Regenwetter kein Interesse an Open-Air-Veranstaltungen. Aus diesem Grund schlagen Adomavicius et al. [Ado+05] einen mehrdimensionalen Ansatz für Recommender-Systeme vor, die *multidimensionalen* (MD) oder *kontextsensitiven Recommender-Systeme*.

Durch die Integration des Kontexts kann die Benutzerin bzw. der Benutzer nicht mehr nur *eine* Bewertung für ein Element vergeben, sondern die Bewertung kann, je nach Situation, variieren. Während bei 2D-Recommender-Systemen (ohne Kontextintegration) die Empfehlung von neuen Elementen im Vordergrund steht, ist bei kontextsensitiven Recommender-Systemen zusätzlich interessant, aus allen Elementen diejenigen Elemente zu empfehlen, die für *den aktuellen Kontext* interessant sind.

Um Kontextinformationen in Recommender-Systeme zu integrieren, schlagen Adomavicius und Tuzhilin verschiedene Paradigmen vor. Dazu gehören *Contextual Pre-Filtering*, *Contextual Post-Filtering* und *Contextual Modeling* [AT11]. Beim *Contextual Pre-Filtering* (Abbildung 2.2a) werden nur die Bewertungen berücksichtigt, die den gleichen Kontext haben. Alle anderen werden herausgefiltert. Anschließend

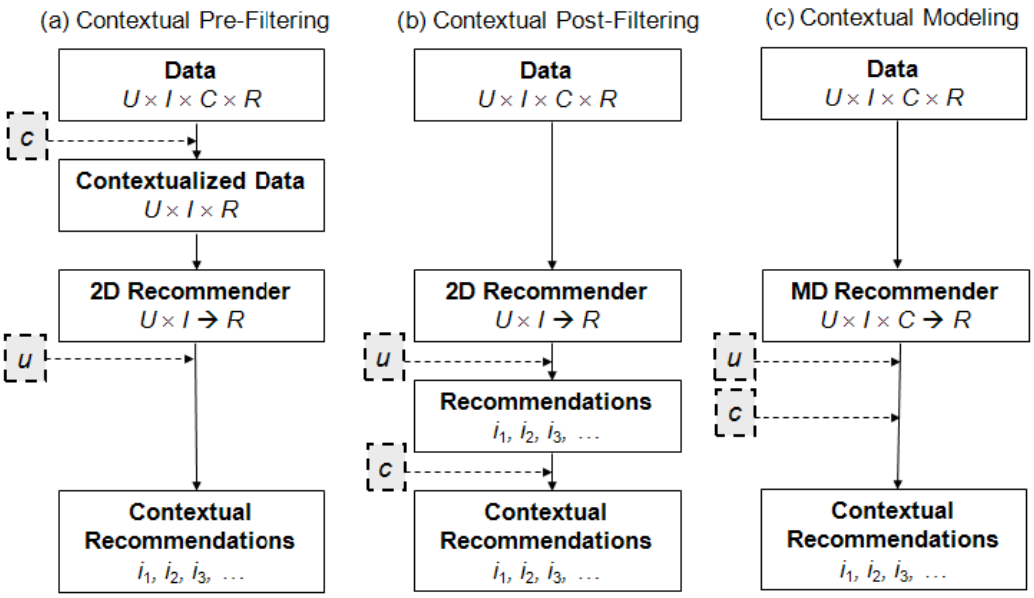


Abbildung 2.2 Paradigmen für die Integration von Kontextinformationen in Recommender-Systeme (Quelle: [AT11])

ist das Vorgehen identisch mit dem der traditionellen Recommender-Systeme. Das *Contextual Post-Filtering* (Abbildung 2.2b) verhält sich vorerst wie ein traditionelles Recommender-System. Dabei werden die Kontextdaten anfangs ignoriert. Anschließend werden die Empfehlungen aufgrund der Kontextdaten neu gewichtet bzw. gefiltert. Das *Contextual Modeling* (Abbildung 2.2c) passt die Recommender-Methoden an, um Kontextdaten direkt in die Bewertungsschätzung einzubeziehen. Letzteres erweitert die Bewertungsfunktion um weitere Dimensionen. Somit erweitert sich die Bewertungsfunktion bei der Berücksichtigung von N verschiedenen Kontextinformationen $X_1, \ldots, X_N$ wie folgt:

$$f_R : U \times I \times X_1 \times \ldots \times X_N \rightarrow R \tag{2.6}$$

Zeitbasierte Recommender-Systeme Das Interesse der Benutzerinnen und Benutzer an Objekten kann sich im Laufe der Zeit ändern. Man spricht in diesem Zusammenhang von Concept Drift. Abbildung 2.3 gibt einen Überblick über die verschiedenen Arten von Concept Drift. Bezogen auf Recommender-Systeme kann sich das Interesse an Objekten plötzlich ändern (*sudden/abrupt*), langsam verschieben (*incremental*), mit einer Übergangsphase allmählich ändern (*gradual*) oder sich nur für einen begrenzten Zeitraum ändern (*reoccuring concepts*). Einmalige Ausreißer von den üblichen Interessen werden dabei nicht als Concept Drift behandelt.

Einen Überblick über Recommender-Systeme, die den zeitlichen Zusammenhang von historischen Bewertungen bei der Schätzung einer Bewertung berücksichtigen

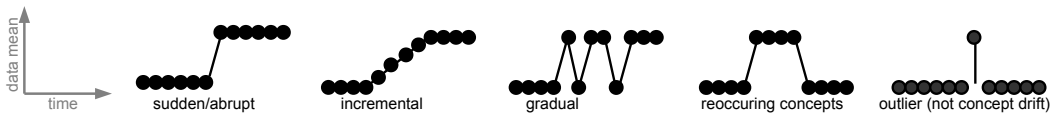


Abbildung 2.3 Arten von Concept Drift (Quelle: [Gam+13])

(*zeitbasierte Recommender-Systemen*), findet man bei Campos et al. [CDC14]. Im Gegensatz zu Campos et al. werden in dieser Arbeit nur Recommender-Systeme als zeitbasiert angesehen, die die Zeit als einen Punkt auf einem unendlichen Zeitstrahl berücksichtigen. Wiederkehrende Zeitpunkte oder -spannen wie Tageszeiten, Jahreszeiten etc. werden als Kontextinformationen angesehen. Recommender-Systeme, die diese Informationen verarbeiten, fallen somit unter die Kategorie der kontextsensitiven Recommender-Systeme.

Die Schätzung der Bewertung in einem zeitbasierten Recommender-System ist somit abhängig von dem Zeitpunkt $t \in T$, zu dem eine Empfehlung gegeben werden soll. t kann dabei der jeweils aktuelle Zeitpunkt sein (Welche Objekte sind für die anfragende Benutzerin bzw. den anfragenden Benutzer zum jetzigen Zeitpunkt interessant?) oder ein Zeitpunkt in der Zukunft (Welche Objekte werden für die anfragenden Benutzerin bzw. den anfragenden Benutzer morgen Abend interessant sein?). Eine Möglichkeit, den Zeitpunkt der Empfehlung zu Berücksichtigen, ist diesen analog zu kontextsensitiven Recommender-Systemen in die Bewertungsfunktion zu integrieren:

$$f_R : U \times I \times T \to R \tag{2.7}$$

Alternativ kann für jeden Zeitpunkt t eine eigene Bewertungsfunktion definiert werden:

$$f_{R,t} : U \times I \to R \tag{2.8}$$

2.1.5 Evaluation von Recommender-Systemen

Die Evaluation von Recommender-Systemen kann offline mit Hilfe eines Datensatzes oder online mittels Evaluation durch Benutzerinnen und Benutzer erfolgen (siehe z. B. [Her+04, S. 12 f]). Der Vorteil der Offline-Evaluation ist, dass diese schnell, ökonomisch und in gleicher Art auf verschiedenen Datensätzen oder mit verschiedenen Algorithmen/Methoden durchgeführt werden kann. Das hat den Vorteil, dass die Evaluation gut vergleichbar und nachvollziehbar ist [Her+04]. Die Online-Evaluation spiegelt dafür näher die Realität wider und berücksichtigt zusätzlich die Beeinflussung der Benutzerinnen und Benutzer durch die Empfehlungen.

Bei Recommender-Systemen können verschiedene Aspekte evaluiert werden. Weit verbreitet ist die Evaluation der Genauigkeit bei der Schätzung der Bewertung. Andere Aspekte sind zum Beispiel Diversität, Abdeckung und Zufriedenheit der Benutzerinnen und Benutzer [Her+04; Lü+12]. Shani und Gunawardana beschäftigen sich in [SG11] ausführlich mit dem Thema der Evaluation von Recommender-Systemen.

Evaluationsmethoden für die Evaluation der Bewertungsschätzung Um die Genauigkeit der Schätzung einer Bewertung im Rahmen der Offline-Evaluation zu bewerten, werden Daten eines Datensatzes $\mathcal{R}$ mit bekannten Bewertungsdaten eines Recommender-Systems als Lerndaten $\mathcal{L}$ von der Bewertungsfunktion $\hat{f}_R$ zur Schätzung unbekannter Bewertungen genutzt. Des Weiteren werden Daten aus $\mathcal{R}$ als Testdaten $\mathcal{T}$ genutzt. Für jedes Testdatum $(u, i, r) \in \mathcal{T}$ wird die Schätzung $\hat{f}_R(u, i)$ mit der wahren Bewertung r mit einer Evaluationsmetrik (siehe nächster Abschnitt) verglichen.

Die Evaluationsmethode bestimmt, welche Daten als Lerndaten und welche als Testdaten genutzt werden. Dabei ist es wichtig, dass Daten, die bereits als Lerndaten genutzt wurden, nicht mehr als Testdaten genutzt werden dürfen, da die Bewertungsfunktion die wahren Bewertungen für diese Daten bereits kennt. Eine einfach Evaluationsmethode teilt deshalb die Daten zufällig in Lern- und Testdaten auf (*Hold-out*). Um auszuschließen, dass zufällig leicht zu schätzende Daten als Testdaten ausgewählt wurden, teilt die *K-fache Kreuzvalidierung* Daten in K gleichgroße Mengen. In K Durchläufen wird anschließend jeweils ein Teil als Testdaten und die restlichen $K - 1$ Mengen als Lerndaten genutzt. Das Evaluationsergebnis ist das arithmetische Mittel aller K Durchläufe.

Bei zeitbasierten Recommender-Systemen hat die Kreuzvalidierung den Nachteil, dass die zeitliche Reihenfolge der Daten nicht erhalten bleibt und somit nicht sinnvoll anwendbar ist. Eine Alternative stellt die Evaluationsmethode *Interleaved Test-Then-Train* (ITTT) dar [CDC14]. Mit dieser Methode werden die Daten in zeitlicher Reihenfolge eingelesen. Bevor ein neues Datum $(u, i, r) \in \mathcal{R}$ in der Bewertungsfunktion zur Schätzung einer Bewertung berücksichtigt wird, wird dieses genutzt, um die bisherige Bewertungsfunktion zu testen. So wird zum Beispiel die Bewertung mit der bisherigen Bewertungsfunktion geschätzt und diese geschätzte Bewertung $\hat{r}$ wird mit der tatsächlichen Bewertung r verglichen. Erst nach diesem Schritt wird das neue Datum zu den Lerndaten hinzugefügt und für die nachfolgenden Bewertungsschätzungen berücksichtigt. Dadurch wird sichergestellt, dass zum Zeitpunkt des Testens die Bewertungsfunktion die wahre Bewertung noch nicht kennt, zur Schätzung aber alle zeitlich vor dem Testdatum liegende Lerndaten genutzt werden.

Diese Methode eignet sich daher besonders zur Evaluation eines Recommender-Systems im Datenstrom.

Evaluationsmetriken für die Evaluation der Bewertungsschätzung Die einfachste Möglichkeit der Beurteilung der Bewertungsschätzung für die Testdaten ist die Berechnung eines Fehlerwertes. Üblich ist die Berechnung des arithmetischen Mittels der absoluten Differenz (*Mean Absolute Error*, MAE), der quadrierten Differenz (*Mean Square Error*, MSE), sowie der Quadratwurzel aus dem MSE (*Root Mean Square Error*, RMSE) [Her+04]. Diese sind für einen Testdatensatz $\mathcal{T}$ und eine Bewertungsfunktion $\hat{f}_R$ wie folgt definiert:

$$\mathrm{MAE} := \frac{1}{|\mathcal{T}|} \sum_{(u,i,r)\in\mathcal{T}} |r - \hat{f}_R(u,i)| \tag{2.9}$$

$$\mathrm{MSE} := \frac{1}{|\mathcal{T}|} \sum_{(u,i,r)\in\mathcal{T}} \left(r - \hat{f}_R(u,i)\right)^2 \tag{2.10}$$

$$\mathrm{RMSE} := \sqrt{\mathrm{MSE}} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,i,r)\in\mathcal{T}} \left(r - \hat{f}_R(u,i)\right)^2} \tag{2.11}$$

Evaluationsmetriken für die Evaluation der Empfehlungsmenge Darüber hinaus gibt es Metriken, die weniger die Bewertung r, sondern die Reihenfolge der nach Bewertung sortierten Empfehlungen in den Vordergrund stellen. Dazu gehören die Metriken *Mean Reciprocal Rank (MRR)* und *Hit Rate*, die in dieser Arbeit genutzt werden.

Gegeben sei eine Menge von Objekten I'. Die Objekte $i' \in I'$ sind genau diese Objekte, die von einer Benutzerin bzw. von einem Benutzer genutzt bzw. positiv bewertet wurden. Ein Recommender-System, welches die Empfehlungskandidaten in eine Reihenfolge bringt, in der die besten Empfehlungen an der Spitze der Liste stehen (z. B. durch Sortierung nach geschätzter Bewertung $\hat{r}$), kann die Position p des ersten Vorkommens eines Objektes i' aus der Menge I' bestimmen. Der *Reciprocal Rank* ist somit $\mathrm{RR} = \frac{1}{p}$. Das arithmetische Mittel des Reciprocal Ranks für eine Menge von Empfehlungsberechnungen ist der *Mean Reciprocal Rank (MRR)*.

Alternativ wird mit der *Hit Rate* das Verhältnis der Empfehlungen angegeben, in denen sich mindestens ein Objekt $i' \in I'$ befindet. Sei e eine Empfehlungsliste mit K Empfehlungen, so ist die Anzahl der „erfolgreichen“ Empfehlungen n' die Anzahl der Empfehlungen, in denen sich mindestens ein Objekt i' aus I' befindet. Das Verhältnis zu allen Empfehlungen n ist die Hit Rate: $\mathrm{HitRate} = \frac{n'}{n}$

2.2 Datenstromverarbeitung und Datenstrommanagementsysteme

Seit den 1970er/1980er Jahren sind relationale Datenbankmanagementsysteme (DBMS) die bedeutendste Technik zur Speicherung und Organisation von Daten. Ein Datenbankmanagementsystem ermöglicht es, dass sich Entwickler von Informationssystemen auf den Kern der Anwendung fokussieren können, ohne selber Strukturen zur Speicherung von und zum Zugriff auf Daten schaffen zu müssen. Ein Datenbankmanagementsystem ermöglicht u. a. die einheitliche Verwaltung der Daten (*Datenintegration*), stellt Operationen zum Hinzufügen, Abfragen, Verändern und Löschen von Daten bereit (sog. *CRUD-Operationen*: Create, Receive, Update und Delete), übernimmt die Überwachung der Datenkonsistenz (*Integrationssicherung*) sowie der Zugriffskontrolle und sorgt für ein Transaktionsmanagement [HS00, S. 6 ff].

In einigen Informationssystemen ist die langfristige Speicherung von Daten allerdings nicht von Bedeutung, sondern das (kontinuierliche) Gewinnen von Informationen aus einem kontinuierlich wachsenden bzw. veränderten Datenbestand. Dazu müssen die Daten nicht zwangsläufig in einer Datenbank gespeichert werden. Insbesondere, wenn neue Daten in einer hohen Datenrate erzeugt werden oder die Verarbeitung datengetrieben erfolgen soll, kann es effizienter sein, wenn das verarbeitende System auf die kontinuierliche Verarbeitung der Daten statt auf die Speicherung der Daten und deren Zugriff optimiert ist. Dazu gehört zum Beispiel die Haltung der Daten im Hauptspeicher mit kurzen Zugriffszeiten statt auf Festplatten.

Ist die Datengrundlage eines Informationssystems eine kontinuierliche Sequenz an Daten, so spricht man von der Verarbeitung von Datenströmen. Dazu gibt es eine Vielzahl an Systemen und Programmierframeworks, die die Entwicklung von datenstromverarbeitenden Systemen vereinfachen. Unter dem Begriff Datenstrommanagementsystem (DSMS) sind Systeme entstanden, die die Konzepte von Datenbankmanagementsystemen auf die Verarbeitung von Datenströmen übertragen.

2.2.1 Datenströme und Datenstromelemente

Während ein Datenbankmanagementsystem wahlfreien Zugriff auf Daten auf der Festplatte oder im Arbeitsspeicher hat, erhält ein Datenstrommanagementsystem die Daten von einem kontinuierlichen Datenstrom. Ein Datenstrom ist eine Sequenz von Datenstromelementen. Das Datenstrommodel ist durch folgende Eigenschaften gekennzeichnet:

Aktive Datenquellen:

Elemente eines Datenstroms werden von einer aktiven Datenquelle erzeugt und versendet [Krä07]. Während ein Datenbankmanagementsystem aktiv Daten aus einer Datenbank ausliest (pull), verarbeitet ein Datenstrommanagementsystem die Daten, die von einer Datenquelle geliefert werden (push).

Reihenfolge der Datenstromelemente:

Das verarbeitende System hat keine Kontrolle über die Reihenfolge, in der die Datenstromelemente von der Quelle gesendet werden – weder innerhalb eines Datenstroms, noch zwischen verschiedenen Datenströmen [Bab+02].

Unendlichkeit von Datenströmen:

Datenströme sind potenziell unbegrenzt in ihrer Größe [Bab+02]. Das bedeutet, dass ein Datenstrommanagementsystem davon ausgehen muss, dass ein zu verarbeitender Datenstrom eine unendliche Anzahl Elemente liefert.

One-pass-Paradigma:

Sobald ein Element verarbeitet ist, wird es verworfen oder archiviert. Ein verarbeitendes System kann nicht erneut auf bereits verarbeitete Elemente zugreifen, es sei denn, es wird explizit im Arbeitsspeicher gehalten. Dies kann allerdings nur für einen begrenzten Teil der Datenstromelemente geschehen, da der Speicher im Gegensatz zum Datenstrom in der Größe begrenzt ist [Bab+02].

Ein Datenstromelement enthält Nutzdaten, die in verschiedenen Datenformaten vorliegen können. In dieser Arbeit werden nur Nutzdaten in relationaler Form betrachtet, bei denen die Daten jedes Elements eines Datenstroms ein festgelegtes Schema haben. Ein Datenstrom kann somit als eine Relation mit einer unendlichen Anzahl an Tupeln interpretiert werden. Ein Datenstromelement entspricht einem Tupel dieser Relation. Alternative Datenformate für die Nutzdaten der Datenstromelemente sind zum Beispiel Schlüssel-Wert-Paare, XML oder RDF.

2.2.2 Kontinuierliche Anfragen, Anfragepläne und Operatoren

Die Verarbeitung der Daten in einem Datenbankmanagementsystem erfolgt üblicherweise mit Hilfe von sogenannten Einmalqueries. Diese werden auf den aktuellen Datenbestand ausgeführt und führen zu einer einmaligen Antwort sobald die Ausführung der Query terminiert. Bei der Verarbeitung von kontinuierlichen Datenströmen wird eine kontinuierliche Query gestellt. Diese wird einmalig dem Datenstrommanagementsystem übergeben und produziert kontinuierlich Ergebnisse [Bab+02]. Eine

Gegenüberstellung der Anfrageverarbeitung eines Daten*bank*managementsystems und eines Daten*strom*managementsystems zeigt Abbildung 2.4.

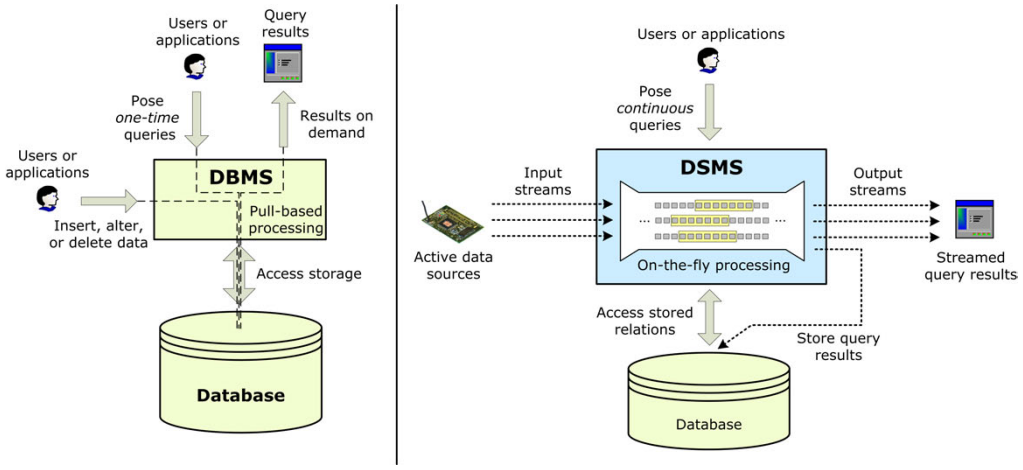


Abbildung 2.4 Gegenüberstellung der Anfrageverarbeitung in einem DBMS und einem DSMS (Quelle: [Krä07])

Eine kontinuierliche Query wird, wie bei Queries in Datenbankmanagementsystemen, in einer Anfragesprache (engl. *Query Language*) formuliert. Beispiele für Datenstrommanagementsystem-Anfragesprachen sind die SQL-ähnliche Sprache CQL („continuous query language“, [ABW06]) oder die PQL („procedural query language“). Die Query wird anschließend in einen Anfrageplan (engl. *Query Plan*, auch engl. *Query Execution Plan*) übersetzt. Dieser besteht aus Operatoren, die durch Warteschlangen miteinander verbunden sind [Bab+02]. Ein Operator bekommt die Datenstromelemente sequentiell übergeben und erzeugt wiederum einen Datenstrom als Ausgabe. Zur Verarbeitung von relationalen Datenströmen kann ein Datenstrommanagementsystem die Operatoren einer datenstrombasierten Variante der relationalen Algebra [Cod70] implementieren.

Ein Anfrageplan kann als gerichteter Graph interpretiert werden, dessen Knoten die Operatoren und deren Kanten die Warteschlangen darstellen. Ein Beispiel für einen einfachen Anfrageplan zeigt Abbildung 2.5. Dieser hat zwei Eingabedatenströme, auf denen mehrere Projektionen (π), eine Selektion (σ) und eine Verbund ($\bowtie$) ausgeführt werden. Ein Anfrageplan kann von einem Datenstrommanagementsystem durch äquivalente Umformungen optimiert werden (z. B. durch die Veränderung der Operatorreihenfolge), um den Ressourcenbedarf der Ausführung zu minimieren. Die Ausführung der Operatoren koordiniert ein Scheduler [Bab+02].

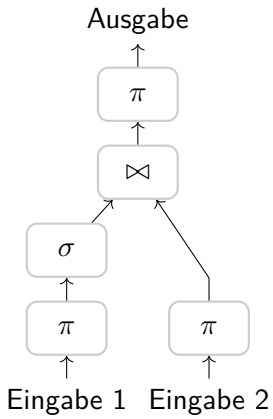


Abbildung 2.5 Beispiel eines Anfrageplans mit Operatoren der relationalen Algebra

2.2.3 Verarbeitung unendlicher Datenströme

Eine Anfrage eines Datenstrommanagementsystems verarbeitet kontinuierlich alle eingehenden Datenstromelemente und gibt bei jeder Änderung des Anfrageergebnisses dieses als ein neues Ergebnisdatenstromelement aus. So gibt zum Beispiel eine Anfrage, die das arithmetische Mittel aller Werte eines Attributes berechnet, bei jedem neuen Datenstromelement ein Element mit dem aktualisierten arithmetischen Mittel aus.

Da eine Anfrage eines Datenstrommanagementsystems nicht einmalig auf einen endlichen Datenbestand, sondern kontinuierlich auf nicht dauerhaft gespeicherte Daten ausgeführt wird, ist die Unterscheidung zwischen zustandslosen und zustandsbehafteten Operatoren wichtig. Ein zustandsloser Operator benötigt zur Verarbeitung der eingehenden Datenstromelemente nur das jeweils aktuelle Element. Ein Beispiel für einen zustandslosen Operator ist der SELECT-Operator. Dieser benötigt nur das aktuelle Datenstromelement, um zu entscheiden, ob die Selektionsbedingung erfüllt ist, und das Element ausgegeben oder verworfen werden soll. Im Gegensatz dazu benötigen zustandsbehaftete Operatoren weitere Daten, die aus der Verarbeitung vorheriger Datenstromelemente stammen. Diese müssen als Zustand im Speicher vorgehalten werden. Ein Beispiel für einen zustandsbehafteten Operator ist der JOIN-Operator. Dieser muss alle bisherigen Datenstromelemente im Speicher halten, die noch als Join-Partner infrage kommen, um bei jedem eingehenden Datenstromelement zu überprüfen, ob die Bedingung zur Kombination mit einem Element aus dem anderen Datenstrom erfüllt ist. Ein anderes Beispiel ist der Aggregationsoperator, der das arithmetische Mittel eines Wertes aller Datenstromelemente berechnet. Je nach Implementierung muss dieser zum Beispiel die Summe sowie die Anzahl der Werte im Speicher halten.

Die zu speichernde Datenmenge von vielen zustandsbehafteten Operatoren wächst mit der Menge der bereits verarbeiteten Datenstromelemente. Geht man von unendlichen Datenströmen aus, so wächst der benötigte Speicher unendlich an. Um das zu verhindern, gibt es verschiedene Ansätze. Eine Möglichkeit besteht darin, die bisherigen Daten zu verdichten. Dazu gehören Sampling-, Sketching-, Histogram- und Wavelet-Techniken [AP07]. Diese Methoden geben statt präzisen Antworten lediglich approximierte Antworten. Alternativ kann man sich auf einen Ausschnitt des Datenstroms beschränken. Man spricht in diesem Fall von einem Fenster-Ansatz. Üblich ist beispielsweise die Beschränkung auf die jüngsten Elemente eines Datenstroms (z. B. Elemente der letzten 15 Minuten oder die letzten 100 Elemente) [PS06]. Abbildung 2.6 stellt beispielhaft einen Datenstrom mit einem Fenster dar, welches 5 Zeiteinheiten umfasst. Zur vereinfachten Darstellung kommt zu jedem Zeitpunkt t ein neues Datenstromelement hinzu. Zum Zeitpunkt $t = 10$ werden alle Elemente betrachtet, die innerhalb der letzten 5 Zeiteinheiten hinzugekommen sind.

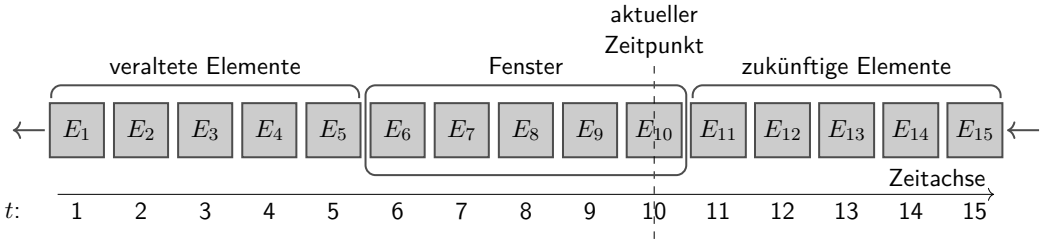


Abbildung 2.6 Ein Fenster der Breite $w = 5$ auf einen Datenstrom zum Zeitpunkt $t = 10$

Die Realisierung eines Fensteransatzes kann auf Operatorebene [Cra+03] oder auf Anfrageplanebene [KS09; Aba+05] erfolgen. Während sich der erste Ansatz nur auf die Verarbeitung innerhalb eines Operators auswirkt, gilt letzterer für alle Operatoren eines Anfrage(teil)plans. Eine gängige Methode zur Definition eines Fensters auf Anfrageplanebene ist das Einfügen eines Fensteroperators. Dieser hängt Metadaten an das Datenstromelement an, mit dem alle nachfolgenden Operatoren entscheiden können, ob das Element noch berücksichtigt werden soll. Die Methode der Positiv-Negativ-Marker [Aba+05] gibt die Datenstromelemente zunächst mit einem sogenannten Positiv-Marker aus, der bestimmt, dass das Element ab diesem Zeitpunkt gültig ist. Wird das Element ungültig, so wird dieses erneut ausgegeben, diesmal mit einem sogenannten Negativ-Marker. Sobald die nachfolgenden Operatoren das Element mit Negativ-Marker bekommen, wird das Element von diesen nicht mehr berücksichtigt. Eine Alternative stellt der Intervall-Ansatz [KS09] dar. Bei diesem annotiert ein Fensteroperator jedes Element mit einem halboffenen Gültigkeitsintervall $[t_s, t_e)$: Das Element ist ab dem Zeitpunkt t_s gültig und ab dem Zeitpunkt t_e ungültig. Der Zeitfortschritt wird bei diesem Ansatz nicht durch die Systemzeit, sondern durch das Eintreffen der Datenstromelemente bestimmt (Applikationszeit). Das bedeutet, dass nur Elemente mit überlappenden Gültigkeitsintervallen zur Anfrageverarbei-

tung zusammen betrachtet werden. Erreicht ein neues Element einen Operator, so werden alle Elemente nicht mehr berücksichtigt, deren Endzeitpunkt t_e kleiner oder gleich dem Startzeitpunkt t_s des neues Elements sind. Das setzt voraus, dass die Elemente grundsätzlich von allen Operatoren chronologisch nach Startzeitpunkt t_s sortiert ausgegeben werden.

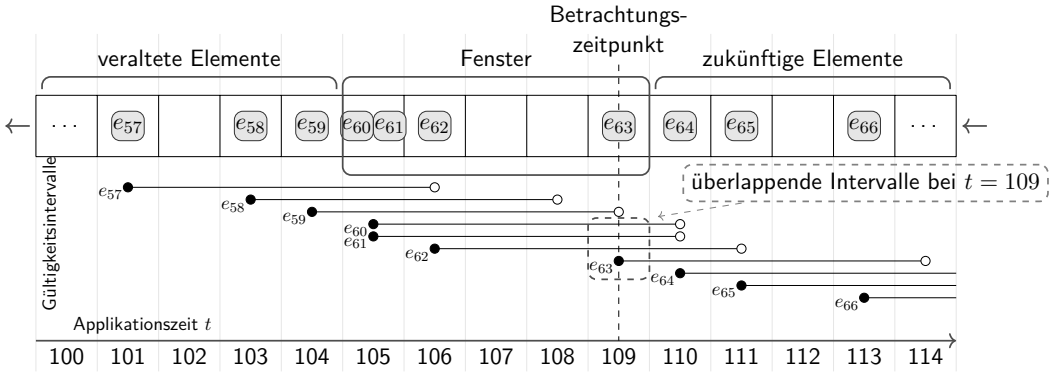


Abbildung 2.7 Ein Fenster der Breite $w = 5$ auf einen Datenstrom zum Zeitpunkt $t = 10$ mit Gültigkeitsintervallen der Datenstromelemente

Abbildung 2.7 zeigt das Zusammenspiel von Gültigkeitsintervallen und Fenstern. Oben ist erneut der Datenstrom aus Abbildung 2.6 mit dem Fenster zum Zeitpunkt $t = 10$ dargestellt. Unter dem Datenstrom befinden sich die Gültigkeitsintervalle $[t_s, t_e)$ der Datenstromelemente, wie sie ein Fensteroperator den Elementen als Metadaten anfügen würde. Der Startzeitpunkt des jeweiligen Intervalls wird als ausgefüllter, der Endzeitpunkt als nicht ausgefüllter Kreis dargestellt. Demnach wird das Element e_5 zum Zeitpunkt $t_s = 5$ gültig und ist ab $t_e = 10$ ungültig. Ein Operator, der das Element e_{10} erhält, betrachtet den Startzeitpunkt $t_s = 10$ von e_{10} als Indikator für den Zeitfortschritt (aufgrund der chronologischen Sortierung der Elemente ist darauffolgend kein Element mit einem geringerem Startzeitpunkt zu erwarten). Anders ausgedrückt könnte man sagen, dass der Operator mit Eintreffen von e_{10} die Applikationszeit von $t = 10$ erreicht hat. Bei jedem Eintreffen eines neuen Elements betrachtet ein zustandsbehafteter Operator somit alle Elemente, deren Gültigkeitsintervalle sich zum Zeitpunkt t_s des neuen Elements überlappen. In Abbildung 2.7 sind diese zum Zeitpunkt $t = 10$ die Elemente $e_6 - e_{10}$. Das Element e_5 hat als Endzeitpunkt $t_e = 10$ und ist somit bereits ungültig, da sich die Gültigkeitsintervalle von e_5 und e_{10} nicht überlappen.

Ohne die Beschränkung der Gültigkeitsintervalle durch einen Fensteroperator sind die Datenstromelemente unendlich gültig. Das Intervall kann in diesem Fall implizit als $[t_s, \infty)$ angenommen werden. Eine besondere Rolle spielt außerdem das Now-Fenster. In diesem Fall sind die Elemente nur für genau einen Zeitpunkt t gültig. Das zugehörige Gültigkeitsintervall $[t, t+1)$ wird im Folgenden mit der alleinigen Angabe von t dargestellt. Weitere Arten von Fenstern unterscheiden sich beispielsweise in

der Definition der Fensterbreite (zeitbasiert oder basierend auf die Anzahl der zu enthaltenden Elemente) [Krä07].

Ein Problem dieses Ansatzes, der die Information über den Zeitfortschritt allein aus den eingehenden Datenstromelementen gewinnt, ist das Fehlen dieser Information zwischen zwei aufeinanderfolgenden Datenstromelementen in einem Datenstrom. Betrachtet man zum Beispiel den UNION-Operator, der die Elemente aus zwei Datenströmen $\mathcal{S}_1$ und $\mathcal{S}_2$ zu einem Datenstrom $\mathcal{S}_3$ zusammenfügt, so kann dieser ein Element aus $\mathcal{S}_1$ erst ausgeben, wenn er sicher sein kann, dass kein weiteres Element in $\mathcal{S}_2$ mit geringerem Startzeitpunkt zu erwarten ist. Andernfalls kann die chronologische Ordnung in $\mathcal{S}_3$ nicht gewährleistet werden. Das bedeutet also, dass ein eingehendes Element e_1 aus $\mathcal{S}_1$ erst ausgegeben werden kann, wenn ein Element in $\mathcal{S}_2$ mit gleichem oder größerem Startzeitpunkt als t_s von e_1 eingeht. Solange kein Element aus $\mathcal{S}_2$ eingeht, hat der Operator keine Informationen über den Zeitfortschritt in $\mathcal{S}_2$ und kann somit e_1 nicht ausgeben. Das bedeutet, dass der Operator blockiert. Während bei hoher Datenrate die Blockierung kaum bemerkbar ist, kann bei geringer Datenrate die Verzögerung bei einem blockierenden Operator die Latenz der Datenstromverarbeitung deutlich vergrößern.

Um die Verzögerung durch blockierende Operatoren zu begrenzen, können Heartbeats [Tuc+03] eingesetzt werden. Heartbeats sind Datenstromelemente ohne Nutzdaten, die lediglich den Zeitfortschritt anzeigen. Zum Beispiel kann die Quelle, die die Datenstromelement aus $\mathcal{S}_2$ aus dem vorherigen Beispiel erzeugt, in regelmäßigen Abständen einen Heartbeat ausgeben. Erreicht ein Heartbeat den UNION-Operator, so kann dieser sichergehen, dass kein weiteres Element mit geringerem Startzeitpunkt in $\mathcal{S}_2$ zu erwarten ist, und das Element e_1 ausgeben.

2.3 Zusammenfassung

Diese Arbeit basiert auf den beiden Forschungsbereichen *Recommender-Systeme* sowie *Datenstrommanagementsysteme*. Ziel von Recommender-Systemen ist es, die Nützlichkeit von Objekten für Benutzerinnen und Benutzer zu schätzen, die häufig als Bewertung ausgedrückt wird, die eine Benutzerin bzw. ein Benutzer einem Objekt geben würde. Aus diesem Grund wird in dieser Arbeit allgemein von der Bewertung gesprochen, wenn die quantifizierte Nützlichkeit gemeint ist. Zur Schätzung der Bewertung wird eine Näherung an eine wahre aber unbekannte Bewertungsfunktion gesucht. Mit dieser Funktion können Bewertungen für Empfehlungskandidaten (in der Regel alle Objekte, die die Benutzerin bzw. der Benutzer noch nicht bewertet hat) geschätzt werden, um der Benutzerin bzw. dem Benutzer die K Objekte zu

empfehlen, die die höchsten geschätzten Bewertungen haben (Abschnitt 2.1). Recommender-Systeme haben Ähnlichkeiten mit IR-Systemen. Während letztere den Fokus auf das Finden von Objekten zu angegebene Suchwörter legen, werden bei Recommender-Systemen Objekte eher aufgrund der Historie der von Benutzerinnen und Benutzern verwendeten Objekte ausgewählt (Abschnitt 2.1.1). Recommender-Systeme kann man in inhaltsbasierte und kollaborative (Abschnitt 2.1.2), sowie in speicherbasierte und modellbasierte (Abschnitt 2.1.3) unterteilen. Außerdem können Informationen über den Kontext der Benutzerin bzw. des Benutzers sowie die Zeit der Bewertungen die Empfehlungen verbessern (Abschnitt 2.1.4). Zur Evaluation von Recommender-Systemen kann man verschiedene Methoden und Metriken nutzen (Abschnitt 2.1.5).

Datenstrommanagementsysteme bieten ähnlich den Daten*bank*managementsystemen einen abstrahierten Zugriff auf Daten (Abbildung 2.2). Während Datenbankmanagementsysteme Datensätze verwalten, die persistent gespeichert werden, verarbeiten Datenstrommanagementsysteme kontinuierliche Datenströme im Hauptspeicher. Dazu setzen sie ähnlich den Datenbankmanagementsystemen Anfragepläne und Operatoren ein (Abschnitt 2.2.2). Ein Problem stellt dabei dar, dass ein Datenstrom potenziell unendlich viele Elemente liefert. Aus diesem Grund kann die Gültigkeitsdauer von Elementen begrenzt werden (Abschnitt 2.2.3).

