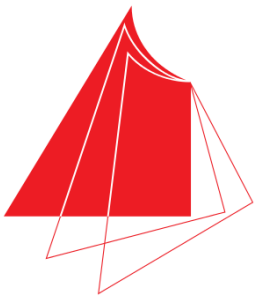


Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Recommendersysteme mit Elasticsearch

Masterthesis im Fach Informatik

Anna Roes

Karlsruhe, 30. März 2015

Referent: Prof. Dr. Heiko Körner

Korreferent: Prof. Dr.-Ing. Astrid Laubenheimer

Inhaltsverzeichnis

1 Einführung

Die vorliegende Masterthesis entstand bei der *inovex GmbH* in Karlsruhe und untersucht, wie sich Recommendersysteme zur Generierung von Empfehlungen für Benutzer mit *Elasticsearch* umsetzen lassen. In den folgenden Abschnitten wird zunächst die Firma vorgestellt, um anschließend die Motivation für die Arbeit sowie deren Ziele zu erläutern.

1.1 inovex

Die *inovex GmbH* wurde 1999 in Pforzheim gegründet und ist bis heute inhabergeführt und fremdkapitalfrei. An den vier Standorten Karlsruhe, Pforzheim, München und Köln arbeiten mehr als 150 Mitarbeiter.[1] Als IT-Projekthaus unterstützt *inovex* Unternehmen wie die 1&1 Internet AG, maxdome, Bosch, Daimler und EnBW bei Enterprise-IT-Aufgaben in den Bereichen Consulting, Application Development, Data Management & Analytics sowie IT Engineering & Operations.[2]

Daneben bietet *inovex* Schulungen an und die Mitarbeiter geben ihr Wissen auf Fachkonferenzen und in Fachzeitschriften weiter.[3]

1.2 Motivation

Bereits seit Mitte der 90er Jahre bilden Recommendersysteme ein eigenständiges Forschungsgebiet an der Schnittstelle von Maschinenlernen, Data Mining, Information Retrieval und Mensch-Maschine-Interaktion, das folgende Fragestellungen berührt:

[1] `https://www.inovex.de/de/ueber-uns/`
[2] `https://www.inovex.de/de/referenzen/kunden/`, `https://www.inovex.de/de/ueber-uns/abteilungen/`
[3] `https://www.inovex.de/de/leistungen/trainings/`, `https://www.inovex.de/de/content-pool/`

Maschinenlernen Wie lassen sich aus Daten Modelle gewinnen, die zur Vorhersage oder als Grundlage für Entscheidungen geeignet sind? Wie lassen sich Muster erkennen, die die Einordnung von bislang Unbekanntem in bekannte Kategorien erlauben?

Data Mining Wie lassen sich aus großen Datenmengen neue und nützliche Informationen gewinnen?

Information Retrieval Informationsrückgewinnung: Wie lassen sich Informationen in unstrukturierten Daten ganz unterschiedlicher Art möglichst schnell zugänglich machen? Wie geht man mit komplexen Daten und/oder vagen Suchkriterien um? Wie lassen sich die Ergebnisse nach Relevanz ordnen?

Mensch-Maschine-Interaktion Wie lassen sich interaktive Systeme so gestalten, dass sie benutzungsfreundlich und intuitiv sind?

Je nachdem welcher Aspekt von Recommendersystemen betrachtet wird, ist man näher am einen oder anderen Gebiet. Wo es um die Präsentation der Empfehlungen für den Nutzer geht oder um die Möglichkeiten des Nutzers, durch Auswahl oder andere Eingaben Einfluss auf die Empfehlungen zu nehmen, sind Methoden und Erkenntnisse der Mensch-Maschine-Interaktion gefragt. Sollen einem Nutzer Texte empfohlen werden, die aufgrund ihres Inhalts interessant für ihn sind, dann sind Verfahren des Information Retrieval nützlich, um Textinhalte zu analysieren und zu vergleichen. Welche Artikel für einen Nutzer interessant sein könnten, muss das System erst lernen. Dabei werden Algorithmen aus dem Bereich Maschinenlernen eingesetzt. Die Daten, auf deren Grundlage dieses Lernen erfolgt, müssen häufig vorbereitet werden, wobei Informationen über die Gruppenzugehörigkeit von Nutzern oder Mengen von ähnlichen Gegenständen erst gewonnen werden. Hierbei helfen Methoden des Data Mining.

Das lässt bereits erahnen, dass die Grenzen zwischen dem Forschungsbereich Recommendersysteme und den anderen Bereichen je nach Betrachtungswinkel und Schwerpunktsetzung fließend sind. So ist es zum Beispiel nicht so einfach, zwischen einer „aufgepeppten“ Suchmaschine, die dem klassischen Information Retrieval zuzuordnen ist, und einem Recommendersystem, das auf Nutzereingaben reagiert, zu unterscheiden. Ab wann sind die Personalisierung und der Einfluss von Nutzervorlieben auf die Suchergebnisse und deren Rangfolge stark genug, damit der Bereich des reinen Information Retrieval verlassen wird? Burke [1, S. 377] versucht, diese Grenze anhand semantischer Kriterien und des Personalisierungsgrades sowie der handeln-

den Instanz zu ziehen: Für ihn zeichnet sich ein Recommendersystem dadurch aus, dass dessen zurückgelieferte Ergebnisse als Empfehlungen gesehen werden, die es zu überdenken gilt, und nicht als Antwort auf eine Nutzeranfrage. Generell wird, so seine Argumentation, von einem Recommendersystem mehr erwartet als die Reaktion auf Anfragen. Es soll eine informierende Instanz sein, die hochgradig personalisierte Empfehlungen ausspricht. Ein Blick auf die Nutzungsszenarien moderner Suchmaschinen zeigt: So klar sind die Grenzen nicht (mehr). Die Ergebnisse, die Google als Antwort auf eine Nutzeranfrage zurückliefert, sind auch personalisiert. Umso stärker natürlich, je mehr Kontext- und Nutzerinformationen verwendet werden können. Aber auch im „normalen“ Betrieb und ohne dass der Nutzer bei Google angemeldet ist, sorgen Cookies und Browsereinstellungen dafür, dass die zurückgelieferten Ergebnisse durchaus als Empfehlungen angesehen werden können, welche Seiten ein Nutzer, der jene Spracheinstellungen vorgenommen hat und sich außerdem in dieser Region befindet, besuchen sollte, um Antworten auf seine Fragen oder Informationen zum gesuchten Thema zu erhalten. Allerdings wird längst nicht alles ausgeschöpft, was möglich wäre. In Kapitel 18 des RS-Handbuchs [2] stellen Smyth et al. mögliche Erweiterungen der klassischen Websuche vor, in denen zum Beispiel soziale Gruppen und Nutzer mit ähnlichen Interessen eine große Rolle spielen.

1.3 Zielsetzung

Die vorliegende Arbeit verknüpft das Forschungsgebiet Recommendersysteme mit der aktuell vielbeachteten und im industriellen Umfeld beliebten Open-Source-Suchmaschine *Elasticsearch*.

Elasticsearch ist für den Einsatz auf verteilten Systemen gebaut und erlaubt es, große Datenmengen nahezu in Echtzeit zu verarbeiten. Eingesetzt wird *Elasticsearch* sowohl zur Volltextsuche, als auch zur Suche in strukturierten Daten und zu Analysezwecken. Auch Suchwortvervollständigung, Vorschläge („Meinten Sie vielleicht ...“) und die Anzeige verwandter Einträge lassen sich mit *Elasticsearch* umsetzen und werden häufig verwendet. In vielen Anwendungskontexten sind darüber hinaus personalisierte Empfehlungen wünschenswert und für die Anbieter der entsprechenden Dienste wäre es natürlich von Vorteil, wenn sich ein solches Empfehlungssystem direkt mit *Elasticsearch* und möglichst ohne großen Aufwand als Plug-and-Play-Lösung umsetzen ließe.

Wie sich dies erreichen lässt, ist das Thema dieser Masterthesis.

1.4 Aufbau

Die beiden auf diese Einleitung folgenden Kapitel führen in die Bereiche Recommendersysteme und *Elasticsearch* ein. Kapitel 2 gibt einen Überblick über die Aufgaben von Recommendersystemen sowie die gängigen Methoden, wobei insbesondere inhaltsbasierte Empfehlungen und kollaboratives Filtern im Vordergrund stehen, und stellt abschließend anhand zweier Fallbeispiele die Arbeitsweise bekannter moderner Recommendersysteme vor. In Kapitel 3 geht es um *Elasticsearch*. Auch hier werden zunächst Grundlagen und Basisfunktionalitäten erläutert. Der Suche und deren Funktionsweise ist ein ganzes Unterkapitel gewidmet, ebenso den Aggregationen. Das im Rahmen der Thesis entstandene *Elasticsearch*-Plugin wird in Kapitel 4 vorgestellt. Anschließend folgt in Kapitel 5 ein Überblick über Ansätze zur Evaluation der generierten Empfehlungen und in Kapitel 6 dann die tatsächliche Auswertung. Die Arbeit schließt mit einem Fazit, in dem der aktuelle Stand mit den Zielen der Arbeit abgeglichen wird. Außerdem werden Weiterentwicklungsmöglichkeiten und offene Fragen aufgezeigt.