Aufbauend auf diesen Grundlagen werden im nächsten Kapitel Konzepte zur Umsetzung von Recommender-Systemen auf Basis von Datenstrommanagementsystemen beschrieben. Dazu betrachten wir die Ausgangssituationen, grundsätzliche Verarbeitungsparadigmen, das Datenmodell für die Datenströme und Algorithmen sowie den Aufbau von Queryplänen.

3 Konzepte

Dieses Kapitel führt verschiedene Konzepte für die Umsetzung von Online-Recommender-Systeme auf Basis eines Datenstrommanagementsystems ein. Zunächst betrachten wir die Ausgangssituation für die Konzeption und den typischen Systemkontext, in den das System eingebettet wird (Abschnitt 3.1). Dazu gehören Anforderungen an das eingesetzte Datenstrommanagementsystem sowie ggf. beteiligte Legacy-Systeme und die Kommunikation mit denselbigen. Dieser Abschnitt beschreibt die Basis für die nachfolgenden Konzepte.

Für die datengetriebene Verarbeitung in einem Datenstrommanagementsystem ist es entscheidend, wann welche Berechnung durch welchen Auslöser durchgeführt wird. Aus diesem Grund werden in Abschnitt 3.2 drei grundsätzliche Paradigmen zur Umsetzung eines datenstrombasierten Online-Recommender-Systems definiert und dessen Vor- und Nachteile diskutiert. Diese Paradigmen unterscheiden sich in der Art, wann und durch welchen Auslöser die benötigten Berechnungen durchgeführt werden. Es zeigt sich, dass für verschiedene Recommender-System-Methoden die einzelnen Paradigmen unterschiedliche Vor- und Nachteile haben.

Als Basis für die weiteren Konzepte wird in Abschnitt 3.3 zunächst ein Datenmodell für Online-Recommender-Systeme definiert. Dieses Datenmodell beschreibt die verschiedenen Entitätstypen, die in den Datenströmen des Datenstrommanagementsystems genutzt werden. Um den flexiblen Einsatz der Konzepte zu gewährleisten, enthalten die Entitätstypen neben einigen wenigen erforderlichen Attributen eine Menge optionaler Attribute, die je nach Anwendungsszenerio unterschiedlich genutzt werden können.

In Abschnitt 3.4 werden für die Umsetzung von Recommender-Systemen in einem Datenstrommanagementsystem Basisalgorithmen definiert, die variable Teile für die Implementierung verschiedener Recommender-Systeme enthalten. Diese Basisalgorithmen limitieren somit auch die Menge der Recommender-System-Methoden, die sich mit dem Konzept dieser Arbeit umsetzen lassen. Lediglich Methoden, die sich durch Implementierung der variablen Teile umsetzen lassen, können im Rahmen dieses Konzepts mit einem Datenstrommanagementsystem umgesetzt werden.

In Abschnitt 3.5 betrachten wir die Architektur der Datenstromverarbeitung für Recommender-Systeme. Dazu unterteilen wir diese in sieben Phasen – vom Datenzugriff bis hin zur Datenausgabe. Zu jeder Phase betrachten wir die ein- und

ausgehenden Datenströme und deren Schemata sowie die Aufgaben der Phasen. Diese Phasen werden im späteren Verlauf der Arbeit immer wieder aufgegriffen.

Abschnitt 3.6 stellt konkrete Querypläne für die Berechnung von Empfehlungen vor. Diese Querypläne zeigen, wie sich mit der Datenstromtechnologie und Operatoren die verschiedenen Empfehlungsberechnungen durchführen lassen. Diese Erkenntnisse fließen im darauffolgenden Kapitel in die Definition von Templates für Datenstromqueries ein.

Abschnitt 3.7 gibt zum Abschluss dieses Kapitels eine Zusammenfassung.

3.1 Ausgangssituation und Voraussetzungen

In diesem Abschnitt betrachten wir zunächst die Anforderungen, die wir an das zugrundeliegende Datenstrommanagementsystem stellen und definieren den Systemkontext des datenstrombasierten Recommender-Systems.

3.1.1 Charakterisierung des Datenstrommanagementsystems

Ziel dieser Arbeit ist die Nutzung eines Datenstrommanagementsystems als Grundlage für ein flexibles Recommender-System. Dazu betrachten wir zunächst die Ausgangslage: Wir bauen unser Recommender-System auf Basis eines erweiterbaren Datenstrommanagementsystems auf, welches als Datengrundlage das relationale Modell nutzt[1]. Ein Datenstrommanagementsystem bietet bereits Funktionen für die Datenstromverarbeitung, deren Konzeption und Implementierung nicht Teil dieser Arbeit ist. Folgende Funktionen werden als gegeben angenommen:

- Eine Anfragesprache, in der ein Anfrageausführungsplan modelliert werden kann.
- Eine Anfrageausführungsumgebung, in der der modellierte Anfrageausführungsplan ausgeführt werden kann.
- Die Unterstützung des in Abschnitt 2.2.3 vorgestellten Intervallansatzes, der mithilfe eines Operators die Datenstromelemente mit Gültigkeitsintervallen annotiert:

[1] In dieser Arbeit nutzen wir dafür Odysseus (`https://odysseus.uni-oldenburg.de`). Das Konzept ist allerdings allgemeingültig, solange die in diesem Abschnitt beschriebene Ausgangssituation vorliegt.

  - Der WINDOW-Operator $\omega_w(\mathcal{S})$ annotiert Datenstromelemente aus dem Datenstrom $\mathcal{S}$ mit Gültigkeitsintervallen. Um beispielsweise ein gleitendes Zeitfenster der Breite w umzusetzen, wird ein Datenstromelement zum Zeitpunkt t mit einem Intervall $[t_s, t_e)$ mit $t_s = t$ und $t_e = t + w$ annotiert.

- Operatoren der relationalen Algebra, die für die datenstrombasierte Verarbeitung mittels Gültigkeitsintervallen (vgl. Abschnitt 2.2.3) ausgelegt sind. Dazu gehören:
  - Eine SELECTION $\sigma_\varphi(\mathcal{S})$ entfernt alle Tupel $x \in \mathcal{S}$, für die das Prädikat φ nicht gilt.
  - Eine PROJECTION $\pi_{a_1,\ldots,a_n}(\mathcal{S})$ schränkt die Menge der Attribute aller $x \in \mathcal{S}$ auf die Attribute $\{a_1, \ldots, a_n\}$ ein. Eine erweiterte Version, MAP genannt, erlaubt die Nutzung von Funktionen (z. B. $\pi_{f(a_1),a_2,f(a_3,a_4),\ldots,a_n}(\mathcal{S})$), die auf eine oder mehrere Attribute angewendet werden.
  - Eine AGGREGATION $\gamma_{G,F}(\mathcal{S})$ bekommt eine Menge an Tupeln $x \in \mathcal{S}$ und gibt für jede Gruppe[2], definiert durch die Gruppierungsattribute $g \in G$, ein Tupel aus, das die Ergebnisse der Aggregationsfunktionen $f \in F$ enthält. Typische Aggregationsfunktionen sind SUM, COUNT, AVG, MAX und MIN. Die datenstrombasierte Variante aggregiert kontinuierlich und inkrementell Tupel, die im selben Fenster liegen (was bedeutet, dass sie überlappende Gültigkeitsintervalle haben).
  - Ein JOIN $\mathcal{S}_1 \bowtie_\theta \mathcal{S}_2$ kombiniert Tupel von $\mathcal{S}_1$ und $\mathcal{S}_2$, für die das Prädikat θ gilt. Die datenstrombasierte Variante verlangt außerdem, dass Tupel überlappende Gültigkeitsintervalle haben. Das Gültigkeitsintervall des Ausgabetupels ist die Schnittmenge der beiden Intervalle der Eingabetupel.

- Während die relationale Algebra auf Mengen im mathematischen Sinne ohne Duplikate definiert ist, lassen übliche Datenstrommanagementsystem-Implementierungen mehrere Datenstromelemente mit den gleichen Attributwerten zu. Um diese Duplikate herauszufiltern, gibt es einen DISTINCT-Operator $\eta(\mathcal{S})$, der dafür sorgt, dass es keine zwei gleichen Datenstromelemente gibt, die zum selben Zeitpunkt gültig sind.

- Operatoren, die Daten in verschiedenen Formaten (z. B. CSV oder JSON) und über verschiedene Schnittstellen (z. B. HTTP, Socket, Dateisystem, DBMS/SQL) entgegennehmen und ausgeben. Da die Datenanbindung originäre Aufgabe eines Datenstrommanagementsystem ist, ist die Konzeption und Implementierung dieser Funktion explizit nicht Teil dieser Arbeit.

[2] Eine Gruppe meint die Segmentierung der Tupel anhand von Gruppierungsattributen. So lassen sich zum Beispiel die Nachrichtenaufrufe nach dem Ort gruppieren, an dem sich die Leserin bzw. der Leser aufhält.

3.1.2 Systemkontext

Recommender-Systeme haben die Aufgabe Empfehlungen zu berechnen, damit diese den Benutzerinnen und Benutzern angezeigt werden können. Ein wichtiger Aspekt des Systemkontexts eines Recommender-Systems ist somit die Kommunikation mit der Anwendung, die die Schnittstelle zu den Benutzerinnen und Benutzern darstellt (im Folgenden nur noch *Anwendung* genannt). Beispiele für Anwendungen sind Webportale von Nachrichtenwebseiten, Apps zum Betrachten von Videos oder zum Anhören von Musik sowie Webportale für die Empfehlung von Restaurants.

Die Anwendung informiert das Recommender-System über die Interaktionen der Benutzerinnen und Benutzer (*das Feedback*), damit es diese Informationen in die Empfehlungsberechnung einbeziehen kann. Des Weiteren fordert die Anwendung die Empfehlungen beim Recommender-System an (*die Requests*). Die berechneten Empfehlungen werden schlussendlich vom Recommender-System zurück an die Anwendung übertragen (*die Empfehlungen*).

Neben der Kommunikation mit der Anwendung greifen Recommender-Systeme häufig auf weitere Daten zurück. Dazu gehören zum Beispiel Objektinformationen aus Datenbanken, Kontextdaten aus externen Quellen (z. B. Wetterinformationen) oder gespeicherte Benutzerpräferenzen. Zusätzlich zu diesen operativen Schnittstellen eines Recommender-Systems ist es für Betreiber/-innen und Entwickler/-innen von Recommender-Systemen von Interesse, Daten über die Performance des Recommender-Systems zu erhalten. Dazu betrachten wir neben der Kommunikation mit der Anwendung und den internen und externen Datenquellen auch die Ausgabe von Performanceinformationen über das Recommender-System.

Der wesentliche Unterschied in der Verarbeitung von Daten durch ein Datenbankmanagementsystem und ein Datenstrommanagementsystem wird bereits in der Betrachtung des Systemkontextes deutlich: Während die Verarbeitung in einem Datenbankmanagementsystem durch die Ausführung einer Query ausgelöst wird, agiert ein Datenstrommanagementsystem datengetrieben (vgl. auch die Gegenüberstellung der Anfrageverarbeitung in der Abbildung 2.4 in Kapitel 2 *Grundlagen*). Das bedeutet, dass der Datenaustausch und die Kommunikation mit einem datenstrombasierten Recommender-System über Datenströme stattfindet.

Abbildung 3.1 zeigt einen typischen Systemkontext eines datenstrombasierten Recommender-Systems. In der Mitte befindet sich das Recommender-System. Links werden die Eingabedatenströme und rechts die Ausgabedatenströme dargestellt. Das Recommender-System bekommt einen Datenstrom mit Feedback von den Benutzerinnen und Benutzern zu den Objekten als Datengrundlage für die Empfehlungsberechnung und zum anderen einen Datenstrom mit Anfragen nach Empfehlungen

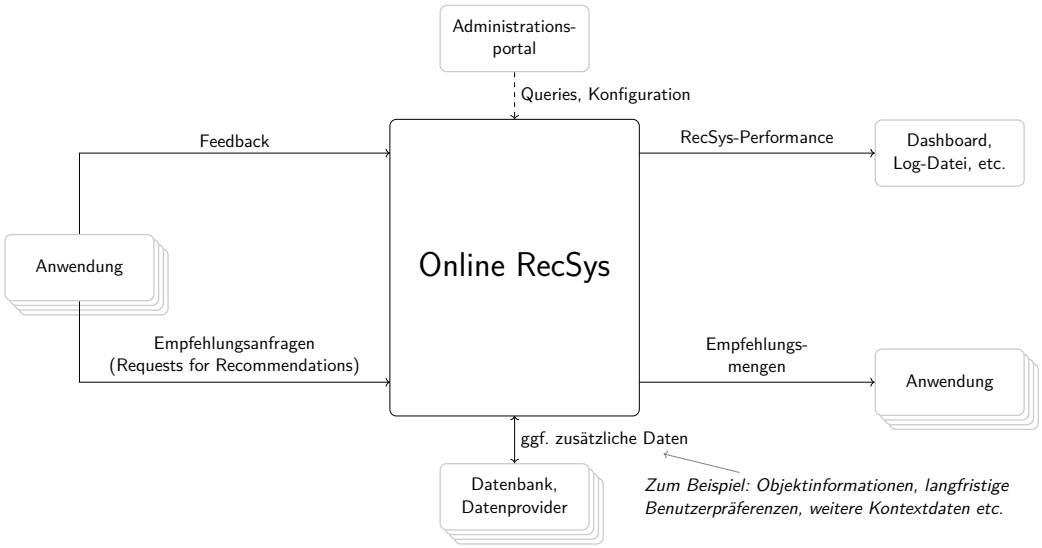


Abbildung 3.1 Systemkontext eines datenstrombasierten Recommender-Systems

(Requests for Recommendations). Ein Request stellt dabei ein Ereignis dar, welches dem Recommender-System signalisiert, dass es eine auf diesen Request zugeschnittene Menge an Empfehlungen ausgeben soll. Die berechneten Empfehlungslisten werden wiederum als Datenstrom an die Anwendung übertragen (unten rechts in Abbildung 3.1). Die Anwendung auf der linken und auf der rechten Seite ist in der Regel (aber nicht zwangsläufig) die selbe.

Der Zugriff auf weitere Daten, die nicht von der Anwendung zur Verfügung gestellt werden, ist in Abbildung 3.1 unter dem Online-Recommender-System dargestellt. Diese Schnittstelle kann vom Recommender-System dazu genutzt werden, um die eingehenden Datenströme mit Zusatzinformationen anzureichern. Dazu zählen beispielsweise Informationen über die Objekte aus einer Objektdatenbank, Kontextinformationen wie z. B. das aktuelle Wetter am Aufenthaltsort des Benutzers etc. Neben dem lesenden Zugriff auf Daten aus einer Datenbank, einem Webservice oder Datenstrom, kann hier auch das Speichern von Daten realisiert werden, die nicht im Hauptspeicher gehalten werden sollen oder können.

Oben rechts in Abbildung 3.1 ist die Ausgabe von Performance-Daten des Recommender-Systems dargestellt. Dazu gehört zum Beispiel die Ausgabe von Evaluationsmetriken (z. B. die Qualität der Empfehlungen) aber auch der Latenz der Datenverarbeitung. Diese Daten können beispielsweise in einem Dashboard visualisiert oder in eine Log-Datei geschrieben werden.

Neben der eigentlichen Datenein- und -ausgabe bietet ein Recommender-System häufig eine Schnittstelle zur Administration an. Für den Einsatz eines Datenstrommanagementsystems ist insbesondere das Hinzufügen, Entfernen und Ändern von

Queries von Bedeutung. Diese Schnittstelle ist in Abbildung 3.1 oberhalb des Online-Recommender-Systems dargestellt.

3.2 Verarbeitungsparadigmen

Um herkömmliche Ansätze zur Berechnung von Empfehlungen auf eine datenstrombasierte Berechnung zu übertragen, betrachten wir in diesem Abschnitt drei grundsätzliche Ansätze, wie man die Berechnung datenstrombasiert umsetzen kann. Dabei klären wir insbesondere, wann bzw. durch welchen Auslöser die Empfehlungsmengen für einen Request berechnet werden. Da bei der Datenstromverarbeitung die effiziente Verarbeitung großer, kontinuierlich erzeugter Datenmengen angestrebt wird, spielt dieser Aspekt bei dem Entwurf eines Online-Recommender-Systems eine besonders große Rolle.

3.2.1 Requestgetriebene Empfehlungsberechnung

In Recommender-Systemen wird durch einen Request die Ausgabe einer Empfehlungsliste ausgelöst. In einem datenbankbasierten Recommender-System wird zu dem eintreffenden Request die Kandidatenmenge bestimmt und für jeden Kandidaten die Lerndatenmenge gebildet bzw. das zuvor trainierte Modell zugeordnet, um anschließend die Kandidaten zu bewerten und zu ordnen. Diesen Ansatz kann man auf die datenstrombasierten Recommender-Systeme übertragen. Abbildung 3.2 zeigt diesen Ansatz: Zu jedem Request werden mittels Join-Operator die zum Zeitpunkt des Requests gültigen Feedbacks zugeordnet und anschließend die Empfehlungen berechnet. Dieser Ansatz wird in dieser Arbeit *requestgetriebene Empfehlungsberechnung* genannt.

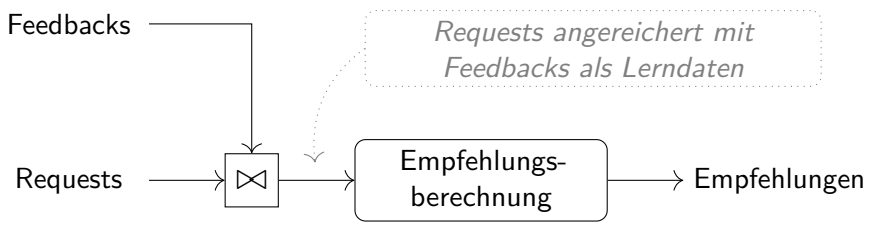


Abbildung 3.2 Requestgetriebene Empfehlungsberechnung

Der Nachteil der requestgetriebenen Empfehlungsberechnung für ein Online-Recommender-System ist, dass für jeden Request die Empfehlungsberechnung von Grund auf neu durchgeführt wird und somit die Beantwortung eines Requests viel Zeit in

Anspruch nimmt. Außerdem müssen alle gültigen Feedbacks im Speicher gehalten werden, damit diese für die Empfehlungsberechnung zur Verfügung stehen. Dies entspricht nicht dem, was man bei der Datenstromverarbeitung anstrebt: Einhaltung des One-Pass-Paradigmas, inkrementelle Verarbeitung, konstante Speichernutzung etc.

3.2.2 Feedbackgetriebenen Empfehlungsberechnung

Für die Datenstromverarbeitung wäre ein Ansatz ideal, welcher die Empfehlungen kontinuierlich und inkrementell berechnet. Dazu führen wir den Ansatz der *feedbackgetriebenen Empfehlungsberechnung* ein (Abbildung 3.3). Ziel dieses Ansatzes ist es, dass bei jeder Änderung der Lerndatenmenge (der Feedbackdaten), die Empfehlungsmengen inkrementell aktualisiert werden, sodass bei Eintreffen eines Requests auf eine bereits berechnete Empfehlungsmenge zurückgegriffen werden kann. Dies ist nur dann möglich, wenn die Menge der möglichen Request-Instanzen zum Zeitpunkt der Empfehlungsberechnung bekannt, endlich und klein genug ist, sodass sie die Speicherkapazitäten nicht überschreiten. Sollen beispielsweise die Empfehlungen daraufhin angepasst werden, in welchem Land sich die Benutzerin bzw. der Benutzer derzeit aufhält, so kann für jedes Land, in dem das Recommender-System eingesetzt wird, eine Empfehlungsmenge kontinuierlich und inkrementell berechnet werden. Sobald ein Request aus einem bestimmten Land eintrifft, wird dieser mit der Empfehlungsmenge mittels Join-Operator verknüpft und ausgegeben.

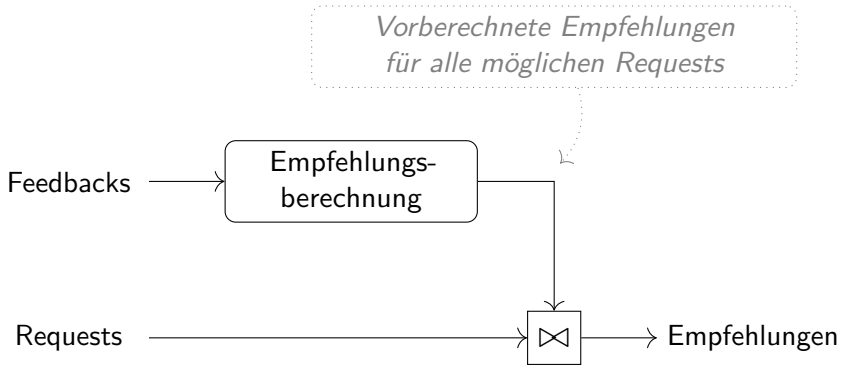


Abbildung 3.3 Feedbackgetriebene Empfehlungsberechnung

Soll allerdings eine Empfehlung berechnet werden, die individuell auf die Geokoordinaten der Benutzerin oder des Benutzers angepasst ist (z. B. für die Berücksichtigung von Geoobjekten in der Nähe als Empfehlungen), kommt es je nach Betrachtungsweise zu einer unbekannten Menge (Zu welchen Koordinaten werden Requests gestellt?) oder zu einer zu großen Menge an möglichen Request-Instanzen (wenn zu jeder möglichen Geokoordinate eine Empfehlungsmenge berechnet werden soll). Das führt dazu, dass eine kontinuierliche Berechnung von Empfehlungen für jede mögliche

Geokoordinate nicht möglich ist. Allgemein kann man sagen, je individueller die Empfehlungen sein sollen, desto weniger ist dieser Ansatz geeignet.

Um die Ansätze der requestgetriebenen Empfehlungsberechnung auf die feedbackgetriebene Empfehlungsberechnung zu übertragen, führen wir das Konzept der *impliziten Requests* ein. Dieses sieht vor, dass aus einem Feedback eine mögliche Requestinstanz abgeleitet wird, zu der die Empfehlungsmenge aktualisiert werden soll. So wird zum Beispiel aus einen Feedback der Form $\{i, r, u, l\}_t$ mit dem bewerteten Objekt i und der Bewertung r von einer Benutzerin oder einem Benutzer u am Ort l zum Zeitpunkt t der implizite Request $\{u, l\}_t$ abgeleitet. Zu diesem impliziten Request kann anschließend die Empfehlungsmenge mit dem neu hinzugekommenen Feedback aktualisiert werden. Trifft ein tatsächlicher Request für den Benutzer u am Ort l ein, so kann dieser direkt mit der vorberechneten Empfehlungsmenge beantwortet werden. Das genaue Vorgehen zum Ableiten eines impliziten Requests wird in Abschnitt 3.6.2 erläutert.

3.2.3 Hybride Empfehlungsberechnung

Einen Kompromiss zwischen den beiden Ansätzen stellt der Ansatz der *hybriden Empfehlungsberechnung* dar (Abbildung 3.4). Dieser sieht vor, dass bei Veränderung der Lerndatenmenge Teile der Empfehlungsberechnung inkrementell erfolgen. Dieser Ansatz ist beispielsweise für modellbasierte Ansätze geeignet: Ändert sich die Lerndatenmenge, so wird das Modell neu berechnet oder im Idealfall inkrementell aktualisiert. Zu einem neuen Request wird das jeweils gültige Modell zugeordnet und die Empfehlungsmenge wird anschließend durch Anwendung des Modells gebildet.

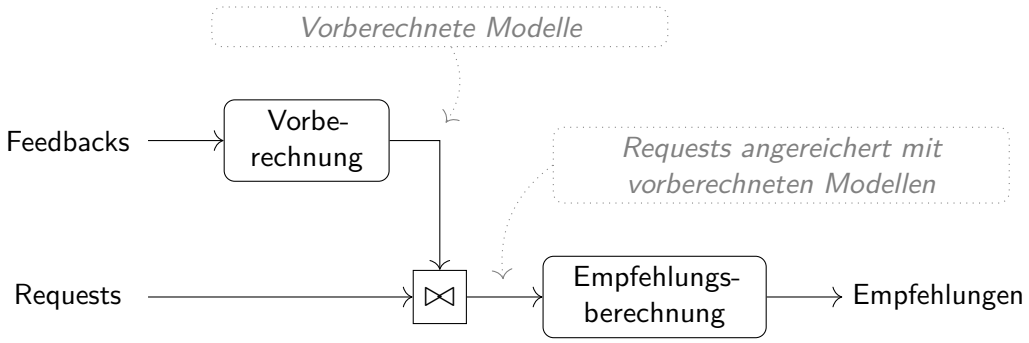


Abbildung 3.4 Hybride Empfehlungsberechnung

3.3 Datenmodell

In diesem Abschnitt werden die Datenmodelle der wichtigsten Entitäten mit ihren Attributen vorgestellt. Attribute können skalare Werte aber auch komplexe Werte wie Listen und Tupel annehmen.

Das Datenmodell gibt die Schemata der Datenströme innerhalb des Datenstrommanagementsystems wieder. Das bedeutet, dass jede Entität dem Schema eines Datenstromelements in einem Datenstrom entspricht. Die Entitäten enthalten somit alle Informationen, die für den jeweiligen Verarbeitungsschritt nötig sind. Das bedeutet insbesondere, dass die Datenmodelle nicht normalisiert im Sinne der Normalisierung relationaler Datenbankschemata, sondern bereits denormalisiert sind.

3.3.1 Empfehlungsobjekte

Eine bedeutende Entität eines jeden Recommender-Systems sind die zu empfehlenden Objekte (Abbildung 3.5). Ein Empfehlungsobjekt bezeichnen wir in dieser Arbeit mit einem i, eine Menge an Empfehlungsobjekten mit I. Die Attribute eines Empfehlungsobjekts sind in den verschiedenen Recommender-Systemen sehr unterschiedlich, je nachdem, um was für Objekte es sich handelt, die empfohlen werden sollen (Nachrichtenartikel, Produkte, Filme, ...). In unserem Modell haben alle Empfehlungsobjekte gemein, dass sie eine ID k besitzen (*primary key*; in der Regel ein numerischer Wert oder eine UUID), mit der das Objekt eindeutig identifiziert werden kann, sowie – wie jede Entität im Datenstrommanagementsystem – einen Zeitstempel t. Der Zeitstempel bezeichnet in der Regel des Zeitpunkt, zu dem das Empfehlungsobjekt erstellt bzw. veröffentlicht wurde, z. B. der Veröffentlichungszeitpunkt eines Nachrichtenartikels oder den Zeitpunkt, an dem ein Produkt in einem Online-Shop hinzugefügt wurde.

Empfehlungsobjekt $i \in I$	
▪ Zeitstempel	t
▪ Objekt-ID	k
o Objektattribute	$\vec{y} = \{y_1, y_2, \ldots, y_m\}$

Abbildung 3.5 Datenmodell für Empfehlungsobjekte (o = optional)

Neben dem Zeitstempel und der Objekt-ID kann ein Empfehlungsobjekt eine Menge an beliebigen Attributen $\vec{y} = \{y_1, y_2, \ldots, y_m\}$ besitzen. Das Attributschema ist innerhalb eines Recommender-Systems fest definiert. Die zusätzlichen Objektattribute

$\vec{y}$ können von bestimmten Recommender-System-Methoden genutzt werden, um Ähnlichkeiten zwischen Empfehlungsobjekten zu bestimmen.

Die Objektattribute $\vec{y}$ sind in Abbildung 3.5 als Vektor bzw. Liste modelliert. Für die konkrete Instanziierung des Datenmodells bietet es sich wie im nachfolgenden Beispiel an, die Attribute direkt als Objektattribute im Schema zu definieren. In diesem Fall dient $\vec{y}$ als abgekürzte Schreibweise für die Menge der Objektattribute $y_1, y_2, \ldots, y_m$. Diese verkürzte Schreibweise nutzen wir im weiteren Verlauf auch für andere semantisch verwandte Attributmengen, wie zum Beispiel die Kontextinformationen $\vec{x}_F$.

Abbildung 3.6 zeigt ein Beispiel für ein konkretes Datenmodell für Empfehlungsobjekte zur Empfehlung von Nachrichtenartikeln. Es enthält für jeden Nachrichtenartikel zusätzliche Informationen wie zum Beispiel den Titel und Autor des Artikels, das Veröffentlichungsdatum, den eigentlichen Text des Artikels etc. Das Datenmodell in dieser Arbeit macht keine Vorgaben über die Art und Anzahl der zusätzlichen Attribute. Das Schema kann für jedes Recommender-System beliebig festgelegt werden.

Nachrichtenartikel	
▪ Zeitstempel	t
▪ Objekt-ID	k
▪ Titel	y_1
▪ Autor	y_2
▪ Veröffentlichungsdatum	y_3
▪ Kategorie	y_4
▪ Schlagwörter	y_5
▪ Text	y_6

Abbildung 3.6 Beispiel-Datenmodell für Empfehlungsobjekte: Nachrichtenartikel

3.3.2 Feedbackdaten

Feedbackdaten (Abbildung 3.7) geben uns Informationen über die Interaktion von Benutzerinnen und Benutzern mit den Empfehlungsobjekten. Mögliche Interaktionen sind beispielsweise das Aufrufen eines Nachrichtenartikels („Click“), das Abspielen eines Films, der Kauf eines Produkts oder die quantitative Bewertung eines Musikalbums. Die einfachste – wenn in der Praxis auch eher selten vorkommende – Form eines Feedbacks f hat neben dem obligatorischen Zeitstempel t ein Attribut zur

Identifikation des Empfehlungsobjekts i (*foreign key*), mit dem die Benutzerin bzw. der Benutzer interagiert hat. So ein Feedback kann zum Beispiel bedeuten: „Irgendeine Benutzerin oder irgendein Benutzer hat das Objekt i aufgerufen / abgespielt / gekauft etc."

Handelt es sich bei der Interaktion um eine quantitative Bewertung des Objekts (explizites Feedback), so kann dem Feedback die Bewertung mithilfe des optionalen Attributs r mitgegeben werden. Hat keine qualitative Bewertung stattgefunden, können für Algorithmen zur Berechnung von Empfehlungsmengen, die einen Wert für eine Bewertung benötigen, der Wert für r für alle Interaktionen auf 1 gesetzt werden (implizites Feedback). Die Bewertung kann auch von der Anwendung implizit ermittelt werden. Zum Beispiel könnte die Bewertung eines Videos dadurch bestimmt werden, wie viel die Benutzerin bzw. der Benutzer von dem Video angeschaut hat, in der Hoffnung darauf zu schließen, ob der Film gefallen hat (ganz angeschaut) oder nicht (nach der Hälfte abgebrochen).

Feedback $f \in F$	
▪ Zeitstempel	t
o Feedback-ID	k
▪ Objekt-ID	i
o Objektattribute	$\vec{y} = \{y_1, y_2, \ldots, y_m\}$
o Bewertung	r
o Kontextdaten	$\vec{x}_F = \{x_1, x_2, \ldots, x_n\}$

Abbildung 3.7 Datenmodell für Feedbacks (o = optional)

Für einige Algorithmen sind zu den Feedbacks Informationen zu dem jeweiligen Empfehlungsobjekt von Interesse. Diese können dem Feedback als Objektattribute $\vec{y} = \{y_1, y_2, \ldots, y_m\}$ mitgegeben werden. Wenn diese nicht zusammen mit dem Feedback von der Anwendung mitgesendet werden, kann das Datenstrommanagementsystem im Rahmen der Datenvorverarbeitung (Abschnitt 3.5.2) die Feedbackdaten mit Objektinformationen aus externen Quellen (z. B. Objektdatenbank) anreichern.

An dieser Stelle sieht man die eingangs erwähnte denormalisierte Form des Datenmodells. In einem normalisierten Datenbankschema wäre die Aufnahme der Objektattribute in das Feedback-Datenschema aus Abbildung 3.7 eine ungewollte und somit zu vermeidende Redundanz. Bei den hier vorgestellten Datenmodellen handelt es sich allerdings um Schemata der Datenströme zwischen den einzelnen Verarbeitungsoperatoren, bei denen die Denormalisierung bereits stattgefunden hat. Operatoren, die die Feedbackdaten verarbeiten sollen, sollen sich explizit nicht

mehr um die Denormalisierung der Daten kümmern müssen. Jegliche Denormalisierung erfolgt im Vorfeld durch eigens dafür eingesetzte Operatoren (z. B. durch den JOIN-Operator). Dies stellt die Verteilung der Zuständigkeiten auf die einzelnen Operatoren sicher (Single Responsibility Principle).

Neben den Objektattributen kann die Anwendung zusätzliche Informationen zu dem Kontext $\vec{x}_F = \{x_1, x_2, \ldots, x_n\}$ des Feedbacks mitgeben. Typische Kontextattribute sind bspw. der Aufenthaltsort der Benutzerin bzw. des Benutzers oder das Wetter zum Zeitpunkt der Objektnutzung. Kontextsensitive Recommender-Systeme können diese Informationen nutzen, um die Empfehlungen auf den Kontext anzupassen. Neben den klassischen Kontextdaten betrachten wir beispielsweise die Benutzer-ID und Informationen über die Benutzerin oder den Benutzer ebenso als Kontext, in dem das Feedback gegeben wurde. Kontextsensitive Recommender-Systeme unterscheiden traditionell in der Regel zwischen Benutzerprofildaten und Informationen über den Benutzerkontext. In diesem Konzept findet diese Unterscheidung nicht statt, da es hier aus technischer Sicht keinen Unterschied macht, ob die Empfehlungen auf die Interessen der Benutzerin bzw. des Benutzer oder auf die Interessen von Benutzerinnen und Benutzern an einem Aufenthaltsort (oder beides) zugeschnitten werden. In beiden Fällen handelt es sich um Attribute zur Personalisierung der Empfehlungsmenge.

Um verschiedene Arten von Interaktionen zu unterscheiden, kann dem Feedback ein Feedbacktyp als Kontextinformation mitgegeben werden. So kann beispielsweise unterschieden werden, ob die Benutzerin bzw. der Benutzer ein Produkt im Online-Shop lediglich angesehen oder sogar gekauft hat.

Abbildung 3.8 zeigt ein Beispiel für ein konkretes Datenmodell für Feedbackdaten. In diesem Beispiel werden Feedbacks in Form von Bewertungen zu Nachrichtenartikeln von Benutzerinnen und Benutzern gegeben.

3.3.3 Requests for Recommendations

Eine Anfrage einer Benutzerin oder eines Benutzers an ein Recommender-System, Empfehlungen zu liefern, bezeichnen wir als *Request for Recommendation* oder kurz *Request* (Abbildung 3.9). Neben dem Zeitstempel t enthält ein Request eine Request-ID k, mit der der Request eindeutig identifiziert werden kann. Des Weiteren können einem Request Kontextdaten $\vec{x}_Q = \{x_1, x_2, \ldots, x_n\}$ mitgegeben werden, anhand derer die Empfehlungsmenge individualisiert werden kann.

Nachrichtenartikelbewertung	
▪ Zeitstempel	t
▪ Feedback-ID	k
▪ Objekt-ID	i
▪ Titel	y_1
▪ Autor	y_2
▪ Veröffentlichungsdatum	y_3
▪ Kategorie	y_4
▪ Schlagwörter	y_5
▪ Text	y_6
▪ Bewertung	r
▪ Benutzer-ID	x_1
▪ Alter	x_2
▪ Aufenthaltsort	x_3
▪ Endgerätart	x_4

Abbildung 3.8 Beispiel-Datenmodell für Feedbacks: Bewertung von Nachrichtenartikeln

Request $q \in Q$	
▪ Zeitstempel	t
▪ Request-ID	k
o Kontextdaten	$\vec{x}_Q = \{x_1, x_2, \ldots, x_n\}$

Abbildung 3.9 Datenmodell für Requests for Recommendations (o = optional)

Abbildung 3.10 zeigt ein Beispiel für einen Request für die Berechnung von Nachrichtenempfehlungen. Als Kontextdaten enthält er Informationen zu der Benutzerin bzw. dem Benutzer ($x_1 - x_2$) und ihrer bzw. seiner Situation ($x_3 - x_4$), zu dem Nachrichtenartikel, den sich die Benutzerin bzw. der Benutzer gerade anschaut ($x_5 - x_{11}$), sowie Suchterme, die die Benutzerin bzw. der Benutzer angegeben hat (x_{12}).

Nachrichtenempfehlungenrequest	
▪ Zeitstempel	t
▪ Request-ID	k
▪ Benutzer-ID	x_1
▪ Alter	x_2
▪ Aufenthaltsort	x_3
▪ Endgerätart	x_4
▪ Objekt-ID	x_5
▪ Titel	x_6
▪ Autor	x_7
▪ Veröffentlichungsdatum	x_8
▪ Kategorie	x_9
▪ Schlagwörter	x_{10}
▪ Text	x_{11}
▪ Suchterme	x_{12}

Abbildung 3.10 Beispiel-Datenmodell für Requests for Recommendations: Request für Nachrichtenempfehlungen

3.3.4 Empfehlungskandidaten

Aus der Menge aller Empfehlungsobjekte I wird für einen Request q eine Teilmenge $C \subseteq I$ gebildet, die Objekte enthält, die zur Empfehlung infrage kommen. Im einfachsten Fall sind dies alle Empfehlungsobjekte. Jeder Kandidat $c \in C$ wird anschließend durch ein Recommendation-Verfahren quantitativ bewertet. Diese berechnete Bewertung (engl. *rating*) bezeichnen wir mit $\hat{r}$. Dabei ist der Begriff *Bewertung* nicht allzu wörtlich zu nehmen. Bei diesem berechneten Wert handelt es sich eher um einen Relevance Score, anhand dem die bewerteten Kandidaten sortiert werden, um die K höchst-bewerteten Kandidaten den Benutzerinnen und Benutzern zu präsentieren. Bei rankingbasierten Verfahren kann die Bewertung dazu genutzt werden, den Rang innerhalb der sortierten Liste zu ermitteln. Für das

Konzept dieser Arbeit ist es lediglich erforderlich, dass sich die Kandidaten anhand des Attributes $\hat{r}$ in eine Reihenfolge bringen lassen, um die sortierte Top-K-Menge als Liste der Empfehlung zu bestimmen. Die Reihenfolge von Kandidaten, die die gleiche geschätzte Bewertung haben, ist dabei nicht definiert und per Zufall bestimmt.