2 Recommendersysteme

Jeder, der schon einmal in einem größeren Onlineshop eingekauft oder in einer Online-Mediathek einen Film oder ein Musikstück gesucht hat, hatte schon mit Recommendersystemen zu tun. Solche Systeme generieren Empfehlungen für den Nutzer. Im simpelsten Fall empfehlen sie, was allen gefällt: „Die Top-10-Downloads diese Woche sind ...“, sehr häufig sind aber auch auf den Nutzer zugeschnittene Empfehlungen zu finden: „Wem „Firefly – Der Aufbruch der Serenity“ gefällt, der mag auch „Serenity – Flucht in neue Welten“, „Dieser Staubsauger wird oft zusammen gekauft mit dem Zehnerpack Staubsaugerbeutel“.

Dieses Kapitel beschreibt die Aufgaben und die Arbeitsweise von Recommendersystemen, stellt gängige Kaltstartprobleme vor und gibt einen kurzen Überblick über den aktuellen Stand der Technik: Welche der theoretisch relevanten Ansätze kommen in der Praxis überhaupt zum Einsatz? Welche Verfahren nutzen Unternehmen wie Netflix[4] oder Amazon[5]?

2.1 Grundlagen

Die Hauptaufgabe eines Recommendersystems ist es, aus einem großen Angebot all jenes herauszusuchen, was für einen bestimmten Nutzer interessant ist. In welchem Kontext und auf welche Art und Weise diese Aufgabe erfüllt wird, ist unterschiedlich. Abschnitt 2.1.2 gibt einen Überblick über die Kriterien, nach denen Recommendersysteme klassifiziert werden können. Welche Aufgaben solche Systeme neben ihrer eigentlichen Hauptaufgabe noch erfüllen, ist das Thema von Abschnitt 2.1.3, einen genaueren Überblick über die Methoden, die bei der Erstellung der Empfehlungen zum Einsatz kommen, liefert Abschnitts 2.2. Zuvor müssen aber einige terminologische Fragen geklärt werden.

[4] `www.netflix.com`
[5] `www.amazon.de`

2.1.1 Terminologie

Da der Ausdruck auch im Deutschen zwischenzeitlich recht verbreitet ist, werden Empfehlungsdienste in dieser Arbeit als „Recommendersysteme“ bezeichnet. „Anbieter“ sind diejenigen, die solche Dienste bereitstellen, gerichtet sind die Empfehlungen an die „Nutzer“. In Ermangelung eines ähnlich griffigen Wortes wie „item“ im Englischen sprechen wir von „Artikeln“, die empfohlen werden.

2.1.2 Taxonomie von Recommendersystemen

Recommendersysteme sind sehr verbreitet und werden zu ganz unterschiedlichen Zwecken und in verschiedenen Kontexten eingesetzt. Entsprechend unterscheiden sich auch die verwendeten Verfahren, die Art der Präsentation der Empfehlungen sowie die verwendeten Daten. Um die große Masse an bestehenden Systemen ordnen zu können, schlagen Joseph A. Konstan und Michael D. Ekstrand von der Universität Minnesota in ihrem Onlinekurs „Introduction to Recommender Systems“ [3] die folgenden acht Analysedimensionen vor:

1. Bereich: Was wird empfohlen? Handelt es sich um textuelle Information irgendeiner Art, um Produkte bzw. Produktpakete (Flug plus Hotel plus Mietwagen), um Personen (soziale Netwerke, Partnerbörsen) oder um ganze Folgen von Artikeln (alle Alben einer Band, die Fortsetzungsbände eines Romans)? Werden nur neue Artikel empfohlen oder auch solche, die der Nutzer bereits kennt/erworben hat (Letzteres kann zum Beispiel bei Verbrauchsartikeln sinnvoll sein)?

2. Zweck: Sollen die Empfehlungen die Verkaufszahlen erhöhen, den Nutzer informieren oder weiterbilden, gemeinschaftsfördernd wirken? (Vgl. hierzu auch 2.1.3.)

3. Kontext der Empfehlung: Was tut der Nutzer zur Zeit der Empfehlung? Beschränkt seine Tätigkeit das Recommendersystem, zum Beispiel, weil das Anzeigen der Empfehlungen einen Arbeitsfluss unterbrechen würde? Erfolgen die Empfehlungen explizit und werden dem Nutzer zum Beispiel in Form einer Liste angezeigt, oder folgt der Nutzer den Empfehlungen automatisch, wie etwa bei *lastfm*[6] oder *Pandora*[7], wo ausgehend vom aktuell gespielten Musikstück

[6] `http://www.lastfm.de/`
[7] `http://www.pandora.com/`

eine neue Empfehlung generiert und das entsprechende Stück im Anschluss automatisch abgespielt wird?

4. Meinungen: Wessen Meinung wird zur Generierung von Empfehlungen herangezogen? Die von Experten? Die von (allen) anderen Nutzern? Werden nur Nutzer berücksichtigt, die dem aktuellen Nutzer in irgendeiner Form ähnlich sind?

5. Personalisierung: Werden nur generische Empfehlungen generiert, die für alle Nutzer gleich sind (Top 10-Listen), sind die Empfehlungen auf eine bestimmte Zielgruppe zugeschnitten (zum Beispiel nach demographischen Gesichtspunkten), handelt es sich um Empfehlungen, die aufgrund kurzfristiger (aktuell betrachteter Artikel) oder langfristiger Interessen (Kaufhistorie) berechnet wurden?

6. Privatsphäre/Vertrauen: Welche Nutzerinformationen werden wo zu welchem Zweck gespeichert? In welchem Maße wird der Nutzer darüber informiert und kann Einfluss nehmen (Transparenz)? Wie wahrscheinlich ist externe Manipulation? Sind die generierten Empfehlungen „ehrlich“ oder von Geschäftsinteressen beeinflusst?

7. Interfaces: Welche Art von Output wird produziert? Handelt es sich um Vorhersagen (zum Beispiel der Nutzerwertung oder des Kaufverhaltens), um Empfehlungen im eigentlichen Sinne oder eher um das Filtern einer Suchliste? Wird der Nutzer explizit mit den Empfehlungen konfrontiert? Kann er Einfluss nehmen (System als „Agent“)?

   Welche Art von Input verarbeitet das System? Werden nur explizite Bewertungen berücksichtigt oder werden aus dem Nutzerverhalten implizite Bewertungen generiert?

8. Algorithmen: Welche Verfahren werden verwendet, um Empfehlungen zu generieren? (Vgl. Abschnitt 2.2.)

2.1.3 Funktionen

Im vorhergehenden Abschnitt wurden bereits einige Funktionen genannt, die ein Recommendersystem (RS) erfüllen kann, und zwar sowohl für den Anbieter des

Systems als auch für den Nutzer. Diese werden im Folgenden zusammen mit weiteren Funktionen noch einmal übersichtlich aufgelistet, wobei ich mich an [2, Kap. 1, S. 4 ff.] orientiere.

- Das RS kann umsatzsteigernd wirken.
- Der Anbieter kann das Interesse der Nutzer auf ein breiteres Spektrum von Artikeln lenken.
- Im Idealfall erhöht sich die Nutzerzufriedenheit und die Nutzungsfrequenz.
- Gute Empfehlungen erhöhen die Bindung des Nutzers an den Anbieter.
- Ein RS erlaubt es, die Vorlieben der Nutzer besser kennen zu lernen und passende Angebote/Aktionen anzubieten.
- Es hilft dem Nutzer, mit dem Überangebot an Informationen/Waren umzugehen.
- Das RS lenkt im Idealfall den Fokus auf für den Nutzer relevante Artikel (alle oder eine Auswahl, je nach Kontext).
- Neben verwandten Artikeln können auch ganze Pakete empfohlen werden, wie etwa die passenden Flüge zum ausgesuchten Hotelaufenthalt, ein ergänzendes Mietwagenangebot oder Ähnliches.
- Der Nutzer wird beim Durchsuchen des Angebots unterstützt und stößt dabei im Idealfall auch auf überraschende Artikel, möglicherweise aus Gebieten, mit denen er vorher nicht viel zu tun hatte.
- Nicht zuletzt kann ein RS für den Nutzer auch eine soziale Funktion erfüllen: Er kann sich selbst ausdrücken, sein Profil verbessern, eine Meinung abgeben und anderen Nutzern helfen.

Aus diesen Beschreibungen geht auch hervor, dass sich der Nutzen von Recommendersystemen nicht auf Verkaufsangebote beschränkt. Auch wenn kein Geld zwischen Nutzer und Anbieter fließt, können Recommendersysteme die Kundenbindung erhöhen und sich positiv auf die Anzahl der Klicks und die Streuung derselben auswirken, was zum Beispiel für werbefinanzierte Webseiten von Vorteil sein kann.

2.1.4 Modellierung

Damit dieser Nutzen tatsächlich zum Tragen kommt, muss das RS auf den Bereich, in dem es eingesetzt wird, zugeschnitten sein. Bei der Empfehlung von Zeitungsartikeln müssen andere Dinge beachtet werden als bei der Empfehlung von Finanzanlagen. Außerdem können unterschiedliche Informationen über die Nutzer verwendet werden. In den folgenden Absätzen soll die Modellierung von Gegenständen, Nutzern sowie der Interaktion zwischen Nutzer und RS näher besprochen werden, orientiert an [2, S. 7 ff.].

Artikel

Die zu empfehlenden Artikel können von unterschiedlicher Komplexität sein (CD mit 15 Musikstücken vs. Portfolio mit Finanzanlagen, die alle eine gewissen Laufzeit haben und unterschiedlich riskant sind). Für den Nutzer haben diese Artikel einen bestimmten Wert oder Nutzen, allerdings entstehen ihm auch Kosten für die Verwendung bzw. Inbesitznahme derselben. Diese Kosten können finanzieller Art sein, wenn es sich bei dem Gegenstand um eine Ware handelt, die gekauft wird, aber auch kognitiver Art, wie etwa beim Lesen eines Nachrichtenartikels.

Wie die Artikel innerhalb des Recommendersystems repräsentiert werden, ist einerseits natürlich davon abhängig, um welche Art von Artikel es sich handelt, andererseits auch davon, welches Verfahren zur Generierung von Empfehlungen eingesetzt wird. Einige Verfahren, wie zum Beispiel das klassische kollaborative Filtern (vgl. 2.2.3), kommen mit wenig Information aus. Zur Berechnung von Empfehlungen reicht es, wenn jeder Artikel durch eine eindeutige ID repräsentiert wird. Inhaltsbasierte Verfahren hingegen benötigen sehr viel mehr Informationen, um arbeiten zu können (vgl. 2.2.2). Zur Artikelbeschreibung kann dann eine Menge von Attributen verwendet werden. Einige Ansätze gehen sogar noch weiter und verwenden ausgefeilte ontologische Konzepte zur Repräsentation der zu empfehlenden Artikel. Neben den Produkteigenschaften können zusätzlich auch nutzergenerierte Schlagworte (*tags*) zur Empfehlungsberechnung herangezogen werden, wenn es den Nutzern möglich ist, solche anzulegen.

Nutzer

Auch die Repräsentation der Nutzer im Recommendersystem ist von der Art der verwendeten Ansätze abhängig. In jedem Fall erhält der Nutzer eine eindeutige ID. Wenn die Empfehlungen nicht personalisiert sind, sondern zum Beispiel nur die meistverkauften Produkte empfohlen werden, müssen darüber hinaus im Nutzerprofil keine Daten gespeichert werden. Für Empfehlungen, die nicht wirklich personalisiert, aber auf bestimmte Zielgruppen zugeschnitten sind, werden soziodemographische Daten des Nutzers benötigt, die im Nutzerprofil gespeichert werden. Bei Ansätzen, die auf Vertrauensverhältnisse zwischen Nutzern bauen („Ihren Freunden gefällt ...“), müssen die Beziehungen zwischen den Nutzern gespeichert werden.

Für personalisierte Empfehlungen muss das System über Informationen verfügen, welche Artikel dem Nutzer gefallen. Die Interaktionen des Nutzers mit dem System (vgl. Abschnitt 2.1.4) geben Aufschluss darüber, an welchen Artikeln der Nutzer interessiert ist. Herangezogen werden können Bewertungen, die der Nutzer explizit vorgenommen hat, aber auch solche, die sich nur implizit dadurch zeigen, dass zum Beispiel ein Lesezeichen gesetzt oder ein bestimmtes Produkt gesucht, gekauft, betrachtet oder in den Warenkorb gelegt wird. Werden kollaborative Filter verwendet, um die Empfehlungen zu generieren, dann werden die Bewertungen des Nutzers in der Regel nicht im Nutzerprofil, sondern in einer für alle Artikel und Nutzer gemeinsamen, in der Regel sehr dünn besetzten Matrix gespeichert (vgl. 2.2.3).

Anders sieht es bei inhaltsbasierten Empfehlungen aus. Bei diesen Verfahren wird für jeden Nutzer in dessen Profil ein Vektor gespeichert, in dem kodiert ist, wie stark ihn bestimmte Eigenschaften von Artikeln ansprechen, um diesen für den Nutzer prototypischen Vektor dann mit den Eigenschaftsvektoren von Artikeln vergleichen zu können.

Häufig werden mehrere Ansätze verwendet, um Empfehlungen für den Nutzer zu generieren (vgl. hierzu auch 2.4). Entsprechend müssen dann auch mehr und unterschiedliche Daten gespeichert werden.

Transaktion

Als Transaktionen werden alle aufgezeichneten Interaktionen zwischen Nutzern und dem Recommendersystem bezeichnet.

Zu diesen Interaktionen gehören zum Beispiel die oben bereits angesprochenen expliziten und impliziten Bewertungen, die der Nutzer vornimmt und die eine Beziehung zwischen Nutzer und Artikel herstellen. Diese werden gespeichert und fließen dann wie oben erläutert in die Nutzerprofile und/oder die Generierung der Empfehlungen ein. Häufig werden dazu aber nicht die Rohdaten verwendet, sondern diese werden zuvor normiert und/oder gewichtet. So sind zum Beispiel explizite Bewertungen entweder binär (gefällt mir – gefällt mir nicht) oder auf einer Skala angesiedelt oder es können nur positive Bewertungen, etwa in Form von „Likes“, abgegeben werden. Die impliziten Bewertungen müssen auf die jeweilige Skala umgerechnet werden, um mit den expliziten vergleichbar zu sein und verrechnet werden zu können. Eine andere zu berücksichtigende Größe sind Verzerrungen in den Daten. So gibt es zum Beispiel Nutzer, die die Bewertungsskala nie voll ausschöpfen, deren Bewertungen aber dennoch mit denen von Nutzern, die die ganze Bandbreite von Bewertungen vergeben, vergleichbar sein sollen. Mehr dazu in späteren Abschnitten.

Weitere Transaktionen sind etwa das Versehen von Artikeln mit Schlagworten sowie Kommentare oder Rezensionen zu Artikeln. Diese können verwendet werden, um die Eigenschaften des betreffenden Artikels zu erweitern, oder sie können in das Nutzermodell einfließen, um die Präferenzen des Nutzers zu präzisieren.

Neben den genannten Transaktionen, die der Nutzer in der Regel gar nicht als Interaktion mit dem Recommendersystem wahrnimmt, gibt es auch noch Interaktionen, bei denen der Nutzer wissentlich Einfluss auf die generierten Empfehlungen nimmt. Dies geschieht zum Beispiel, wenn Artikel explizit von der Empfehlungsgenerierung ausgenommen werden können (möglich zum Beispiel bei *Amazon*) oder wenn der Nutzer bei einem sogenannten wissens- oder fallbasierten Empfehlungssystemen Einschränkungen vorgibt („Empfiehl mir Kameras, die weniger als 300€ kosten und einen optischen Zoom haben“) und gegebenenfalls auf Nachfragen des Systems reagiert, die dazu dienen, die generierten Emfpehlungen zu verfeinern.

Kontext

Je nachdem, welche Artikel empfohlen werden sollen, kann es wichtig sein, Kontextinformationen zur Generierung von Empfehlungen heranzuziehen. Bei Restaurantempfehlungen oder der Empfehlung von Ausflugszielen/-paketen können zum Beispiel der aktuelle Aufenthaltsort, die Uhrzeit, aber auch die Jahreszeit eine wichtige Rolle spielen.

2.2 Methoden

Seit Beginn der Forschungen zu Recommendersystemen wurden verschiedene Empfehlungsverfahren vorgeschlagen und getestet, von denen sich kollaboratives Filtern (*collaborative filtering*) und inhaltsbasierte Empfehlungen (*content based recommendation*) als De-facto-Standards durchgesetzt haben. Spätestens durch den Netflix-Wettbewerb 2006[8], durch den ein verbesserter Filmempfehlungsalgorithmus für die Onlinemediathek Netflix gefunden werden sollte, kam wieder Bewegung in die Forschung und es wurden eine Reihe weiterer Ansätze vorgestellt, die aber größtenteils noch nicht im großen Stil eingesetzt werden. Die verbreitetsten Ansätze, orientiert an Ricci, Rokach und Shapira [2, S. 10 ff.] sowie [3, VL 1.4], sind:

1. Nicht-personalisiert: Die Empfehlungen sind für alle Nutzer gleich. Empfohlen werden zum Beispiel die beliebtesten Artikel.
2. Inhaltsbasiert: Die Empfehlung beruht auf Inhalten/Eigenschaften, empfohlen werden Gegenstände, die denen ähneln, die der Nutzer mag.
3. Kollaboratives Filtern: Was Nutzern mit ähnlichem Geschmack gefällt, wird empfohlen.
4. Demographisch: Die Empfehlung beruht auf soziodemographischen Daten.
5. Wissensbasiert: Empfehlungen werden aufgrund von Fachwissen darüber ausgesprochen, wie bestimmte Produkteigenschaften sich auf die Befriedigung von Kundenbedürfnissen und -vorlieben auswirken. Dabei werden Ähnlichkeitsfunktionen oder Wissensdatenbanken mit expliziten Regeln genutzt.
6. Gemeinschaftsbasiert: Empfehlungen von Freunden werden besser angenommen als von anonymen Nutzern, daher werden Netzwerke genutzt nach dem Motto „Sag mir, wer deine Freunde sind, und ich sage dir, wer du bist“.
7. Hybrid: Kombinationen der genannten Ansätze werden zur Empfehlung genutzt.