Abbildung 3.11 zeigt das Datenmodell für bewertete Empfehlungskandidaten. Es enthält neben dem Zeitstempel t und der ID des Requests q die Objekt-ID i des als Empfehlung infrage kommenden Objekts sowie die geschätzte Bewertung $\hat{r}$. Enthält der Request Kontextdaten, so können diese dem Empfehlungskandidat ebenso angefügt werden ($\vec{x}_Q$), damit diese in den nachfolgenden Verarbeitungsschritten unmittelbar zur Verfügung stehen. Genauso verhält es sich mit den Objektattributen $\vec{y}$.

Bewerteter Empfehlungskandidat $c \in C$	
▪ Zeitstempel	t
▪ Request-ID	q
o Request-Kontext	$\vec{x}_Q = \{x_1, x_2, \ldots, x_n\}$
▪ Objekt-ID	i
o Objektattribute	$\vec{y} = \{y_1, y_2, \ldots, y_m\}$
▪ Bewertung	$\hat{r}$

Abbildung 3.11 Datenmodell für bewertete Empfehlungskandidaten (o = optional)

3.3.5 Lerndaten

Die Schätzung einer Bewertung eines Empfehlungskandidaten erfolgt in der Regel mit Hilfe von Feedbackdaten. Die Feedbackdaten, die bei einer Schätzung berücksichtigt wurden, bezeichnen wir als Lerndaten $L \subseteq F$. Im einfachsten Fall werden alle existierenden Feedbackdaten als Lerndaten genutzt. Um beispielsweise ein kontextsensitives Recommender-System umzusetzen, können die Feedbackdaten bzgl. einer Kontextdimension partitioniert werden. So können beispielsweise für die Empfehlung von Nachrichtenartikeln einer Leserin bzw. eines Lesers aus Deutschland nur Feedbacks aus ebendiesem Land berücksichtigt werden.

Das Datenmodell für Lerndaten ist in den einzelnen Verarbeitungsparadigmen (vgl. Abschnitt 3.2) unterschiedlich. Beim requestgetriebenen sowie beim feedbackgetriebenen Paradigma entsteht das Datenmodell durch einen Verbund eines (möglicherweise impliziten) Requests mit Feedbacks. Zu einem eingehenden Request werden alle Feedbackdaten zugeordnet, die zur Bestimmung der Empfehlungen

genutzt werden sollen. Das Ergebnis sind Lerndaten, die aus den Attributen des Feedback-Datenmodells (Abbildung 3.7) sowie der Request-ID mit dem Request-Kontext bestehen (Abbildung 3.12). Beim hybriden Verarbeitungsparadigma gehen die Feedbackdaten mit unverändertem Datenmodell in den Vorberechnungsschritt ein. Je nach Verfahren werden die Feedbackdaten ggf. gefiltert und/oder partitioniert.

Lerndatum $l \in L$	
▪ Zeitstempel	t
▪ Request-ID	q
o Request-Kontext	$X_R = \{x_1, x_2, \ldots, x_n\}$
o Feedback-ID	f
o Feedbacktyp	a
▪ Objekt-ID	i
o Objektattribute	$\vec{y} = \{y_1, y_2, \ldots, y_m\}$
o Bewertung	r
o Feedback-Kontext	$\vec{x}_F = \{x_1, x_2, \ldots, x_n\}$

Abbildung 3.12 Datenmodell für Lerndaten beim requestgetriebenen Verarbeitungsparadigma (o = optional)

3.3.6 Recommendation-Modelle

Bei modellbasierten Recommender-Systemen werden mithilfe der Lerndaten Modelle berechnet, mit deren Informationen man Bewertungen für Empfehlungskandidaten berechnen kann. Das Datenmodell für Modellinstanzen beinhaltet einen Zeitstempel t, ab dem das Modell gelten soll, optional eine ID k, um die Modellinstanz eindeutig identifizieren zu können (Modell-ID) und die eigentlichen Modelldaten $\vec{z}$.

Modell $m \in M$	
▪ Zeitstempel	t
o Modell-ID	k
▪ Modelldaten	$\vec{z} = \{z_1, z_2, \ldots, z_n\}$

Abbildung 3.13 Datenmodell für Recommender-System-Modelle (o = optional)

Die Form der Modelldaten $\vec{z}$ ist abhängig von dem eingesetzten Recommender-System-Verfahren. Abbildung 3.14 zeigt ein Beispiel für ein Modell für die Matrix-Faktorisierung. Das Modell besteht aus User-Feature-Vektoren (z_1) sowie Item-Fea-

ture-Vektoren (z_2). Diese können genutzt werden, um die Bewertung für ein Objekt zu schätzen.

Matrixfaktorisierungsmodell	
▪ Zeitstempel	t
▪ User-Feature-Vektoren	z_1
▪ Item-Feature-Vektoren	z_2

Abbildung 3.14 Beispiel-Datenmodell für Matrixfaktorisierungsmodelle

3.3.7 Empfehlungen

Empfehlung $e \in E$	
▪ Zeitstempel	t
▪ Request-ID	q
o Request-Kontext	$\vec{x}_Q = \{x_1, x_2, \ldots, x_n\}$
▪ Empfehlung 1	i_1
o Objektattribute 1	$\vec{y}_1 = \{y_1, y_2, \ldots, y_m\}$
o Bewertung 1	$\hat{r}_1$
▪ Empfehlung 2	i_2
o Objektattribute 2	$\vec{y}_2 = \{y_1, y_2, \ldots, y_m\}$
o Bewertung 2	$\hat{r}_2$
⋮	⋮
▪ Empfehlung K	i_K
o Objektattribute K	$\vec{y}_K = \{y_1, y_2, \ldots, y_m\}$
o Bewertung K	$\hat{r}_K$

Abbildung 3.15 Datenmodell für Empfehlungen (o = optional)

Ein Request wird mit einer sortierten Liste von K Empfehlungen beantwortet. Das Datenmodell einer Empfehlung enthält mindestens den Zeitstempel und die Request-ID zu dem Request, der mit dieser Empfehlung beantwortet werden soll, sowie die K Empfehlungen in Form von Objekt-IDs. Zusätzlich können die Objektattribute und Bewertungen zu den Empfehlungen beigefügt werden, um diese in den nachfolgenden Verarbeitungsschritten unmittelbar nutzen und beispielsweise den Benutzerinnen und Benutzern anzeigen zu können. In den meisten Fällen ist die Ausgabe der Objekt-IDs ausreichend, sodass die Anwendung bei Bedarf die erfor-

derlichen Informationen mithilfe der ID aus der Objektdatenbank abfragen kann. Abbildung 3.15 zeigt das Datenmodell.

3.4 Basisalgorithmen

Mit dem Konzept dieser Arbeit sollen sich verschiedene (im Idealfall beliebige) Recommender-Systeme umsetzen können. Bei der Konzeption der Lösung in dieser Arbeit orientieren wir uns an der folgenden Auswahl an Recommender-System-Verfahren. Mit dieser Auswahl ist ein breites Spektrum an verschiedenen Methoden abgedeckt, sodass diese eine gute Basis bildet, um verschiedenste Recommender-Systeme umsetzen zu können.

3.4.1 Speicherbasierte Verfahren

Speicherbasierte Verfahren (vgl. auch Abschnitt 2.1.3) basieren auf der Idee, dass zur Bewertung von potenziellen Empfehlungen (die Kandidaten) über einige oder alle Feedbacks (die Lerndaten) iteriert wird. Gegeben eines Requests q, zeigt Algorithmus 1 das allgemeine Vorgehen zur Berechnung der Empfehlungsmenge bei speicherbasierten Verfahren, wie es im Rahmen des Konzepts in dieser Arbeit umgesetzt worden ist.

Algorithmus 1 : Speicherbasiertes Vorgehen

Trigger (in) : Ein Request $q \in Q$ aus dem Datenstrom $\mathcal{Q}$
Output : Datenstrom $\mathcal{E}$ mit Datenstromelementen, die jeweils eine Empfehlung e als Antwort auf einen Request q transportieren

$C \leftarrow$ Auswahl von Objekten, die für eine Empfehlung infrage kommen (Kandidaten), mit der Funktion $f_C(q, I)$
for $c \in C$ **do**
  $L \leftarrow$ Auswahl von Feedbacks, die für die Bewertung des Objekts c benötigt werden (Lerndaten), mit der Funktion $f_L(q, c, F)$
  $\hat{r} \leftarrow$ Berechnung der Bewertung des Objekts c anhand der Lerndaten und der Bewertungsfunktion $\hat{f}_R(q, c, L)$
$e \leftarrow$ Berechnung der Empfehlungsmenge (Top-K-Menge der bewerteten Kandidaten) unter Verwendung der berechneten Bewertungen $\hat{r}$ und Ausgabe als Datenstromelement

Die verschiedenen Verfahren unterscheiden sich in der Implementierung der drei Funktionen f_C, f_L und $\hat{f}_R$.

Die Kandidatenselektionsfunktion f_C bildet zu einem Request q aus der Menge der Empfehlungsobjekte I die Kandidatenmenge:

$$f_C : (q, I) \mapsto C \tag{3.1}$$

Die Lerndatenselektionsfunktion f_L bestimmt für jeden Kandidaten $c \in C$ passend zu dem Request q unter Nutzung der Feedbackdaten F die Lerndatenmenge L:

$$f_L : (q, c, F) \mapsto L \tag{3.2}$$

Die Bewertungsschätzfunktion $\hat{f}_R$ ist eine Annäherung an die wahre, aber unbekannte, Bewertungsfunktion f_R. Sie erlaubt mithilfe der Lerndatenmenge L die Berechnung einer auf den Request q zugeschnittenen Schätzung $\hat{r}$ für die wahre, aber unbekannte Bewertung r für einen Kandidaten c:

$$\hat{f}_R : (q, c, L) \mapsto \hat{r} \tag{3.3}$$

Daraus resultiert die Menge der *bewerteten* Kandidaten c' für den Request q. Die Elemente in c' werden absteigend nach $\hat{r}$ sortiert und die K Elemente an der Spitze bilden die Empfehlung e (Top-K-Menge).

Empfehlung der am häufigsten genutzten Objekte. Als erste speicherbasierte Methode betrachten wir die Empfehlung der am häufigsten genutzten Objekte. Als Objektnutzung betrachten wir beispielsweise das Aufrufen eines Nachrichtenartikels im Webbrowser durch eine Benutzerin oder einen Benutzer. Jede Objektnutzung führt zu einem Feedbackdatum $f \in F$.

Um jeder Benutzerin bzw. jedem Benutzer die am häufigsten genutzten Objekte zu empfehlen, werden alle Objekte in I als Kandidaten genutzt. Die Kandidatenselektionsfunktion (Gleichung 3.1) sieht in diesem Fall wie folgt aus:

$$f_C : (q, I) \mapsto I \tag{3.4}$$

Die Lerndatenmenge für einen Kandidaten c bildet die Menge an Feedbacks $f \in F$, die zu den Kandidaten abgegeben wurden. Die Lerndatenselektionsfunktion (Gleichung 3.2) selektiert alle Feedbacks, deren Objekt-ID $f.i$ der Objekt-ID des Kandidaten $c.k$ entspricht, als Lerndatenmenge L_c für die Schätzung der Bewertung

für den Kandidaten c:

$$f_L : (q, c, F) \mapsto \{f \in F \mid f.i = c.k\} \tag{3.5}$$

Diese Funktion kann auch dazu genutzt werden, um das Contextual Pre-Filtering umzusetzen (vgl. Abschnitt 2.1.4). Dazu kann eine auf den Request zugeschnittene Lerndatenmenge gebildet werden. Haben beispielsweise sowohl die Requests als auch die Feedbackdaten ein Kontextattribut x_1, welches den derzeitigen Aufenthaltsort der Benutzerin bzw. des Benutzers beinhaltet, so kann je Aufenthaltsort eine eigene Lerndatenmenge wie folgt bestimmt werden:

$$f_L : (q, c, F) \mapsto \{f \in F \mid f.i = c.k \wedge f.x_1 = q.x_1\} \tag{3.6}$$

Nachdem die Lerndaten wie beschrieben ausgewählt wurden, lässt sich die Häufigkeit der Nutzung eines Kandidaten, und somit dessen geschätzte Bewertung $\hat{r}$, durch die Mächtigkeit der Lerndatenmenge L_c wie folgt bestimmen:

$$\hat{f}_R : (q, c, L_c) \mapsto |L_c| \tag{3.7}$$

Empfehlung der am besten bewerteten Objekte. Bei dieser Methode gehen wir davon aus, dass Benutzerinnen und Benutzer Objekte explizit oder implizit quantitativ bewerten. Die Feedbackdaten enthalten somit zu einem Objekt i eine Bewertung $r \in R$. In diesem Fall findet wie im vorherigen Beispiel keine Einschränkung der Kandidatenmenge statt. Gleichung 3.4 gilt hier ebenso. Genauso entspricht die Lerndatenselektionsfunktion der Funktion aus dem vorherigen Abschnitt (Gleichung 3.5). Die geschätzte Bewertung $\hat{r}$ für einen Kandidaten c und die dazugehörige Lerndatenmenge L_c berechnet sich anschließend durch das arithmetische Mittel wie folgt:

$$\hat{f}_R : (q, c, L_c) \mapsto \frac{1}{|L_c|} \sum_{l \in L_c} l.r \tag{3.8}$$

3.4.2 Modellbasierte Verfahren

Bei den modellbasierten Verfahren unterscheiden wir zwischen der requestgetriebenen und der hybriden Methode (vgl. Abschnitt 3.2).

Requestgetriebenes, modellbasiertes Verfahren. Der Algorithmus zur Umsetzung eines requestgetriebenen, modellbasierten Verfahrens mit diesem Konzept ist in Algorithmus 2 dargestellt.

Algorithmus 2 : Modellbasiertes Vorgehen (requestgetrieben)

Trigger (in) : Ein (impliziter oder expliziter) Request $q \in Q$ aus dem Datenstrom $\mathcal{Q}$

Output : Empfehlungen e als Datenstromelement des Datenstroms $\mathcal{E}$

$L \leftarrow$ Auswahl von Feedbacks, die für das Lernen des Modells benötigt werden (Lerndaten), mit der Funktion $f_L(q, F)$

$m \leftarrow$ Lernen eines Modells anhand der Lerndaten L, mit der Funktion $f_M(q, L)$

$C \leftarrow$ Auswahl von Objekten, die für eine Empfehlung infrage kommen (Kandidaten), mit der Funktion $f_C(q, I)$

for $c \in C$ **do**

  $\hat{r} \leftarrow$ Berechnung der Bewertung des Objekts c anhand des Modells m und der Bewertungsfunktion $\hat{f}_R(c, m)$

$e \leftarrow$ Berechnung der Empfehlungsmenge (Top-K-Menge der bewerteten Kandidaten) unter Verwendung der berechneten Bewertungen $\hat{r}$ und Ausgabe als Datenstromelement

Im Gegensatz zum speicherbasierten Vorgehen (vgl. Gleichung 3.2) wird beim modellbasierten Verfahren die Lerndatenselektion nicht für jeden Kandidaten, sondern je Request durchgeführt. Das Modell wird anhand der Lerndaten einmalig je Request gelernt und anschließend für jeden Kandidaten angewandt. Die Lerndatenselektionsfunktion ist somit wie folgt definiert:

$$f_L : (q, F) \mapsto L \tag{3.9}$$

In den meisten Fällen werden bei modellbasierten Verfahren alle Feedbackdaten als Lerndaten genutzt. In diesem Fall ist die Lerndatenselektionsfunktion wie folgt definiert:

$$f_L : (q, F) \mapsto F \tag{3.10}$$

Diese Funktion kann auch dazu genutzt werden, um das Contextual Pre-Filtering umzusetzen (vgl. Abschnitt 2.1.4). Dazu kann eine auf den Request zugeschnittene Lerndatenmenge gebildet werden. Haben beispielsweise sowohl die Requests als auch die Feedbackdaten ein Kontextattribut x_1, welches den derzeitigen Aufenthaltsort der Benutzerin bzw. des Benutzers beinhaltet, so kann je Aufenthaltsort ein eigenes Modell mit der folgenden Lerndatenselektionsfunktion gelernt werden:

$$f_L : (q, F) \mapsto \{f \in F \mid f.x_1 = q.x_1\} \tag{3.11}$$

Das Lernen des Modells wird durch folgende Funktion f_M ausgedrückt:

$$f_M : (q, L) \mapsto m \tag{3.12}$$

Im Rahmen der Matrixfaktorisierung implementiert diese Funktion ein Lernverfahren, welches als Modell sowohl die User-Feature-Vektoren als auch die Item-Feature-Vektoren zurückgibt.

Anschließend wird analog zu Gleichung 3.1 mit der Kandidatenselektionsfunktion f_C zu dem Request q die Menge der Kandidaten C aus der Menge der Objekte I ausgewählt.

Für jeden Kandidaten $c \in C$ wird anschließend die geschätzte Bewertung $\hat{r}$ durch Anwendung des Modells m bestimmt. Zum Schluss wird die Top-K-Menge als Empfehlungsmenge e gebildet.

Hybrides, modellbasiertes Verfahren. Der requestgetriebene Ansatz ist in der Regel nicht das Vorgehen, welches man bei einem modellbasierten Verfahren anstrebt. Das Neulernen des Modells bei jedem Request ist zeitaufwändig sowie rechenintensiv und verzögert die Beantwortung des Requests. Beim modellbasierten Ansatz ist es in der Regel erstrebenswert, dass das Modell einmalig gelernt und aktualisiert oder regelmäßig in größeren Abständen gelernt wird. Dazu kann der Ansatz der hybriden Empfehlungsberechnung eingesetzt werden (vgl. Abschnitt 3.2.3).

Das hybride, modellbasierte Vorgehen besteht aus den beiden Teilen *Modelllernen* (Algorithmus 3) und *Empfehlungsberechnung* (Algorithmus 4).

Algorithmus 3 : Modellbasiertes Vorgehen (hybrid: Modelllernen)

Trigger (in) : Ein neues Feedbackdatum $f \in F$ aus dem Datenstrom $\mathcal{F}$
Output : Eine neue Modellinstanz $m \in M$ als Datenstromelement des Datenstroms $\mathcal{M}$

$q' \leftarrow$ Ableitung eines impliziten Requests aus dem Feedbackdatum f, mit der Funktion $f_Q(f)$
$m \leftarrow$ Lernen einer neuen Modellinstanz m auf Basis der vorherigen Modellinstanz m' zu dem impliziten Request q' und des neuen Lerndatums l, mit der Funktion $f_M(q', m', l)$

Im ersten Schritt wird für eingehende Feedbackdaten das Modell gelernt bzw. aktualisiert. Dazu wird zunächst mit der Funktion

$$f_Q : (f) \mapsto q' \tag{3.13}$$

aus dem neuen Feedback f ein impliziter Request q' abgeleitet (vgl. die Einführung des Konzept von impliziten Requests in Abschnitt 3.2.2; das genaue Vorgehen zum Ableiten eines impliziten Requests wird in Abschnitt 3.6.2 erläutert). Die Ableitung des impliziten Request ermöglicht, dass beispielsweise für jeden Aufenthaltsort eines Benutzers (z. B. das Land) ein eigenes Modell gelernt wird. In diesem Beispiel

enthält der implizite Request q' als einzigen Parameter das Kontextattribut x_1 des Feedbackdatums. Im einfachsten Fall wird nur ein globales Modell gelernt. In diesem Fall ist der implizite Request q' für alle eingehenden Feedbackdaten identisch und hat keine Attribute.

Im nächsten Schritt wird zu dem impliziten Request q' eine Modellinstanz m gelernt. In Idealfall wird dazu die vorherige Modellinstanz m' herangezogen und mit der Funktion f_M inkrementell aktualisiert (z. B. mit dem BRISMF-Algorithmus [Tak+09]):

$$f_M : (q', m', l) \mapsto m \tag{3.14}$$

Soll ein Lernalgorithmus eingesetzt werden, der die inkrementelle Aktualisierung nicht vorsieht, so kann m' auch ignoriert und das Modell von Grund auf neu gelernt werden. Limitierender Faktor ist dabei die Latenz für die Neuberechnung des Modells.

Als Ergebnis wird für jede aktualisierte bzw. neugelernte Modellinstanz m ein Datenstromelement mit ebendieser Modellinstanz über den Modelldatenstrom $\mathcal{M}$ ausgegeben. Zur Berechnung der Empfehlungen kann dieser anschließend als Quelldatenstrom genutzt werden und somit auf die Modellinstanzen zugegriffen werden. Da die Datenstromelemente mit einem Zeitstempel t annotiert sind, wird durch das Datenstrommanagementsystem sichergestellt, dass die temporal korrekte Modellinstanz zur Empfehlungsberechnung genutzt wird. Das stellt eine deterministische Verarbeitung (und somit auch Reproduzierbarkeit) sicher.

Algorithmus 4 : Modellbasiertes Vorgehen (hybrid: Empfehlungsberechnung)

Trigger (in) : Ein Request $q \in Q$ aus dem Datenstrom $\mathcal{Q}$
Output : Empfehlungen e als Datenstromelement des Datenstroms $\mathcal{E}$

$m \leftarrow$ Auswahl des passenden Modells für den Request q aus dem Datenstrom $\mathcal{M}$
$C \leftarrow$ Auswahl von Objekten, die für eine Empfehlung infrage kommen (Kandidaten), mit der Funktion $f_C(q, I)$
for $c \in C$ **do**
  $\hat{r} \leftarrow$ Berechnung der Bewertung des Objekts c anhand des Modells m und der Bewertungsfunktion $\hat{f}_R(c, m)$
$e \leftarrow$ Berechnung der Empfehlungsmenge (Top-K-Menge der bewerteten Kandidaten) unter Verwendung der berechneten Bewertungen $\hat{r}$ und Ausgabe als Datenstromelement

Der zweite Teil ist die eigentliche Empfehlungsberechnung, die von einem Request q angestoßen wird. Dazu wird zunächst aus dem Modelldatenstrom $\mathcal{M}$ das zum Request q passende Modell m ausgewählt. Das weitere Vorgehen ist analog zum requestgetriebenen Ansatz: Es werden die Kandidaten C ausgewählt, für jeden

Kandidaten $c \in C$ eine geschätzte Bewertung $\hat{r}$ durch Anwendung des Modells m berechnet und anschließend die Top-K-Menge als Empfehlung ausgegeben.

3.5 Query-Architektur

In diesem Abschnitt wird die Datenverarbeitung eines Online-Recommender-Systems in sieben aufeinanderfolgende Phasen unterteilt, die von den Feedback- und Requestdaten durchlaufen werden. Abbildung 3.16 stellt diese Phasen grafisch dar: Auf der linken Seite befinden sich die Datenquellen mit den eingehenden Daten (Feedbacks und Requests, Objektinformationen, Kontextdaten), auf der rechten Seite die ausgehenden Daten (Empfehlungen). Dazwischen sind die sieben Phasen für die Datenstromverarbeitung dargestellt.

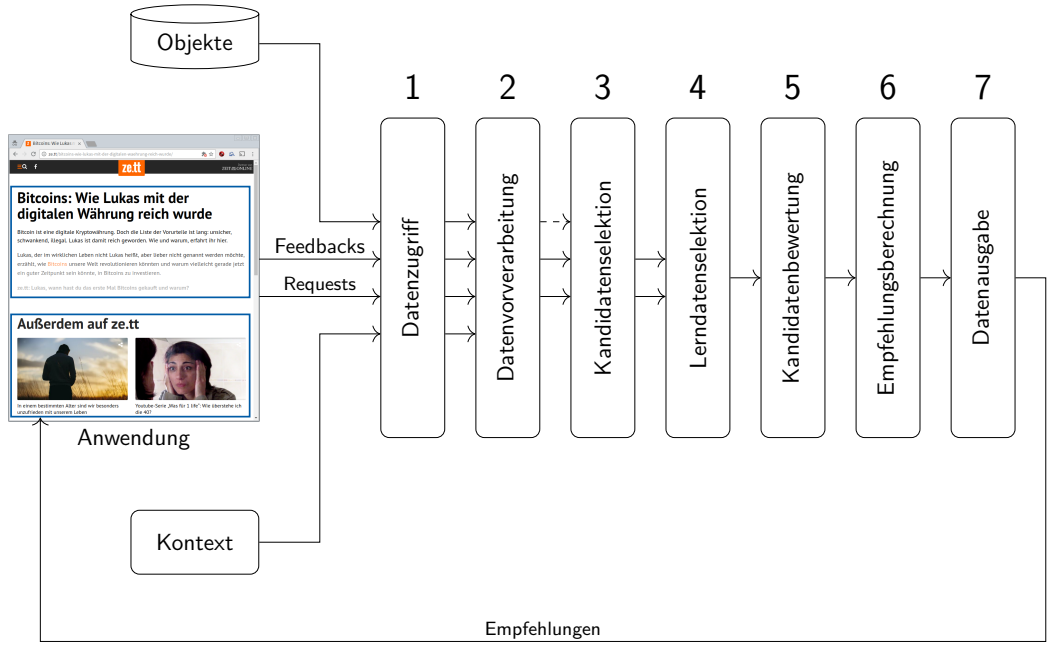


Abbildung 3.16 Phasen eines Online-Recommender-Systems

3.5.1 Phase 1: Datenzugriff

Den erste Verarbeitungsschritt eines Online-Recommender-Systems stellt der Zugriff auf die Eingabedaten dar. An dieser Stelle werden die eingehenden Datenströme an das Datenstrommanagementsystem angebunden, die Daten kontinuierlich entgegengenommen, sowie die Anbindung von Datenbanken und Services festgelegt. Zu den Aufgaben dieser Phase gehören beispielsweise:

- Das Aufbauen und Halten einer Verbindung zur Datenquelle,
- das Entgegennehmen von Daten von der Datenquelle,
- das periodische Abrufen von Daten, wenn die Datenquelle nicht in der Lage ist, Daten zu pushen,
- die Verbindung zu einer Datenbank oder einem Webservice,
- das Parsen der Daten (JSON, CSV, ...) sowie
- das Erstellen von Datenstromelementen im internen Datenformat des Datenstrommanagementsystems (in unserem Fall als Tupel).

Die Datenanbindung ist eine originäre Aufgabe eines Datenstrommanagementsystems, für die auf die Standardfunktionen ohne spezifische Anpassungen zurückgegriffen werden kann. Die eingehenden Daten werden häufig nicht in genau dem Format übergeben, wie sie der Recommender-System-Algorithmus benötigt. Deshalb werden die Daten an die Phase *Datenvorverarbeitung* übergeben, die die Aufgabe hat, die Daten in das geforderte Format zu überführen.

3.5.2 Phase 2: Datenvorverarbeitung

Während bei der Batchverarbeitung die Daten vor der Nutzung einmalig vorverarbeitet werden müssen, ist dies bei Online-Recommender-System ein kontinuierlicher Prozess: Jedes neue Datenelement muss für die Verarbeitung durch den Recommender-System-Algorithmus vorbereitet werden. Dazu gehören beispielsweise:

- Das Anreichern von Feedback- und Requestdaten um Informationen aus Datenbanken und Webservices (Kontextinformationen, beispielsweise das Wetter am Ort der Objektbewertung),
- das Normalisieren sowie Diskretisieren von Werten,
- das Überführen der Daten vom Quelldatenformat in das Datenformat, das von den nachfolgenden Recommender-System-Algorithmen benötigt wird und
- das Filtern von ungewollten Daten (z. B. Ausreißer).

Ziel der Phasen *Datenzugriff* und *Datenvorverarbeitung* ist es, den nachfolgenden Phasen einen einheitlichen Zugriff auf die Daten zu ermöglichen. Das Ergebnis sind die folgenden Datenströme:

Objektdatenstrom

Der Objektdatenstrom enthält die Empfehlungsobjekte, die das Recommender-System empfehlen kann. Der Objektdatenstrom kann entfallen, wenn nur die Objekte empfohlen werden sollen, zu denen es Feedbackdaten gibt. In diesem

Fall können die Empfehlungsobjekte aus dem Feedbackdatenstrom abgeleitet werden (vgl. Datenmodell in Abschnitt 3.3.1).

Feedbackdatenstrom

Der Feedbackdatenstrom enthält die Feedbacks, die von der Anwendung übermittelt werden, ggf. angereichert mit Kontextdaten und Objektinformationen (vgl. Datenmodell in Abschnitt 3.3.2).

Requestdatenstrom

Der Requestdatenstrom enthält die Requests for Recommendations, die von der Anwendung übermittelt werden, ggf. angereichert mit Kontextdaten und Objektinformationen (vgl. Datenmodell in Abschnitt 3.3.3).

Objektdatenstrom Eine der wichtigsten Entitätstypen in einem Recommender-System sind die Objekte, die von den Benutzerinnen und Benutzern genutzt und ebendiesen empfohlen werden. Die Objekte in einem Recommender-System bestehen aus einer Objekt-ID i, aus einer von der Anwendung abhängigen und hier nicht näher spezifizierten Menge an Attributen $\vec{y} = \{y_1, y_2, \ldots, y_m\}$ sowie dem Metadatum t. Daraus ergibt sich das Schema

$$\mathcal{I} : \{i, y_1, y_2, \ldots, y_m\}_t \tag{3.15}$$

für ein Objekt im Objektdatenstrom $\mathcal{I}$, welches zum Zeitpunkt t erstellt wurde.

Feedbackdatenstrom Die Datengrundlage für die Berechnung von Empfehlungen ist das Feedback von Benutzerinnen und Benutzern zu Objekten. Dazu zählt sowohl explizites Feedback (z. B. die explizite, quantitative Bewertung von Nachrichtenartikeln) sowie implizites Feedback in Form von beobachteten Aktionen von Benutzerinnen und Benutzern (z. B. das Aufrufen eines Nachrichtenartikels). Tabelle 3.1 zeigt ein typisches Beispiel für Feedbackdaten mit expliziten Bewertungen von Nachrichtenartikeln. Jedes Feedback besteht aus einer Bewertung zu einen Objekt mit einer Objekt-ID zu einem bestimmten Zeitpunkt. Häufig enthalten Feedbackdaten weitere Informationen, wie zum Beispiel die ID der Benutzerin oder des Benutzers, die/der das Feedback abgegeben hat, der Aufenthaltsort der Benutzerin oder des Benutzers, das Wetter zum Zeitpunkt der Bewertung etc. Diese Attribute können direkt durch die Anwendung mitgegeben oder durch Anreicherung aus anderen Datenquellen in einer Vorverarbeitungsphase den Feedbacks hinzugefügt werden.

Welche Informationen in den Feedbackdaten enthalten sind, unterscheidet sich von Anwendung zu Anwendung. Da wir die Daten als Datenstrom betrachten, haben wir immer einen Zeitstempel der Bewertung. Wird dieser von der Anwendung nicht mitgegeben, so wird dieser vom Datenstrommanagementsystem gesetzt. Die

Time-stamp	Item	Title	Category	Rating	User	Age	City	Weather
5209	8658	Inside College Bas...	Sports	4.0	504	23	Boston	Rainy
5228	9806	Bob Dylan Says ...	Arts	2.0	950	66	Oldenburg	Sunny
5239	2275	In Estonia, Cautio...	Politics	3.5	651	48	Boston	Cloudy
5251	6308	Across China, Wa...	Politics	4.5	499	16	Paris	Sunny
...	...	...	...	...	...	...	...	...

Tabelle 3.1 Feedback-Eingabedatenstrom

minimalste Form von Feedback ist die alleinige Angabe einer Objekt-ID mit der Bedeutung: *„Irgendeine Benutzerin oder irgendein Benutzer hat mit dem Objekt i eine Interaktion durchgeführt“*, beispielsweise den Nachrichtenartikel i aufgerufen. Diese Information genügt, um ein Recommender-System zu realisieren, das die derzeitig am häufigsten aufgerufenen Artikel empfiehlt.

Die Informationen eines Feedbackelements verallgemeinern wir wie folgt:

Zeitstempel

Der Zeitpunkt t, zu dem das Feedback abgegeben wurde. Wird diese Information nicht von der Anwendung zur Verfügung gestellt, so wird die aktuelle Systemzeit beim Eintreffen in das Datenstrommanagementsystem als Zeitstempel genutzt.

Feedback-ID

Die Feedback-ID f ermöglicht die eindeutige Identifikation eines Feedbacks. Da diese von den hier betrachteten Recommender-System-Algorithmen und diesem Konzept nicht benötigt wird, entfällt diese in der Regel und ist hier nur der Vollständigkeit halber aufgeführt.

Objekt-ID

Die Objekt-ID i identifiziert das Objekt, zu dem das Feedback abgegeben wird.

Objektattribute

Eine Menge von Objektinformationen $\vec{y} = \{y_1, y_2, \ldots, y_m\}$, die das Objekt näher beschreiben. Werden diese nicht von der Anwendung mitgeliefert, so kann das Datenstrommanagementsystem im Rahmen der Vorverarbeitung die Feedbackdaten mit Informationen aus einer Objektdatenbank anreichern.

Bewertung

Eine explizite oder implizite Bewertung r des Objekts.

Kontextdaten

Eine Menge von Informationen $\vec{x}_F = \{x_1, x_2, \ldots, x_n\}$, die den Kontext der Bewertung näher beschreiben. Dazu können gehören:

- Eine Identifikation der Benutzerin oder des Benutzers, von der/dem das Feedback stammt.
- Informationen über die Benutzerin oder den Benutzer (Alter, Geschlecht, Wohnort, Vorlieben, ...).
- Informationen über die Situation der Benutzerin oder des Benutzers zum Zeitpunkt der Bewertung (Aufenthaltsort, Wetter, Stimmung, ...).
- Explizite Angaben der Benutzerin oder des Benutzers zu dem aktuellen Interesse (Angabe von Schlagwörtern, Auswahl einer Kategorie, ...).
- Bei implizitem Feedback: die Art der Interaktion mit dem Objekt (Artikel aufgerufen, an Freundin oder Freund empfohlen, ausgedruckt ...).

Aus diesen Angaben definieren wir das folgende, generische Relationenschema $\mathcal{F}$ für Feedbackdaten:

$$\mathcal{F} : \{i, r, x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m\}_t \tag{3.16}$$

Das Schema enthält die Objekt-ID i, die Bewertung r, die Menge aller Kontextinformationen zu einem Feedback $\vec{x}_F = \{x_1, x_2, \ldots, x_n\}$ und die Menge aller Objektinformationen $\vec{y} = \{y_1, y_2, \ldots, y_m\}$, sowie als Metadatum den Zeitstempel t. Das Schema lässt sich auch verkürzt wie folgt darstellen:

$$\mathcal{F} : \{i, r, \vec{x}_F, \vec{y}\}_t \tag{3.17}$$

Requestdatenstrom Ein Request ist der Auslöser, eine Empfehlung auszugeben. Ebenso wie ein Feedback kann ein Request Attribute enthalten, die den Kontext des Requests charakterisieren. Diese können vom Recommender-System genutzt werden, um die Empfehlungen zu personalisieren. Tabelle 3.2 zeigt ein Beispiel für Requestdaten.

Time-stamp	ReqID	User	Age	City	Weather	Categories	Preferences
5210	364	504	23	Boston	Rainy	Arts, Sports	Boston Celtics
5226	365	950	66	Oldenburg	Sunny	Politics	
5232	366	651	48	Boston	Cloudy	Arts, Politics	Picasso
5249	367	499	16	Paris	Sunny	Sports	
...	...	...	...	...	...	...	...

Tabelle 3.2 Request-Eingabedatenstrom (Beispiel)

Ein minimaler Request enthält keine Nutzdaten, sondern lediglich eine Request-ID q sowie als Metadatum den Zeitstempel t. In diesem Fall ist ein Request lediglich ein Ereignis welches signalisiert, eine (nicht personalisierte) Empfehlung auszugeben. Zum Zwecke der Personalisierung werden dem Request in der Regel weitere Kontext-

daten mitgegeben. Diese können die gleichen Angaben sein, die auch den Feedbacks mitgegeben werden (z. B. Benutzerinnen-/Benutzer-ID, Aufenthaltsort, ...) aber auch darüber hinausgehende Angaben, wie beispielsweise das aktuell genutzte Objekt, zu dem weitere Empfehlungen geben werden sollen oder explizite Angaben der Benutzerin oder des Benutzers über das aktuelle Interesse.

Das allgemeine Relationenschema eines Requests stellt sich damit wie folgt dar:

$$\mathcal{Q} : \{q, x_1, x_2, \ldots, x_n\}_t \tag{3.18}$$

mit einer Request-ID q und einer ggf. leeren Menge an Attributen $\vec{x}_Q = \{x_1, x_2, \ldots, x_n\}$. In verkürzter Schreibweise ergibt sich damit das Schema:

$$\mathcal{Q} : \{q, \vec{x}_Q\}_t \tag{3.19}$$

Ein Request wird meist durch die Anwendung ausgelöst, beispielsweise durch das Öffnen der Seite, die die Empfehlungen anzeigen soll. Alternativ kann ein Request auch zeitgesteuert ausgelöst werden, z. B. für Empfehlungen im Rahmen eines Newsletters.

3.5.3 Phase 3: Kandidatenselektion

Für die Berechnung der Empfehlungen muss als Erstes entschieden werden, welche Objekte zur Empfehlung infrage kommen. Diese Objekte werden in dieser Arbeit als *Kandidaten* bezeichnet. Grundlage der Kandidatenmenge bildet die Menge aller Objekte $\mathcal{I}_t$, die dem Recommender-System zum Zeitpunkt t bekannt sind. Als Quellen für die Objektmenge $\mathcal{I}_t$ kommen folgende in Betracht:

1. Objekte aus einer Objektdatenbank, die zum Zeitpunkt t angefragt wird.
2. Objekte, die ein Webservice zur Verfügung stellt, der zum Zeitpunkt t angefragt wird.
3. Ein Datenstrom von Objekten, der aus einer Objektdatenbank oder einem Webservice erzeugt wird.
4. Bewertete Objekte im Feedbackdatenstrom. In diesem Fall werden als Objektmenge alle Objekte betrachtet, zu denen in einem bestimmten Fenster Feedbackdaten existieren.