In den folgenden Abschnitten werden diese Ansätze vorgestellt. Der Schwerpunkt liegt dabei auf inhaltsbasierten Methoden und dem kollaborativen Filtern, weil diese besonders weit verbreitet sind und in der Vergangenheit gute Ergebnisse geliefert haben.

[8] `http://www.netflixprize.com/rules`

2.2.1 Nicht-personalisierte Empfehlungen

Die nicht-personalisierten Empfehlungen sind für die vorliegende Arbeit von untergeordneter Bedeutung, weil sie nicht den einzelnen Nutzer und dessen Vorlieben berücksichtigen. Die Empfehlungen, die generiert werden, sind für alle Nutzer gleich. Beispiele umfassen Bestseller-Listen, die Empfehlung der k bestbewerteten Artikel, aber auch die Anzeige von Überblicksstatistiken: 98% der Nutzer, die dieses Hotel bewertet haben, empfehlen es weiter; dieser Artikel wird durchschnittlich mit $3,75$ Sternen bewertet, wobei sich die Bewertungen wie folgt verteilen ...

Auch die Empfehlung von Artikeln, die dem aktuell betrachteten Artikel ähnlich sind oder häufig zusammen mit diesem gekauft werden, sind höchstens insofern personalisiert, als die aktuellen Interessen des Nutzers, der einen bestimmten Artikel betrachtet, berücksichtigt werden. Bei der Generierung solcher Empfehlungen gibt es aber Überschneidungen mit den inhaltsbasierten Methoden, weil Artikeleigenschaften betrachtet werden, oder mit kollaborativen Filtern, weil Kookkurrenzen (Wer A kauft, kauft auch B) ausgewertet werden. Diese beiden sehr verbreiteten Ansätze und die Methoden, deren sie sich bedienen, werden in den folgenden Abschnitten besprochen.

2.2.2 Inhaltsbasierte Empfehlungen

Inhaltsbasierte Empfehlungen stellen den einzelnen Nutzer und dessen Präferenzen ins Zentrum und haben das Ziel, Artikel zu finden, die den Ansprüchen des Nutzers genügen. Das Verhalten anderer Nutzer spielt für solche Ansätze keine Rolle, sondern die Empfehlungen basieren im einfachsten Fall ausschließlich auf den Artikeln, die der Nutzer bereits positiv bewertet hat. Ausgehend von diesen Artikeln sucht das RS im Bestand nach ähnlichen Artikeln, um diese dem Nutzer empfehlen zu können. Die Aufgabe, die dabei gelöst werden muss, entspricht der „More like this“-Query im Information Retrieval. Sie kann als Klassifizierungsproblem betrachtet werden, bei dem jeder Artikel einer der Klassen „Nutzer mag das“ oder „Nutzer mag das nicht“ zugeordnet werden muss (vgl. hierzu [1, S. 378]).

Eine Übersicht über die Grundlagen und neuere Ansätze findet sich im Artikel von Pasquale Lops, Marco de Gemmis und Giovanni Semeraro [2, Kap. 3], an dem ich mich im Folgenden orientieren werde.

Schematischer Ablauf

Grundlage für die Empfehlungen bildet das Nutzerprofil, das aus den Eigenschaften der vom Nutzer bewerteten Artikel generiert wird. Je besser das Profil die Interessen des Nutzers abbildet, desto genauer kann die Relevanz neuer Artikel für den Nutzer eingeschätzt werden, indem man deren Eigenschaften mit den im Nutzerprofil festgehaltenen vergleicht. Letztlich geht es also darum, trennungswirksame Merkmale zu finden, um Artikel als „interessant“ oder „uninteressant“ einstufen zu können.

Im Allgemeinen sind mindestens drei Schritte zur Generierung von Empfehlungen nötig:

1. Inhaltsanalyse: Beim Verarbeiten unstrukturierter Daten ist häufig eine Vorverarbeitung nötig, bevor Nutzerprofil und Empfehlungen erstellt werden können. Die Eigenschaften von Artikeln müssen analysiert und in eine einheitliche Form gebracht werden, um diese Informationen im nächsten Schritt zur Verfügung zu haben. Wie eine solche Analyse ablaufen kann, ist das Thema von Abschnitt 2.2.2.

2. Profillernen: Alle Daten, die auf die Präferenzen des Nutzers schließen lassen, werden verwendet, um ein Nutzerprofil zu erstellen. Explizites und implizites Feedback spielen hier eine wichtige Rolle, zusätzlich kann der Nutzer auch die Möglichkeit haben, das eigene Profil um die Angabe von Interessen zu ergänzen. Aus all diesen Eingaben müssen generalisierte Informationen gewonnen werden. Welche Methoden dabei eingesetzt werden können, wird in Abschnitt 2.2.2 beschrieben.

3. Filtern: Im letzten Schritt werden Nutzerprofil und Artikelinformationen genutzt, um dem Nutzer die Artikel vorzuschlagen, die für ihn relevant sind. Diese Auswahl erfolgt gestützt auf ein Ähnlichkeitsmaß.

Repräsentation von Artikeln

Artikel werden durch Eigenschaften beschrieben. Im einfachsten Fall gibt es eine feste Anzahl von Attributen, deren mögliche Werte bekannt sind. In diesem Fall können die Artikel durch strukturierte Daten repräsentiert werden und es stehen eine Menge von Algorithmen zur Verfügung, um aufbauend auf diesen Daten Nutzerprofile zu erlernen.

Etwas schwieriger wird es, wenn Artikel durch Freitexte beschrieben werden. In diesem Fall gibt es keine im Voraus bekannten Attribute mit einer festen Menge möglicher Belegungen und zur Erstellung von Artikelrepräsentationen müssen andere Methoden herangezogen werden. Ein sehr verbreiteter Ansatz sind Vektorraummodelle (engl. *Vector Space Models*), in denen jedes (Text-)Dokument als Vektor repräsentiert wird, dessen Einträge sogenannte Termgewichte sind, die angeben, wie dominant die einzelnen Wörter aus dem Gesamtvokabular der zugrunde liegenden Textmenge im repräsentierten Dokument sind. Etwas formaler (vgl. [2, S. 81]): Gegeben eine Menge von Dokumenten $D = \{d_1, d_1, \ldots, d_N\}$ sowie die Menge der in diesem Korpus vorkommenden Terme $T = \{t_1, \ldots, t_n\}$, die durch Textanalyse gewonnen werden, dann wird Dokument d_j repräsentiert als $d_j = \{w_{1j}, w_{2j}, \ldots, w_{nj}\}$, wobei w_{kj} das Gewicht von Term t_k in Dokument d_j ist. Für die Berechnung der Gewichte gibt es verschiedene erprobte Verfahren, von denen TF-IDF (Term Frequency-Inverse Document Frequency) eines der verbreitetsten ist. Details zu diesem Verfahren werden in 3.2.3 erläutert, die Grundidee lässt sich aber wie folgt zusammenfassen: Wörter, die in einem Dokument häufig auftauchen, im Rest des Korpus aber nur selten, sind mit hoher Wahrscheinlichkeit relevant, um das Thema des Dokuments zu beschreiben, und erhalten daher ein hohes Gewicht. Um kurze Dokumente nicht zu benachteiligen, werden die Gewichte außerdem normalisiert.

Für die Berechnung von Empfehlungen werden die Nutzerprofile ebenfalls als gewichtete Termvektoren dargestellt. Wie gut ein gegebenes Dokument den Interessen des Nutzers entspricht, kann dann mittels der Kosinusähnlichkeit bestimmt werden. Für zwei Vektoren $\vec{d_1}$ und $\vec{d_2}$ gilt:

$$\frac{\vec{d_1} \cdot \vec{d_2}}{\|\vec{d_1}\| \cdot \|\vec{d_2}\|} = \cos \alpha,$$

wobei α der Winkel zwischen den beiden Vektoren ist. Im allgemeinen Fall liegen die Ähnlichkeitswerte zwischen -1 und 1. Da die TF-IDF-Vektoren aber nur positive Einträge enthalten, liegen die Ähnlichkeitswerte in diesem Fall zwischen 0 und 1, so dass ein Dokument dann als besonders unpassend zum Nutzerprofil angesehen werden muss, wenn das Ergebnis des obigen Ausdrucks sehr klein ist, während maximale Ähnlichkeit dann gegeben ist, wenn sich die Vektoren nur in ihrer Länge, nicht aber in ihrer Richtung unterscheiden und das Ergebnis deshalb 1 ist.