Ein Kandidat ist immer einem Request (ggf. einem impliziten Request) zugeordnet, weswegen sich das Relationenschema aus dem Schema des Requests und dem der

Empfehlungsobjekte wie folgt zusammensetzt:

$$\mathcal{C} : \{q, x_1, x_2, \ldots, x_n, i, y_1, y_2, \ldots, y_m\}_t \tag{3.20}$$

mit einer Request-ID q und einer ggf. leeren Menge an Requestattributen $\vec{x}_Q = \{x_1, x_2, \ldots, x_n\}$ sowie der Objekt-ID i des Kandidaten mit einer ggf. leeren Menge an Objektattributen $\vec{y} = \{y_1, y_2, \ldots, y_m\}$. In verkürzter Schreibweise ergibt sich damit das Schema:

$$\mathcal{C} : \{q, \vec{x}_Q, i, \vec{y}\}_t \tag{3.21}$$

Time-stamp	ReqID	User	Age	...	ItemID	Title	Category	...
5210	364	504	23	...	8659	Inside College Bas...	Sports	...
5210	364	504	23	...	2275	In Estonia, Cautio...	Politics	...
5210	364	504	23	...	9806	Bob Dylan Sys...	Arts	...
...	...	...	...	...	...	...	...	
5226	365	950	66	...	2275	In Estonia, Catio...	Politics	...
...	...	...	...	...	...	...	...	

Tabelle 3.3 Kandidatendatenstrom (Beispiel)

Für die Kandidatenauswahl sei die folgende Funktion f_C definiert:

$$f_C : X_R \times I_t \rightarrow C_t \tag{3.22}$$

Diese Funktion bildet die Menge aller Objekte I_t, die dem System zum Zeitpunkt t bekannt sind, auf die Menge der Kandidaten C_t zum Zeitpunkt t für einen Request mit den Attributen X_R ab.

Im einfachsten Fall werden alle bekannten Objekte als Kandidaten für eine Empfehlung genutzt. Eine Einschränkung der Kandidatenmenge kann sinnvoll sein, um die Empfehlungsberechnung zu beschleunigen oder die Empfehlungen durch ein Vorfiltern zu beeinflussen. Beispiele für ein Vorfiltern von Kandidaten ist die Beschränkung auf Objekte, mit denen in den vergangenen 24 Stunden eine Interaktion stattgefunden hat (Fokussierung auf *aktuelle* Objekte) oder Objekte, zu denen es am Aufenthaltsort der Benutzerin oder des Benutzers eine Interaktion gegeben hat (*Context Filtering*; z. B. Fokussierung auf Nachrichtenartikel, die in der Region relevant sind).

3.5.4 Phase 4: Lerndatenselektion

Die Phase der Lerndatenselektion unterscheidet sich danach, ob eine speicherbasierte oder modellbasierte Methode eingesetzt wird. Bei einer speicherbasierten Methode werden zu jedem *Kandidaten* die Feedbackdaten als Lerndaten zugeord-

net, die zur Bewertung des Kandidaten benötigt werden. Bei einer modellbasierten Methode werden zunächst zu jedem (impliziten oder expliziten) *Request* die Lerndaten zugeordnet, mit denen dann ein Modell gelernt wird. Anschließend wird zu jedem Kandidaten das Modell zugeordnet, mit dem in der nächsten Phase die Kandidatenbewertung erfolgt.

Im einfachsten Fall werden alle zur Verfügung stehenden Interaktionen zu Objekten als Lerndatenmenge ausgewählt. Zur Berücksichtigung von Concept Drifts oder Kontextinformationen können diese auch gefiltert werden, z. B. nur Interaktionen in einem bestimmten Zeitraum oder Interaktionen mit bestimmten Kontextinformationen (z. B. alle Interaktionen zur selben Tageszeit).

Für die Lerndatenauswahl sei die folgende Funktion f_L definiert:

$$f_L : F_t \rightarrow L_t \tag{3.23}$$

Diese Funktion bildet die Menge aller gültigen Feedbackdaten F_t, die dem System zum Zeitpunkt t bekannt sind, auf die Menge der Lerndaten F_t zum Zeitpunkt t ab.

3.5.5 Phase 5: Kandidatenbewertung

Mithilfe der Lerndaten werden nun die Kandidaten bewertet. Abhängig davon, wie die Kandidatenbewertung erfolgen soll, kann eine globale Lerndatenmenge oder eine Lerndatenmenge je Kandidat oder Kontextausprägung ausgewählt werden. Sollen beispielsweise die am meisten genutzten Objekte empfohlen werden, kann zu jedem Kandidaten c die Menge der Interaktionen als Lerndatenmenge zugeordnet werden, die eine Interaktion zu dem Kandidaten c darstellen (z. B. der Aufruf des Nachrichtenartikels). Die Anzahl der Interaktionen ergibt dann die Bewertung.

Soll zum Beispiel ein Modell trainiert werden, welches sensibel für Concept Drifts sein soll, können als Lerndatenmenge alle Interaktionen der letzten 24 Stunden genutzt werden. Das Modell wird bei Veränderung der Lerndatenmenge inkrementell aktualisiert. Alternativ können auch für verschiedene Orte unterschiedliche Modelle trainiert werden. In diesem Fall muss im vorherigen Schritt für jeden Ort eine eigene Lerndatenmenge ausgewählt werden.

Eine Alternative zum modellbasierten Ansatz stellt die Aggregation von Lerndaten je Kandidat dar. Bestehen die Lerndaten beispielsweise aus Interaktionen in Form von Objektbewertungen (z. B. die Auswahl von 1 bis 5 »Sternen«), so kann man die durchschnittliche Bewertung je Objekt berechnen, um die am höchsten bewerteten

Objekte zu empfehlen. In diesem Fall wird für jeden Kandidaten eine Lerndatenmenge mit allen Interaktionen zu diesem Kandidaten gebildet (ggf. kombiniert mit der Begrenzung auf einen bestimmten Zeitraum und/oder Kontext).

3.5.6 Phase 6: Empfehlungsberechnung

Nachdem die Kandidaten bewertet wurden, wird daraus die Empfehlungsmenge gebildet. Dazu wird aus allen Kandidaten die Top-K-Menge der K Kandidaten ermittelt, die im vorherigen Schritt am besten bewertet wurden.

Als Empfehlung übergibt das Recommender-System eine geordnete Menge an Objekt-IDs an die Anwendung, damit die dazugehörigen Objekte der Benutzerin oder dem Benutzer als Empfehlungen angezeigt werden können. Die Empfehlungen sind nach geschätzter Bewertung sortiert. Neben den Objekt-IDs können auch die Bewertungen zu jedem Objekt mit übertragen werden. Tabelle 3.4 zeigt ein Beispiel für einen Datenstrom mit Empfehlungen.

Time-stamp	ReqID	Recommendations	
		2275	4.5
5210	364	8658	4.0
		1379	3.9
		8658	4.9
5226	365	6308	4.5
		2275	4.4
		6455	5.0
5232	366	1379	4.5
		2275	4.4
		9582	4.5
5249	367	8658	4.0
		6308	3.0
...	...	...	

Tabelle 3.4 Empfehlungen-Ausgabedatenstrom (Beispiel) mit Objekt-IDs und geschätzten Bewertungen der Empfehlungen

Üblicherweise ist die Anzahl der Empfehlungen begrenzt (z. B. die K nützlichsten Objekte). Alternativ kann die Ausgabe auch auf Objekte mit einem Mindest-Score begrenzt werden. Ist die (maximale) Anzahl der Empfehlungen festgelegt, so kann das minimale Relationenschema $\mathcal{E}$ für K Empfehlungen wie folgt definiert werden:

$$\mathcal{E} : \{q, i_1, i_2, \ldots, i_K\}_t \tag{3.24}$$

Dieses Schema enthält die Request-ID q zu dem Request, der mit diesen Empfehlungen beantwortet wird, sowie die Empfehlungen $i_1, i_2, \ldots, i_K$. Bei Bedarf kann

das Schema um die Objektattribute $\vec{y}$ sowie die geschätzten Bewertungen $\hat{r}$ der empfohlenen Objekte erweitert werden:

$$\mathcal{E} : \{q, i_1, \vec{y}_1, \hat{r}_1, i_2, \vec{y}_2, \hat{r}_2, \ldots, i_K, \vec{y}_K, \hat{r}_K\}_t \tag{3.25}$$

3.5.7 Phase 7: Datenausgabe

Zum Schluss werden die Empfehlungen als Datenstrom ausgegeben. Dazu muss, analog zu den Eingabedatenströmen, spezifiziert werden, wie die Daten übertragen werden sollen.

3.6 Querypläne

In den bisherigen Abschnitten des Kapitels *Konzepte* haben wir allgemeine Konzepte für die Umsetzung von Recommender-Systemen auf Basis eines Datenstrommanagementsystems betrachtet. In diesem Abschnitt werden Querypläne zur Lösung konkreter Probleme innerhalb des Konzepts vorgestellt. Dazu betrachten wir zunächst ein Beispiel für einen einfachen speicherbasierten Ansatz am konkreten Beispiel der requestgetriebenen Empfehlung von häufig gelesenen Nachrichtenartikeln (Abschnitt 3.6.1). Anschließend geht der Abschnitt 3.6.2 auf die Berechnung von abgeleiteten Requests ein. Abschnitt 3.6.3 stellt eine Lösung für den konkreten Fall der feedbackgetriebenen Empfehlungsberechnung bei disjunkter Kandidaten- bzw. Lerndatenmenge vor. Zum Schluss wird in Abschnitt 3.6.4 ein Queryplan für die Integration von modellbasierten Methoden vorgestellt.

3.6.1 Requestgetriebene Empfehlung von häufig gelesenen Nachrichtenartikeln in der Umgebung

Gegeben sei ein Requestdatenstrom, der zu jedem Request die gegenwärtige Position x_1 der Benutzerin bzw. des Benutzers enthält sowie ein Feedbackdatenstrom der neben der Objekt-ID i ebenso die Position x_1 der Benutzerin bzw. des Benutzers liefert. Empfohlen werden sollen die K Objekte, die im Umkreis von 50 km innerhalb der letzten 60 Minuten am häufigsten gelesen wurden. Dazu sei eine Funktion f_Δ gegeben, die zu zwei Positionen die Distanz bestimmt.

Der erste Schritt nach der Datenanbindung und -vorverarbeitung ist die Bestimmung der Kandidaten für jeden Request. Die Menge der Kandidaten $\mathcal{C}$ für einen Request R ist die Menge der Objekte, für die es ein Feedback innerhalb von 50 km gibt:

$$C_q = \{q.k, q.x_1, f.i \in F \times R \mid f_\Delta(q.x_1, f.x_1) \leq 50\,\text{km}\} \tag{3.26}$$

mit $q.k$ als Request-ID, x_1 als Request- bzw. Feedback-Kontext-Attribut, welches die Position der Benutzerin bzw. des Benutzers angibt, sowie $f.i$ als Objekt-ID des Feedbacks.

Als nächstes wird zu jedem Kandidaten die Menge der Lerndaten bestimmt, die benötigt wird, um die Bewertung zu berechnen. Die Bewertung berechnet sich in diesem Beispiel durch die Anzahl der Feedbacks, die es zu einem Kandidaten im Umkreis von 50 km gibt. Somit ergibt sich die Lerndatenmenge zu einem Kandidaten C wie folgt:

$$L_c = \{c.q, c.x_1, c.i, f.k \in C \times R \mid f_\Delta(c.x_1, f.x_1) \leq 50\,\text{km}\} \tag{3.27}$$

mit $c.q$ als Request-ID des Kandidaten, x_1 als Request- bzw. Feedback-Kontext-Attribut, welches die Position der Benutzerin bzw. des Benutzers angibt, $c.i$ als Objekt-ID des Kandidaten sowie $f.k$ als Feedback-ID des Lerndatums. Die Bewertung eines Kandidaten erfolgt nun über die Aggregation mit den Gruppierungsattributen q und i sowie der Aggregationsfunktion COUNT.

Abbildung 3.17 stellt den Ablauf in Form eines Queryplans dar. Die Berechnung der Empfehlungen besteht aus sieben Operatoren:

- Kandidatenselektion
  1. JOIN $\bowtie$ von Requests $q \in Q$ und Feedbacks $f \in F$ mit dem Prädikat $\theta : f_\Delta(q.x_1, f.x_1) \leq 50\,\text{km}$ (vgl. Gleichung 3.26).
  2. PROJECTION π auf die Attribute $\{q.k, q.x_1, f.i\}$ (vgl. Gleichung 3.26).
  3. DISTINCT η zur Entfernung von Duplikaten.
     Ergebnis ist die Menge der Kandidaten C.
- Lerndatenselektion
  4. JOIN $\bowtie$ von Empfehlungskandidaten $c \in C$ und Feedbacks $f \in F$ mit dem Prädikat $\theta : f_\Delta(c.x_1, f.x_1) \leq 50\,\text{km}$ (vgl. Gleichung 3.27).
  5. PROJECTION π auf die Attribute $\{c.q, c.x_1, c.i, f.k\}$ (vgl. Gleichung 3.27).
     Ergebnis ist die Menge der Lerndaten L.
- Kandidatenbewertung

6. AGGREGATION γ mit den Gruppierungsattributen $\{q, i\}$ und der Aggregationsfunktion $\hat{r} = \mathsf{COUNT}()$.
   Ergebnis sind die bewerteten Kandidaten C'.

- Empfehlungsberechnung
  7. AGGREGATION γ mit dem Gruppierungsattribut $\{q\}$ und der Aggregationsfunktion $\mathsf{TOP\_K}(\hat{r})$.
     Ergebnis sind die K best-bewerteten Kandidaten und somit die Empfehlungsmenge e.

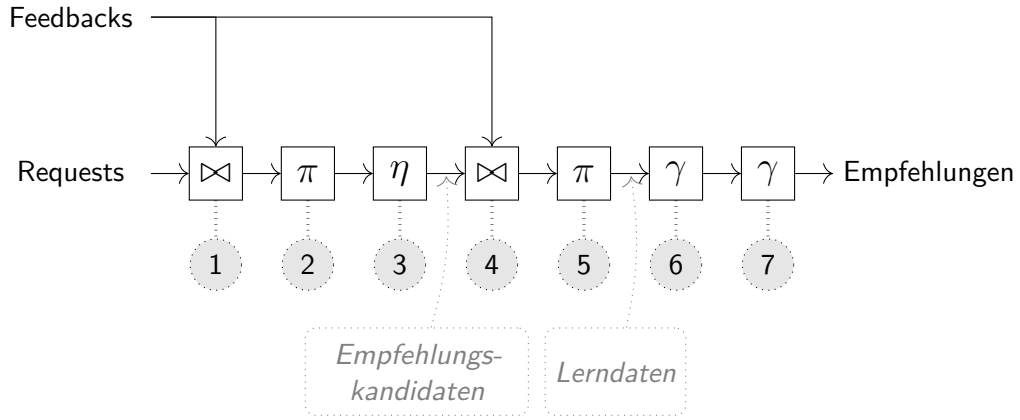


Abbildung 3.17 Queryplan für die requestgetriebene Empfehlung

3.6.2 Ableitung eines impliziten Requests

Bei der Umsetzung eines Recommender-Systems mit dem Ansatz der feedbackgetriebenen Empfehlungsberechnung (vgl. Abschnitt 3.2.2), werden bei jeder Änderung der Feedbackdaten die Empfehlungsmengen für jeden möglichen Request, der von der Änderung der Feedbackdaten betroffen ist, inkrementell aktualisiert. Trifft ein Request ein, so werden dem Request die vorberechneten Empfehlungen zugeordnet und an die Benutzerin bzw. den Benutzer ausgegeben. Um die Empfehlungen für Requests vorberechnen zu können, müssen die möglichen Requests abzählbar endlich sein und aus den Feedbackdaten geschlossen werden können. Die aus Feedbackdaten abgeleiteten Requests nennen wir *implizite Requests*.

Betrachten wir dazu zunächst ein Beispiel. Gegeben sei ein Requestdatenstrom mit dem einzigen Kontextattribut x_1^Q, welches das Land angibt, in dem sich die Benutzerin bzw. der Benutzer derzeit aufhält, für die bzw. für den eine Empfehlung berechnet werden soll. Des Weiteren sei ein Feedbackdatenstrom gegeben, der neben der ID i des Nachrichtenartikels, den die Benutzerin bzw. der Benutzer gerade geöffnet hat, als einziges Kontextattribut x_1^F das Land enthält, in dem sich die Benutzerin bzw. der Benutzer aufgehalten hat, als sie bzw. er den Nachrichtenartikel gelesen hat. Eine mögliche Recommender-System-Implementierung gibt nun als

Empfehlung die fünf Objekte zurück, die in dem entsprechenden Land in der letzten Stunde am häufigsten gelesen wurden.

Betrachtet man dieses Beispiel, so stellt man fest, dass zu einem Request aus einem Land x' nur dann Empfehlungen gegeben werden können, wenn es auch Feedbackdaten aus ebendiesem Land gibt. Requests aus einem Land ohne Feedbackdaten müssen im Zweifelsfall mit einer Fallbackempfehlung beantwortet werden (z. B. die meistgelesenen Artikel überhaupt). Betrachtet man die Feedbackdaten F als eine Tupelmenge mit dem Schema $\mathcal{F} : \{i, r, \vec{x}_F, \vec{y}\}$, wobei die Attribute $\vec{x}_F = \{x_1, x_2, \ldots, x_n\}$ mit den Requestattributen $\vec{x}_Q = \{x_1, x_2, \ldots, x_n\}$ übereinstimmen, so lässt sich mit folgenden Operationen die Menge der möglichen Requests bestimmen (vgl. Abbildung 3.18):

1. Entfernung aller Attribute der Feedbackdaten F außer den Kontextdaten $\vec{x}_F = \{x_1, x_2, \ldots, x_n\}$ (PROJECT-Operation $\pi_{x_1,x_2,\ldots,x_n}(\mathcal{F})$).
2. Eliminierung aller Duplikate (DISTINCT-Operation $\eta(\ldots)$).

Die Ergebnismenge ist die Menge der impliziten Requests. Diese sind, bezogen auf das Beispiel, alle Länder, zu denen Feedbackdaten existieren (ohne Duplikate).

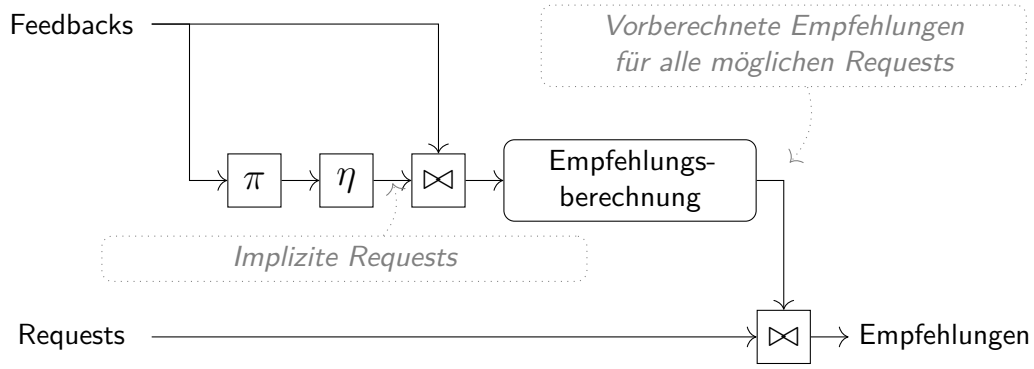


Abbildung 3.18 Berechnung von Empfehlungen mit impliziten Requests

3.6.3 Feedbackgetriebene Empfehlungsberechnung bei disjunkter Kandidaten- bzw. Lerndatenmengen

Betrachtet man das Beispiel der Empfehlung der meistgelesenen Artikel je Land aus dem letzten Abschnitt, so stellt man fest, dass die Feedbackdaten zur Bestimmung der Empfehlungskandidaten bzw. Lerndaten in disjunkte Teilmengen aufgeteilt werden: Die Feedbackdaten zur Bestimmung der Empfehlungskandidaten bzw. Lerndaten für ein Request q' aus einem Land x'_1 werden nur für den Request q' benötigt und nicht für einen anderen Request mit einem anderen Land $x''_1 \neq x'_1$. Ein Gegenbeispiel ist das Beispiel aus Abschnitt 3.6.1, in dem die meistgelesenen Artikel im Umkreis von

50 km empfohlen werden sollen: Die Feedbackdaten, die im Umkreis eines Request q' mit der Position x_1' liegen, können sich mit denen überschneiden, die im Umkreis eines zweiten Requests q'' mit einer anderen Position $x_1'' \neq x_1'$ liegen.

Die Situation, in der wir die Feedbackdaten zur Bestimmung der Kandidaten- bzw. Lerndatenmenge in disjunkte Teilmengen aufteilen können, erlaubt uns für die feedbackgetriebene Empfehlungsberechnung die Optimierung des Queryplans. Die Schritte der Kandidatenselektion, Lerndatenselektion und Kandidatenbewertung, wie sie in Abschnitt 3.6.1 beschrieben sind, sowie die Berechnung der impliziten Requests aus dem vorherigen Abschnitt, können in diesem Fall zu einem Schritt vereint werden.

Abbildung 3.19 zeigt das Vorgehen: Die Vorberechnung der Empfehlungsmengen besteht aus zwei AGGREGATION-Operationen. Die erste Aggregation mit den Gruppierungsattributen $\{f.i, f.x_1\}$ und der Aggregationsfunktion COUNT berechnet für jedes Objekt in jedem Land die Anzahl der Vorkommnisse in den Feedbackdaten. Die zweite Aggregation berechnet die Top-K-Menge. Der JOIN verbindet die Requests mit den vorberechneten Empfehlungsmengen mit dem Prädikat $\theta : f.x_1 = q.x_1$.

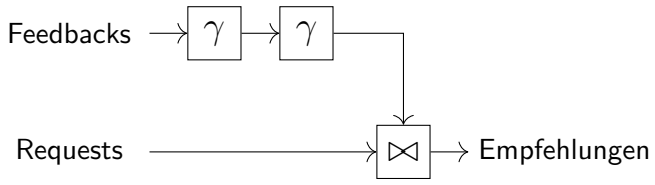


Abbildung 3.19 Queryplan für die feedbackgetriebene Empfehlungsberechnung bei disjunkten Kandidaten- bzw. Lerndatenmengen

3.6.4 Modellbasierte Empfehlung am Beispiel der Matrixfaktorisierung

Für die modellbasierte Empfehlungsberechnung bietet sich das Verarbeitungsparadigma der hybriden Empfehlungsberechnung an. Mit ihr können die Modelle inkrementell aus den Feedbackdaten berechnet und aktualisiert werden, um diese mit den Requests zu verbinden und zur Berechnung der Empfehlungen zu nutzen (vgl. Abschnitt 3.2.3). Um dies zu verdeutlichen, betrachten wir das Vorgehen am Beispiel der Matrixfaktorisierung.

Die Methode der Matrixfaktorisierung ordnet die Bewertungen von Benutzerinnen und Benutzern zu Objekten als Matrix $\mathbf{R}$ an, in der die Benutzerinnen und Benutzer auf den Zeilen und die Objekte auf den Spalten abgebildet werden. Diese Matrix wird in zwei Matrizen $\mathbf{P}$ und $\mathbf{Q}$ niedrigen Rangs zerlegt, sodass in $\mathbf{P}$ zu jeder Benutzerin bzw. jeden Benutzer ein sogenannter User-Feature-Vektor $\vec{p}_u$ und in $\mathbf{Q}$ zu jedem

Objekt ein Item-Feature-Vektor $\vec{q}_i$ entsteht. Eine unbekannte Bewertung lässt sich anschließend durch die Multiplikation des entsprechenden User-Feature-Vektors mit dem entsprechenden Item-Feature-Vektor schätzen (vgl. Abschnitt 2.1.3).

Das Vorgehen im modellbasierten Ansatz im Allgemeinen und der Matrixfaktorisierung im Speziellen gliedert sich in zwei Teile: die Berechnung des Modells (Zerlegung von $\mathbf{R}$ in $\mathbf{P}$ und $\mathbf{Q}$) und die Anwendung des Modells (Multiplikation von $\vec{p}_u$ der Benutzerin bzw. des Benutzers aus dem Request jeweils mit den Vektoren $\vec{q}_i$ der Empfehlungskandidaten). Übertragen auf den Queryplan lässt sich der erste Teil mit einer AGGREGATION und der zweite Teil mit einem MAP realisieren (Operator 4 resp. 6 in Abbildung 3.20).

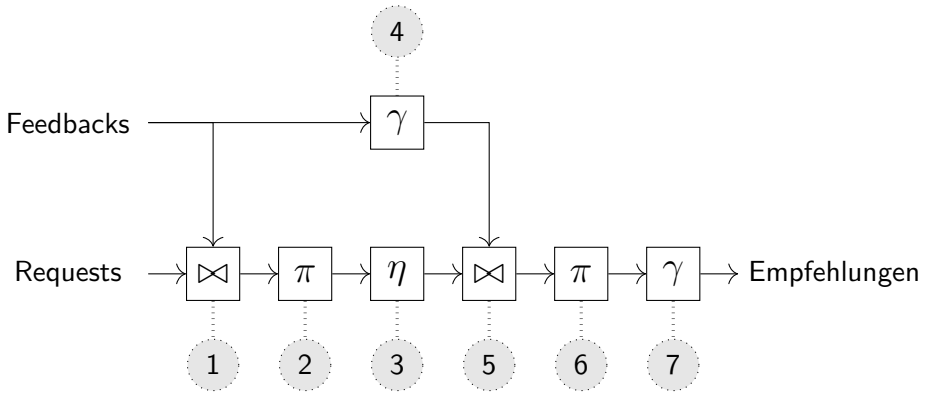


Abbildung 3.20 Queryplan für die modellbasierte Empfehlungsberechnung mit dem hybriden Ansatz

Abbildung 3.20 stellt den Ablauf in Form eines Queryplans dar. Die Berechnung der Empfehlungen besteht aus sieben Operatoren:

- Kandidatenselektion
  1. JOIN $\bowtie$ von Requests $r \in \mathcal{R}$ und Feedbacks $f \in \mathcal{F}$. Hier kann ggf. noch eine Vorfilterung der Kandidaten mithilfe eines Join-Prädikats erfolgen.
  2. PROJECTION π auf die Attribute $\{q.k, q.x_1, f.i\}$, mit $q.x_1$ als ID der Benutzerin bzw. des Benutzers und $q.x_2$ vom Request und $f.i$ als Empfehlungskandidaten.
  3. DISTINCT η zur Entfernung von Duplikaten.

  Ergebnis ist der Datenstrom $\mathcal{C}$ mit den Kandidaten $c \in C$.
- (Lerndatenselektion entfällt, da alle Feedbackdaten als Lerndaten genutzt werden.)
- Kandidatenbewertung
  4. AGGREGATION γ erstellt aus der Menge aller gültigen, im aktuellen Fenster liegenden Feedbackdaten ein Modell. Ändert sich die Menge der gültigen Feedbackdaten wird das Modell neu berechnet oder, wenn

möglich, inkrementell aktualisiert. Die Ausgabe ist ein Datenstrom von Modellinstanzen annotiert mit einem Gültigkeitszeitraum (Zeitintervall), sodass zu jedem Zeitpunkt genau eine Modellinstanz gültig ist. Im Falle der Matrixfaktorisierung erzeugt der Aggregationsoperator einen Ausgabedatenstrom, dessen Elemente die Matrizen $\mathbf{P}$ und $\mathbf{Q}$ enthalten.

5. JOIN $\bowtie$ von Empfehlungskandidaten $c \in \mathcal{C}$ mit dem jeweils temporal passenden Modell.
6. MAP π wendet das Modell an. Im Falle der Matrixfaktorisierung wird aus $\mathbf{P}$ der User-Feature-Vektor $\vec{p}_{x_1}$ für die Benutzerin bzw. den Benutzer $R.x_1$ und aus dem $\mathbf{Q}$ der Item-Feature-Vektor $\vec{q}_i$ für den Kandidaten $F.i$ selektiert und als geschätzte Bewertung als Attribut $\hat{r} = \vec{p}_{x_1}\vec{q}_i$ ausgegeben.

Ergebnis ist der Datenstrom der bewerteten Kandidaten $\mathcal{C}'$.

- Empfehlungsberechnung
  7. AGGREGATION γ mit dem Gruppierungsattribut $\{q\}$ und der Aggregationsfunktion TOP_K($\hat{r}$).

  Ergebnis sind die K am besten bewerteten Kandidaten und somit der Datenstrom der Empfehlungsmengen $\mathcal{E}$.

Für die Integration eines modellbasierten Lernverfahrens als Aggregationsfunktion in den Aggregationsoperator ist ein inkrementelles Verfahren ideal, welches es ermöglicht, dem Modell ein neues Feedbackdatum hinzuzufügen und zu entfernen, ohne dass das Modell von Grund auf neu gelernt werden muss. Für die Matrixfaktorisierung bietet sich eine Adaption des BRISMF-Algorithmus [Tak+09] an. Das übliche, nicht-inkrementelle Vorgehen sieht vor, dass die Matrizen $\mathbf{P}$ und $\mathbf{Q}$ zufällig initialisiert und anschließend durch ein Optimierungsverfahren, wie beispielsweise das Gradientenabstiegsverfahren, optimiert werden. Die grundsätzliche Idee von BRISMF ist es, dass das Modell nicht komplett, sondern nur teilweise neu initialisiert wird (z.B. nur $\mathbf{Q}$). Dies beschleunigt die Optimierung, wenn sich nur wenig in $\mathbf{R}$ geändert hat. Während BRISMF lediglich vorsieht, dieses Verfahren bei neuen Feedbackdaten einzusetzen, wird in diesem Konzept die teilweise Neuinitialisierung des Modells auch eingesetzt, wenn Feedbackdaten nicht mehr relevant sind. Somit kann der BRISMF-Algorithmus auch für Verarbeitungsfenster angewendet werden, bei dem ein Modell das Interesse von den Benutzerinnen und Benutzern in einem bestimmten Zeitfenster widerspiegeln soll.

3.7 Zusammenfassung

In diesem Kapitel haben wir Konzepte für die Umsetzung von Recommender-Systemen betrachtet. Zunächst haben wir den Systemkontext definiert, dessen wichtigster Teil die Anwendung darstellt, von der die Daten zur Empfehlungsberechnung übertragen werden und zu der die berechneten Empfehlungen gesendet werden. Anschließend haben wir drei unterschiedliche Verarbeitungsparadigmen definiert: die requestgetriebene, die feedbackgetriebene und die hybride Empfehlungsberechnung. Dabei stellt die requestgetriebene Empfehlungsberechnung eine einfache Möglichkeit dar, um klassische Recommender-System-Methoden in ein Datenstrommanagementsystem zu integrieren. Um vom inkrementellen Vorgehen von Recommender-Systemen zu profitieren, ist die Umsetzung als feedbackgetriebene Empfehlungsberechnung vorzuziehen. Die hybride Empfehlungsberechnung stellt einen Mittelweg dar, der sich insbesondere für die modellbasierten Recommender-Systeme anbietet.

In Abschnitt 3.3 haben wir Datenmodelle für die Datenströme definiert, die in einem Recommender-System auf Basis eines Datenstrommanagementsystems vorkommen. Diese Modelle sind denormalisiert, sodass die Daten direkt von den verarbeiteten Operatoren genutzt werden können. Das Vorgehen der Recommender-System-Methoden haben wir in Abschnitt 3.4 als Basisalgorithmen eingeführt und anschließend eine Architektur für die Queries darauf aufgebaut (Abschnitt 3.5). Diese Architektur besteht aus sieben Phasen – vom Zugriff der Eingabedaten bis hin zur Ausgabe der Empfehlungen. Zum Schluss haben wir für konkrete Probleme Beispielquerypläne betrachtet, die für die Umsetzung eines Recommender-Systems benötigt werden.

4 Framework

Dieses Kapitel beschreibt, wie ein Datenstrommanagementsystem als Framework für Online-Recommender-Systeme eingesetzt werden kann. Grundlage für das Framework bieten zwei Sprachen, mit denen Queries in Recommender-Systemen definiert werden können, die in Odysseus bereits implementiert sind. Das sind die an funktionalen Konzepten orientierte Querysprache *PQL* (Abschnitt 4.1) sowie die an SQL angelehnte Sprache *CQL* (Abschnitt 4.3). CQL hat gegenüber PQL den Vorteil, dass sie für SQL-erfahrene Personen intuitiver zu verstehen ist. Dafür ist ihre Ausdrucksfähigkeit nicht so mächtig wie bei PQL.

Basierend auf den Konzepten aus dem vorherigen Kapitel werden in den Abschnitten 4.2 resp. 4.4 Templates für die Umsetzung von Recommender-Systemen in PQL resp. CQL vorgestellt. Diese Templates können von Anwenderinnen und Anwendern des Datenstrommanagementsystems übernommen und angepasst werden, um ein individuelles Recommender-System umzusetzen. Für jede der Phasen aus Abschnitt 3.5 werden Queries zur Umsetzung der erforderlichen Datenverarbeitungsschritte definiert. Die Anwenderin bzw. der Anwender kann dabei frei wählen, ob er die PQL- oder die CQL-Templates nutzt oder beide Arten von Templates mischt. In den beiden Abschnitten zu den Templates werden wir sehen, dass sich die erforderlichen Queries weitestgehend mit beiden Sprachen umsetzen lassen. Ausnahmen bilden die Templates, die spezielle Funktionen nutzen, die sich mit der Syntax von CQL nicht umsetzen lassen. Dazu gehört beispielsweise das Anreichern von Daten aus einer Objektdatenbank (in PQL kommt dabei der DBENRICH-Operator zum Einsatz, für den es in der auf SQL basierenden Sprache CQL keine Entsprechung gibt). Im Abschnitt 4.4 (CQL-Templates) wird darauf hingewiesen, wenn sich die in Abschnitt 4.2 vorgestellten PQL-Templates nicht mit einem CQL-Template umsetzen lassen.

Um für Anwenderinnen und Anwender die Konfigurationsmöglichkeiten und somit die variablen Teile der Templates zu verdeutlichen, ist jedem Template ein Block von Konfigurationsparametern vorangestellt.

Die Beschreibungen in diesem Kapitel zeigen den neuartigen Ansatz, die Querysprachen PQL bzw. CQL als Frameworksprache im Allgemeinen und im Speziellen für die Umsetzung von datenstrombasierten Recommender-Systemen einzusetzen. Dazu wird die Templatefunktion von Odysseus genutzt.

Jedes Template besteht aus zwei Abschnitten: Die Angabe von Konfigurationsparametern für das Template sowie das eigentliche Template, welches eine Query oder Teile

Quelltext 4.1 Beispiel für Konfigurationsparameter in den Templates

```
1 #DEFINE predicate user == 1234
2 #DEFINE limit 5
3
4 SELECT * FROM feedbacks WHERE ${predicate} LIMIT ${limit};
```

einer Query definiert. Quelltext 4.1 zeigt ein Beispiel für ein Template, welches mit CQL maximal fünf Feedbacks von der Benutzerin bzw. dem Benutzer mit der ID *1234* selektiert. In Zeile 1 ist der Konfigurationsparameter `predicate` definiert, der in diesem Beispiel auf `user == 1234` gesetzt ist. In Zeile 2 ist der Parameter `limit` definiert, der auf `5` gesetzt ist. Eingeleitet wird ein Konfigurationsparameter mit dem Schlüsselwort `#DEFINE`, gefolgt von dem Parameternamen und dem zu setzenden Wert (eine beliebige Zeichenkette, beginnend mit dem ersten druckbaren Zeichen nach dem Parameternamen und endend mit dem letzten druckbaren Zeichen vor dem Zeilenende). In Zeile 4 werden die Parameter in einem CQL-Statement durch die Angabe `${predicate}` bzw. `${limit}` verwendet. Diese Angaben werden durch die Werte in Zeile 1 resp. 2 ersetzt. Dabei wird die komplette Zeichenkette an dieser Stelle eingesetzt, genauso wie sie in den Zeilen 1 bzw. 2 auf die Parameternamen folgt.

4.1 Die Querysprache PQL

Die Querysprache PQL besteht aus einer Verkettung von Datenstromoperatoren. Die Syntax der Sprache sei an dem Beispiel in Quelltext 4.2 demonstriert. Die Definition einer Operatorinstanz besteht aus Parametern (Key-Value-Paare) und einem oder mehreren Eingabedatenströmen. Um den Ausgabedatenstrom eines Operators als Eingabedatenstrom eines anderen Operators zu nutzen, kann diesem ein Name gegeben werden (`output_stream` in Quelltext 4.2).

Quelltext 4.2 Syntax der PQL-Operatordefinition

```
1 output_stream = OPERATOR_NAME({
2     PARAMETER_1 = value_1,
3     PARAMETER_2 = value_2,
4     ...,
5     PARAMETER_N = value_N,
6   }, input_stream_1, input_stream_2, ..., input_stream_M)
```

Quelltext 4.3 zeigt als Beispiel die Definition eines SELECT-Operators, der aus dem Datenstrom aller Feedbackdaten einen Datenstrom als Ausgabe erzeugt, der nur die Feedbacks der Benutzerin bzw. des Benutzers mit der ID = 1234 enthält.

Quelltext 4.3 Beispiel einer PQL-Operatordefinition (Filter-Operator SELECT)

```
1 feedbacks_user_1234 = SELECT({PREDICATE = "user == 1234"}, ↩
    feedbacks)
```

Um diese Query als Template zu nutzten, zeigt Quelltext 4.4 die Verwendung eines Parameters für die Konfiguration des Prädikats des SELECT-Operators. Dazu wird in Zeile 3 die Zeichenkette `${predicate}` durch die komplette Zeichenkette (inkl. der Anführungszeichen) am Ende der Zeile 1 ersetzt.

Quelltext 4.4 Konfigurationsparameter in PQL

```
1 #DEFINE predicate "user == 1234"
2
3 filtered_feedbacks = SELECT({PREDICATE = ${predicate}}, ↩
    feedbacks)
```

Die Sprache PQL kann um eigene Operatoren erweitert werden. Das Datenstrommanagementsystem *Odysseus* stellt dafür Java-Schnittstellen zur Verfügung. Anhand dieser können die Parameter, die erforderlichen Eingabedatenströme sowie die Verarbeitungslogik angegeben werden.

4.2 Templates für die Umsetzung von Recommender-Systemen in PQL

In diesem Abschnitt werden PQL-Templates für die Umsetzung von Recommender-Systemen vorgestellt. Die Gliederung der Unterabschnitte orientiert sich an den sieben Phasen, die in Abschnitt 3.5 eingeführt wurden.