Dieser Ansatz hat sich in der Praxis bewährt und liefert gute Ergebnisse, wenn auch mit den Einschränkungen, die in Abschnitt 2.2.2 (*Nachteile und Herausfor-*

derungen) beschrieben sind. Was allerdings bei dieser Vorgehensweise überhaupt nicht berücksichtigt wird, ist die Semantik der zugrunde liegenden Texte. Dies kann zu unerwünschten Ergebnissen führen. Ein einfaches, wenn auch sehr konstruiertes Beispiel sind die beiden folgenden Sätze, die konträre Aussagen machen, aber eine Kosinusähnlichkeit von Eins aufweisen:

> *Dieser Text handelt nicht von Kühen, sondern von Pferden.*
> *Dieser Text handelt nicht von Pferden, sondern von Kühen.*

Generell können Verneinungen oder Abgrenzungen unter Umständen zu Problemen führen. Etwa, wenn eine Filmbeschreibung beginnt mit „Anders als klassische Hollywoodromanzen setzt dieser Film ...“ Unter Umständen erhält „Hollywoodromanze“ im resultierenden Termvektor ein vergleichsweise hohes Gewicht, obwohl es sich bei dem Film gar nicht um eine solche handelt. Denkbar ist auch, dass zwei Artikel anhand ihrer Beschreibungen als weniger ähnlich eingestuft werden, als sie es de facto sind, weil sie sich nicht derselben Terminologie bedienen. Ein Buch, das sich laut Beschreibung mit der Ökonomie der Vereinigten Staaten beschäftigt, weist Ähnlichkeiten mit einer Veröffentlichung zum Wirtschaftssystem der USA auf. Diese Ähnlichkeit wird aber unter Umständen unterschätzt, weil die Verwendung von Synonymen zu unterschiedlichen Termvektoren führt. Wünschenswert ist es auch, dass einem Nutzer, der Interesse am französischen Impressionismus hat, Artikel mit Bezug zu Monet oder Renoir vorgeschlagen werden (vgl. [2, S. 85]).

Neuere Ansätze setzen daher auf semantische Analysen, bei denen auf Ontologien oder externe Wissensdatenbanken zurückgegriffen wird, um Mehrdeutigkeiten auszuräumen, Synonyme zu erkennen oder andere zusätzliche Informationen bei der Textanalyse und dem Erstellen der Benutzerprofile zu berücksichtigen.

Lernen von Nutzerprofilen

Wie weiter oben bereits angemerkt, entspricht das Lernen eines Nutzerprofils bei inhaltsbasierten Recommendern im Endeffekt einer binären Klassifizierungsaufgabe: Für jeden Artikel muss entschieden werden können, ob er dem Nutzer gefällt oder nicht.

Um diese Aufgabe zu lösen, haben sich in der Praxis verschiedene Ansätze bewährt. Im Folgenden sollen drei häufig verwendete Arten von Ansätzen kurz angerissen

werden. Für eine detailliertere Darstellung sei auf die einschlägige Literatur, wie zum Beispiel [2] und [4], verwiesen.

Eine erste und intuitive Möglichkeit stellt der Nächste-Nachbarn-Ansatz dar [4, S. 58 ff.], [2, S. 48 ff.], der auch beim kollaborativen Filtern eingesetzt wird (s. 2.2.3). Um einzuschätzen, ob dem Nutzer ein bestimmter Artikel gefällt, werden unter den Artikeln, für die Bewertungen des Nutzers bekannt sind, die k herausgesucht, die dem einzuschätzenden Artikel am ähnlichsten sind. Werden die Artikel durch TF-IDF-Vektoren repräsentiert, kann als Ähnlichkeitsmaß zum Beispiel die Kosinusähnlichkeit herangezogen werden. Aus den Bewertungen dieser Artikel kann im nächsten Schritt die erwartete Bewertung des aktuellen Artikels geschätzt werden, um den Artikel dann, wenn das Ergebnis positiv ausfällt, zu empfehlen. Hat ein Nutzer sehr viele Artikel bewertet, muss die Bestimmung der nächsten Nachbarn gegebenenfalls optimiert werden, weil sie bei naiver Implementierung des Algorithmus sehr rechenintensiv ist.

Eine Alternative zum Nächste-Nachbarn-Ansatz ist der Rocchio-Algorithmus [4, S. 60 ff.], [2, S. 92 f.]. Bei diesem handelt es sich um ein Relevance-Feedback-Verfahren, das darauf basiert, dass Nutzer die Empfehlungen, die sie vom System erhalten, bewerten können, um so eine ständige Verfeinerung ihres Profils und eine Verbesserung der generierten Empfehlungen zu ermöglichen. Bei diesem Algorithmus kommen TF-IDF-Vektoren zum Einsatz und die Klassifikation eines Dokuments (gefällt dem Nutzer oder nicht) erfolgt anhand eines Vergleichs mit Prototypenvektoren. Sowohl für Artikel, die dem Nutzer gefallen, als auch für solche, bei denen das nicht der Fall ist, gibt es einen solchen Vektor, der wie ein „normaler“ Dokumentenvektor aufgebaut ist, dessen Einträge aber aus den Vektoren von bekanntermaßen zur Klasse gehörenden Dokumenten und bekanntermaßen nicht zur Klasse gehörenden Dokumenten berechnet wird. Die positiven Beispiele erhöhen den Wert der entsprechenden Komponente, die negativen Beispiele verringern den Wert der entsprechenden Komponente. Für ein zu klassifizierendes Dokument kann dann die Kosinusähnlichkeit mit den Prototypenvektoren berechnet werden, um es der Klasse mit der größeren Ähnlichkeit zuzuordnen.

Eine weitere Familie von Algorithmen, die zur Artikelklassifizierung (gefällt dem Nutzer oder nicht) herangezogen werden können, sind die probabilistischen Ansätze. Unter diesen sticht der naive Bayes-Klassifikator durch seine Einfachheit hervor. In frühen inhaltsbasierten Recommendersystemen wurde dieser Algorithmus häufig eingesetzt, heute ist er in diesem Bereich nicht mehr so verbreitet. Ausgehend von

Trainingsdaten schätzt der naive Bayes-Ansatz die Aposteriori-Wahrscheinlichkeit dafür, dass ein Dokument d einer bestimmten Klasse c angehört, wie folgt: $P(c \mid d) = \frac{P(c)P(d|c)}{P(d)}$, wobei $P(c)$ die Apriori-Wahrscheinlichkeit ist, mit der ein Dokument der Klasse c angehört, $P(d \mid c)$ die Wahrscheinlichkeit, mit der Dokument d auftritt, wenn Klasse c gegeben ist, und $P(d)$ die Wahrscheinlichkeit, Dokument d überhaupt zu beobachten.

Dokument d wird dabei als Vektor dargestellt, der für jedes Wort aus dem Vokabular des gesamten Korpus entweder nur angibt, ob es in d auftaucht (multivariates Bernoulli-Ereignismodell), oder aber, wie häufig es in d vorkommt (multinomiales Ereignismodell).

Weil $P(d)$ unabhängig von der Klassenzugehörigkeit ist, wird es in der Regel weggelassen. Bleiben noch $P(c)$ und $P(d \mid c)$, die aus den Trainingsdaten geschätzt werden müssen. Die Wahrscheinlichkeiten für die Klassen zu schätzen, ist einfach. $P(d \mid c)$ zu schätzen ist hingegen schwierig bis unmöglich, weil man davon ausgehen kann, dass man es mit individuellen Dokumenten zu tun hat, die weder in den Trainingsdaten noch in der späteren Empfehlungssituation mehrfach auftauchen. Aus diesem Grund wird an dieser Stelle eine Annahme getroffen, die zwar in der Realität nicht zutrifft, sich aber als hilfreich erwiesen hat: Für die einzelnen Terme im das Dokument repräsentierenden Vektor wird statistische Unabhängigkeit angenommen. Dann werden die Wahrscheinlichkeiten wortweise berechnet. Das Dokument wird der Klasse zugeordnet, für die die Aposteriori-Wahrscheinlichkeit maximal ist.

Weitere Methoden, die bei inhaltsbasierten Recommendersystemen zum Einsatz kommen können, umfassen zum Beispiel Support-Vektor-Maschinen, Entscheidungsbäume und Methoden zur Regelableitung für Empfehlungen, wie zum Beispiel den RIPPER-Algorithmus.

Vorteile

Inhaltsbasierte Recommendersysteme haben den Vorteil, dass auch neue Artikel empfohlen werden können, für die bislang noch keine Bewertungen vorliegen. Solange eine Beschreibung existiert, werden diese Artikel behandelt wie alle anderen auch.

Die Empfehlungen sind außerdem unabhängig von anderen Nutzern. Es besteht also nicht die Gefahr, dass mangels Nutzer mit wirklich ähnlichem Geschmack auch die „entfernte“ Nachbarschaft zur Generierung von Empfehlungen herangezogen wird.

Entschieden wird nach den Vorlieben des aktuellen Nutzers, wie ein Artikel bei anderen Nutzern ankommt, spielt keine Rolle.

Ein weiterer Vorteil des inhaltsbasierten Ansatzes ist, dass das Zustandekommen der Empfehlungen transparent und für den Nutzer nachvollziehbar ist. Solange keine neuen Artikel hinzukommen und sich die Artikelbeschreibungen nicht ändern, sind die Empfehlungen darüber hinaus deterministisch und der Nutzer bekommt immer dieselben Empfehlungen.