4.2.1 Datenzugriff

Der Zugriff auf externe Daten erfolgt über ACCESS-Operatoren. Diese erlauben die flexible Anbindung von externen Datenquellen. Eine ACCESS-Operation besteht aus einem Transport-Handler, der die Art der Datenverbindung bestimmt (z. B. *Socket*, *HTTP*, *File*), dem Protokoll-Handler, der das Datenformat festlegt (z. B. *CSV*, *JSON*, *XML*), dem Data-Handler, der den internen Datentyp bestimmt (in dieser Arbeit immer *Tuple*, alternativ auch *Key-Value* möglich) und einem Wrapper, der angibt, ob die Datenquellen aktiv Daten senden (*GenericPush*) oder ob die Daten vom Datenstrommanagementsystem kontinuierlich abgefragt werden müssen (*GenericPull*).

Zugriff auf CSV-Dateien

Zur Offline-Evaluation oder zur Anbindung von statischen Daten ermöglicht das Datenstrommanagementsystem den Zugriff auf CSV-Dateien. Quelltext 4.5 zeigt ein Template zur Simulation eines Feedbackdatenstroms aus einer CSV-Datei. Das Template bietet folgende Konfigurationsmöglichkeiten:

Quelltext 4.5 Template zur Anbindung von Daten aus einer CSV-Datei (Beispiel: Feedbackdaten)

```
1  #DEFINE file "feedbacks.csv"
2  #DEFINE data_stream_name feedbacks
3  #DEFINE schema [["user", "Long"], ["item", "Long"], ["rating",  ↩
       "Double"]]
4  #DEFINE csv_delimiter ";"
5  #DEFINE csv_read_first_line "false"
6
7  ${data_stream_name} = ACCESS({
8      SOURCE = ${data_stream_name},
9      WRAPPER = "GenericPull",
10     TRANSPORT = "File"
11     PROTOCOL = "CSV",
12     DATAHANDLER = "Tuple",
13     OPTIONS = [
14       ["filename", ${file}],
15       ["delimiter", ${csv_delimiter}],
16       ["readfirstline", ${csv_read_first_line}]
17     ],
18     SCHEMA = ${schema}
19   })
```

- Den Dateipfad zur CSV-Datei (Zeile 1). Dieser wird als Option in Zeile 14 verwendet, welche vom File-Transport-Handler ausgelesen wird.

- Den Namen des Datenstroms (Quellenname), welcher aus den Daten der CSV-Datei erzeugt werden soll (Zeile 2). Zusätzlich zur Zuweisung des Ausgabedatenstroms des ACCESS-Operators in Zeile 7 wird dieser Name auch als eindeutiger Identifikator der Quelle genutzt (Zeile 8).
- Das Datenschema der Daten in der CSV-Datei (Zeile 3). Dieses wird dem ACCESS-Operator in Zeile 18 übergeben und vom CSV-Protokoll-Handler zum Parsen der CSV-Zeile sowie vom Tuple-Data-Handler zur Erzeugung der Tupel genutzt. Das Schema besteht aus einer Liste in der Form

  ```
  [..., ..., ...]
  ```

  in der die Attributdefinitionen als Paare in der Form

  ```
  ["Attributname", "Datentyp"]
  ```

  angegeben werden.
- Das Trennzeichen der Spalten in der CSV-Datei (Zeile 4). Dieses wird als Option in Zeile 15 genutzt und vom CSV-Protokoll-Handler gelesen.
- Ein Flag, ob die erste Zeile mitgelesen werden soll oder ob sich in der ersten Zeile die Spaltennamen befinden und diese Zeile somit ignoriert werden soll (Zeile 5). Dieses wird als Option in Zeile 16 genutzt und vom CSV-Protokoll-Handler gelesen.

Fest definiert sind der Wrapper *GenericPull* in Zeile 9, der in einer Schleife den Transport-Handler nach neuen Daten fragt, bis dieser meldet, dass alle Daten gelesen wurden, den Transport-Handler *File* in Zeile 10, der die Datei zeilenweise einliest, den Protokoll-Handler *CSV*, der die Zeilen als CSV parst, und der Data-Handler *Tuple* in Zeile 12, der aus den geparsten Daten Tupel-Objekte erzeugt.

Bereitstellung einer HTTP-Schnittstelle

Wenn Datenobjekte von einer aktiven Quelle per HTTP-Requests an das Datenstrommanagementsystem übertragen werden sollen, so kann wie im Template in Quelltext 4.6 der Transport-Handler *HttpServer* genutzt werden (Zeile 10). Das Beispiel in Quelltext 4.6 startet einen HTTP-Server, der unter dem Ressourcennamen `/requests` (Zeile 3) auf Port 8080 (Zeile 2) unter Nutzung des Hostnamens `api.example.com` (Zeile 1) JSON-Objekte entgegennimmt. Anhand der Schemadefinition in Zeile 4 werden die Werte aus dem JSON-Objekt extrahiert: Der Attributname wird als JSON-Pfad interpretiert. In Zeile 5 ist der Name des Datenstroms definiert, der vom ACCESS-Operator erzeugt wird.

Quelltext 4.6 Template zur Bereitstellung einer HTTP-Schnittstelle für das Entgegennehmen von Daten im JSON-Format (Beispiel: Requests)

```
1  #DEFINE hostname "api.example.com"
2  #DEFINE port 8080
3  #DEFINE path "/requests"
4  #DEFINE schema [["user.id", "Long"], ["context.location.city", "String"]]
5  #DEFINE data_stream_name requests
6
7  ${data_stream_name} = ACCESS({
8      SOURCE = ${data_stream_name},
9      WRAPPER = "GenericPush",
10     TRANSPORT = "HttpServer"
11     PROTOCOL = "JSON",
12     DATAHANDLER = "Tuple",
13     OPTIONS = [
14       ["hostname", ${hostname}],
15       ["port", ${port}],
16       ["path", ${path}],
17       ["content_type", "application/json"]
18     ],
19     SCHEMA = ${schema}
20   })
```

Abfragen einer HTTP-Schnittstelle

Häufig bieten bereits vorhandene Legacy-Systeme, in denen sich das Recommender-System einbetten soll, HTTP-Schnittstellen an, von denen Daten abgerufen werden können. Quelltext 4.7 zeigt ein Template, welches Daten von einer HTTP-Schnittstelle abruft und diese als Datenstrom zur Verfügung stellt. Dazu muss die URI der HTTP-Schnittstelle (Zeile 1) sowie die HTTP-Methode (Zeile 2; bspw. `GET`, `POST`, `PUT`, ...) angegeben werden. Wie im vorherigen Template werden Daten als JSON-Objekte übertragen und die Schemadefinition (Zeile 4) gibt den JSON-Pfad der Attributwerte und die Datentypen an. In Zeile 4 wird der Datenstromname definiert.

4.2.2 Datenvorverarbeitung

Die benötigten Operationen in der Phase der Datenvorverarbeitung sind stark davon abhängig, in welcher Form die Daten von den Legacy-Systemen zur Verfügung gestellt werden. Die Templates in diesem Abschnitt können somit nur eine Inspiration für mögliche Vorverarbeitungsschritte geben und das Ziel verfolgen, die wichtigsten Operationen abzudecken. Betrachtet werden:

Quelltext 4.7 Template zum Abrufen von Daten von einer HTTP-Schnittstelle (Beispiel: Feedbacks)

```
1  #DEFINE uri "http://api.example.com/feedbacks"
2  #DEFINE method "GET"
3  #DEFINE schema [["item.id", "Long"], ["item.text", "String"],
       ["user.id", "Long"], ["context.location.city", "String"]]
4  #DEFINE data_stream_name requests
5
6  ${data_stream_name} = ACCESS({
7      SOURCE = ${data_stream_name},
8      WRAPPER = "GenericPull",
9      TRANSPORT = "HTTP"
10     PROTOCOL = "JSON",
11     DATAHANDLER = "Tuple",
12     OPTIONS = [
13       ["uri", ${uri}],
14       ["method", ${method}],
15     ],
16     SCHEMA = ${schema}
17   })
```

- Das Umbenennen von Datenattributen, damit die Templates der nachfolgenden Phasen die Attribute unter den Namen wiederfinden, wie sie in den Templates definiert sind.
- Das Transformieren von Daten, bspw. um diese zu normalisieren.
- Das Herausfiltern von nicht benötigten Daten.
- Das Verknüpfen von Daten aus unterschiedlichen Datenquellen, bspw. um Datenelemente mit Bestandsdaten anzureichern.

Umbenennen von Datenattributen

Die Umbenennung von Datenattributen ist streng genommen nichts anderes als eine Transformation der Datenelemente. Somit verbirgt sich hinter dieser Operation ein MAP-Operator, wie er in Quelltext 4.8 dargestellt ist. In diesem Beispiel werden die Attribute des Requestdatenstroms, wie sie von dem Beispiel in Quelltext 4.6 erzeugt werden, umbenannt. Dieser Schritt ist erforderlich, da in Quelltext 4.6 die verwendeten Attributnamen den JSON-Pfad-Ausdrücken entsprechen. Die Umbenennung erfolgt anhand der Angabe in Zeile 1, welche aus einer Liste von Paaren der Form `["Alter_Name", "Neuer_Name"]` besteht.

Quelltext 4.8 Template zur Umbenennung von Attributen mit dem MAP-Operator (Beispiel: Requests)

```
1 #DEFINE mappings [["user.id", "user"],  ↩
      ["context.location.city", "city"]]
2 #DEFINE input_data_stream_name requests_with_wrong_names
3 #DEFINE output_data_stream_name requests
4
5 ${output_data_stream_name} = MAP({
6     EXPRESSIONS = ${mappings}
7   }, ${input_data_stream_name})
```

Der MAP-Operator kann nicht nur zum Umbenennen von Attributen, sondern auch zum Transformieren von Attributwerten genutzt werden. Anstelle des neuen Attributnamens kann ein beliebiger Ausdruck angeben werden. Um beispielsweise aus einer numerischen Benutzer-ID eine Zeichenkette der Form „User“ + Benutzer-ID zu erzeugen, kann der Ausdruck `"'User ' + ToString(user.id)"` eingesetzt werden. In diesem Fall wird deutlich, dass der MAP-Operator in Wirklichkeit eine Transformation der Eingabedaten durchführt.

Neben der Möglichkeit, den MAP-Operator für die Umbenennung einzusetzen, gibt es auch noch einen speziellen Operator mit dem Namen RENAME. Die Verwendung dieses Operators hat den Vorteil, dass die Query-Ausführungsumgebung weiß, dass hier lediglich eine Umbenennung stattfinden soll. Mit dieser Zusatzinformation ist diese in der Lage, die Ausführung der Umbenennung zu optimieren: Die Ausführungsumgebung verändert lediglich das Schema, ohne dass die Datenelemente selber transformiert werden müssen.

Die Verwendung des RENAME-Operators ist im Template in Quelltext 4.9 dargestellt. Die neuen Attributnamen werden in Zeile 1 in der Reihenfolge des Vorkommens angegeben.

Quelltext 4.9 Template zur Umbenennung von Attributen mit dem RENAME-Operator (Beispiel: Requests)

```
1 #DEFINE new_names ["user", "city"]
2 #DEFINE input_data_stream_name requests_with_wrong_names
3 #DEFINE output_data_stream_name requests
4
5 ${output_data_stream_name} = RENAME({
6     ALIASES = ${new_names}
7   }, ${input_data_stream_name})
```

Transformieren von Daten

Abhängig davon, wie die Legacy-Systeme (z. B. die Anwendung, in der die Empfehlungsobjekte angezeigt werden) die Daten an das Datenstrommanagementsystem senden, sind verschiedene Transformationsschritte nötig, damit die Recommender-System-Implementierung die Daten verarbeiten kann. Als Beispiel sei hier die Diskretisierung von Daten angeführt (Quelltext 4.10). Gegeben sei ein Feedbackdatenstrom, der als Kontextattribut `temperature` einen Temperaturwert enthält. In diesem einfachen Beispiel soll der Wert in die beiden Fälle `low` (kleiner als 10 °C) und `high` (größer oder gleich 10 °C) diskretisiert werden. Dazu wird der MAP-Operator eingesetzt. Als Parameter sind in Zeile 1 die Transformationen definiert – in diesem Beispiel die Transformation der Temperatur mit der Funktion `ifthenelse`, welche drei Parameter übergeben bekommt: als ersten Parameter die Bedingung, als zweiten Parameter den Wert, der gesetzt werden soll, wenn die Bedingung wahr ist und als dritten Parameter den Wert, der gesetzt werden soll, wenn die Bedingung falsch ist.

Der MAP-Operator gibt normalerweise nur die neuen Attribute aus, die ihm als `EXPRESSIONS` (Zeile 6) übergeben werden. Um auch alle eingehenden Attribute zu behalten, setzen wir den Parameter `KEEPATTRIBUTES` auf `true` (Zeile 7). Somit wird in dem Beispiel das Attribut `temperature` durch den neuen, diskretisierten Wert ersetzt.

Quelltext 4.10 Template zur Transformation von Daten (Beispiel: Diskretisierung von Kontextdaten)

```
1 #DEFINE transformations [["ifthenelse(temperature < 10, 'low',
    'high')", "temperature"]]
2 #DEFINE input_data_stream_name feedbacks_before_transformation
3 #DEFINE output_data_stream_name feedbacks
4
5 ${output_data_stream_name} = MAP({
6     EXPRESSIONS = ${transformations},
7     KEEPATTRIBUTES = "true"
8   }, ${input_data_stream_name})
```

Filtern von Daten

Gelegentlich müssen Eingabedaten bereinigt werden. Als Beispiel sei ein Feedbackdatenstrom gegeben, der gelegentlich fehlerhafte Feedbacks sendet, die keine Objekt-

IDs beinhalten. Das Beispiel in Quelltext 4.11 entfert mithilfe des SELECT-Operators alle Datenstromelemente, bei denen das Attribut `item` mit der Objekt-ID *„null“* ist. Dazu enthält das Template in Zeile 1 als Konfigurationsparameter das Prädikat für die Elemente, die im Datenstrom erhalten bleiben sollen.

Quelltext 4.11 Template zum Filtern von Daten (Beispiel: Herausfiltern von Feedbacks ohne Objekt-ID)

```
1 #DEFINE predicate "!isNull(item)"
2 #DEFINE input_data_stream_name unfiltered_feedbacks
3 #DEFINE output_data_stream_name feedbacks
4
5 ${output_data_stream_name} = SELECT({
6     PREDICATE = ${predicate}
7   }, ${input_data_stream_name})
```

Anreicherung von Datenströmen mit Daten aus Datenbanken

Wenn Datenströme mit Daten aus Datenbanken angereichert werden sollen, kann dies mit dem DBENRICH-Operator erfolgen. Quelltext 4.12 zeigt ein Template, in dem Feedbackdaten um weitere Objektinformationen aus einer Objektdatenbank angereichert werden. Dazu wird der Operator DBENRICH genutzt. Dieser Operator ist logisch gesehen ein JOIN-Operator kombiniert mit einem ACCESS-Operator, der aus einer Datenbanktabelle einen Datenstrom erzeugt. Der Vorteil der gesonderten Implementierung des DBENRICH-Operators ist, dass er nicht die gesamte Datenbanktabelle einlesen und im Speicher halten muss, sondern sich die vorhandenen Zugriffsstrukturen des Datenbanksystems zunutze machen kann, und somit nur die Daten abfragen muss, die benötigt werden.

Die Verbindung zur Datenbank wird über eine JDBC-URL konfiguriert (Zeile 1). In dem Beispiel wird die Datenbank mit dem Namen `database` eines lokalen PostgreSQL-Datenbanksystems genutzt. Für die Verbindung müssen Benutzername und Passwort (Zeilen 2 und 3) konfiguriert werden. Die Zuordnung der Objektinformationen erfolgt über eine Datenbankquery (Zeile 4), in deren Prädikat sowohl die Attribute des Feedbackdatenstroms als auch die der Datenbankrelation der Objekte genutzt werden können. In dem Beispiel findet die Zuordnung über die Objekt-ID (Primär-/Fremdschlüssel) statt. In der Query werden die Attribute `category` und `author` der Objekte ausgewählt, die vom DBENRICH-Operator mit den Attributen des Feedbackdatenstroms verknüpft werden.

Quelltext 4.12 Template zur Anreicherung von Objektinformationen aus einer Objektdatenbank zu Feedbackdaten

```
1  #DEFINE jdbc "jdbc:postgresql://localhost:5432/database"
2  #DEFINE db_user_name "dbuser"
3  #DEFINE db_password "dbpassword"
4  #DEFINE itemdata_db_query "SELECT items.category, items.author
       FROM items WHERE feedbacks.item == items.id;"
5  #DEFINE feedbacks_in_data_stream_name feedbacks_without_item_data
6  #DEFINE feedbacks_out_data_stream_name feedbacks
7
8  ${feedbacks_out_data_stream_name} = DBENRICH({
9      JDBC = ${jdbc},
10     USER = ${db_user_name},
11     PASSWORD = ${db_passord},
12     QUERY = ${itemdata_db_query}
13   }, ${feedbacks_in_data_stream_name})
```

Ableitung impliziter Requests

Die Ableitung impliziter Requests, wie sie in Abschnitt 3.6.2 beschrieben ist, kann mit dem Template in Quelltext 4.13 erfolgen. Dazu werden in Zeile 1 die Kontextattribute der Feedbacks angegeben, aus denen der implizite Request erstellt werden soll. In diesem Beispiel sind das die Kontextattribute `city` und `time_of_day` als Stadt, in der sich die Benutzerin bzw. der Benutzer aufhält, sowie die Tageszeit (morgens, mittags, abends) als diskretisierte Zeitangabe. Mit dieser Ableitung kann für jeden Ort und zu jeder Tageszeit eine angepasste Empfehlung berechnet werden.

Neben den Kontextattributen benötigt das Template den Namen des Feedbackdatenstroms, aus dem die Requests abgeleitet werden sollen, sowie den Namen des Requestdatenstroms, der von dem Template als Ausgabe erzeugt werden soll.

Die Umsetzung erfolgt analog zu der Beschreibung in Abschnitt 3.6.2 mit dem PROJECT-Operator, der die Kontextattribute für die Requests herausfiltert und dem DISTINCT-Operator, der Duplikate entfernt.

Quelltext 4.13 Template zur Ableitung impliziter Requests

```
1 #DEFINE context_attributes ["city", "time_of_day"]
2 #DEFINE feedbacks_data_stream_name feedbacks
3 #DEFINE impl_requests_data_stream_name implrequests
4
5 only_context_attributes = PROJECT({
6     ATTRIBUTES = ${context_attributes}
7   }, ${feedbacks_data_stream_name})
8 ${impl_requests_data_stream_name} = ↩
    DISTINCT(only_context_attributes)
```

4.2.3 Kandidatenselektion

In der Phase der Kandidatenselektion werden zu einem (expliziten oder impliziten) Request Objekte verknüpft, die ein bestimmtes Prädikat erfüllen. Diese Objekte sind die Kandidaten. Dazu muss die Kandidatenselektionsfunktion f_C (vgl. Gleichung 3.1) umgesetzt werden. Eine mögliche Quelle für Kandidaten ist eine Objektdatenbank. Das Template in Quelltext 4.14 reichert den Requestdatenstrom mit Objekten aus einer relationalen Datenbank an. Dazu wird erneut der DBENRICH-Operator analog zu dem Template in Quelltext 4.12 eingesetzt. In dem Beispiel werden durch die Datenbankquery in Zeile 4 alle Objekte ausgewählt, die einen Bezug zu der Stadt haben, in der der Request abgesetzt wurde. Das entspricht der folgenden Kandidatenselektionsfunktion für alle Objekte I aus einer Objektdatenbank:

$$f_C : (q, I) \mapsto \{c \in I \mid c.y_1 = q.x_1\} \tag{4.1}$$

Dabei steht das Attribut x_1 der Query q für die Stadt, in der sich die Benutzerin bzw. der Benutzer derzeit aufhält und das Objektattribut y_1 für die Stadt, die dem Objekt zugeordnet ist (z. B. den Ort des Geschehens bei einem Nachrichtenartikel).

Um die Effizienz bei der Datenanreicherung mit dem DBENRICH-Operator zu steigern, hält der Operator eine dauerhafte Verbindung zur Datenbank und speichert für einen schnelleren Zugriff häufig angefragte Daten in einem Cache.

Quelltext 4.14 Template zur Kandidatenselektion aus einer Objektdatenbank

```
1 #DEFINE jdbc "jdbc:postgresql://localhost:5432/database"
2 #DEFINE db_user_name "dbuser"
3 #DEFINE db_password "dbpassword"
4 #DEFINE candidates_db_query "SELECT * FROM items WHERE ↩
    requests.city == items.city;"
5 #DEFINE requests_data_stream_name requests
6 #DEFINE candidates_data_stream_name candidates
```

```
7
8  ${candidates_data_stream_name} = DBENRICH({
9      JDBC = ${jdbc},
10     USER = ${db_user_name},
11     PASSWORD = ${db_passord},
12     QUERY = ${candidates_db_query}
13   }, ${requests_data_stream_name})
```

Ein Problem mit der Anreicherung von Objekten aus einer Objektdatenbank kann entstehen, wenn zu einem Objekt keine Feedbackdaten vorhanden sind, die Recommender-System-Methode diese aber zur Bewertung eines Kandidatens benötigt. Das ist beispielsweise der Fall, wenn in der Objektdatenbank auch alte, von Benutzerinnen und Benutzern nicht mehr genutzte Objekte gespeichert sind, das Recommender-System aber nur Feedbacks der letzten 24 Stunden berücksichtigt.

Dieses Problem kann von vornherein vermieden werden, wenn die Kandidaten direkt aus dem Feedbackdatenstrom gewonnen werden. Dazu kann das Template in Quelltext 4.15 genutzt werden. Es setzt die Kandidatenselektion analog zu der Definition in Gleichung 3.26 (Seite 66) bzw. Abbildung 3.17 (Seite 67) um. Das Template besteht aus einem JOIN-Operator, der die Requests mit den Feedbacks verknüpft, einem PROJECT-Operator, der von den Feedbackdaten nur die Objektinformationen übrig lässt, und einem DISTINCT-Operator, der die Duplikate entfernt.

Quelltext 4.15 Template zur Kandidatenselektion aus dem Feedbackdatenstrom

```
1  #DEFINE candidates_predicate "requests.city == feedbacks.city"
2  #DEFINE candidates_attributes ["requests.id", "requests.city",
       "feedbacks.item", "feedbacks.item_author"]
3  #DEFINE requests_data_stream_name requests
4  #DEFINE feedbacks_data_stream_name feedbacks
5  #DEFINE candidates_data_stream_name candidates
6
7  requests_with_feedbacks = JOIN({
8      PREDICATE = ${candidates_predicate}
9    }, ${requests_data_stream_name}, ${feedbacks_data_stream_name})
10 candidates_with_duplicates = PROJECT({
11     ATTRIBUTES = ${candidates_attributes}
12   }, requests_with_feedbacks)
13 ${candidates_data_stream_name} =
       DISTINCT(candidates_with_duplicates)
```

In Zeile 1 wird das Prädikat konfiguriert, welches die Zuordnung von Requests und Feedbacks definiert. In dem Beispiel sollen alle Objekte als Kandidaten genutzt wer-

den, zu denen es mindestens ein Feedback gibt, welches in der selben Stadt gegeben wurde, aus der der Request kommt. In Zeile 2 werden die Attribute definiert, die für den Kandidatenstrom übrig sein sollen. Das sind analog zu Gleichung 3.26 die Attribute $\{r.k, r.X, f.i, f.Y\}$, also die Request-ID, die Requestkontextdaten, die Item-ID aus dem Feedbackdatenstrom und ggf. die Objektattribute aus dem Feedbackdatenstrom.

4.2.4 Lerndatenselektion

Bei der Lerndatenselektion muss zwischen dem speicherbasierten und modellbasierten Vorgehen unterschieden werden. Bei speicherbasierten Methoden wird zu jedem *Kandidaten* die Menge der Lerndaten bestimmt (vgl. Gleichung 3.2), bei modellbasierten Methoden wird zu jedem *Request* die Menge der Lerndaten bestimmt (vgl. Gleichung 3.9).

Das Template in Quelltext 4.16 zeigt ein Beispiel für ein speicherbasiertes Vorgehen, welches die Lerndaten aus dem Feedbackdatenstrom gewinnt. Es verbindet mithilfe eines JOIN-Operators die Elemente des Feedbackdatenstroms, die das Prädikat in Zeile 1 erfüllen, mit jeweils einem Kandidaten.

Quelltext 4.16 Template zur Lerndatenselektion aus dem Feedbackdatenstrom (speicherbasiert)

```
1 #DEFINE learning_data_predicate "candidates.item ==
    feedbacks.item AND candidates.city == feedbacks.city"
2 #DEFINE candidates_data_stream_name candidates
3 #DEFINE feedbacks_data_stream_name feedbacks
4 #DEFINE learning_data_data_stream_name learning_data
5
6 ${learning_data_data_stream_name} = JOIN({
7     PREDICATE = ${learning_data_predicate}
8   }, ${candidates_data_stream_name},
    ${feedbacks_data_stream_name})
```

Die Auswahl der Lerndaten erfolgt in diesem Beispiel über die Kontextinformation `city`, die, analog zu Gleichung 3.6, die Feedbackdaten als Lerndaten selektiert, die zu dem Kandidaten (`candidates.item == feedbacks.item`) und in der selben Stadt abgegeben wurde, aus der der Request kommt (`candidates.city == feedbacks.city`). Das Attribut `item` aus den `candidates`-Datenstrom bezieht sich in diesem Beispiel auf die Objekt-ID des Kandidaten, das Attribut `city` aus dem `candidates`-Datenstrom auf das Kontextattribut des Requests.

Die Selektion der Lerndaten im modellbasierten Fall geschieht analog (Template in Quelltext 4.17). Der Unterschied liegt in den eingehenden Datenströmen. Während bei speicherbasierten Methoden der JOIN-Operator zu den Kandidaten die Feedbacks als Lerndaten zuordnet, ordnet der JOIN-Operator bei modellbasierten Methoden die Feedbackdaten den (impliziten) Requests zu (Zeile 2 in Quelltext 4.16 vs. Zeile 2 in Quelltext 4.17). Das Beispiel setzt die Funktion in Gleichung 3.11 um, die zu jedem Request die Feedbacks aus der selben Stadt als Lerndaten zuordnet. Somit kann die Lerndatenselektion als Contextual Pre-Filtering eingesetzt werden.

Quelltext 4.17 Template zur Lerndatenselektion aus dem Feedbackdatenstrom (modellbasiert)

```
1 #DEFINE learning_data_predicate "implrequests.city ==  ↩
    feedbacks.city"
2 #DEFINE requests_data_stream_name implrequests
3 #DEFINE feedbacks_data_stream_name feedbacks
4 #DEFINE learning_data_data_stream_name learning_data
5
6 ${learning_data_data_stream_name} = JOIN({
7     PREDICATE = ${learning_data_predicate}
8   }, ${requests_data_stream_name}, ${feedbacks_data_stream_name})
```

4.2.5 Kandidatenbewertung

Bei der Kandidatenbewertung wird zu jedem Kandidaten c eine Bewertung $\hat{r}$ bestimmt. Die Umsetzung der verschiedenen Methoden zur Bewertung der Kandidaten stützt sich in dieser Arbeit auf die Nutzung einer Aggregationsoperation über

(a) alle Lerndaten eines Kandidatens (speicherbasierte Methode) oder

(b) alle Lerndaten eines (impliziten) Requests (modellbasierte Methode).

Die Unterscheidung liegt darin begründet, dass bei speicherbasierten Methoden direkt zu jedem Kandidaten durch Iteration über die Lerndaten eine Bewertung für diesen Kandidaten berechnet wird, während bei modellbasierten Methoden zunächst ein Modell gelernt wird, welches für verschiedene Kandidaten eine Bewertung schätzen kann. Für modellbasierte Methoden besteht die Kandidatenbewertung somit aus zwei Teilen: das Lernen des Modells sowie die Anwendung des Modells (das eigentliche Bewerten der Kandidaten). Das Lernen des Modells wird analog zur speicherbasierten Methode mit einem Aggregationsoperator umgesetzt, der aus der Menge der Lerndaten ein Modell lernt. Für die eigentliche Bewertung der Kandidaten wird anschließend ein MAP-Operator eingesetzt, der das Modell anwendet.

Die folgenden Templates setzen vor der Aggregation ein Zeitfenster ein, um die Lerndatenmenge, die für die Bewertung der Kandidaten herangezogen wird, auf die Menge der Lerndaten der jüngsten Vergangenheit zu begrenzen (beispielsweise die Lerndaten der letzten 24 Stunden oder der letzten 10 Minuten). Auf dieses Zeitfenster kann verzichtet werden, sodass grundsätzlich alle Lerndaten genutzt werden. Um zu verhindern, dass die Lerndatenmenge unendlich wächst und um den Fokus auf die jüngeren Daten zu legen und damit auf Concept Drifts reagieren zu können, ist es in der Regel bei der Datenstromverarbeitung gewollt, die Gültigkeitsdauer der Daten zu begrenzen. Mit dem Konzept dieser Arbeit liegt die Entscheidung bei der Anwenderin bzw. dem Anwender des Systems, ob der Fensteroperator genutzt werden soll oder nicht.

Das Template in Quelltext 4.18 zeigt ein Beispiel für die Empfehlung der am häufigsten genutzten Objekte mit der speicherbasierten Methode, wie sie in Abschnitt 3.4.1 beschrieben wurde. Als Konfigurationsparameter werden die Fenstergröße für die Gültigkeitsdauer der Lerndaten (Zeile 1), der Name des Datenstroms mit den Lerndaten (Zeile 2), das Attribut des Lerndatenstroms, welches die Request-ID (Zeile 3) sowie die Objekt-ID des Kandidaten enthält (Zeile 4), der Name des Ausgabedatenstroms mit den bewerteten Kandidaten (Zeile 5) und der Name des Attributs mit der Bewertung im Ausgabedatenstrom (Zeile 6) angegeben. Der TIMEWINDOW-Operator annotiert zunächst die Lerndaten mit Gültigkeitsintervallen entsprechend der Konfiguration in Zeile 1 bzw. Zeile 9.

Der mit Gültigkeitsintervallen annotierte Datenstrom von Lerndaten wird an den AGGREGATION-Operator übergeben. Dieser zählt (Aggregationsfunktion count, Zeile 14) die Anzahl der Tupel (Feedback- bzw. Lerndaten) *je Kandidat und Request* (durch die Gruppierung in den Zeilen 18 und 19). Die Ausgabe des Aggregationsoperators besteht aus der Request-ID, der Objekt-ID des Kandidaten (Attribute, welche in Zeile 3 resp. 4 definiert und in Zeile 18 resp. 19 als Gruppierungsattribut genutzt werden) und dem Ergebnis der Aggregationsfunktion count, welches die Anzahl der Lerndaten für den Kandidaten innerhalb des Zeitfensters widerspiegelt, als Attribut mit dem Namen, der in Zeile 6 konfiguriert und in Zeile 15 dem Aggregationsoperator übergeben wird.

Dieses Template kann auch dazu genutzt werden, um den Ansatz aus Abschnitt 3.6.3 (siehe S. 68; *Feedbackgetriebene Empfehlungsberechnung bei disjunkter Kandidaten- bzw. Lerndatenmengen*) umzusetzen. In diesem Fall existiert zwar keine Request-ID im Datenstrom, dafür gibt es aber andere Attribute, die den (impliziten) Request eindeutig identifizieren. Quelltext 4.19 zeigt ein Beispiel, wie die Templateparameter gesetzt werden müssen. In diesem Beispiel sollen die am häufigsten genutzen

Quelltext 4.18 Template zur Kandidatenbewertung mit der speicherbasierten Methode, die die am häufigsten genutzten Objekte empfiehlt.

```
1  #DEFINE window_size [10, "MINUTES"]
2  #DEFINE learning_data_data_stream_name learning_data
3  #DEFINE request_id_attribute_name "request"
4  #DEFINE candidate_item_id_attribute_name "item"
5  #DEFINE rated_data_stream_name rated
6  #DEFINE est_rating_attribute_name "rating"
7
8  windowed_learning_data = TIMEWINDOW({
9      SIZE = ${window_size}
10   }, ${learning_data_data_stream_name})
11
12 ${rated_data_stream_name} = AGGREGATION({
13     AGGREGATIONS = [[
14       FUNCTION = "count",
15       OUTPUT_ATTRIBUTE = ${est_rating_attribute_name}
16     ]],
17     GROUP_BY = [
18       ${request_id_attribute_name},
19       ${candidate_item_id_attribute_name}
20     ]
21   }, windowed_learning_data)
```

Objekte empfohlen werden, die in der Stadt ($f.x_1$) genutzt wurden, in der sich die Benutzerin bzw. der Benutzer des späteren Requests befindet. Außerdem sollen nur die Feedbacks berücksichtigt werden, die zu der Kategorie ($f.x_2$) gehören, die die Benutzerin bzw. der Benutzer im späteren Request angegeben haben wird. Als Lerndatenstrom wird nun direkt der Feedbackdatenstrom genutzt (Zeile 2). Die beiden Attribute *Stadt* ($f.x_1$) und *Kategorie* ($f.x_2$) werden als Request-ID gesetzt (Zeile 3). Zusammen mit der Objekt-ID (Zeile 4) bilden diese die Gruppierungsattribute für den Aggregationsoperator (vgl. Quelltext 4.18). Das Ergebnis ist ein Datenstrom mit bewerteten Kandidaten je Stadt / Kategorie, die nach der Empfehlungsberechnung mit den tatsächlichen Requests per Join verknüpft werden können (vgl. auch Abbildung 3.19 auf Seite 69).

Das Template in Quelltext 4.20 zeigt ein Beispiel für die Berechnung der Bewertung von Kandidaten anhand der durchschnittlichen Bewertung des Kandidaten. Dazu werden analog zu dem vorherigen Template die Fenstergröße für die Lerndaten, die Namen der Datenströme und der Attribute konfiguriert (Zeilen 1-7). Gegenüber dem vorherigen Template ist die Angabe des Attributnamens der Bewertung des eingehenden Lerndatenstroms hinzugekommen (Zeile 5). Diese Information wird in Zeile 16 als Konfiguration für die Aggregationsfunktion avg (Zeile 15) genutzt und bestimmt das Attribut, für das der Durchschnittswert (arithmetisches Mittel)

Quelltext 4.19 Template-Variante zur Kandidatenbewertung bei disjunkter Kandidaten- und Lerndatenmenge mit der speicherbasierten Methode, die die am häufigsten genutzten Objekte empfiehlt.

```
1  #DEFINE window_size [10, "MINUTES"]
2  #DEFINE learning_data_data_stream_name feedbacks
3  #DEFINE request_id_attribute_name "feedbacks.city",  ↩
       "feedbacks.category"
4  #DEFINE candidate_item_id_attribute_name "item"
5  #DEFINE rated_data_stream_name rated
6  #DEFINE est_rating_attribute_name "rating"
7
8
9  windowed_learning_data = TIMEWINDOW({
10     SIZE = ${window_size}
11   }, ${learning_data_data_stream_name})
12
13 ${rated_data_stream_name} = AGGREGATION({
14     AGGREGATIONS = [[
15       FUNCTION = "count",
16       OUTPUT_ATTRIBUTE = ${est_rating_attribute_name}
17     ]],
18     GROUP_BY = [
19       ${request_id_attribute_name},
20       ${candidate_item_id_attribute_name}
21     ]
22   }, windowed_learning_data)
```

berechnet werden soll. Das Ergebnis ist somit ein Datenstrom mit Elementen, die die Request-ID, Objekt-ID des Kandidaten sowie die durchschnittliche Bewertung des Objekts in dem Zeitfenster enthalten.

Der Einsatz einer modellbasierten Methode wird anhand des Templates in Quelltext 4.21 verdeutlicht. Dieses Template besteht aus drei Teilen:

- In den Zeilen 1 bis 14 werden die Konfigurationsparameter des Templates angegeben.
- In den Zeilen 16 bis 27 wird das Modell gelernt.
- In den Zeilen 29 bis 40 werden die Kandidaten unter Anwendung des Modells bewertet.

Für das Lernen des Modells werden zunächst mithilfe des TIMEWINDOW-Operators den Lerndaten Gültigkeitsintervalle zugeordnet. Somit repräsentieren die gelernten Modelle jeweils die Lerndaten in den jeweiligen Zeitfenstern. Die Größe des Zeitfensters wird im Konfigurationsparameter in Zeile1 angegeben.

Quelltext 4.20 Template zur Kandidatenbewertung mit der speicherbasierten Methode, die die am besten bewerteten Objekte empfiehlt.

```
1  #DEFINE window_size [10, "MINUTES"]
2  #DEFINE learning_data_data_stream_name learning_data
3  #DEFINE request_id_attribute_name "request"
4  #DEFINE candidate_item_id_attribute_name "item"
5  #DEFINE rating_attribute_name "rating"
6  #DEFINE rated_data_stream_name rated
7  #DEFINE est_rating_attribute_name "rating"
8
9  windowed_learning_data = TIMEWINDOW({
10     SIZE = ${window_size}
11   }, ${learning_data_data_stream_name})
12
13 ${rated_data_stream_name} = AGGREGATION({
14     AGGREGATIONS = [[
15       FUNCTION = "avg",
16       INPUT_ATTRIBUTE = ${rating_attribute_name},
17       OUTPUT_ATTRIBUTE = ${est_rating_attribute_name}
18     ]],
19     GROUP_BY = [
20       ${request_id_attribute_name},
21       ${candidate_item_id_attribute_name}
22     ]
23   }, windowed_learning_data)
```

Das Lernen des Modells ist mit einem Aggregationsoperator umgesetzt. In diesem Beispiel wird die Methode der Matrixfaktorisierung eingesetzt. Dazu wird die Aggregationsfunktion MatrixFactorization genutzt. Diese Funktion wurde eigens für den Einsatz des Datenstrommanagementsystems als Recommender-System entwickelt und implementiert eine inkrementelle Matrixfaktorisierungsmethode basierend auf dem BRISMF-Algorithmus [Tak+09]. Als Konfigurationsparameter erhält diese Aggregationsfunktion die Attribute, die die ID der Benutzerin bzw. des Benutzers, die die ID des Objektes und die die Bewertungen in den Lerndaten enthalten. Diese werden in den Zeilen 4, 5 und 6 konfiguriert.