Nachteile und Herausforderungen

Wie weiter oben bereits angeklungen ist, werden trennungswirksame Merkmale benötigt, um Artikel, die einem bestimmten Nutzer gefallen, von solchen zu unterscheiden, auf die das nicht zutrifft. Dazu müssen genügend Informationen über den Artikel vorhanden sein. Die Auswahl der relevanten Merkmale und die Wahl des Analysealgorithmus setzt Domänenwissen voraus. Nicht immer reicht der hier vorgestellte „Bag of Words“-Ansatz, bei dem jeder Artikel durch einen Termvektor repräsentiert wird, aus, um ähnliche Artikel aufzufinden. Schwierig wird es zum Beispiel, wenn die Artikelbeschreibungen in verschiedenen Sprachen vorliegen. In diesem Fall müssen externe Wissensquellen herangezogen werden. Einige Ansätze arbeiten mit enzyklopädischem Wissen, andere erzeugen Ontologien der Domäne, in der die Empfehlungen erstellt werden (vgl. hierzu z. B. [2, S. 78 sowie S. 85 ff.]).

Ein großer Nachteil inhaltsbasierter Recommendersysteme besteht in ihrer Überspezialisierung. Weil alle Empfehlungen auf den Nutzerinteressen beruhen und nur Artikel empfohlen werden, die den bereits bekannten ähneln, werden dem Nutzer methodenbedingt keine wirklich überraschenden Ergebnisse geliefert. Im Englischen spricht man vom „serendipity problem“ (dt. Zufallsfund, Entdeckung), weil es nahezu unmöglich ist, dem Nutzer Artikel zu empfehlen, die aus einem Bereich stammen, von dem er bislang noch gar nicht wusste, dass dieser interessant sein könnte. Zu den Lösungsansätzen für dieses Problem gehören die Einführung von Zufall, zum Beispiel durch genetische Algorithmen, die Wahl „schlechter“ Ähnlichkeitsmaße oder die Empfehlung von Artikeln, bei denen nicht eindeutig entschieden werden kann, ob der Nutzer sie mag oder nicht, weil für diese die Wahrscheinlichkeit, dass es sich um wirkliche Neuheiten für den Nutzer handelt, am höchsten ist.

Bei der Modellierung der Nutzerpräferenzen stellt sich die Frage, wie der Faktor Zeit

berücksichtigt werden soll: Gewichtet man alle Bewertungen gleich stark oder misst man ihnen mehr Gewicht bei, je aktueller sie sind, um sich ändernde Präferenzen zu berücksichtigen? (Vgl. hierzu [1, S. 379 f.].)

Neuere Ansätze versuchen, dem Umstand Rechnung zu tragen, dass immer mehr nutzergenerierte Inhalte zur Verfügung stehen, etwa in Form von Schlagworten, mit denen einzelne Artikel versehen werden. Diese Inhalte lassen sich nutzen, um Ähnlichkeiten zwischen Artikeln festzustellen. Damit entfernt man sich aber auch ein Stück weit von rein inhaltsbasierten Ansätzen, weil andere Nutzer und deren Vorlieben oder Einschätzungen von Artikeln ins Spiel kommen. Ansätze, die vollständig auf der Auswertung von Nutzerbeziehungen bzw. -ähnlichkeiten beruhen, sind das Thema des folgenden Abschnitts.

2.2.3 Kollaboratives Filtern

Während bei den inhaltsbasierten Ansätzen der Fokus auf dem einzelnen Nutzer und den Eigenschaften der Artikel, die ihm gefallen, liegt, nehmen kollaborative Filter die Nutzergemeinschaft ins Visier. Dem einzelnen Nutzer wird das empfohlen, was andere Nutzer mit einem ähnlichen Geschmack in der Vergangenheit gut bewertet haben. Damit wird nachgebildet, was in der Realität ohnehin geschieht: Viele Menschen vertrauen bei der Auswahl neuer Artikel dem Urteil anderer Menschen mit ähnlichem Geschmack.

Weil die Ähnlichkeit des Geschmacks durch den Vergleich vergangener Bewertungen ermittelt wird, wird kollaboratives Filtern bisweilen auch als *People-to-People Correlation* bezeichnet [2, Kap. 1, S. 2]. Bei der Beschreibung des kollaborativen Filterns orientiere ich mich an einem Artikel von Yehuda Koren und Robert Bell [2, Kap. 5].

Im Gegensatz zu inhaltsbasierten Methoden benötigen kollaborative Filter jenseits des Nutzerverhaltens keine weiteren Informationen. Zum Nutzerverhalten zählt in erster Linie explizites Feedback in Form von Bewertungen, gegebenenfalls aber auch implizite Rückmeldungen ans System durch Käufe, Browserverlauf, Suchmuster oder Mausbewegungen. Die Einbeziehung dieser impliziten Informationen kann zu signifikanten Verbesserungen führen.

Ziel des kollaborativen Filterns ist es, Artikel zu finden, die einem bestimmten Nutzer gefallen könnten, oder sogar die Bewertung eines Nutzers u für einen ihm bislang

noch unbekannten Artikel i vorherzusagen. Bei dieser Vorhersage werden systematische Abweichungen (Artikel i wird immer besser bewertet als der Durchschnitt, Nutzer u vergibt nie mehr als 4 Sterne, ...), die unabhängig von Interaktionen sind, durch sogenannte *baseline predictors* berücksichtigt. Eine solche Normalisierung ist notwendig, weil sonst inkompatible Bewertungen miteinander verrechnet würden.

Zur Erstellung kollaborativer Filter haben sich zwei hauptsächliche Herangehenweisen etabliert: nachbarschaftsbasierte Ansätze und latente Variablenmodelle (engl. latent factor models).[9] Deren Funktionsweise wird in den folgenden Unterabschnitten kurz umrissen.

Nachbarschaftsbasierte Ansätze

Nachbarschaftsbasierte Ansätze sind die intuitivere der beiden Vorgehensweisen. Durch Beobachtungen des Nutzerverhaltens werden Nachbarschaften bestimmt. Bei klassischen kollaborativen Filtern sind dies Nachbarschaften von Nutzern. Zwei Nutzer werden dann als Nachbarn eingestuft, wenn sie ein ähnliches Bewertungsverhalten aufweisen. Als Ähnlichkeitsmaß kann zum Beispiel die Kosinusähnlichkeit verwendet werden (vgl. [5]). Dazu wird jeder Nutzer als Vektor repräsentiert, in dem für jeden Artikel ein Eintrag besteht, der positiv ist, wenn der Nutzer den besagten Artikel positv bewertet hat, negativ bei schlechter Bewertung und 0 sonst. Dabei sollte auf jeden Fall auch durch *baseline predictors* normiert werden, um eine Vergleichbarkeit zu gewährleisten. Alternativ kann zum Beispiel auch die Pearsonkorrelation (normierte Kovarianz) verwendet werden, die weiter unten erläutert wird.

Hat man die k Nutzer gefunden, die dem aktuellen Nutzer am ähnlichsten sind, dann kann man deren (gut) bewertete Artikel heranziehen, um sie dem aktuellen Nutzer zu empfehlen. Im einfachsten Fall sammelt man alle Artikel, die von mindestens einem der Nachbarn bewertet wurden, sortiert diejenigen aus, die der aktuelle Nutzer bereits bewertet hat, und empfiehlt die übrigen, wobei diejenigen Artikel mit besonders hoher Priorität empfohlen werden, die von möglichst vielen und/oder möglichst ähnlichen Nutzern gut bewertet wurden. Alternativ ist es auch möglich, die zu empfehlenden Artikel nach geschätzten Bewertungen zu sortieren. Dazu nimmt man die Bewertungen benachbarter Nutzer als Grundlage und mittelt diese gewichtet nach

[9] In [2, Kap. 5] zeigen Koren und Bell, dass diese Modelle an sich äquivalent sind. Zum Verständnis ist es dennoch hilfreich, sich die unterschiedlichen Ansätze vor Augen zu führen.

Nähe zum aktuellen Nutzer. Die Artikel mit den höchsten geschätzten Bewertungen werden empfohlen.

Bezeichne R die Bewertungsmatrix. Diese ist in der Regel sehr dünn besetzt und setzt sich aus den oben bereits erwähnten Nutzervektoren zusammen: Eintrag r_{ui} beinhaltet dabei die Bewertung von Nutzer u für Artikel i. Weiterhin sei s_{uv} die berechnete Ähnlichkeit von Nutzer u und Nutzer v und N_{ui}^k die Menge der k Nutzer, die von all jenen Nutzern, die Artikel i bewertet haben, diejenigen sind, die Nutzer u am meisten ähneln, also dessen Nachbarn. b_{ui} ist der *baseline predictor* für die Bewertung von Artikel i durch Nutzer u. Die erwartete Bewertung wird dann wie folgt geschätzt:[10]

$$\hat{r}_{ui} = b_{ui} + \frac{\sum\limits_{v \in N_{ui}^k} s_{uv}(r_{vi} - b_{vi})}{\sum\limits_{v \in N_{ui}^k} s_{uv}}$$