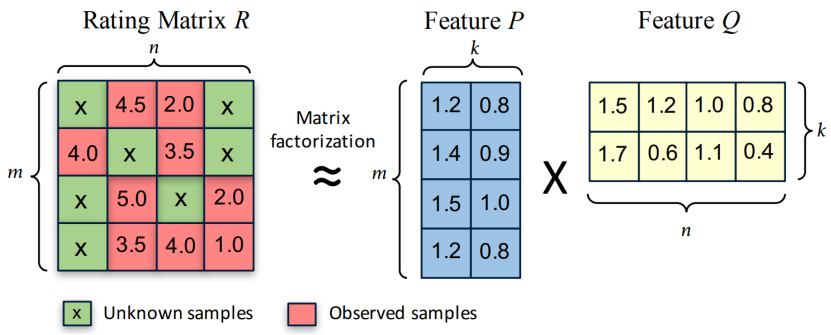


Abbildung 4.1 Matrixfaktorisierung (Quelle: [Xie+16])

Das Ergebnis des Aggregationsoperators sind Tupel mit den beiden Attributen `user_feature_matrix` und `item_feature_matrix`, die die Matrizen P resp. Q beinhalten (vgl. Abbildung 4.1 und Abschnitt 2.1.3).

Jedem Kandidaten aus dem in Zeile 8 konfigurierten Datenstrom wird durch einen JOIN-Operator (Zeile 29) ein Modell zugeordnet. Die Zuordnung genau eines Modells zu einem Kandidaten ist gewährleistet, da der Aggregationsoperator die Aggregationsergebnisse mit Gültigkeitsintervallen ausgibt, sodass jedes Modell genau so lange gültig ist, bis es durch ein neues abgelöst wird (z. B. weil ein neues Lerndatum / Feedback hinzukommt und sich somit das Modell verändert). Das bedeutet, in dem Modelldatenstrom gibt es keine zwei Elemente, deren Gültigkeitsintervalle sich überlappen. Ebenso gibt es auch keinen Zeitpunkt, der nicht in ein Gültigkeitsintervall genau eines Modells fällt. Der JOIN-Operator berücksichtigt bei der Zuordnung von Elementen die Gültigkeitsintervalle. Somit wird zu einem Zeitpunkt t, an dem der Kandidat c gültig ist (entspricht dem Zeitpunkt, an dem der zum Kandidaten gehörende Request q gestellt wurde), genau das Modell zugeordnet, in dessen Gültigkeitsintervall der Zeitpunkt t fällt. Dieses Modell wurde genau mit den Lerndaten gelernt, die zum Zeitpunkt t gültig sind.

Die eigentliche Schätzung der Bewertung für die Kandidaten erfolgt durch einen MAP-Operator. Dieser nutzt die von der Aggregation ausgegebenen User-Feature- und Item-Feature-Matrizen, um damit die geschätzte Bewertung $\hat{r}$ zu berechnen (Zeile 37). Dazu werden die Funktionen row resp. column genutzt. Diese geben zu den Feature-Matrizen die zur Benutzerin / zum Benutzer bzw. zum Objekt passenden Zeilen- bzw. Spaltenvektor als Feature-Vektor zurück. Die Multiplikation des User-Feature-Vektors mit dem Item-Feature-Vektor ergibt die geschätzte Bewertung für den Kandidaten (vgl. Abschnitt 2.1.3). Das Ergebnis ist ein Datenstrom mit Elementen, die jeweils die Request-ID, die Objekt-ID des Kandidaten und die geschätzte Bewertung enthalten.

Für die Umsetzung der modellbasierten Methode wurde vollständig auf Basisoperatoren (im Wesentlichen AGGREGATION, JOIN und MAP) gesetzt und diese ggf. erweitert (insbesondere durch die neue Aggregationsfunktion; im Rahmen dieser Arbeit wurden auch die Matrixfunktionen für die MAP-Operation erweitert). Die Alternative dazu wäre, spezielle Recommender-System-Operatoren zu entwickeln. Der Ansatz in dieser Arbeit hat den Vorteil, dass vorhandene Funktionalitäten in den genutzten Operatoren wiederverwendet werden können.

Quelltext 4.21 Template zur Kandidatenbewertung mit der modellbasierten Methode der Matrixfaktorisierung

```
#DEFINE window_size [10, "MINUTES"]

#DEFINE learning_data_data_stream_name learning_data
#DEFINE learning_data_user_attribute_name "user"
#DEFINE learning_data_item_attribute_name "item"
#DEFINE learning_data_rating_attribute_name "rating"

#DEFINE candidates_stream_name candidates
#DEFINE candidates_request_attribute_name "request"
#DEFINE candidates_user_attribute_name "user"
#DEFINE candidates_item_attribute_name "item"

#DEFINE rated_data_stream_name rated
#DEFINE est_rating_attribute_name "rating"

windowed_learning_data = TIMEWINDOW({
    SIZE = ${window_size}
  }, ${learning_data_data_stream_name})

models = AGGREGATION({
    AGGREGATIONS = [[
      FUNCTION = "MatrixFactorization",
      USER_ATTRIBUTE = ${learning_data_user_attribute_name},
      ITEM_ATTRIBUTE = ${learning_data_item_attribute_name},
      RATING_ATTRIBUTE = ${learning_data_rating_attribute_name}
    ]]
  }, windowed_learning_data)

candidates_with_model = JOIN(${candidates_stream_name}, models)

${rated_data_stream_name} = MAP({
    EXPRESSIONS = [
      ${candidates_request_attribute_name},
      ${candidates_item_attribute_name},
      [
        ${est_rating_attribute_name},
        "row(user_feature_matrix, ${candidates_user_attribute_name}) * column(item_feature_matrix, ${candidates_item_attribute_name})"
      ]
    ]
  }, candidates_with_model)
```

4.2.6 Empfehlungsberechnung

Nachdem im vorherigen Schritt die Kandidaten mit geschätzten Bewertungen versehen wurden, wird anschließend die Empfehlungsmenge für jeden Request berechnet. Für die Empfehlungsberechnung muss aus der Menge der bewerteten Empfehlungskandidaten ein Datenstromelement erstellt werden, welches die K Kandidaten enthält, die die besten geschätzten Bewertungen haben. Diese Operation – eine Menge Datenstromelemente zu einem Datenstromelement zusammenzufassen – entspricht einer Aggregationsoperation. Aus diesem Grund wird für diesen Schritt wieder der Aggregationsoperator eingesetzt, diesmal mit der Aggregationsfunktion TopK.

Die Aggregationsfunktion TopK hat zwei Konfigurationsparameter: Das Attribut, nach dem die Elemente in absteigender Reihenfolge sortiert werden sollen und die Anzahl der Elemente, die in die Auswahl übernommen werden sollen. Da für jeden Request eine eigene Empfehlung berechnet werden soll, partitioniert der Aggregationsoperator die bewerteten Kandidaten nach dem Attribut `request_id`, sodass für die Berechnung der Top-K-Menge nur Kandidaten zum selben Request als Empfehlung zusammengefasst werden.

Quelltext 4.22 Template zur Berechnung der Top-K-Empfehlungen für die requestgetriebene Empfehlungsberechnung

```
1  #DEFINE number_of_recommendations 6
2  #DEFINE rated_data_stream_name rated
3  #DEFINE est_rating_attribute_name "rating"
4  #DEFINE request_id_attribute_name "request_id"
5  #DEFINE recommendations_data_stream_name recommendations
6  #DEFINE recommendations_attribute_name "recommendations"
7
8  ${recommendations_data_stream_name} = AGGREGATION({
9      AGGREGATIONS = [[
10       FUNCTION = "TopK",
11       TOP_K = ${number_of_recommendations},
12       SCORING_ATTRIBUTE = ${est_rating_attribute_name},
13       OUTPUT_ATTRIBUTE = ${recommendations_attribute_name}
14     ]],
15     GROUP_BY = ${request_id_attribute_name}
16   }, ${rated_data_stream_name})
```

Das Template in Quelltext 4.22 zeigt die Berechnung der Empfehlung für einen Request. Die Konfiguration in Zeile 1 gibt an, wie viele Objekte die Empfehlung beinhalten soll. Dieser Parameter wird in Zeile 11 der TopK-Aggregationsfunktion als Parameter `TOP_K` übergeben.

Der Name des Eingabedatenstroms mit den bewerteten Empfehlungskandidaten wird in Zeile 2 konfiguriert. Zeile 3 gibt den Namen des Attributes an, welches die geschätzte Bewertung des Kandidaten enthält. Dieses Attribut wird in Zeile 12 als Scoring-Attribut verwendet. Dieses Attribut wird von der Top-K-Funktion als Kriterium für die absteigende Sortierung der Empfehlungskandidaten genutzt. Gibt es zwei Elemente mit demselben Wert, ist die Reihenfolge dieser Elemente nicht festgelegt.

Das Attribut, welches die Request-ID enthält, ist in Zeile 4 konfiguriert. Dieses Attribut wird vom Aggregationsoperator in Zeile 15 genutzt, um die Daten zu partitionieren (Gruppierung).

Der Name des ausgehenden Datenstroms wird in Zeile 5 angegeben. Das Attribut mit dem in Zeile 6 konfigurierten Namen enthält die Empfehlungen.

Feedbackgetriebene Empfehlungsberechnung Bei der feedbackgetriebenen Empfehlungsberechnung werden die Empfehlungen vorberechnet und bei Eintreffen eines Requests wird dieser mit der passenden Empfehlung verknüpft und ausgegeben (vgl. Abschnitt 3.2.2).

Das Template in Quelltext 4.23 zeigt ein Beispiel für die Umsetzung der feedbackgetriebenen Empfehlungsberechnung. Dazu werden wie im vorherigen Beispiel die Anzahl der Empfehlungsobjekte je Empfehlung sowie die eingehenden und ausgehenden Datenströme konfiguriert. In diesem Beispiel wird für jede Stadt (Kontextattribut `city`) je Tageszeit (Kontextattribut `time_of_day`) eine Empfehlung vorberechnet. Dazu müssen zuvor wie im Template in Quelltext 4.13 die impliziten Requests bestimmt werden. Anschließend werden die impliziten Requests dazu genutzt, um Kandidaten und Lerndaten zu selektieren sowie die Kandidaten zu bewerten. Der resultierende Datenstrom wird in Zeile 2 als Eingabe konfiguriert. Die Kontextattribute, die bei der Berechnung der impliziten Requests genutzt wurden, müssen in dem Template als Gruppierungsattribute in Zeile 4 konfiguriert werden. Diese werden in der Top-K-Aggregation genutzt (Zeile 19), um für jeden impliziten Request eine eigene Empfehlung zu berechnen. Diese Attribute ersetzen die Request-ID aus dem vorherigen Template (Quelltext 4.22), da ein impliziter Request keine Request-ID besitzt.

Die Zuordnung der eigentlichen Requests zu den vorberechneten Empfehlungen erfolgt mithilfe eines JOIN-Operators. Dieser erhält ein Prädikat, wie es in Zeile 5 konfiguriert ist. Dieses Prädikat sorgt dafür, dass die Kontextattribute aus den Requests mit denen aus den vorkonfigurierten Empfehlungen übereinstimmen. Die

temporale Logik des JOIN-Operators sorgt erneut dafür, dass die zeitlich passenden Empfehlungen den Requests zugeordnet werden.

Quelltext 4.23 Template zur Berechnung der Top-K-Empfehlungen für die feedbackgetriebene Empfehlungsberechnung

```
#DEFINE number_of_recommendations 6
#DEFINE rated_data_stream_name rated
#DEFINE est_rating_attribute_name "rating"
#DEFINE grouping_attributes ["city", "time_of_day"]
#DEFINE join_predicate "city == city AND time_of_day ==
    time_of_day"
#DEFINE requests_data_stream_name requests
#DEFINE recommendations_data_stream_name recommendations
#DEFINE recommendations_attribute_name "recommendations"

pre_calculated_recommendations = AGGREGATION({
    AGGREGATIONS = [
      [
        FUNCTION = "TopK",
        TOP_K = ${number_of_recommendations},
        SCORING_ATTRIBUTE = ${est_rating_attribute_name},
        OUTPUT_ATTRIBUTE = ${recommendations_attribute_name}
      ]
    ],
    GROUP_BY = ${grouping_attributes}
  }, ${rated_data_stream_name})

${recommendations_data_stream_name} = JOIN({
    PREDICATE = ${join_predicate}
  }, ${requests_data_strea_name}, pre_calculated_recommendations)
```

4.2.7 Datenausgabe

Nachdem die Empfehlungen für einen Request berechnet worden sind, sieht der letzte Schritt die Ausgabe der Empfehlungen vor. Das Template in Quelltext 4.24 zeigt, wie sich Daten in eine CSV-Datei schreiben lassen. Dies kann beispielsweise sinnvoll sein, um eine Log-Datei zu schreiben oder um zu Evaluationszwecken das Recommender-System zu simulieren und die Ergebnisse zu analysieren.

Quelltext 4.24 Template zum Schreiben von Daten in eine CSV-Datei

```
1  #DEFINE file "recommendations.csv"
2  #DEFINE data_stream_name recommendations
3  #DEFINE csv_delimiter ";"
4
5  SENDER({
6      WRAPPER = "GenericPull",
7      TRANSPORT = "File"
8      PROTOCOL = "CSV",
9      DATAHANDLER = "Tuple",
10     OPTIONS = [
11       ["filename", ${file}],
12       ["delimiter", ${csv_delimiter}]
13     ]
14   }, ${data_stream_name})
```

Für die Ausgabe von Daten ist der SENDER-Operator zuständig. Dieser ähnelt dem SOURCE-Operator mit dem Unterschied, dass dieser einen Datenstrom als Eingabe erhält (Zeile 14) und *keinen* Datenstrom als Ausgabe zur Verfügung stellt. Ebenso wie der SOURCE-Operator erhält der SENDER-Operator die Angaben `Wrapper`, `Transport`, `Protocol`, `DataHandler` und eine Liste an Parametern (`Options`). Gegenüber dem SOURCE-Operator entfällt die Angabe des Schemas, da dieses aus dem eingehenden Datenstrom abgeleitet werden kann.

Für das Schreiben einer CSV-Datei wird als Transport-Handler *File* und als Protocol-Handler *CSV* gesetzt. Der Dateiname wird in Zeile 1 konfiguriert und in Zeile 11 als Optionsparameter dem SENDER-Operator mitgegeben. Ebenso kann das Trennzeichen konfiguriert werden (Zeile 3 bzw. 12).

Im Produktivbetrieb ist das Übertragen der berechneten Empfehlungen an die Anwendung relevant, die die Empfehlungen dem Benutzer anzeigen soll. Das Template in Quelltext 4.25 zeigt ein Beispiel, in dem die Empfehlungen als JSON-Objekte an eine HTTP-Schnittstelle (REST-Service) übertragen werden. Dazu werden die URI sowie die HTTP-Methode konfiguriert (Zeile 1 resp. 2) und dem SENDER-Operator als Optionen übergeben (Zeile 11 resp. 12).

Quelltext 4.25 Template zum Senden von Daten an einen HTTP-Service im JSON-Format

```
#DEFINE uri "http://api.example.com/recommendations"
#DEFINE method "GET"
#DEFINE data_stream_name recommendations

SENDER({
    WRAPPER = "GenericPush",
    TRANSPORT = "HTTP"
    PROTOCOL = "JSON",
    DATAHANDLER = "Tuple",
    OPTIONS = [
      ["uri", ${uri}],
      ["method", ${method}]
    ]
  }, ${data_stream_name})
```

4.3 Die Querysprache CQL

Die Querysprache CQL ist eine SQL-ähnliche Sprache, die um temporale Aspekte erweitert wurde. Der Vorteil von CQL gegenüber PQL ist, dass er für SQL-erfahrene Personen intuitiver zu verstehen ist. Der Nachteil ist, dass komplexe Queries in CQL unübersichtlicher werden und sich spezielle Operatoren in CQL nicht so einfach integrieren lassen. Häufig ist es dafür nötig, die Sprache zu erweitern. Dies wird insbesondere bei der Datenanbindung (Datenzugriff und Datenausgabe) deutlich (weiteres siehe Abschnitte 4.4.1 und 4.4.7).

Quelltext 4.26 zeigt ein Beispiel für die Definition eines CQL-Queries. Durch diesen Query werden die Attribute `user`, `item` und `rating` des `feedbacks`-Datenstroms der Elemente der letzten 10 Minuten ausgewählt, bei denen die Bewertung mindestens einen Wert von 3 hat.

Quelltext 4.26 Beispiel eines CQL-Queries

```
SELECT user, item, rating
FROM feedbacks [SIZE 10 Minutes TIME]
WHERE rating >= 3;
```

Um das Ergebnis des Datenstroms in weiteren Queries nutzen zu können, muss der resultierende Datenstrom benannt werden. Dies erfolgt durch die Definition einer *View*. Ein Beispiel ist in Quelltext 4.27 angegeben. Der resultierende Datenstrom wird von einer weiteren Query unter dem Namen `selected_feedback` weiterver-

wendet, in dem die Anzahl der ausgewählten Feedbacks in den 10-Minuten-Fenstern ausgegeben wird.

Quelltext 4.27 Beispiel eines CQL-Queries als View

```
1 CREATE VIEW selected_feedbacks FROM (
2   SELECT user, item, rating
3   FROM feedbacks [SIZE 10 Minutes TIME]
4   WHERE rating >= 3
5 );
6
7 SELECT count(*) FROM selected_feedbacks;
```

Um die verschiedenen Templates miteinander zu einem Recommender-System verknüpfen zu können, werden im nächsten Abschnitt alle Templates als Views definiert.

4.4 Templates für die Umsetzung von Recommender-Systemen in CQL

In diesem Abschnitt werden CQL-Templates für die Umsetzung von Recommender-Systemen vorgestellt. Die Gliederung der Unterabschnitte orientiert sich an den sieben Phasen, die in Abschnitt 3.5 eingeführt wurden.

4.4.1 Datenzugriff

Quelltext 4.28 zeigt ein Template für das Einlesen einer CSV-Datei als Datenstrom analog zu dem PQL-Template in Quelltext 4.5. Die Konfigurationsparameter `file`, `data_strea_name`, `csv_delimiter` und `csv_read_first_line` entsprechen den Parametern aus dem PQL-Template. Bei dem Parameter `schema` unterscheidet sich die Syntax der Schemadefinition von der Angabe in dem PQL-Statement.

Der Datenzugriff mit CQL erfolgt über ein `CREATE-STREAM`-Statement. Dieses Statement ist die datenstrombasierte Variante eines `CREATE-TABLE`-Statements aus SQL. Aus diesem Grund entspricht die Syntax der Schemadefinition auch der Schema-

angabe in einem CREATE-TABLE-Statement. Zusätzlich zu dem Namen und dem Datenschema muss zur Erstellung eines Datenstroms die Quelle der Daten für den Datenstrom angegeben werden. In Quelltext 4.28 ist das eine CSV-Datei, deren Inhalt zur Simulation eines Datenstroms eingelesen werden soll. Zur Definition der Datenquelle hat das CREATE-STREAM-Statement neben dem Namen und Datenschema die Schlüsselwörter WRAPPER, PROTOCOL, TANSPORT, DATAHANDLER und OPTIONS. Mit diesen wird analog zu dem PQL-Template in Quelltext 4.5 die Datenquelle spezifiziert.

Quelltext 4.28 Template zur Quellenanbindung mit CQL

```
1  #DEFINE file "feedbacks.csv"
2  #DEFINE data_stream_name feedbacks
3  #DEFINE schema user LONG, item LONG, rating DOUBLE
4  #DEFINE csv_delimiter ";"
5  #DEFINE csv_read_first_line "false"
6
7  CREATE STREAM ${data_stream_name} (${schema})
8    WRAPPER "GenericPull"
9    PROTOCOL "CSV"
10   TRANSPORT "File"
11   DATAHANDLER "Tuple"
12   OPTIONS (
13     "filename" ${file},
14     "delimiter" ${csv_delimiter},
15     "readfirstline" ${csv_read_first_line}
16   )
17 ;
```

Analog zu dem PQL-Template in Quelltext 4.6, in dem eine HTTP-Schnittstelle bereitgestellt wird, über die andere Softwaresysteme ihre Daten an das Datenstrommanagementsystem übergeben können, zeigt Quelltext 4.29 ein CQL-Template, mit dem die gleiche Aufgabe erledigt werden kann. Bis auf den anderen Syntax unterscheidet sich das Template kaum von dem PQL-Template und ist somit selbsterklärend.

Quelltext 4.30 zeigt analog zu dem PQL-Template in Quelltext 4.7 ein CQL-Template für das Abfragen von Daten von einer HTTP-Schnittstelle. Mit diesem Template werden Daten per *GET* von der als Parameter uri definierten HTTP-Schnittstelle abgefragt und als Datenstrom zur Verfügung gestellt.

Quelltext 4.29 Template zur Quellenanbindung über eine HTTP-Schnittstelle (Server) mit CQL

```
#DEFINE hostname "api.example.com"
#DEFINE port 8080
#DEFINE path "/requests"
#DEFINE schema user.id LONG, context.location.city STRING
#DEFINE data_stream_name requests

CREATE STREAM ${data_stream_name} (${schema})
  WRAPPER "GenericPush"
  PROTOCOL "JSON"
  TRANSPORT "HttpServer"
  DATAHANDLER "Tuple"
  OPTIONS (
    "hostname" ${hostname},
    "port" ${port},
    "path" ${path},
    "content_type" "application/json"
  )
;
```

Quelltext 4.30 Template zur Quellenanbindung über eine HTTP-Schnittstelle (Client) mit CQL

```
#DEFINE uri "http://api.example.com/feedbacks"
#DEFINE method "GET"
#DEFINE schema item.id LONG, item.text STRING, user.id LONG,
    context.location.city STRING
#DEFINE data_stream_name requests

CREATE STREAM ${data_stream_name} (${schema})
  WRAPPER "GenericPull"
  PROTOCOL "JSON"
  TRANSPORT "HTTP"
  DATAHANDLER "Tuple"
  OPTIONS (
    "uri" ${uri},
    "method" ${method}
  )
;
```

4.4.2 Datenvorverarbeitung

Für die Phase der Datenvorverarbeitung haben wir bereits in Abschnitt 4.2.2 die folgenden vier Aufgaben definiert, zu denen wir PQL-Templates definiert haben:

- Das Umbenennen von Datenattributen, damit die Templates der nachfolgenden Phasen die Attribute unter den Namen wiederfinden, wie sie in den Templates definiert sind.
- Das Transformieren von Daten, bspw. um diese zu normalisieren.
- Das Herausfiltern von nicht benötigten Daten.
- Das Verknüpfen von Daten aus unterschiedlichen Datenquellen, bspw. um Datenelemente mit Bestandsdaten anzureichern.

In diesem Abschnitt betrachten wir, wie sich diese Templates mit CQL umsetzen lassen.

Analog zu dem PQL-Template in Quelltext 4.8, übernimmt das CQL-Template in Quelltext 4.31 die Aufgabe, Datenattribute eines Datenstroms umzubenennen. Die Attribute, die in dem resultierenden Datenstrom enthalten sein sollen, werden mit dem Konfigurationsparameter in Zeile 1 definiert. Die Syntax entspricht der Auswahlliste (Select-Part) des SQL-Select-Statements.

Quelltext 4.31 Template zur Umbenennung von Attributen mit CQL

```
1 #DEFINE mappings user.id AS user, context.location.city AS city
2 #DEFINE input_data_stream_name requests_with_wrong_names
3 #DEFINE output_data_stream_name requests
4
5 CREATE VIEW ${output_data_stream_name} FROM (
6   SELECT ${mapping}
7   FROM ${input_data_stream_name}
8 );
```

Die bei den PQL-Templates in Abschnitt 4.2.2 vorgenommene Unterscheidung zwischen der Nutzung des MAP- und des RENAME-Operators entfällt bei den CQL-Statements. An dieser Stelle zeigt sich der Unterschied zwischen CQL und PQL: In CQL als deklarativer Sprache gibt die Anwenderin bzw. der Anwender das gewünschte Ergebnis an. Der CQL-Parser entscheidet selbstständig, ob intern der MAP- oder der RENAME-Operator genutzt werden soll.

Als Beispiel für die Transformation von Daten wurde im PQL-Template in Quelltext 4.10 der stetige Temperaturwert in einen diskreten Temperaturwert überführt, der den Wert `low` (kleiner als 10 °C) oder `high` (größer oder gleich 10 °C) annehmen kann. Alle anderen Attribute sollen beibehalten werden. Quelltext 4.32 zeigt, wie dieses Template in CQL umgesetzt wird. Der Konfigurationsparameter in Zeile 1 ermöglicht die Angabe der Transformationen. Dabei entspricht die Syntax dieses Parameters der der Auswahlliste (Select-Part) des SQL-Select-Statements (wie im vorherigen Template).

Quelltext 4.32 Template zur Transformation von Daten mit CQL

```
1 #DEFINE transformations ifthenelse(temperature < 10, 'low',  ↩
      'high') AS temperature
2 #DEFINE input_data_stream_name feedbacks_before_transformation
3 #DEFINE output_data_stream_name feedbacks
4
5 CREATE VIEW ${output_data_stream_name} FROM (
6   SELECT *, ${transformations}
7   FROM ${input_data_stream_name}
8 );
```

In dem PQL-Template in Quelltext 4.10 haben wir den Parameter `KEEPATTRIBUTES` des MAP-Operators auf `true` gesetzt. Damit haben wir erreicht, dass alle Attribute des eingehenden Datenstroms erhalten bleiben, ohne dass wir alle Attribute als „Transformation“ aufzählen müssen. Um das gleiche Verhalten in dem CQL-Template zu erzielen, stellen wir den Transformationen in dem CQL-Statement in Zeile 6 die Zeichenkette „`*,`“ voran. Das Wildcard-Sternchen sorgt dafür, dass alle vorhandenen Attribute beibehalten werden.

Quelltext 4.33 zeigt ein CQL-Template zum Filtern von Daten analog zum PQL-Template in Quelltext 4.11. Der Konfigurationsparameter in Zeile 1 bestimmt das Prädikat, welches in Zeile 8 in der `WHERE`-Clause des Select-Statements genutzt wird.

Die Anreicherung eines Datenstroms mit Daten aus einer Datenbank (z. B. eine Datenbank mit Objektinformationen), wie sie in Quelltext 4.12 als PQL-Template definiert ist, lässt sich mit CQL nicht ohne weiteres umsetzen. Wird diese Funktion benötigt, muss auf ein PQL-Template zurückgegriffen werden.

Quelltext 4.33 Template zum Filtern von Daten mit CQL

```
1 #DEFINE predicate "!isNull(item)"
2 #DEFINE input_data_stream_name unfiltered_feedbacks
3 #DEFINE output_data_stream_name feedbacks
4
5 CREATE VIEW ${output_data_stream_name} FROM (
6   SELECT *
7   FROM ${input_data_stream_name}
8   WHERE ${predicate}
9 );
```

Die Ableitung impliziter Requests, wie sie in Abschnitt 3.6.2 beschrieben ist und mit dem PQL-Template in Quelltext 4.13 bereits umgesetzt wurde, ist als CQL-Template in Quelltext 4.34 dargestellt. Die wichtigste Konfiguration ist die Angabe der Kontextattribute, für die durch den impliziten Request individuelle Empfehlungen vorberechnet werden sollen (Zeile 1). Diese Attribute werden im Select-Statement in Zeile 6 zusammen mit dem Schlüsselwort `DISTINCT` genutzt.

Quelltext 4.34 Template zur Ableitung impliziter Requests mit CQL

```
1 #DEFINE context_attributes city, time_of_day
2 #DEFINE feedbacks_data_stream_name feedbacks
3 #DEFINE impl_requests_data_stream_name implrequests
4
5 CREATE VIEW ${impl_requests_data_stream_name} FROM (
6   SELECT DISTINCT ${context_attributes}
7   FROM ${feedbacks_data_stream_name}
8 );
```

4.4.3 Kandidatenselektion

Die Kandidatenselektion aus einer Objektdatenbank, wie sie im PQL-Template in Quelltext 4.14 definiert ist, lässt sich in CQL nicht umsetzen, da hierfür erneut der DBENRICH-Operator benötigt wird, der sich mit der an SQL angelehnten CQL-Syntax nicht ausdrücken lässt.

Sollen die Kandidaten aus den Feedbackdaten abgeleitet werden, so ist in Quelltext 4.35 ein CQL-Template definiert, welches diese Aufgabe analog zu dem PQL-Template in Quelltext 4.15 erledigt. Das Kreuzprodukt des Request- und Feedbackdatenstroms in Zeile 9, welches in dem Datenstrommanagementsystem grundsätzlich die Gültigkeitsintervalle der Requests und Feedbacks berücksichtigt, ergibt zusam-

men mit dem Prädikat, welches in Zeile 1 definiert und in Zeile 10 verwendet wird, eine temporale Join-Operation. Die Angabe der Attribute in Zeile 2 und die Verwendung derselbigen als Projektion zusammen mit dem Schlüsselwort `DISTINCT` in Zeile 8 führen zu einem Datenstrom analog zu der Definition in Gleichung 3.26 (Seite 66) bzw. Abbildung 3.17 (Seite 67).

Quelltext 4.35 Template zur Kandidatenselektion aus dem Feedbackdatenstrom mit CQL

```
1  #DEFINE candidates_predicate requests.city == feedbacks.city
2  #DEFINE candidates_attributes requests.id, requests.city,  ↩
       feedbacks.item, feedbacks.item_author
3  #DEFINE requests_data_stream_name requests
4  #DEFINE feedbacks_data_stream_name feedbacks
5  #DEFINE candidates_data_stream_name candidates
6
7  CREATE VIEW ${candidates_data_stream_name} FROM (
8    SELECT DISTINCT ${candidates_attributes}
9    FROM ${requests_data_stream_name},  ↩
       ${feedbacks_data_stream_name}
10   WHERE ${candidates_predicate}
11 );
```

4.4.4 Lerndatenselektion

Für die Lerndatenselektion haben wir in Abschnitt 4.2.4 zwei PQL-Templates definiert: Ein Template für die Lerndatenselektion aus dem Feedbackdatenstrom für die speicherbasierte Methode (Quelltext 4.16) und ein Template für die modellbasierte Methode (Quelltext 4.17).

Quelltext 4.36 zeigt das CQL-Template analog zu dem PQL-Template in Quelltext 4.16. In diesem Template werden die Lerndaten für das speicherbasierte Vorgehen aus dem Feedbackdatenstrom gewonnen. Das Kreuzprodukt in Zeile 8 verbindet zusammen mit dem Prädikat, welches in Zeile 1 definiert und in Zeile 9 verwendet wird, einen Kandidaten mit Feedbacks des Feedbackdatenstroms, welche als Lerndaten für die Bewertung des Kandidaten genutzt werden sollen.

Die Lerndatenselektion der modellbasierten Methode unterscheidet sich zur Lerndatenselektion der speicherbasierten Methode in dem Datenstrom, zu dem die Lerndaten zugeordnet werden sollen. Während dies bei der speicherbasierten Me-

Quelltext 4.36 Template zur Lerndatenselektion aus dem Feedbackdatenstrom (speicherbasiert) mit CQL

```
#DEFINE learning_data_predicate candidates.item ==
    feedbacks.item AND candidates.city == feedbacks.city
#DEFINE candidates_data_stream_name candidates
#DEFINE feedbacks_data_stream_name feedbacks
#DEFINE learning_data_data_stream_name learning_data

CREATE VIEW ${learning_data_data_stream_name} FROM (
  SELECT *
  FROM ${candidates_data_stream_name},
    ${feedbacks_data_stream_name}
  WHERE ${learning_data_predicate}
);
```

thode der Kandidatendatenstrom ist, ist es bei der modellbasierten Methode der Datenstrom der (impliziten) Requests. Quelltext 4.37 zeigt das CQL-Template analog zum PQL-Template in Quelltext 4.17, welches die Lerndaten den (impliziten) Requests zuordnet. Der Aufbau des Templates ist analog zu dem vorherigen Template und somit selbsterklärend.

Quelltext 4.37 Template zur Lerndatenselektion aus dem Feedbackdatenstrom (modellbasiert) mit CQL

```
#DEFINE learning_data_predicate implrequests.city ==
    feedbacks.city
#DEFINE requests_data_stream_name implrequests
#DEFINE feedbacks_data_stream_name feedbacks
#DEFINE learning_data_data_stream_name learning_data

CREATE VIEW ${learning_data_data_stream_name} FROM (
  SELECT *
  FROM ${requests_data_stream_name},
    ${feedbacks_data_stream_name}
  WHERE ${learning_data_predicate}
);
```

4.4.5 Kandidatenbewertung

Wie bereits in den PQL-Templates zur Kandidatenbewertung in Abschnitt 4.2.5 zu sehen ist, basiert die Kandidatenbewertung auf der Aggregation von Lerndaten. Aggregationsfunktionen lassen sich in CQL analog zu SQL in der Attributauswahlliste (Select-Part) angeben. Quelltext 4.38 zeigt ein CQL-Template, welches mithilfe der

count-Funktion die Anzahl der Feedbacks zu einem Kandidaten bestimmt und diesen Wert als Bewertung des Kandidaten festsetzt. Dieses Template entspricht dem PQL-Template in Quelltext 4.18.

Die Berechnung der Bewertung erfolgt über die Aggregation von Lerndaten, die in einem Zeitfenster liegen. In dem Beispiel in Quelltext 4.38 ist das Zeitfenster in Zeile 1 auf 10 Minuten konfiguriert. Das Zeitfenster wird zu dem Lerndatenstrom im FROM-Teil in Zeile 13 gesetzt. Da für jeden Kandidaten zu jedem Request eine eigene Bewertung berechnet werden soll, werden die Daten im Datenstrom nach Request-ID und Kandidaten-ID gruppiert (Zeile 14). Die Berechnung der Bewertung erfolgt in Zeile 11 durch die Anwendung der count-Funktion.

Quelltext 4.38 Template zur Kandidatenbewertung mit der speicherbasierten Methode, die die am häufigsten genutzten Objekte empfiehlt (CQL).

```
1  #DEFINE window_size 10 MINUTES
2  #DEFINE learning_data_data_stream_name learning_data
3  #DEFINE request_id_attribute_name request
4  #DEFINE candidate_item_id_attribute_name item
5  #DEFINE rated_data_stream_name rated
6  #DEFINE est_rating_attribute_name rating
7
8  CREATE VIEW ${rated_data_stream_name} FROM (
9    SELECT ${request_id_attribute_name},
10     ${candidate_item_id_attribute_name},
11     count(*) AS ${est_rating_attribute_name}
12   FROM ${learning_data_data_stream_name}
13     [SIZE ${window_size} TIME]
14   GROUP BY ${request_id_attribute_name},  ↩
       ${candidate_item_id_attribute_name}
15 );
```

Die Schätzung der Bewertung für einen Kandidaten durch die Berechnung der durchschnittlichen Bewertung aus den Feedbackdaten ist in dem Template in Quelltext 4.39 dargestellt. Dieses CQL-Template entspricht dem PQL-Template aus Quelltext 4.20 und unterscheidet sich zu dem vorherigen Template lediglich durch die Verwendung der avg-Funktion anstelle der count-Funktion in Zeile 13.

Als typischer Vertreter einer modellbasierten Methode wird im Template in Quelltext 4.40 analog zum PQL-Template in Quelltext 4.21 ein Beispiel für die Anwendung der Matrixfaktorisierung vorgestellt. Neben der Konfiguration in den Zeilen 1 bis 14

Quelltext 4.39 Template zur Kandidatenbewertung mit der speicherbasierten Methode, die die am häufigsten genutzten Objekte empfiehlt (CQL).

```
1  #DEFINE window_size 10 MINUTES
2  #DEFINE learning_data_data_stream_name learning_data
3  #DEFINE request_id_attribute_name request
4  #DEFINE candidate_item_id_attribute_name item
5  #DEFINE rating_attribute_name rating
6  #DEFINE rated_data_stream_name rated
7  #DEFINE est_rating_attribute_name rating
8
9  CREATE VIEW ${rated_data_stream_name} FROM (
10   SELECT
11     ${request_id_attribute_name},
12     ${candidate_item_id_attribute_name},
13     avg(${rating_attribute_name}) AS ↩
       ${est_rating_attribute_name}
14   FROM ${learning_data_data_stream_name}
15     [SIZE ${window_size} TIME]
16   GROUP BY ${request_id_attribute_name}, ↩
       ${candidate_item_id_attribute_name}
17 );
```

besteht das Template aus dem Lernen des Modells in den Zeilen 16 bis 25 und der Anwendung des Modells in den Zeilen 27 bis 33.

Für die Berechnung der geschätzten Bewertung wird in Zeile 18 die Aggregationsfunktion MatrixFactorization eingesetzt, die im Rahmen dieser Arbeit entwickelt wurde. Im Gegensatz zu den Funktionen count und avg ist das Ergebnis der Matrixfaktorisierung, so wie sie in dieser Arbeit umgesetzt wurde, kein skalarer Wert, sondern besteht aus den beiden Werten Item- sowie User-Feature-Matrix. Da diese Aggregationsfunktion zwei Werte liefert, ist diese nicht mit dem AS-Syntax kompatibel, welches erlaubt, die Namen der resultierenden Attribute frei zu setzen. Auf die Feature-Matrizen lässt sich somit nur über die von der Funktion als Standard festgelegten Bezeichner `user_feature_matrix` resp. `item_feature_matrix` zugreifen, wie es in der Anwendung des Modells in Zeile 31 zu sehen ist.

Für die Anwendung des Modells und zur Berechnung der geschätzten Bewertung werden die Funktionen row und column eingesetzt, welche die User-Feature-Vektoren bzw. Item-Feature-Vektoren für die/den entsprechende(n) Benutzer/-in bzw. das entsprechende Objekt aus den Feature-Matrizen auswählen und aus diesen das Skalarprodukt bilden (Zeile 31).

Quelltext 4.40 Template zur Kandidatenbewertung mit der modellbasierten Methode der Matrixfaktorisierung mit CQL

```
#DEFINE window_size 10 MINUTES

#DEFINE learning_data_data_stream_name learning_data
#DEFINE learning_data_user_attribute_name user
#DEFINE learning_data_item_attribute_name item
#DEFINE learning_data_rating_attribute_name rating

#DEFINE candidates_stream_name candidates
#DEFINE candidates_request_attribute_name request
#DEFINE candidates_user_attribute_name user
#DEFINE candidates_item_attribute_name item

#DEFINE rated_data_stream_name rated
#DEFINE est_rating_attribute_name rating

CREATE VIEW models FROM (
  SELECT
    MatrixFactorization(
      ${learning_data_user_attribute_name},
      ${learning_data_item_attribute_name},
      ${learning_data_rating_attribute_name}
    )
  FROM ${learning_data_data_stream_name}
    [${window_size}]
);

CREATE VIEW ${rated_data_stream_name} FROM (
  SELECT
    ${candidates_request_attribute_name},
    ${candidates_item_attribute_name},
    row(user_feature_matrix, ↩
   ${candidates_user_attribute_name}) * ↩
   column(item_feature_matrix, ↩
   ${candidates_item_attribute_name}) AS ↩
   ${est_rating_attribute_name}
  FROM ${candidates_stream_name}, models
);
```

4.4.6 Empfehlungsberechnung

Nachdem im vorherigen Schritt die Kandidaten bewertet wurden, müssen aus diesen die Empfehlungen mit den K Kandidaten mit den höchsten geschätzten Bewertungen ausgewählt und zu einer geordneten Liste zusammengefügt werden. In den PQL-Templates (Quelltext 4.22 und 4.23) wurde dazu ein Aggregationsoperator mit der Aggregationsfunktion TopK eingesetzt. Das gleiche Ergebnis erhalten wir mit den folgenden CQL-Templates, auch wenn die Verwendung der TopK-Funktion nicht auf Anhieb ersichtlich ist.