Nutzerbasierte Nachbarschaftsansätze waren lange Zeit der De-facto-Standard bei kollaborativen Filtern. Zwischenzeitlich wurden sie aber von artikelbasierten Nachbarschaftsansätzen abgelöst. Diese Ansätze haben den Vorteil, dass sie in Fällen, wo es wesentlich mehr Nutzer als Artikel gibt, weniger rechen- und speicherintensiv sind. Die Grundidee besteht darin, Artikel zu suchen, die denjenigen ähneln, die der aktuelle Nutzer bereits gekauft/bewertet hat, um sie ihm empfehlen zu können. Anders als bei den inhaltsbasierten Ansätzen aus Abschnitt 2.2.2 wird die Ähnlichkeit der Artikel aber nicht aufgrund ihrer endogenen Eigenschaften bestimmt, sondern durch Auswertung des Nutzerverhaltens: Ähnlich sind diejenigen Artikel, die oft zusammen gekauft werden und/oder ähnliche Bewertungen erhalten. Zur Berechnung der Ähnlichkeit kann wiederum die Kosinusähnlichkeit verwendet werden, indem jeder Artikel als ein Vektor dargestellt wird, der für jeden Kunden einen Eintrag enthält, der dann ungleich 0 ist, wenn der Kunde den Artikel bewertet/gekauft hat. Häufig werden nicht nur binäre Vektoren verwendet, sondern die Bewertungen, die die Nutzer abgegeben haben, werden korrigiert um die *baseline predictors* für jeden Nutzer. Die Kosinusähnlichkeit der korrigierten Bewertungsvektoren der Artikel i und j ist der oben bereits erwähnte Pearson-Korrelationskoeffizient (normierte

[10] Zur Gewichtung der Bewertungen, die in die Schätzung einfließen, werden hier die berechneten Ähnlichkeit zwischen u und den anderen Nutzern verwendet. Es kann zu besseren Ergebnissen führen, die Ähnlichkeiten vorher zu transformieren oder aber die optimalen Gewichte separat zu lernen. Vgl. hierzu [2, S. 163 ff.].

Kovarianz), der wie folgt geschätzt wird:

$$\hat{\rho}_{ij} = \frac{\sum\limits_{u \in U(i,j)} (r_{ui} - b_{ui})(r_{uj} - b_{uj})}{\sqrt{\sum\limits_{u \in U(i,j)} (r_{ui} - b_{ui})^2 \cdot \sum\limits_{u \in U(i,j)} (r_{uj} - b_{uj})^2}}$$

$U_{i,j}$ bezeichnet hier die Menge der Nutzer, die sowohl i als auch j bewertet haben, r_{ui} ist wie oben die Bewertung, die Nutzer u für Artikel i abgegeben hat, und b_{ui} ist wieder der *baseline predictor* für die Bewertung des Artikels i durch Nutzer u. Der so erhaltene empirische Korrelationskoeffizient wird dann in der Regel noch so angepasst, dass er größer ist, wenn er auf einer breiten empirischen Basis steht (viele Nutzer haben i und j bewertet), als wenn das nicht der Fall ist. Für jeden Artikel, den der aktuelle Nutzer bewertet hat, können dann ähnliche Artikel bestimmt werden, um sie zu empfehlen. Auch hier müssen natürlich diejenigen Artikel ausgefiltert werden, die der Nutzer schon kennt.

Analog zum obigen Fall kann auch wieder die Bewertung des Nutzers für Artikel i geschätzt werden, allerdings werden diesmal nicht die Bewertungen zu u benachbarter Nutzer als Grundlage genommen, sondern die Bewertungen, die Nutzer u für Artikel abgegeben hat, die i ähneln. Die k Artikel, die von u bewertet wurden und i am ähnlichsten sind, sind in Menge N_{iu}^k enthalten. Die zu erwartende Bewertung berechnet sich dann wie folgt:

$$\hat{r}_{ui} = \frac{b_{ui} + \sum\limits_{j \in N_{iu}^k} s_{ij}(r_{uj} - b_{uj})}{\sum\limits_{j \in N_{iu}^k} s_{ij}}$$

Vergleicht man die Vorgehensweise der beiden nachbarschaftsbasierten Ansätze, so fällt auf, dass sie im Endeffekt dasselbe tun, sich aber unterschiedlicher Abkürzungen bedienen (vgl. [2, S. 184]). Genutzt werden in jeden Fall Nutzer-Artikel-Ketten der Art „Nutzer u mag Artikel i, der gefällt auch Nutzer v, der außerdem Artikel j gut bewertet hat, also empfehlen wir u Artikel j“ (vereinfachte Darstellung). Den ersten Teil der Kette kann man sich sparen, wenn man Ähnlichkeiten zwischen Nutzern im Voraus berechnet, denn dann landet man bei Betrachtung der Nachbarn von u automatisch bei v, der hintere Teil fällt hingegen weg, wenn Artikelähnlichkeiten berechnet wurden, die j als Nachbar von i ausweisen.

Latente Variablenmodelle

Nachbarschaftsbasierte Ansätze haben den Vorteil, dass sie recht leicht nachvollziehbar sind. Allerdings sind viele der Vektoren und Matrizen, mit denen gerechnet wird, zwar sehr dünn besetzt, aber auch sehr groß. Im Zuge des bereits erwähnten Netflix-Wettbewerbs haben latente Variablenmodelle an Popularität gewonnen und zwar insbesondere solche, die auf Matrixfaktorisierung setzen.[11] Modelle dieser Art nehmen eine Dimensionsreduktion vor, indem sie die Daten auf einen „versteckten" Raum niedrigerer Dimension abbilden. Merkmalsausprägungen in diesem Raum werden dann verwendet, um Artikeleigenschaften und Nutzerpräferenzen abzugleichen.

Konkret sieht das wie folgt aus: Artikel i wird repräsentiert durch Vektor $q_i \in \mathbb{R}^f$, Nutzer u durch Vektor $p_u \in \mathbb{R}^f$ und die Interaktion von Nutzer und Artikel wird repräsentiert durch das Produkt dieser zwei Vektoren.

Die Bewertung des Nutzers u für Artikel i wird dann wie folgt geschätzt:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u,$$

wobei μ die durchschnittliche Bewertung für alle Artikel ist, b_i der *baseline predictor* für Artikel i und b_u der *baseline predictor* für Nutzer u.

Um auf diese Weise Bewertungen schätzen zu können, müssen natürlich zunächst die Modellparameter q_i, p_u sowie b_u und b_i bestimmt werden. Dies geschieht gestützt auf Trainingsdaten. Geschätzte und tatsächliche Bewertung werden für verschiedene mögliche Werte der Parameter verglichen mit dem Ziel, den normierten quadratischen Fehler zu minimieren. Hierzu gibt es Standardverfahren, wie zum Beispiel den probabilistischen Gradientenabstieg oder alternierende kleinste Quadrate.

Auch latente Variablenmodelle können natürlich erweitert werden. Zum Beispiel, um implizites Feedback der Nutzer an das RS zu berücksichtigen, oder auch, um Bewertungen unterschiedlich zu gewichten abhängig vom Zeitpunkt, zu dem sie abgegeben wurden.

[11] Andere latente Variablenmodelle sind zum Beispiel neuronale Netze, Latent Dirichlet Allocation oder pLSA (probabilistic Latent Semantic Analysis).

Vorteile

Nachbarschaftsbasierte kollaborative Filter sind einfach umzusetzen, intuitiv und für den Nutzer nachvollziehbar, gerade wenn mit Artikelähnlichkeiten gearbeitet wird. Bei artikelbasierten Ansätzen können neue Bewertungen sofort berücksichtigt werden, ohne neue Berechnungen anstellen zu müssen, jedenfalls wenn man davon ausgeht, dass die Beziehungen zwischen Artikeln recht stabil sind.

Matrixfaktorisierungsmethoden haben den großen Vorteil, dass sie tendenziell etwas bessere Ergebnisse erzielen und zudem speichersparend sind. Zusätzliche Informationen lassen sich integrieren, wie Koren und Bell in [2] zeigen.

Nachteile und Herausforderungen

Neben den im vorangegangenen Abschnitt genannten Vorteilen haben kollaborative Filter natürlich auch Nachteile. Die klassischen Ansätze sind recht speicherintensiv. Außerdem will die Ähnlichkeitsfunktion wohl überlegt sein und die Schätzung der Relevanz eines Artikels für den Nutzer weist je nach Bestimmung der Gewichte unterschiedliche Probleme auf. Bei einigen Ansätzen ist es so, dass zwingend alle Nachbarn bei der Schätzung berücksichtigt werden müssen. Bei eigentlich allen Ansätzen werden verwandte Nachbarn nicht als solche erkannt (Beispiel: Fortsetzungen bei Büchern) und diese verwandten Artikel tragen dann tendenziell zu viel zur gesamten Schätzung bei und schmälern die Diversität der Empfehlungen.

Einige dieser Probleme können leicht gelöst werden, zum Beispiel, indem man Gewichtungsfaktoren einführt, die den Einfluss von Nachbarn, die nur wenig Information beitragen, schmälern. In anderen Fällen ist dies schwieriger (vgl. hierzu [2, S. 164 f.]).

2.2.4 Demographische Ansätze

Demographische Recommendersysteme empfehlen Artikel basierend auf soziodemographischen Daten der Nutzer. Im Kleinen werden solche Ansätze bereits erfolgreich eingesetzt, etwa wenn den Nutzern sprachabhängige Addons empfohlen werden (zum Beispiel Rechtschreib- und Grammatikprüfungsfunktionen passend zur Systemsprache) oder wenn älteren Nutzern andere Reisepakete vorgeschlagen werden als jünge-

ren. Im großen Stil kommen solche Ansätze aber höchstens als Ergänzung zu anderen Systemen zum Einsatz.