Das CQL-Template in Quelltext 4.41 ermittelt für die requestgetriebene Empfehlungsberechnung die Menge der Top-K-Empfehlungen analog zu dem PQL-Template in Quelltext 4.22. Dazu werden die bewerteten Kandidaten (Zeile 12) nach Request-ID gruppiert (Zeile 13), damit für jeden Request eine eigene Empfehlung berechnet wird, und nach geschätzter Bewertung absteigend sortiert (Zeile 14). Nun sollen für jede Gruppe (also je Request) die ersten K (durch die Sortierung also diese mit der höchsten geschätzten Bewertung) ausgewählt werden. SQL und CQL kennen dafür das Schlüsselwort `LIMIT`. Das Problem des Einsatzes von `LIMIT` in dieser Query ist allerdings, dass `LIMIT` nicht *je* Gruppe, sondern *vor* der Gruppierung angewendet wird. Um die Begrenzung der Anzahl der Daten je Gruppe analog umsetzen zu können, definieren wir an diese Stelle ein neues Schlüsselwort `GROUP LIMIT`, welches die Funktionalität von `LIMIT` auf die Daten nach der Gruppierung überträgt. Diese Funktionalität haben wir in dem Datenstrommanagementsystem *Odysseus* entsprechend implementiert. Dadurch werden je Gruppe die Top-K-Kandidaten bzgl. der geschätzten Bewertung ausgewählt (Zeile 15). Um die Top-K-Kandidaten anschließend zu einer Empfehlung je Request zusammenzufassen, wird in Zeile 11 die Aggregationsfunktion nest genutzt, welche eine Tupelmenge zu einer Liste zusammenfügt. Statt des Sternchens kann man an dieser Stelle konkrete Attribute angeben, beispielsweise die Kandidaten-ID. In diesem Falle würden die Empfehlungen nur aus den spezifizierten Attributen bestehen und alle anderen Attribute durch einen PROJECT-Operator entfernt.

Aus diesem Template ergibt sich also, dass die Daten gruppiert werden (`GROUP BY`) und anschließend je Gruppe sortiert (`ORDER BY`), auf die Top-*K*-Liste beschränkt (`GROUP LIMIT`) und in ein Tupel zusammengefasst (nest) werden. Da dieses Verhalten dem Aggregationsoperator mit der TopK-Funktion aus dem PQL-Template entspricht, definieren wir hiermit mit der Einführung des neuen Schlüsselwortes `GROUP LIMIT` die Übersetzungsregel, die dieses Pattern in einen Aggregationsoperator mit TopK-Funktion übersetzt. Mit dieser Erweiterung lässt sich die für Recommender-Systeme benötigte Funktion der Empfehlungsmengenbildung auch mit CQL umsetzen.

Quelltext 4.41 Template zur Berechnung der Top-K-Empfehlungen für die requestgetriebene Empfehlungsberechnung mit CQL

```
1  #DEFINE number_of_recommendations 6
2  #DEFINE rated_data_stream_name rated
3  #DEFINE est_rating_attribute_name rating
4  #DEFINE request_id_attribute_name request_id
5  #DEFINE recommendations_data_stream_name recommendations
6  #DEFINE recommendations_attribute_name recommendations
7
8  CREATE VIEW recommendations_data_stream_name FROM (
9    SELECT
10     ${request_id_attribute_name},
11     nest(*) AS ${recommendations_attribute_name}
12   FROM ${rated_data_stream_name}
13   GROUP BY ${request_id_attribute_name}
14   ORDER BY ${est_rating_attribute_name} DESC
15   GROUP LIMIT ${number_of_recommendations}
16 );
```

Die Zusammenstellung der Empfehlungen für die feedbackgetriebene Empfehlungsberechnung ist in dem CQL-Template in Quelltext 4.42 dargestellt. Dieses Template entspricht dem PQL-Template aus Quelltext 4.23. Die eigentliche Berechnung der Empfehlungen findet in den Zeilen 10 bis 18 statt. Im Gegensatz zur requestgetriebenen Empfehlungsberechnung findet an dieser Stelle eine Vorberechnung der Empfehlungen statt. An Stelle der Request-ID treten die Gruppierungsattribute in den Zeilen 4 und 15 zur Gruppierung der bewerteten Kandidaten. Diese Gruppierungsattribute bestimmen die möglichen Request-Instanzen (vgl. dazu auch die Berechnung der impliziten Feedbacks). Davon abgesehen entspricht die Empfehlungsberechnung der Query aus dem vorherigen Template.

Gegenüber der requestgetriebenen Empfehlungsberechnung ist die Query in den Zeilen 20 bis 24 hinzugekommen. Diese verknüpft die tatsächlichen Requests mit den vorberechneten Empfehlungen mithilfe einer Join-Operation. Für die richtige Zuordnung sorgt das Join-Prädikat, definiert in der Zeile 5 und eingesetzt in der Zeile 23, sowie die temporale Eigenschaft des Join-Operators in dem Datenstrommanagementsystem. Das Join-Prädikat greift die Gruppierungsattribute auf und verbindet Requests, deren Attribute denen der impliziten Requests (und somit den Attributen, die die Gruppierung bestimmen) entsprechen.

Quelltext 4.42 Template zur Berechnung der Top-K-Empfehlungen für die feedbackgetriebene Empfehlungsberechnung mit CQL

```
1  #DEFINE number_of_recommendations 6
2  #DEFINE rated_data_stream_name rated
3  #DEFINE est_rating_attribute_name rating
4  #DEFINE grouping_attributes city, time_of_day
5  #DEFINE join_predicate city == city AND time_of_day ==
       time_of_day
6  #DEFINE requests_data_stream_name requests
7  #DEFINE recommendations_data_stream_name recommendations
8  #DEFINE recommendations_attribute_name recommendations
9
10 CREATE VIEW pre_calculated_recommendations FROM (
11   SELECT
12     ${grouping_attributes},
13     nest(*) AS ${recommendations_attribute_name}
14   FROM ${rated_data_stream_name}
15   GROUP BY ${grouping_attributes}
16   ORDER BY ${est_rating_attribute_name}
17   GROUP LIMIT ${number_of_recommendations}
18 );
19
20 CREATE VIEW ${recommendations_data_stream_name} FROM (
21   SELECT *
22   FROM ${requests_data_strea_name},
       pre_calculated_recommendations
23   WHERE ${join_predicate}
24 );
```

4.4.7 Datenausgabe

Für die Datenausgabe definieren wir analog zu den PQL-Templates in Quelltext 4.24 und 4.25 CQL-Templates zur Ausgabe in eine CSV-Datei und zur Übergabe an ein HTTP-Service.

Das Template in Quelltext 4.43 zeigt die Ausgabe von Empfehlungen in eine CSV-Datei. Die Datenausgabe besteht in CQL aus zwei Teilen: Zum einen aus einer Definition einer Datensenke (Zeilen 7 bis 15) und zum anderen aus einer Query, die die Daten an diese Datensenke sendet (Zeile 17). Letzteres kann auch dadurch ersetzt werden, in dem man zu der Query, die die Empfehlungen berechnet (die Templates aus dem vorherigen Abschnitt), das `CREATE-VIEW`-Konstrukt durch die `STREAM-TO`-Angabe ersetzt und somit die zusätzliche Query in Zeile 17 verzichtbar macht.

Quelltext 4.43 Template zum Schreiben von Daten in eine CSV-Datei mit CQL

```
#DEFINE file "recommendations.csv"
#DEFINE data_stream_name recommendations
#DEFINE sink_name recommendations_sink
#DEFINE csv_delimiter ";"
#DEFINE schema request_id LONG, recomm_1_item LONG, recomm_1_score DOUBLE, recomm_2_item LONG, recomm_2_score DOUBLE, recomm_3_item LONG, recomm_3_score DOUBLE, recomm_4_item LONG, recomm_4_score DOUBLE, recomm_5_item LONG, recomm_5_score DOUBLE, recomm_6_item LONG, recomm_6_score DOUBLE

CREATE SINK ${sink_name} (${schema})
  WRAPPER "GenericPush"
  PROTOCOL "CSV"
  TRANSPORT "File"
  DATAHANDLER "Tuple"
  OPTIONS (
    "filename" ${file},
    "delimiter" ${delimiter}
  );

STREAM TO ${sink_name} SELECT * FROM ${data_stream_name};
```

Der CREATE-SINK-Syntax erinnert an den CREATE-STREAM-Syntax. Vergleicht man das CQL-Template mit dem PQL-Template fällt auf, dass die isolierte Definition der Datensenke (ohne, dass der Eingabedatenstrom der Senke bekannt gemacht wird) erfordert, dass der Senkendefinition das Schema explizit bekannt gemacht wird. Dies ist gegenüber PQL eine redundante Angabe, die bei Änderungen des Datenschemas im Eingabedatenstrom zu Fehlern führen kann.

Das Template zum Senden von Daten an eine HTTP-Schnittstelle, dargestellt in Quelltext 4.44, ist analog aufgebaut und somit selbsterklärend.

Quelltext 4.44 Template zum Senden von Daten an einen HTTP-Service im JSON-Format mit CQL

```
#DEFINE uri "http://api.example.com/recommendations"
#DEFINE method "GET"
#DEFINE data_stream_name recommendations
#DEFINE sink_name recommendations_sink
#DEFINE schema request_id LONG, recomm_1_item LONG, recomm_1_score DOUBLE, recomm_2_item LONG, recomm_2_score DOUBLE, recomm_3_item LONG, recomm_3_score DOUBLE, recomm_4_item LONG, recomm_4_score DOUBLE, recomm_5_item LONG, recomm_5_score DOUBLE, recomm_6_item LONG, recomm_6_score DOUBLE
```

```
6
7  CREATE SINK ${sink_name} (${schema})
8    WRAPPER "GenericPush"
9    PROTOCOL "JSON"
10   TRANSPORT "HTTP"
11   DATAHANDLER "Tuple"
12   OPTIONS (
13     "filename" ${file},
14     "uri" ${uri},
15     "method" ${method}
16   );
17
18 STREAM TO ${sink_name} SELECT * FROM ${data_stream_name};
```

4.5 Zusammenfassung

In diesem Kapitel wurden für die Umsetzung von Recommender-Systemen auf Basis von Datenstrommanagementsystemen Templates in den Query-Sprachen PQL und CQL definiert. Bis auf wenige Ausnahmen ließen sich die Templates in beiden Sprachen gleichermaßen umsetzen. Welche Sprache für die Umsetzung eines Recommender-Systems genutzt wird, bleibt somit den Präferenzen der DSMS-Benutzerin bzw. des DSMS-Benutzers überlassen. Ebenso ist es möglich, die Sprachen zu mischen und auf Datenströme zuzugreifen, die mit der anderen Sprache definiert wurden.

Für die Umsetzung eines Recommender-Systems müssen lediglich die Templates mit den gewünschten Parametern aufgerufen werden. Beispiele dafür werden im folgenden Kapitel gezeigt.

5 Implementierung und Evaluation

In diesem Kapitel werden die vorgestellten Konzepte evaluiert. Die Evaluation besteht aus zwei Teilen: Eine Offline-Evaluation mit der modellbasierten Methode der Matrixfaktorisierung (Abschnitt 5.1) und eine Online-Evaluation mit speicherbasierten Methoden im Rahmen der CLEF-NewsREEL-Challenge (Abschnitt 5.2).

Das allgemeine Ziel der Evaluation ist es zu zeigen, dass sich die vorgestellten Konzepte für den Einsatz eines Datenstrommanagementsystems als Basis für datenstrombasierte Recommender-Systeme eignen. Im ersten Szenario liegt der Fokus darauf zu zeigen, dass sich die Datenstromtechnologie grundsätzlich für die Berechnung von Empfehlungen (insb. auch mit modellbasierten Methoden) eignet und die Implementierung korrekt ist. Dass das vorgestellte Konzept den Anforderungen an einen produktiven Betrieb in einem realistischen Szenario gewachsen ist, zeigt das zweite Szenario.

Es ist ausdrücklich nicht das Ziel dieser Arbeit einen neuen Algorithmus bzw. eine neue Methode zu entwerfen, mit dem / der bessere Empfehlungen bzgl. eines Kriteriums bzw. einer Metrik berechnet werden können. Aus diesem Grund ist es auch nicht das primäre Ziel der Evaluation, eine quantitative Bewertung der berechneten Empfehlungen durchzuführen. Nichtsdestoweniger hat die Evaluation im Rahmen der CLEF-NewsREEL-Challenge (Abschnitt 5.2) gezeigt, dass der in dieser Arbeit vorgestellte Ansatz mit den Ansätzen anderer teilnehmenden Teams mithalten kann und dieser sogar bzgl. der Click Through Rate (CTR) bei hohen Datenraten andere Ansätze schlägt. Dies ist vermutlich darauf zurückzuführen, dass die datengetriebene Verarbeitung der Feedbackdaten ohne Verzögerung dazu führt, dass Trends in den Daten schneller Berücksichtigung finden.

5.1 Inkrementelle Matrixfaktorisierung

In dieser Arbeit wird die Methode der Matrixfaktorisierung als Beispiel für eine modellbasierte Empfehlungsberechnungsmethode herangezogen. Der in dieser Arbeit implementierte Algorithmus für die inkrementelle Matrixfaktorisierung basiert auf dem Algorithmus *BRISMF*, wie er in [Tak+09] beschrieben wurde. Neben diversen

Optimierungen, die die Qualität der Vorhersage gegenüber anderen Verfahren verbessern, ist das Besondere an dem BRISMF-Algorithmus, dass bei kleinen Änderungen in der Bewertungsmatrix $\mathbf{R}$ das Trainieren der Feature-Matrizen $\mathbf{P}$ und $\mathbf{Q}$ schneller konvergiert als bei anderen Matrixfaktorisierungsalgorithmen (vgl. Abschnitt 3.5 in [Tak+09]). BRISMF ist darauf ausgelegt, dass gelernte Feature-Matrizen bei kleinen Änderungen in der Bewertungsmatrix nicht verworfen und komplett neu gelernt werden, sondern nur bestimmte Teile neu initialisiert werden. Für die genaue Funktionsweise von BRISMF sei auf die zitierte Veröffentlichung von Takács et al. verwiesen. Diese Arbeit geht nur auf die Aspekte ein, die für die Implementierung in einem Datenstrommanagementsystem von Bedeutung sind.

5.1.1 Ziel der Evaluation

Ziel dieser Evaluation ist zu zeigen, wie mit diesem Ansatz eine modellbasierte Methode umgesetzt werden kann und zu zeigen, dass diese Implementierung korrekt funktioniert. Dazu wird als Vergleich die Implementierung des Frameworks *Massive Online Analysis (MOA)* [Bif+10] herangezogen.

5.1.2 Vorgehen der Evaluation

Das Datenstrommanagementsystem *Odysseus*, welches als Basis dieser Evaluation herangezogen wurde, ermöglicht die Erweiterung um neue Operatoren. Dazu muss die Methode `processNext` implementiert werden, die die Datenstromelemente entgegennimmt und verarbeitet. Diese Methode wird von Odysseus für jedes neue Datenstromelement in der Reihenfolge aufgerufen, wie diese im Datenstrom vorkommen (chronologische Reihenfolge). Innerhalb dieser Methode kann der Operator die Methode `transfer` aufrufen, und dieser die Ergebnisse als Datenstromelemente für den Ausgabedatenstrom übergeben. Diese Elemente werden dann von Odysseus an den nachfolgenden Operator übergeben.

Um eine modellbasierte Methode zu implementieren, muss der Operator, der für das Lernen des Modells zuständig ist, alle gültigen Lerndaten (d. h. alle Datenstromelemente, die in einem gemeinsamen Verarbeitungsfenster liegen) vorhalten. Für die Implementierung des BRISMF-Algorithmus bedeutet dies, dass alle Bewertungen in Form einer Bewertungsmatrix $\mathbf{R}$ vorgehalten werden müssen, die derzeit gültig sind. Als Alternative zur Implementierung eines eigenen Operators bietet sich die Erweiterung des Aggregationsoperators an (vgl. Abschnitt 3.6.4 auf S. 69). Dieser kann um weitere Aggregationsfunktionen erweitert werden. Eine Aggregationsfunktion bekommt immer dann, wenn sich die Menge der zu berücksichtigenden

Elemente aus dem Eingangsdatenstrom ändert, die Menge aller derzeit gültigen Datenelemente übergeben, damit aus dieser ein neues Aggregationsergebnis berechnet werden kann, das für den Zeitpunkt gültig ist. Das ist immer dann der Fall, wenn ein Datenstromelement neu hinzukommt oder ein Datenstromelement ungültig wird.

Diese Methode eignet sich, um ein modellbasiertes Lernverfahren für Recommender-Systeme umzusetzen, ohne das Management der Lerndaten im Speicher selbst umsetzen zu müssen. Angenommen, man möchte beim Lernen eines Matrixfaktorisierungsmodells die Feedbackdaten der vergangenen 60 Minuten berücksichtigen, so würde man einen Fensteroperator vor einem Aggregationsoperator positionieren. Der Aggregationsoperator bekommt als Aggregationsfunktion eine Implementierung des Lernalgorithmus zugewiesen. Diese Aggregationsfunktion bekommt nun jedes Mal, wenn ein neues Feedbackdatum hinzukommt oder wenn ein Feedbackdatum ungültig wird (älter als 60 Minuten), vom Aggregationsoperator die Menge aller gültigen Lerndaten (diese, die nicht älter als 60 Minuten sind) übergeben, um mit diesen das Modell zu aktualisieren. Die Aggregationsfunktion gibt anschließend eine Kopie der aktualisierten Modellinstanz an den Aggregationsoperator zurück, damit dieser die Modellinstanz als Datenstromelement ausgibt. Diese Modellinstanz kann anschließend genutzt werden, um Empfehlungen zu berechnen.

Um die Machbarkeit dieses Ansatzes zu zeigen, wurde vom Autor die Implementierung des BRISMF-Algorithmus aus dem Projekt *Massive Online Analysis (MOA)* [Bif+10] als Aggregationsfunktion `learn_brismf` in Odysseus integriert. Die MOA-Implementierung des Algorithmus enthält bereits Methoden, um das Modell inkrementell zu aktualisieren. Dazu muss dem Algorithmus lediglich die aktualisierte Bewertungsmatrix übergeben werden, welche von der neu eingeführten Aggregationsfunktion übernommen wird. Als Ergebnis gibt die Aggregation ein Tupel mit zwei Matrizen $\mathbf{P}$ und $\mathbf{Q}$ als Modellinstanzen aus. Diese können von einem MAP-Operator genutzt werden, um die geschätzte Bewertung für eine Benutzerin bzw. einen Benutzer und einem Objekt zu berechnen. Dazu wird der ensprechende Zeilen- bzw. Spaltenvektor aus der Matrix $\mathbf{P}$ bzw. $\mathbf{Q}$ addressiert und beide miteinander multipliziert. Das Resultat ist die geschätzte Bewertung $\hat{r}$.

Das MOA-Framework unterstützt die Evaluation der dort implementierten Algorithmen. Dazu wird die Evaluationsmethode *Interleaved Test-Then-Train* (ITTT) zusammen mit dem *Root Mean Square Error* (RMSE) eingesetzt (beides in Abschnitt 2.1.5 beschrieben). Um die Implementierung des BRISMF-Algorithmus in Odysseus zu validieren, wurden die ITTT-Methode mit einer Datenstromquery umgesetzt und die Ergebnisse mit denen des MOA-Frameworks verglichen. Unter der Annahme, dass die Vergleichsimplementierung korrekt ist, kommt eine korrekte Implementierung zu den selben Ergebnissen.

5.1.3 Umsetzung von ITTT mit einer Datenstromquery

Bei ITTT ist es das Ziel, ein Lerndatum als Testdatum zu nutzen, bevor es als Lerndatum genutzt wird. Im Falle der Matrixfaktorisierung bestehen die Lerndaten aus User-Item-Rating-Tripeln. Zu jedem neuen Lerndatum wird zunächst das vorhandene Modell genutzt, um zu dem User-Item-Pair eine Bewertung $\hat{r}$ zu schätzen und diese mit der echten Bewertung r zu vergleichen. Anschließend wird das Lerndatum genutzt, um das Modell zu aktualisieren.

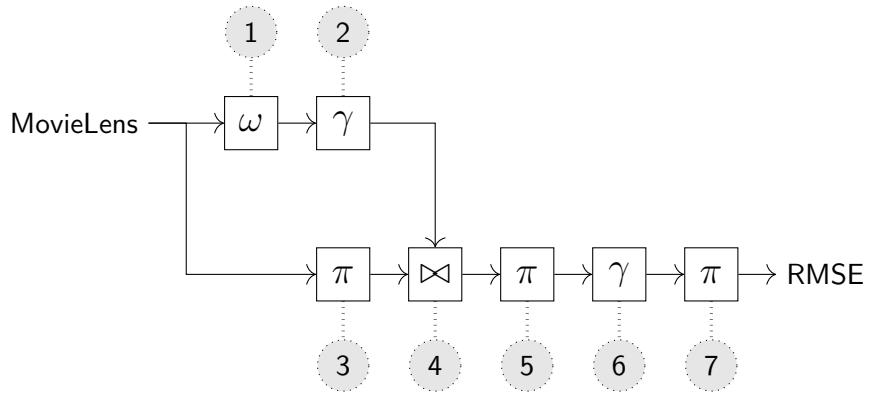


Abbildung 5.1 Queryplan für Evaluation der BRISMF-Implementierung mit der Evaluationsmethode ITTT

Die Umsetzung der ITTT-Evaluation als Datenstromquery ist in Abbildung 5.1 dargestellt. Diese besteht aus den folgenden Operatoren:

1. Der WINDOW-Operator setzt die Gültigkeitsintervalle der Lerndaten, sodass diese beispielsweise 60 Minuten gültig sind, bis sie aus der Bewertungsmatrix entfernt werden.
2. Der AGGREGATION-Operator trainiert das BRISMF-Modell mit der Aggregationsfunktion `learn_brismf` und gibt die Modellinstanzen als Datenstromelemente aus.
3. Der MAP-Operator manipuliert die Gültigkeitsintervalle der Lerndaten, um diese als Testdaten nutzen zu können. Dazu wird die Gültigkeitsdauer so gesetzt, dass dieser Datensatz als Testdatum genau die Zeiteinheit gültig ist, *bevor* dieser Datensatz als Lerndatum gültig wird.
4. Der JOIN-Operator verbindet die Testdaten mit den temporal zuständigen Modellinstanzen.
5. Der MAP-Operator berechnet eine geschätzte Bewertung ($\hat{r} = \vec{pq}$) für die Testdaten sowie eine Abweichung zur echten Bewertung ($(r - \hat{r})^2$ = Square Error).
6. Der AGGREGATION-Operator aggregiert die Square Errors zu einem Mean Square Error (Aggregationsfunktion: AVG).

7. Der MAP-Operator berechnet den Root Mean Square Error mit der Funktion SQRT.

Um die Umsetzung des ITTT mit der Datenstromtechnologie zu verstehen, muss man sich die Bedeutung der Gültigkeitsintervalle der Datenstromelemente bewusst machen. Der Zeitpunkt t_s ist im Rahmen der Evaluation im Datensatz vorgegeben. Dies ist der Zeitpunkt, ab dem das Lerndatum zur Bewertungsmatrix hinzugefügt werden soll und somit der Zeitpunkt, an dem dieses gültig wird. Der Fensteroperator setzt den Zeitpunkt t_e, beispielsweise auf $t_s + 60\,\mathrm{min.}$, um die Gültigkeitsdauer auf 60 Minuten zu begrenzen. Der Aggregationsoperator (Punkt 2 in der Liste bzw. Grafik) gibt zu jedem Zeitpunkt, an dem ein Lerndatum gültig oder ungültig wird (also zu jedem t_e oder t_s eines Lerndatums) eine neue Modellinstanz aus, die bei dessen Berechnung das neue Lerndatum berücksichtigt bzw. das ungültig gewordene Lerndatum nicht mehr berücksichtigt. Das Datenstromelement mit der Modellinstanz, welches vom Aggregationsoperator ausgegeben wird, hat ebenso eine Gültigkeitsdauer. Dieses ist zu dem Zeitpunkt gültig (t_s), an dem das Lerndatum, welches die Neuberechnung ausgelöst hat, gültig bzw. ungültig geworden ist und ist solange gültig (t_e), bis es durch eine neue Modellinstanz abgelöst wird. Somit ist zu jedem Zeitpunkt genau eine Modellinstanz gültig.

Um ein Lerndatum l als Testdatum v zu nutzen, muss sichergestellt werden, dass das Modell, welches zur Schätzung der Bewertung genutzt wird, das Lerndatum l noch nicht kennt. Dies wird dadurch erreicht, dass der MAP-Operator (Punkt 3) die Gültigkeitsdauer des Testdatums anpasst. Für ein Lerndatum mit einem Gültigkeitsstartzeitpunkt von t'_s wird die Gültigkeitsdauer des Testdatums auf $t_s = t'_s - 1$ und $t_e = t'_s$ gesetzt. Das bedeutet, das Testdatum wird eine Zeiteinheit vor dem Lerndatum gültig und verliert seine Gültigkeitsdauer an dem Zeitpunkt, an dem das Lerndatum gültig wird. Der darauffolgende JOIN-Operator ordnet dem Testdatum anschließend das Modell zu, welches zu dem Zeitpunkt t_s des Testdatums gültig ist. Das ist somit genau das Modell, in dem das Lerndatum gerade nicht beim Trainieren des Modells berücksichtigt wurde. Das ist somit das Verhalten, welches bei der Methode ITTT verlangt wird.

5.1.4 Ergebnisse der Evaluation

Um die Implementierung zu validieren wurden die ITTT-Umsetzung in Odysseus und die ITTT-Umsetzung in MOA mit dem MovieLens-Datensatz durchgeführt und verglichen. Da MOA die Begrenzung der Gültigkeitsdauer der Lerndaten nicht unterstützt, wurde die Evaluierung in Odysseus ohne Fensteroperator durchgeführt. Das Ergebnis ist ein Datenstrom, der zu jedem Zeitpunkt, bevor ein Lerndatum gültig wird, einen aktualisierten RMSE ausgibt (akkumuliert). Um es mit den Ergebnissen

mit MOA zu vergleichen, wurde dieser Datenstrom in eine CSV-Datei geschrieben. Die Ergebnisse der Evaluation mit Odysseus sind mit denen des MOA identisch. Somit konnte die Korrektheit der Implementierung gezeigt werden.

5.2 Speicherbasierte Empfehlungen im Live-Betrieb in der CLEF-NewsREEL-Challenge

Mit dem in dieser Arbeit vorgestellten Konzept hat der Autor an der CLEF-NewsREEL-Challenge [Hop+16] in den Jahren 2016 und 2017 teilgenommen und jeweils die ersten Plätze belegt. Die Lösungen wurden vom Autor in [Lud16] (für das Jahr 2016) und in [Lud17a] (für das Jahr 2017) veröffentlicht. In diesem Abschnitt wird die Teilnahme aus dem Jahr 2017 als Teil der Evaluation vorgestellt. Die Gesamtergebnisse der Challenge 2017 wurden vom Veranstalter in [Lom+17] veröffentlicht.

Das CLEF NewsREEL Evaluation Lab ermöglicht Forscherinnen und Forschern im Bereich der Recommender-Systeme die Evaluation von Algorithmen und Systemen zur Empfehlung von Nachrichtenartikeln. Dazu erhalten diese Zugang zur *Open Recommendation Platform (ORP)* des Veranstalters, welche in Kooperation mit der plista GmbH den Teilnehmenden Live-Daten von verschiedenen Nachrichtenportalen (u. a. von sport1.de und dem Kölner Stadtanzeiger) in Echtzeit zur Verfügung stellt. Die Daten werden in Form von drei Datenströmen zur Verfügung gestellt:

- Impressions: Aufruf eines Nachrichtenartikels durch eine Leserin bzw. einen Leser
- Items: Nachrichtenartikel, die neu hinzugefügt werden
- Errors: Fehlermeldungen (zu Debugging-Zwecken, hier nicht weiter von Belang)

Die Daten aller Nachrichtenportale werden über den selben Datenstrom übertragen. Die Zugehörigkeit der Datenelemente zu den Portalen werden durch einen numerischen Fremdschlüssel kenntlich gemacht.

Neben den Daten der Portale werden über einen weiteren Datenstrom die Teilnehmenden aufgefordert, Empfehlungen für eine konkrete Benutzerin bzw. einen konkreten Benutzer eines Portals auszusprechen. Von den Antworten der Teilnehmenden wird zufällig eine ausgewählt und der Leserin bzw. dem Leser live angezeigt. Das Verhältnis der Anzahl der angezeigten Empfehlungen und der Anzahl der Klicks

auf ebendiesen ergibt als *Click Through Rate (CTR)* die Metrik zur Bestimmung der besten Lösung.

Damit die Websites des Nachrichtenportals ohne wesentliche Verzögerung ausgeliefert werden können, muss die Antwort mit den Empfehlungen innerhalb von 100 ms beim ORP-System eintreffen. Andernfalls wird der Request als unbeantwortet gewertet. Um möglichst vielen Teams die Teilnahme zu ermöglichen, wird die Datenrate der Datenströme individuell angepasst. Ist ein Recommender-System vermehrt nicht in der Lage innerhalb der festgelegten Zeitspanne auf Requests zu reagieren, so wird die Datenrate des Impressions-Datenstrom verringert. Ist es in der Lage, den Großteil der Requests rechtzeitig zu beantworten, so wird die Datenrate wieder erhöht. So wird kontinuierlich die Datenrate entsprechend der Möglichkeiten der teilnehmenden Recommender-Systeme angepasst. Der genaue Algorithmus zur individuellen Anpassung der Datenraten wurde vom Veranstalter nicht veröffentlicht.

Um die Echtzeitanforderung an das Recommender-System in der Challenge zu berücksichtigen, haben die Veranstalter ein einfaches Baseline-Recommender-System unter dem Namen „BL2Beat“ an der Challenge teilnehmen lassen. Dieses hat zu jedem Request die letzten Nachrichtenartikel empfohlen, die dem System hinzugefügt wurden. Alle teilnehmenden Recommender-Systeme mussten sich mit der Anzahl der Requests des Baseline-Recommender-Systems messen, die innerhalb der geforderten 100 ms beantwortet werden konnten. Alle Systeme, die nicht in der Lage waren, mindestens 75 % der Anzahl beantworteter Requests des Baseline-Recommender-Systems zu erreichen, wurden disqualifiziert. Das gelang 2016 21 von 55 und 2017 12 von 18 teilnehmenden Recommender-Systemen.

5.2.1 Ziel der Evaluation

Die Teilnahme an der CLEF NewsREEL Challenge als Teil der Evaluation verfolgt das Ziel, den Nachweis der Umsetzbarkeit einer kompletten Lösung in einem Real-World-Szenario zu erbringen. Dabei stehen folgende Fragestellungen im Fokus:

- Ist die Technologie der Datenstromverarbeitung für die Umsetzung eines Recommender-Systems im Live-Betrieb geeignet?
- Kann mit dem Framework aus Kapitel 4 ein Recommender-System für den Live-Betrieb umgesetzt werden?
- Kann die Lösung mit den Ansätzen der anderen Teilnehmenden mithalten?

	Approach	CTR
ody0:	Random sample of 60 min. sliding time window.	0.0072
ody1:	Most viewed articles of 30 min. sliding time window partitioned by publisher.	0.0118
ody2:	Most viewed articles of 5 min. sliding time window partitioned by publisher.	0.0137
ody3:	Most viewed articles of 30 min. sliding time window partitioned by publisher and user location.	0.0137
ody4:	Most clicked recommendations of 12 hour sliding time window partitioned by item.	0.0157
ody5:	Most clicked recommendations of 12 hour sliding time window partitioned by item filled up with most viewed articles of 30 min. sliding time window partitioned by publisher.	0.0156

Tabelle 5.1 Überblick über die Ansätze, die mit dem Konzept dieser Arbeit im Rahmen der CLEF NewsREEL Challenge 2017 umgesetzt wurden (aus [Lud17a]).

5.2.2 Vorgehen der Evaluation

Im Rahmen der Challenge wurden verschiedene Variationen umgesetzt. Tabelle 5.1 gibt einen Überblick. Als interner Baseline-Ansatz wurde als *ody0* ein Recommender-System umgesetzt, welches zufällige Nachrichtenartikel zur Empfehlung ausgewählt hat. Als Kandidaten wurden die Artikel ausgewählt, die in den vergangenen 60 Minuten mindestens von einer Leserin bzw. einem Leser geöffnet wurden.

Als *ody1* und *ody2* wurde jeweils ein Recommender-System umgesetzt, welches für jedes Nachrichtenportal eigene Empfehlungen berechnete, für die jeweils nur die Daten des eigenen Portals berücksichtigt wurden. Bei *ody1* wurden die Daten der vergangenen 30 Minuten, bei *ody2* die Daten der vergangenen 5 Minuten berücksichtigt. Es wurden die Artikel empfohlen, die in den jeweiligen Zeitfenstern am häufigsten aufgerufen wurden.

Der Ansatz *ody3* stellt eine Variante von *ody1* dar, in dem zusätzlich Lerndaten nach der Geoposition der Leserinnen bzw. Leser partitioniert wurden. Das bedeutet, dass nur Artikel empfohlen wurden, die mit der gleichen Geoposition (entsprechend der von der Anwendung genutzten Auflösung) in den vergangenen 30 Minuten aufgerufen wurden. Außerdem wurden zur Bewertung der Kandidaten nur die Aufrufe mit gleicher Geoposition berücksichtigt.

Für die Recommender-Systeme *ody4* und *ody5* wurde zur Empfehlungsberechnung ein anderer Ansatz gewählt: Der Impressions-Datenstrom enthält zu den Seiten-

aufrufen auch die Information, ob es sich bei dem Aufruf um einen Klick auf eine Empfehlung handelt. Die Ansätze *ody4* und *ody5* lernen von diesen erfolgreichen Empfehlungen. Dazu wurden im Rahmen der Datenvorverarbeitung nur die Datenstromelemente herausgefiltert, bei denen es sich um Klicks auf Empfehlungen handelt. Die enthaltenen Artikel wurden als Empfehlungskandidaten behandelt. Die Bewertung der Kandidaten erfolgte durch die Berechnung, wie häufig der jeweilige Kandidat erfolgreich empfohlen wurde. Die Artikel, die in einem Zeitfenster am häufigsten geklickt wurden, nachdem diese empfohlen wurden, wurden anderen Leserinnen und Lesern empfohlen. Als Zeitfenster wurden 12 Stunden gewählt, da kürzere Zeitfenster zu einer zu geringen Anzahl an Empfehlungen geführt haben.

In dem Ansatz *ody5* wurde zusätzlich die Empfehlungsmenge aufgefüllt, wenn aufgrund zu weniger erfolgreicher Empfehlungen keine sechs Empfehlungen zustande gekommen sind. Dazu wurde dieser Ansatz mit dem Recommender-System *ody1* verknüpft, der die Empfehlungen zum Auffüllen der Empfehlungsmenge geliefert hat.

5.2.3 Implementierung einer Lösung für die CLEF-NewsREEL-Challenge mit Odysseus

Zur Evaluation des Konzepts dieser Arbeit wird im Folgenden erläutert, wie mit den vorgestellten Konzepten eine Lösung mit Odysseus für die Challenge umgesetzt wurde. Dazu wurden die Templates aus dem vorherigen Kapitel angewandt. In diesem Abschnitt wird der Ansatz *ody1* beispielhaft vorgestellt. Dieser Ansatz empfiehlt zu jedem Request die Nachrichtenartikel, die auf dem jeweiligen Nachrichtenportal („publisher“) in den vergangenen 30 Minuten am häufigsten aufgerufen wurden.

Datenzugriff und Datenvorverarbeitung

Die Datenströme der ORP-Plattform werden als HTTP-Requests an die Recommender-Systeme übergeben. Dazu muss das Recommender-System als HTTP-Server die Daten entgegennehmen.

Zur Anbindung der Datenströme wird das Template aus Quelltext 4.6 (siehe S. 78) genutzt. Die Instantiierung des Templates ist in Quelltext 5.1 dargestellt. Die Daten werden im JSON-Format übertragen. Das Schema in den Zeilen 4 ff überführt das JSON-Format in eine relationale Struktur (Tupel). Dazu werden die einzelnen Attribute mithilfe von JSON-Pfadausdrücken angegeben. Nach der Definition der Templateparameter wird in Zeile 14 schlussendlich das eigentliche Template aus

Quelltext 4.6 eingebunden. Dazu wird dem `#INPUT`-Befehl die einzubindende Datei mit dem entsprechenden Template übergeben.

Quelltext 5.1 Anbindung des Impressions-Datenstroms für die CLEF-NewsREEL-Challenge

```
1  #DEFINE hostname "172.17.0.2"
2  #DEFINE port 9092
3  #DEFINE path "/impressions"
4  #DEFINE schema [
5    ['type', 'String'],
6    ['timestamp', 'StartTimeStamp'],
7    ['context.simple.25', 'Integer'], /// item
8    ['context.simple.57', 'Long'],  /// user
9    ['context.simple.27', 'Integer'], /// publisher
10   ...
11 ]
12 #DEFINE data_stream_name impressions_input
13
14 #INPUT templates/access-httpserver-json.qry
```

Da die Attribute als Attributnamen die JSON-Pfadausdrücke erhalten, benennen wir diese im nächsten Schritt im Rahmen der Datenvorverarbeitung (vgl. Abschnitt 4.2.2) in sprechendere Attributnamen um. Dazu wenden wir das Template aus Quelltext 4.9 an. Die Instantiierung ist in Quelltext 5.2 zu sehen. Die neuen Attributnamen sind in Zeile 1 angegeben.

Quelltext 5.2 Umbenennung der Impressions-Attribute für die CLEF-NewsREEL-Challenge

```
1  #DEFINE new_names ["type", "timestamp", "item", "user",  ↩
       "publisher", ... ]
2  #DEFINE input_data_stream_name impressions_input
3  #DEFINE output_data_stream_name impressions
4
5  #INPUT templates/rename.qry
```

Die Anbindung der anderen Datenströme erfolgt analog.

Kandidatenbewertung und Empfehlungsberechnung

Als Paradigma wird die feedbackgetriebene Empfehlungsberechnung angewandt (vgl. Abschnitt 3.2.2). Das bedeutet, die Empfehlungsberechnung findet auf Basis

des Feedbackdatenstroms mit Hilfe von impliziten Requests statt. Bei Eintreffen eines tatsächlichen Requests werden diese vorberechneten Empfehlungen mit diesem Request verknüpft.

Die Berechnung der Empfehlungen erfolgt entsprechend des in Abschnitt 3.6.3 (*Feedbackgetriebene Empfehlungsberechnung bei disjunkter Kandidaten- bzw. Lerndatenmengen*) vorgestellten Konzepts. Die explizite Kandidaten- und Lerndatenselektion entfällt in diesem Fall. Sowohl die Lerndaten als auch die Kandidaten werden unmittelbar aus dem Impressions-Datenstrom entnommen. Als Lerndaten werden *alle* Daten aus dem Impressions-Datenstrom genutzt. Die Kandidatenmenge bilden alle Objekte, die im Impressions-Datenstrom vorkommen. Die Kandidatenbewertung erfolgt durch eine gruppierte Aggregationsoperation. Die impliziten Feedbacks werden durch das Gruppierungsattribut bestimmt (vgl. auch Abbildung 3.19 auf Seite 69).

Zur Umsetzung wird das Template aus Quelltext 4.18 bzw. die Variante in Quelltext 4.19 genutzt. Die Instantiierung ist in Quelltext 5.3 zu sehen. Als Request-ID wird in Zeile 3 die Publisher-ID gesetzt. Dies führt dazu, dass im nachfolgenden Schritt für jedes Nachrichtenportal eine eigene Empfehlungsmenge vorberechnet werden kann (implizite Requests). Das Ergebnis ist der Datenstrom `rated`, welches als Attribute die Publisher-ID `publisher`, die Objekt-ID `item` sowie die geschätzte Bewertung `rating` enthält. Das Attribut `rating` enthält dabei die Anzahl der Feedbacks (Seitenaufrufe) je Nachrichtenartikel (`item`) und Nachrichtenportal (`publisher`) der vergangenen 30 Minuten (Zeile 1).

Quelltext 5.3 Kandidatenbewertung für die CLEF-NewsREEL-Challenge

```
1 #DEFINE window_size [30, "MINUTES"]
2 #DEFINE learning_data_data_stream_name feedbacks
3 #DEFINE request_id_attribute_name "publisher"
4 #DEFINE candidate_item_id_attribute_name "item"
5 #DEFINE rated_data_stream_name rated
6 #DEFINE est_rating_attribute_name "rating"
7
8 #INPUT templates/candidates-rating-count.qry
```

Die Berechnung der Empfehlungen erfolgt auf Basis des Templates in Quelltext 4.23. Quelltext 5.4 zeigt die Instantiierung. Durch das Template werden die sechs (Zeile 1) Nachrichtenartikel mit den höchsten geschätzten Bewertungen als Empfehlung für ein Nachrichtenportal zusammengefasst. Als Gruppierungsattribut wird das Attribut `publisher` in Zeile 4 als implizite Request-ID gesetzt. Das Join-Prädikat in

Zeile 5 sorgt dafür, dass zu den Requests die passenden Empfehlungen zugeordnet werden.

Quelltext 5.4 Empfehlungsberechnung für die CLEF-NewsREEL-Challenge

```
1  #DEFINE number_of_recommendations 6
2  #DEFINE rated_data_stream_name rated
3  #DEFINE est_rating_attribute_name "rating"
4  #DEFINE grouping_attributes ["publisher"]
5  #DEFINE join_predicate "requests.publisher == rated.publisher"
6  #DEFINE requests_data_stream_name requests
7  #DEFINE recommendations_data_stream_name recommendations
8  #DEFINE recommendations_attribute_name "recommendations"
9
10 #INPUT templates/calc-recommendations-feedbackdriven.qry
```

Datenausgabe

Die berechneten Empfehlungen zu einem Request werden anschließend per HTTP-Post an die ORP-Plattform übertragen. Dazu wird das Template aus Quelltext 4.25 eingesetzt, dessen Instanziierung in Quelltext 5.5 zu sehen ist. Neben der Adresse des HTTP-Services (Zeile 1) wird die HTTP-Methode gesetzt (Zeile 2) und der Datenstromname angegeben, der die Empfehlungen beinhaltet (Zeile 3).

Quelltext 5.5 Senden der Empfehlungen an die ORP-Plattform

```
1 #DEFINE uri "http://api.example.com/recommendations"
2 #DEFINE method "POST"
3 #DEFINE data_stream_name recommendations
4
5 #INPUT templates/send-http-service-json.qry
```

5.2.4 Ergebnisse der Evaluation

In den folgenden beiden Abschnitten werden die Ergebnisse der Evaluation dargestellt. Der erste Abschnitt beschreibt die Ermittlung der optimalen Fenstergröße. Der zweite Abschnitt stellt die Ergebnisse der CLEF NewsREEL Challenge 2017 vor.

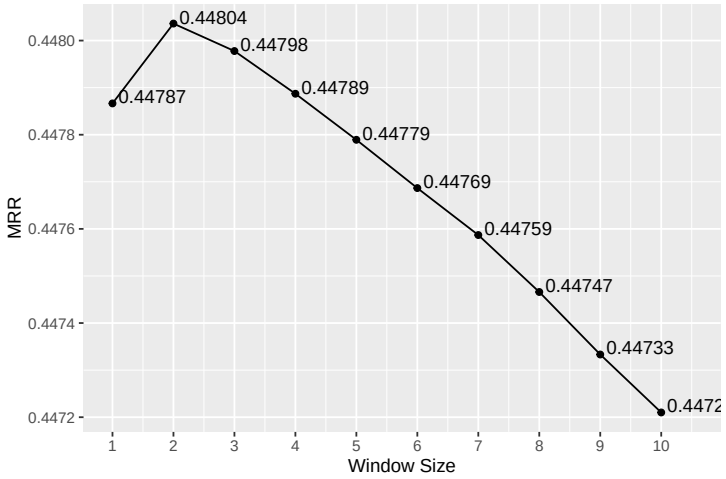
Evaluation geeigneter Fenstergrößen

Ein entscheidender Konfigurationsparameter bei den vorgestellten Ansätzen ist die Fenstergröße. Diese bestimmt, über welchen Zeitraum die Daten zur Berechnung der Empfehlungen vorgehalten und berücksichtigt werden sollen. Im Rahmen der CLEF NewsREEL Challenge 2016 hat der Autor eine ausführliche Evaluation der Fenstergrößen vorgenommen, um den Einfluss der Fenstergrößen zu ermitteln und die optimale Fenstergröße zu bestimmen (vgl. [Lud16]). Dazu wurde der Ansatz *ody1* mit 21 verschiedenen Fenstergrößen ausgeführt (von 1 bis 10 min., sowie 20 min., 30 min., 40 min., 50 min., 60 min., 90 min., 2 Stunden, 3 Stunden, 6 Stunden, 12 Stunden, 24 Stunden). Anschließend wurde die Evaluationsmethode *Interleaved Test-Then-Train* (ITTT, vgl. Abschnitt 2.1.5 auf S. 22) mit den Metriken *Mean Reciprocal Rank* (MRR, vgl. Abschnitt 2.1.5 auf S. 23) und der *Hit Rate* unter den ersten sechs Kandidaten (vgl. Abschnitt 2.1.5 auf S. 23) durchgeführt. Außerdem wurde bestimmt, in wie vielen Fällen aufgrund zu weniger Daten in den jeweiligen Fenstern weniger als sechs Empfehlungen zustandegekommen sind.

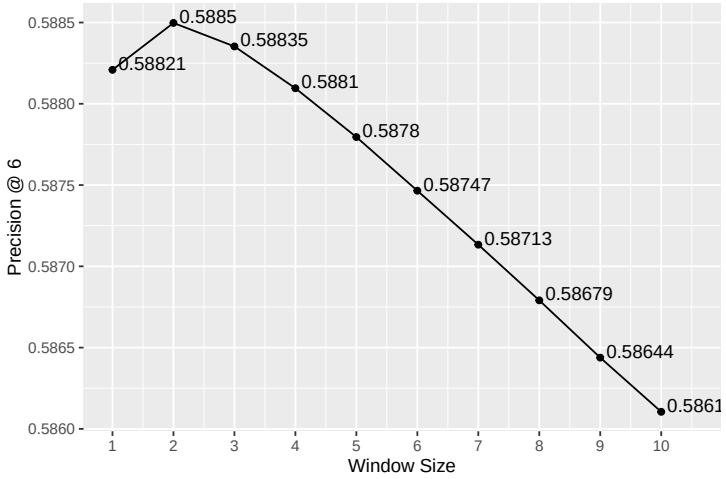
Die Evaluation mit der Methode ITTT wurde wie folgt umgesetzt: Für jeden Seitenaufruf aus dem Impressions-Datenstrom wurde eine Empfehlung berechnet, ohne dabei zu berücksichtigen, welcher Nachrichtenartikel tatsächlich aufgerufen wurde. Genauer gesagt wurde für alle Kandidaten eine geschätzte Bewertung berechnet und die bewertete Kandidatenmenge absteigend nach geschätzter Bewertung sortiert. Anschließend wurde für den tatsächlich aufgerufenen Nachrichtenartikel der Rang innerhalb der sortierten Kandidatenmenge bestimmt. Der Kehrwert dieses Ranges bildet den *Reciprocal Rank*. Je höher der tatsächlich betrachtete Artikel in der sortierten Kandidatenmenge einsortiert wurde (und somit je wahrscheinlicher dieser Artikel zur Empfehlung ausgewählt wäre), desto höher ist der *Reciprocal Rank*. Das arithmetische Mittel dieser Werte bildet den *Mean Reciprocal Rank*.

Anstelle der Position in der sortieren Kandidatenmenge wird bei der Metrik *Hit Rate* bestimmt, ob der tatsächlich aufgerufene Artikel in der Empfehlungsmenge (Top-6-Menge) vorkommt oder nicht. Das Verhältnis der Fälle, in denen der Artikel in der Empfehlungsmenge vorkommt zu den Fällen, in denen er nicht vorkommt, bildet die *Hit Rate*.

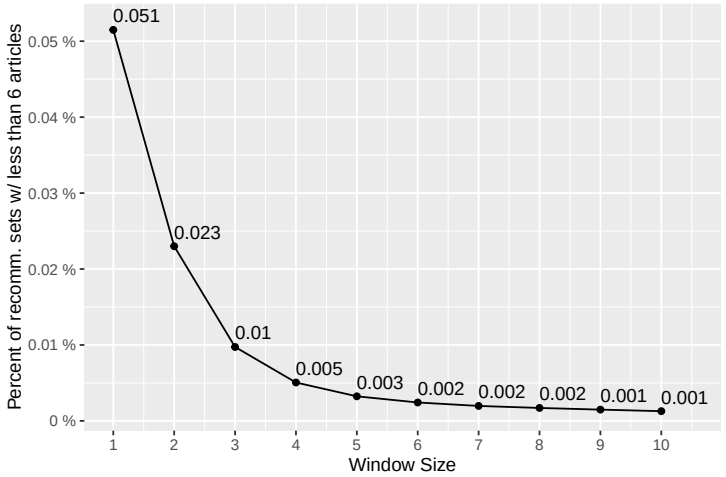
Der *Mean Reciprocal Rank* für die Fenstergrößen 1 bis 10 min. ist in Abbildung 5.2a abgebildet. Bei einer Fenstergröße von 2 Minuten ist dieser am größten. Das bedeutet, dass bei einer Fenstergröße von 2 Minuten die Wahrscheinlichkeit am größten ist, dass der tatsächlich aufgerufene Nachrichtenartikel auch in der Empfehlungsmenge vorgekommen wäre. Das gleiche Bild zeigt sich bei der *Hit Rate* (Abbildung 5.2b). Die Fenstergrößen ab 20 Minuten wurden aus Gründen der Übersichtlichkeit in der Ab-



(a) Mean Reciprocal Rank



(b) Hit Rate bei sechs Objekten je Empfehlung



(c) Prozent der Fälle mit unzureichender Anzahl (< 6) an Empfehlungen

Abbildung 5.2 Evaluation der Fenstergrößen 1 - 10 min. für das Nachrichtenportal mit der Publisher-ID 35774 bei der CLEF NewsREEL Challenge 2016.

bildung weggelassen. Der zu erkennende Trend hat sich bei größeren Fenstergrößen fortgesetzt.

Die Anzahl der Fälle, in denen nicht genügend verschiedene Artikel zur Empfehlung ermittelt werden konnten, ist in Abbildung 5.2c zu sehen. Wie zu erwarten war, nimmt die Anzahl der Fälle mit steigender Fenstergröße ab. Selbst bei einer Fenstergröße von 1 Minute war dies nur in rund 0.05 % der Fälle der Fall.

Die optimale Fenstergröße hängt natürlich entscheidend von der Datenrate des Nachrichtenportals und der Verteilung der Artikelaufrufe ab. Für die CLEF NewsREEL Challenge 2017 hat sich für die Empfehlung der am häufigsten aufgerufenen Artikel je Nachrichtenportal eine über alle Nachrichtenportale geeignete Fenstergröße von 10 Minuten als optimal herausgestellt. Die zusätzliche Partitionierung nach Geoposition der Leserinnen und Leser hat eine optimale Fenstergröße von 30 Minuten ergeben. Dies ist dadurch zu erklären, dass durch die zusätzliche Partitionierung je Fenster weniger Daten zur Verfügung stehen.

Ergebnisse der CLEF NewsREEL Challenge 2017

Tabelle 5.2 zeigt die Ergebnisse der CLEF-NewsREEL-Challenge 2017, wie sie vom Veranstalter zur Verfügung gestellt wurden. Die Spalte *# Recomm.* gibt die Anzahl der Empfehlungen an, die das Empfehlungssystem im Evaluationszeitraum gegeben hat. Um nicht disqualifiziert zu werden, mussten mindestens 75 % der Empfehlungen des Baseline-Empfehlungssystems *BL2Beat* gegeben werden. Ob dies gelang, ist in der letzten Spalte angegeben. Die Spalte *# Clicks* gibt an, auf wie viele Empfehlungen geklickt wurde. Daraus ergibt sich als CTR das Verhältnis zwischen Anzahl der Empfehlungen und Anzahl der Klicks auf ebendiese. Die Ergebnistabelle ist nach CTR absteigend sortiert.

Von den nicht disqualifizierten Ansätzen haben die Lösungen, die vom Autor dieser Arbeit mit dem hier vorgestellten Konzept umgesetzt wurden, die ersten vier Plätze belegt. Besonders erfolgreich waren die beiden Ansätze, die bereits erfolgreich empfohlene Artikel erneut empfohlen haben (*ody4* und *ody5*). Aber auch die Ansätze *ody3* und *ody2*, die die am häufigsten aufgerufen Nachrichtenartikel empfohlen haben, haben gute Ergebnisse erzielt. Der Ansatz *ody1* hat mit einer größeren Fenstergröße wie erwartet schlechter abgeschnitten.

	Recommender	# Recomm.	# Clicks	CTR	75 % BL
	Riadi_NV_01	443	12	0.0271	no
	ORLY_KS	42,786	896	0.0209	no
1.	**ody4**	72,601	1,139	0.0157	yes
	IRS5	3,708	58	0.0156	no
2.	**ody5**	81,245	1,268	0.0156	yes
3.	**ody3**	59,227	813	0.0137	yes
4.	**ody2**	63,950	875	0.0137	yes
5.	IT5	68,582	925	0.0135	yes
6.	eins	61,524	817	0.0133	yes
7.	yl-2	60,814	747	0.0123	yes
8.	WIRG	49,830	600	0.0120	yes
9.	**ody1**	68,768	810	0.0118	yes
10.	BL2Beat	62,052	726	0.0117	yes
11.	RIADI_pn	77,723	879	0.0113	yes
12.	IL	79,120	813	0,0103	yes
13.	RIADI_nehyb	75,535	764	0.0101	yes
	Has logs	816	6	0.0074	no
	ody0	23,023	166	0.0072	no
	RIADI_hyb	349	2	0.0057	no

Tabelle 5.2 Ergebnisse der CLEF-NewsREEL-Challenge 2017. Die Daten wurden vom Veranstalter zur Verfügung gestellt. Hervorhebungen durch den Autor.

5.3 Zusammenfassung

In diesem Kapitel wurden zwei Szenarien umgesetzt und dadurch entsprechend der hier angewandten Forschungsmethodik (vgl. Abschnitt 1.3, Vorgehen und Aufbau der Arbeit) die Konzepte und das erarbeitete Framework demonstriert und evaluiert.

In dem ersten Abschnitt wurde die Evaluation eines modellbasierten Ansatzes mithilfe eines Datensatzes vorgestellt. Dazu wurde eine Adaption des Matrixfaktorisierungsalgorithmus BRISMF in das Datenstrommanagementsystem integriert. Mit einem Datensatz und der Evaluationsmethode Interleaved Test-Then-Train (ITTT) wurde mit der Datenstromtechnologie ein vergleichbares Setting umgesetzt, welches sich in dem Evaluationstool Massive Online Analysis (MOA) wiederfindet. Der Vergleich der Ergebnisse aus der Evaluation hat gezeigt, dass mit dem Einsatz von Datenstromqueries die Umsetzung eines vergleichbaren Recommender-Systems möglich ist und die Implementierung korrekt ist.

Im zweiten Teil dieses Kapitels wurde die Teilnahme an der CLEF NewsREEL Challenge vorgestellt. Durch die erfolgreiche Teilnahme wurde gezeigt, dass mit dem

vorgestellten Ansatz konkrete Probleme in einem realen Setting gelöst werden können. Dazu wurden verschiedene speicherbasierte Methoden umgesetzt. Durch Zeitfenster wurden Concept Drifts berücksichtigt. Dieser Ansatz hat in der Challenge den ersten Platz belegt.

6 Zusammenfassung und Ausblick

Recommender-Systeme haben das Ziel, automatisiert aus einer großen Menge an Objekten (z. B. Nachrichtenartikel oder Produkte) für eine Person eine Auswahl zu treffen. Diese Auswahl soll das Interesse der Person treffen, für die diese Auswahl berechnet wurde. Eine übliche Methode ist das kollaborative Filtern, bei dem auf historische Bewertungen und Nutzungen zu den Objekten zurückgegriffen wird, um aus diesen das Interesse für eine Person oder eine Personengruppe zu lernen.

Neben dem generellen, langfristigen Interesse von Benutzerinnen und Benutzern spielt häufig auch das kurzfristige Interesse eine Rolle. Dieses spiegelt beispielsweise wider, zu welchen Themen eine Person zum gegenwärtigen Zeitpunkt Nachrichtenartikel lesen möchte oder ob eine Person ein Produkt für sich selbst oder als Geschenk sucht. Um das kurzfristige Interesse während der Berechnungen der Empfehlungen zu berücksichtigen, ist es von Bedeutung, neue Informationen (z. B. Bewertungen oder Views zu Objekten, allgemein „Lerndaten“) möglichst schnell und mit geringer Verzögerung in die Empfehlungsberechnung einfließen zu lassen. Dazu ist es hilfreich, die Lerndaten nicht als statischen Datenbestand, sondern als kontinuierlichen Datenstrom aufzufassen, der on-the-fly zur Berechnung von Empfehlungen oder zum Lernen eines Recommendation-Modells verarbeitet wird. Solche Recommender-Systeme nennt man Online-Recommender-Systeme.

Die Forschung an Datenstrommanagementsystemen beschäftigt sich mit der Umsetzung von Systemen, die Daten kontinuierlich im Datenstrom verarbeiten. Dazu nutzt sie Konzepte aus Datenbankmanagementsystemen und überträgt diese auf die kontinuierliche Datenverarbeitung. In dieser Arbeit wurde das Datenstrommanagementsystem „Odysseus“ eingesetzt, welches als Open-Source-Software zur Verfügung steht, auf Erweiterbarkeit ausgelegt ist und viele Operatoren (z. B. die der relationalen Algebra) bereits integriert hat.

Diese Arbeit hat sich mit der Fragestellung auseinandergesetzt, wie Online-Recommender-Systeme mit der Datenstrommanagementsystem-Technologie umgesetzt werden können. Dazu wurden Konzepte zur Konstruktion von Recommender-Systemen anhand von Datenstromqueries erarbeitet. Diese Konzepte wurden in ein Framework bestehend aus Query-Templates überführt. Anschließend wurden zur Evaluation darauf aufbauend zwei Recommender-Systeme umgesetzt. Eine Umsetzung fand im Rahmen der CLEF-NewsREEL-Challenge statt, mit der sich das hier

vorgestellte Konzept gegenüber denen anderer Teilnehmerinnen und Teilnehmer durchgesetzt und den ersten Platz belegt hat.

Für die Zusammenführung der beiden Forschungsfelder der Datenstromverarbeitung und der Recommender-Systeme stellt diese Arbeit einen ersten Schritt dar. Dabei lag der Fokus auf dem grundsätzlichen Aufbau von Datenstromqueries für Recommender-Systeme. Andere Aspekte wurden nur am Rande betrachtet. So wurde beispielsweise auf die eigentlichen Algorithmen zum Lernen von Recommender-System-Modellen nicht weiter eingegangen, sondern lediglich betrachtet, wie vorhandene Algorithmen in ein Datenstrommanagementsystem integriert werden können. Anknüpfungen für Folgearbeiten bieten zum Beispiel die folgenden Fragestellungen:

Fenster in Online-Recommender-Systemen

In dieser Arbeit wurden Zeitfenster für die Verarbeitung der Lerndaten betrachtet. Diese wurden für den Anwendungsfall der Empfehlung von Nachrichtenartikeln im Rahmen der CLEF NewsREEL Challenge evaluiert. Darauf aufbauend stellt sich die Frage, welchen Einfluss die Fenstergrößen in den verschiedenen Anwendungsfällen von Recommender-Systemen haben. Welche Einflussfaktoren bestimmen die Fenstergröße (Art der Objekte, Datenrate, Anzahl der Personen, explizite vs. implizite Bewertung, ...)? Wie können andere Arten von Fenstern (z. B. Session Windows) umgesetzt werden und wie verhalten sich diese im Vergleich zu Zeitfenstern?

Lernen von Recommender-System-Modellen aus dem Datenstrom

Algorithmen zum Lernen eines Recommender-System-Modells benötigen in der Regel einen wahlfreien Zugriff auf die Daten. Ändert sich die Datenbasis, wird das Modell von Grund auf neu gelernt. Da sich bei der Datenstromverarbeitung der Datenbestand kontinuierlich ändert, ist es wünschenswert, dass neue Daten möglichst schnell einem Modell hinzugefügt werden können und Daten, die aus einem Verarbeitungsfenster entfernt wurden, ohne vollständiges Neulernen in dem Modell keine Berücksichtigung mehr finden. In dieser Arbeit wurde der BRISMF-Algorithmus genutzt, um aus den Lerndaten mit der Methode der Matrixfaktorisierung ein Modell zu lernen. Dieser ist dafür ausgelegt, ein vorheriges Modell zu aktualisieren. Weiterer Forschungsbedarf besteht darin, verschiedene Algorithmen für den Einsatz in einem Datenstrommanagementsystem zu evaluieren und weiterzuentwickeln. Dabei stellen sich die Fragen: Welche Algorithmen sind für den Einsatz im Datenstrom geeignet? Welche können angepasst werden? Wo liegen die Grenzen?

Semantik der Lerndaten / Kontextdaten

In dieser Arbeit wurde die Semantik der Lern- und Kontextdaten weitestgehend nicht berücksichtigt. Im Fokus der Arbeit stand die grundsätzliche

Integration der Recommender-System-Funktionalität. In anschließenden Forschungsarbeiten sollten die Semantik der Daten und die damit verbundenen Besonderheiten bei deren Verarbeitung näher untersucht werden. Dazu könnten folgende Aspekte betrachtet werden: Welche Besonderheiten bestehen bei der Verarbeitung von geospatialen Kontextinformationen? Können Lerndaten in geospatialen Verarbeitungsfenstern gruppiert werden? Welche Besonderheiten bestehen bei der Verarbeitung von temporalen Kontextinformationen? Wie können zyklische Trends, beispielsweise bezogen auf die Tageszeit oder Saison, in der Datenstromverarbeitung berücksichtigt werden?

Zusammenfassend lässt sich sagen, dass diese Arbeit gezeigt hat, wie die Methoden der Recommender-Systeme in die Technologie der Datenstrommanagementsysteme integriert werden können, um Online-Recommender-Systeme auf Basis von Datenstromqueries umzusetzen. Diese Grundlage bietet weitreichende Möglichkeiten, um Forschungsarbeiten an der Schnittstelle zwischen Datenstrommanagementsystemen und Recommender-Systemen anzuschließen.

Literatur

[Aba+05] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska et al. „The Design of the Borealis Stream Processing Engine.“ In: *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research (CIDR 2015)*. Bd. 5. 2005, S. 277–289 (zitiert auf Seite 28).

[Abo+99] Gregory D Abowd, Anind K Dey, Peter J Brown et al. „Towards a better understanding of context and context-awareness“. In: *Handheld and ubiquitous computing*. Springer. 1999, S. 304–307 (zitiert auf Seite 18).

[ABW06] Arvind Arasu, Shivnath Babu und Jennifer Widom. „The CQL continuous query language: semantic foundations and query execution“. In: *The VLDB Journal—The International Journal on Very Large Data Bases* 15.2 (2006), S. 121–142 (zitiert auf den Seiten 5, 26).

[Ado+05] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen und Alexander Tuzhilin. „Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach“. In: *ACM Transactions on Information Systems* 23.1 (2005) (zitiert auf den Seiten 15, 19).

[AJT11] Muqeet Ali, Christopher C Johnson und Alex K Tang. „Parallel collaborative filtering for streaming data“. In: *University of Texas Austin, Tech. Rep* (2011) (zitiert auf den Seiten 5, 7).

[AP07] Charu C Aggarwal und S Yu Philip. „A survey of synopsis construction in data streams“. In: *Data Streams*. Springer, 2007, S. 169–207 (zitiert auf Seite 28).

[App+12] H.-Jürgen Appelrath, Dennis Geesen, Marco Grawunder, Timo Michelsen und Daniela Nicklas. „Odysseus: A Highly Customizable Framework for Creating Efficient Event Stream Management Systems“. In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. DEBS ’12. Berlin, Germany: ACM, 2012, S. 367–368 (zitiert auf den Seiten 5, 6).

[AT11] Gediminas Adomavicius und Alexander Tuzhilin. „Context-Aware Recommender Systems“. In: *Recommender Systems Handbook*. Hrsg. von Francesco Ricci, Lior Rokach, Bracha Shapira und Paul B. Kantor. 10.1007/978-0-387-85820-3_7. Springer US, 2011, S. 217–253 (zitiert auf den Seiten 19, 20).

[Bab+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani und Jennifer Widom. „Models and Issues in Data Stream Systems“. In: *Proceedings of 21st ACM Symposium on Principles of Database Systems (PODS 2002)*. ACM. 2002, S. 1–16 (zitiert auf den Seiten 5, 25, 26).

[BHK98] John S. Breese, David Heckerman und Carl Kadie. „Empirical Analysis of Predictive Algorithms for Collaborative Filtering“. In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (Feb. 1998), S. 43–52 (zitiert auf den Seiten 16, 17).

[Bif+10] Albert Bifet, Geoff Holmes, Richard Kirkby und Bernhard Pfahringer. „MOA: Massive online analysis“. In: *The Journal of Machine Learning Research* 11 (2010), S. 1601–1604 (zitiert auf den Seiten 3, 7, 116, 117).

[BR+11] Ricardo Baeza-Yates, Berthier Ribeiro-Neto et al. *Modern Information Retrieval*. Hrsg. von Ricardo Baeza-Yates und Berthier Ribeiro-Neto. 2nd. Pearson Education Limited, 2011 (zitiert auf Seite 15).

[BRL06] Christopher J. C. Burges, Robert Ragno und Quoc Viet Le. „Learning to Rank with Nonsmooth Cost Functions“. In: *Advances in Neural Information Processing Systems*. 2006, S. 193–200 (zitiert auf Seite 15).

[Bur02] Robin Burke. „Hybrid Recommender Systems: Survey and Experiments“. In: *User Modeling and User-Adapted Interaction* 12.4 (2002), S. 331–370 (zitiert auf den Seiten 14, 16).

[CDC14] Pedro G Campos, Fernando Díez und Iván Cantador. „Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols“. In: *User Modeling and User-Adapted Interaction* 24.1-2 (2014), S. 67–119 (zitiert auf den Seiten 21, 22).

[Cha+11] Badrish Chandramouli, Justin J. Levandoski, Ahmed Eldawy und Mohamed F. Mokbel. „StreamRec: A Real-time Recommender System“. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. ACM. 2011, S. 1243–1246 (zitiert auf den Seiten 5, 6).

[Cha07] Soumen Chakrabarti. „Learning to Rank in Vector Spaces and Social Networks“. In: *Internet Mathematics* 4.2-3 (2007), S. 267–298 (zitiert auf Seite 15).

[Cod70] Edgar F Codd. „A relational model of data for large shared data banks“. In: *Communications of the ACM* 13.6 (1970), S. 377–387 (zitiert auf Seite 26).

[Cra+03] Chuck Cranor, Theodore Johnson, Oliver Spataschek und Vladislav Shkapenyuk. „Gigascope: a stream database for network applications“. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. ACM. 2003, S. 647–651 (zitiert auf Seite 28).

[CSS99] William W. Cohen, Robert E. Schapire und Yoram Singer. „Learning to Order Things“. In: *Journal of Artificial Intelligence Research* 10 (1999), S. 243–270 (zitiert auf Seite 15).

[Dia+12] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme und Wolfgang Nejdl. „Real-Time Top-N Recommendation in Social Streams“. In: *Proceedings of the Sixth ACM conference on Recommender Systems*. 2012, S. 59–66 (zitiert auf Seite 7).

[Gam+13] Joao Gama, Indre Zliobaite, Albert Biefet, Mykola Pechenizkiy und Abdelhamid Bouchachia. „A Survey on Concept Drift Adaptation“. In: *ACM Comp. Surveys* 1.1 (2013) (zitiert auf Seite 21).

[Gol+92] David Goldberg, David Nichols, Brian M. Oki und Douglas Terry. „Using collaborative filtering to weave an information tapestry“. In: *Communications of the ACM* 35.12 (1992), S. 61–70 (zitiert auf Seite 13).

[Her+04] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen und John T. Riedl. „Evaluating Collaborative Filtering Recommender Systems“. In: *ACM Transactions on Information Systems (TOIS)* 22.1 (2004), S. 5–53 (zitiert auf den Seiten 21–23).

[Hop+14] Frank Hopfgartner, Benjamin Kille, Andreas Lommatzsch et al. „Benchmarking News Recommendations in a Living Lab“. In: *CLEF'14*. LNCS. Sheffield, UK: Springer Verlag, Sep. 2014, S. 250–267 (zitiert auf Seite 4).

[Hop+16] Frank Hopfgartner, Torben Brodt, Jonas Seiler et al. „Benchmarking News Recommendations: The CLEF NewsREEL Use Case“. In: *ACM SIGIR Forum*. Bd. 49. 2. ACM. 2016, S. 129–136 (zitiert auf Seite 120).

[HS00] Andreas Heuer und Gunter Saake. *Datenbanken: Konzepte und Sprachen*. mitp Verlags GmbH & Co. KG, 2000 (zitiert auf Seite 24).

[JJK18] Michael Jugovac, Dietmar Jannach und Mozhgan Karimi. „Streamingrec: a framework for benchmarking stream-based news recommenders“. In: *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM. 2018, S. 269–273 (zitiert auf Seite 7).

[Joa02] Thorsten Joachims. „Optimizing Search Engines using Clickthrough Data“. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2002, S. 133–142 (zitiert auf Seite 15).

[Jon72] Karen Spärck Jones. „A statistical interpretation of term specificity and its application in retrieval“. In: *Journal of Documentation* 28.1 (1972), S. 11–21 (zitiert auf Seite 16).

[KBV09] Yehuda Koren, Robert Bell und Chris Volinsky. „Matrix factorization techniques for recommender systems“. In: *Computer* 42.8 (2009), S. 30–37 (zitiert auf Seite 17).

[Krä07] Jürgen Krämer. „Continuous Queries over Data Streams – Semantics and Implementation“. Diss. University of Marburg, 2007 (zitiert auf den Seiten 25, 26, 30).

[KS09] Jürgen Krämer und Bernhard Seeger. „Semantics and implementation of continuous sliding window queries over data streams“. In: *ACM Transactions on Database Systems 2009* 34.1 (2009), S. 4 (zitiert auf den Seiten 5, 28).

[LGA15] Cornelius A Ludmann, Marco Grawunder und Hans-Jürgen Appelrath. „OdysseusRecSys: Collaborative Filtering based on a Data Stream Management System.“ In: *Proceedings of the 23rd Conference on User Modeling, Adaptation, and Personalization (UMAP 2015)*. 2015 (zitiert auf Seite 8).

[Lom+17] A. Lommatzsch, B. Kille, F. Hopfgartner et al. „CLEF 2017 NewsREEL Overview: A Stream-Based Recommender Task for Evaluation and Education“. In: *CLEF 2017*. Dublin, Ireland: Springer Verlag, 2017 (zitiert auf Seite 120).

[Lü+12] Linyuan Lü, Matúš Medo, Chi Ho Yeung et al. „Recommender Systems“. In: *Physics Reports* 519.1 (2012), S. 1–49 (zitiert auf Seite 22).

[Lud+15] Cornelius A Ludmann, Marco Grawunder, Timo Michelsen, H Appelrath et al. „Kontinuierliche Evaluation von kollaborativen Recommender-Systeme in Datenstrommanagementsystemen“. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2015)-Workshopband* (2015) (zitiert auf Seite 8).

[Lud15] Cornelius A Ludmann. „Online recommender systems based on data stream management systems“. In: *Proceedings of the 9th ACM Conference on Recommender Systems*. ACM. 2015, S. 391–394 (zitiert auf Seite 8).

[Lud16] Cornelius Ludmann. „Lessons Learned from Using a Data Stream Management System for Real-time Recommendation of Popular News Articles based on Real User Streams“. In: *LSRS at 10th ACM Conference on Recommender Systems*. Boston, USA, 2016 (zitiert auf den Seiten 8, 120, 127).

[Lud17a] Cornelius Ludmann. „Recommending News Articles in the CLEF News Recommendation Evaluation Lab with the Data Stream Management System Odysseus“. In: *CLEF Conference and Labs of the Evaluation Forum 2017*. Dublin, Ireland: Springer Verlag, 2017 (zitiert auf den Seiten 8, 120, 122).

[Lud17b] Cornelius A Ludmann. „Einsatz eines Datenstrommanagementsystems als Framework für Online-Recommender-Systeme am Beispiel der Nachrichtenempfehlungen“. In: *Datenbank-Spektrum* 17.3 (2017), S. 267–276 (zitiert auf Seite 8).

[Lud17c] Cornelius A Ludmann. „Einsatz eines Datenstrommanagementsystems zur Empfehlung von beliebten Nachrichtenartikeln in der CLEF NewsREEL Challenge“. In: *Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband* (2017) (zitiert auf Seite 8).

[LXZ12] Xin Luo, Yunni Xia und Qingsheng Zhu. „Incremental collaborative filtering recommender based on regularized matrix factorization“. In: *Knowledge-Based Systems* 27 (2012), S. 271–280 (zitiert auf Seite 7).

[Pef+07] Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger und Samir Chatterjee. „A Design Science Research Methodology for Information Systems Research“. In: *Journal of Management Information Systems* 24.3 (2007) (zitiert auf den Seiten 8, 9).

[PS06] Kostas Patroumpas und Timos Sellis. „Window specification over data streams“. In: *Current Trends in Database Technology–EDBT 2006*. Springer, 2006, S. 445–464 (zitiert auf Seite 28).

[QCJ18] Massimo Quadrana, Paolo Cremonesi und Dietmar Jannach. „Sequence-Aware Recommender Systems“. In: *ACM Comput. Surv.* 51.4 (Juli 2018), 66:1–66:36 (zitiert auf Seite 7).

[Res+94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom und John Riedl. „GroupLens: an open architecture for collaborative filtering of netnews“. In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM. 1994, S. 175–186 (zitiert auf Seite 13).

[Scu+15] D Sculley, Gary Holt, Daniel Golovin et al. „Hidden technical debt in machine learning systems“. In: *Advances in Neural Information Processing Systems*. 2015, S. 2503–2511 (zitiert auf Seite 4).

[SG11] Guy Shani und Asela Gunawardana. „Evaluating Recommendation Systems“. In: *Recommender Systems Handbook*. Hrsg. von Francesco Ricci, Lior Rokach, Bracha Shapira und Paul B. Kantor. Springer, 2011, S. 257–297 (zitiert auf Seite 22).

[ST94] Bill N. Schilit und Marvin M. Theimer. „Disseminating Active Map Information to Mobile Hosts“. In: *IEEE Network* 8.5 (1994), S. 22–32 (zitiert auf Seite 18).

[Tak+09] Gábor Takács, István Pilászy, Bottyán Németh und Domonkos Tikk. „Scalable collaborative filtering approaches for large recommender systems“. In: *JMLR* 10 (2009), S. 623–656 (zitiert auf den Seiten 3, 7, 55, 71, 91, 115, 116).

[Tuc+03] Peter Tucker, David Maier, Tim Sheard, Leonidas Fegaras et al. „Exploiting punctuation semantics in continuous data streams“. In: *Knowledge and Data Engineering, IEEE Transactions on* 15.3 (2003), S. 555–568 (zitiert auf Seite 30).

[Xie+16] Xiaolong Xie, Wei Tan, Liana L Fong und Yun Liang. „Cumf_sgd: Fast and scalable matrix factorization“. In: *arXiv preprint arXiv:1610.05838* (2016) (zitiert auf den Seiten 18, 91).

Abbildungsverzeichnis

Quelltextverzeichnis

Tabellenverzeichnis