2.2.5 Wissensbasierte Ansätze

Wissensbasierte RS basieren auf Fachwissen über ein bestimmtes Spezialgebiet und empfehlen basierend auf diesem Wissen Artikel, die die Nutzerbedürfnisse befriedigen. Durchgesetzt haben sich zwei Hauptspielarten: Fallbasierte (*case based*) und regel- bzw. beschränkungsbasierte (*constraint based*) Ansätze.

Im ersten Fall wird eine Datenbank mit bereits gelösten Problemen und entsprechenden Lösungen verwendet, um Empfehlungen zu generieren. Dazu wird die aktuelle Anfrage bzw. das durch sie beschriebene Problem mit bereits bekannten Problemen verglichen und die ähnlichsten Probleme werden zur Generierung einer Lösung herangezogen (vgl. z. B. [6]). Regelbasierte Ansätze nutzen Wissensdatenbanken, um Artikel zu finden, die den durch den Nutzer vorgegebenen Einschränkungen entsprechen. Damit solche Systeme auch zur Zufriedenheit der Nutzer funktionieren, ist es wichtig, dass inkompatible Nutzeranforderungen erkannt und bei leerer Ergebnismenge sinnvolle Vorschläge gemacht werden, welche Einschränkungen gelockert werden sollten.

2.2.6 Gemeinschaftsbasierte Ansätze

Da sich viele Nutzer eher auf das Urteil ihnen bekannter Menschen verlassen als auf das fremder Personen, gewinnen Soziale Recommendersysteme zunehmend an Bedeutung. Ricci et al. berichten von gemischten Ergebnissen: Im Großen und Ganzen schneiden soziale RS nicht besser ab als die klassischen Ansätze, außer wenn es sich um sehr umstrittene Artikel handelt, deren Bewertungen stark auseinander gehen, oder wenn eine so genannte Kaltstartsituation vorliegt, in der Nutzerähnlichkeiten noch nicht sinnvoll berechnet werden können, weil nicht genug Daten vorhanden sind. Allerdings kann es von Vorteil sein, klassisches kollaboratives Filtern durch gemeinschaftsbasierte Ansätze zu ergänzen, um bessere Ergebnisse zu erhalten. (Vgl. [2, S. 13].)

2.2.7 Hybride Ansätze

In den vorausgegangenen Abschnitten ist schon mehrfach angeklungen, dass es von Vorteil sein kann, unterschiedliche Ansätze miteinander zu kombinieren, um insgesamt bessere Ergebnisse zu erhalten. Diese Erkenntnis ist nicht neu und so haben sich im Laufe der Jahre eine Reihe von hybriden Ansätzen entwickelt. In seinem 2007 erschienenen Überblicksartikel nennt Robert Burke sieben Möglichkeiten, unterschiedliche Empfehlungskomponenten miteinander zu kombinieren [1, S. 380]:

1. *Gewichtung*: Jede der Komponenten berechnet separat eine Reihe von Empfehlungen, die jeweils mit einer Punktzahl versehen sind (zum Beispiel der geschätzten Bewertung durch den Nutzer). Diese Ergebnisse werden dann mit unterschiedlicher Gewichtung linear miteinander kombiniert, um das endgültige Ergebnis zu erhalten.

2. *Umschaltung (Switching)*: Das System wählt eine der Komponenten aus und verwendet deren Empfehlungen. Um Verbesserungen im Vergleich zur Verwendung eines einzelnen Verfahrens zu erzielen, ist ein geeignetes Auswahlverfahren erforderlich, das entweder externe Daten zur Entscheidung heranzieht oder Maße, die ihm die zur Auswahl stehenden Komponenten liefern, wie etwa p-Werte.

3. *Mischung*: Die Ergebnisse mehrerer Verfahren werden kombiniert und dem Nutzer präsentiert.

4. *Merkmalskombination*: Merkmale aus verschiedenen Quellen werden kombiniert und dann von einem einzelnen Empfehlungsalgorithmus bearbeitet. Ein inhaltsbasiertes RS erhält dann zum Beispiel als zusätzliche Eingaben noch Informationen wie „Nutzer u und Nutzer v mögen Artikel j“, die eigentlich von kollaborativen Filtern verarbeitet werden würden, und verarbeitet diese auf dieselbe Weise wie alle anderen Eingaben auch.

5. *Merkmalsverbesserung*: Um die Eingangsdaten für die Empfehlungen zu vervollständigen oder anzureichern, wird eines der Verfahren dazu verwendet, neue oder ergänzende Merkmale zu berechnen, die dann die Eingabe für die im nächsten Schritt verwendete Empfehlungstechnik erweitern.

6. *Kaskade*: Die Empfehlungsverfahren stehen in einer festen Hierarchie. Weiter unten stehende Verfahren kommen nur dann zum Einsatz, wenn keine eindeutige Rangfolge der Empfehlungen hergestellt werden kann.

7. *Meta-Level*: Das erste Verfahren erzeugt ein Modell, das dann als Eingabe für das nächste Verfahren dient. Burke bezeichnet dies als „eine Art Basiswechsel im Empfehlungsraum“ [1, S. 391], weil das zweite Verfahren nie auf den Ausgangsdaten arbeitet.

In der aktuellen Literatur finden sich viele Beispiele für solche gemischte Systeme. Einige Beispiele seien im Folgenden kurz umrissen:

David M. Blei und Chong Wang stellen in ihrem 2011 erschienenen Beitrag [7] einen Algorithmus zur Empfehlung wissenschaftlicher Artikel vor, der kollaboratives Filtern, umgesetzt als latentes Variablenmodell, mit probabilistischer Themenmodellierung, also einem inhaltsbasierten Ansatz, kombiniert. Ein Artikel, der noch nicht von vielen Nutzern bewertet oder in die Sammlung aufgenommen wurde, wird stärker anhand seines Inhalts empfohlen, bei bekannteren Artikeln wird eher das Verhalten der Nutzergemeinschaft berücksichtigt.

Gayatree Ganu, Yogesh Kakodkar und Amélie Marian [8] haben 2013 einen Ansatz zur Restaurantempfehlung veröffentlicht, der neben der Sternbewertung durch die Nutzer auch Freitextbewertungen nutzt. Der Algorithmus bildet Cluster von Nutzern in Abhängigkeit von den Themen und Gefühlen, die in den Bewertungstexten zum Ausdruck kommen. Die Bewertungen der auf diese Weise gefundenen ähnlichen Nutzer werden dann zur Vorhersage neuer Bewertungen genutzt. Auch diese Vorgehensweise ist also eine Kombination von kollaborativem Filtern mit inhaltsbasierten Methoden.

Auch in der Praxis sind solche gemischten Systeme Standard. In professionellen Recommendersystemen kommen in der Regel mehrere Verfahren zum Einsatz. Dies wird auch anhand der Praxisbeispiele deutlich, die in Abschnitt 2.4 vorgestellt werden.

2.3 Kaltstartprobleme

Unabhängig davon, welche der oben beschriebenen Methoden ein Recommendersystem zu Generierung von Empfehlungen verwendet, kann es mit sogenannten Kaltstartproblemen zu kämpfen haben. Diese treten immer dann auf, wenn noch nicht über genügend Informationen vorliegen, um sinnvolle Empfehlungen generieren zu

können. Weil diese Art von Problemen nicht auf eine bestimmte Klasse von Recommendersystemen beschränkt ist, ist ihr ein eigenes Unterkapitel gewidmet.

Nicht-personalisierte Empfehlungen für neue Nutzer sind unproblematisch. Bei der Anzeige von Toplisten kommt es nur darauf an, dass eine genügend große Anzahl Nutzer überhaupt schon Artikel bewertet, gekauft oder betrachtet hat. Informationen über den neuen Nutzer sind nicht nötig. Sollen dem Nutzer ähnliche Artikel zum gerade betrachteten oder häufig mit diesem zusammen gekaufte Artikel empfohlen werden, dann funktioniert dies problemlos auch bei neuen Nutzern und bei neuen Artikeln dann, wenn zur Ähnlichkeitsbestimmung inhaltliche Kriterien herangezogen werden. Wird die Ähnlichkeit hingegen anhand gemeinsamer Käufe bestimmt oder sollen zusammen gekaufte Artikel angezeigt werden, kommt es zu denselben Kaltstartproblemen wie beim artikelbasierten kollaborativen Filtern.

Bei inhaltsbasierten Recommendersystemen hingegen sind es die neuen Nutzer, über deren Präferenzen noch nichts bekannt ist, für die das System keine sinnvollen Empfehlungen generieren kann. Wer noch keine Bewertungen abgegeben und/oder sein Profil auf andere Weise vervollständigt hat, für den können keine personalisierten Vorschläge gemacht werden. Je mehr Informationen über die Vorlieben und Abneigungen des Nutzers vorliegen, desto besser sind tendentiell die generierten Empfehlungen. Dies heißt aber auch, dass Gelegenheitsnutzer unter Umständen nie in den vollen Genuss der Fähigkeiten eines inhaltsbasierten Recommendersystems kommen, weil über ihre Vorlieben zu wenig bekannt ist.

Genau wie inhaltsbasiert Empfehlungssysteme können auch kollaborative Filter unter Kaltstartproblemen leiden. Das ist dann der Fall, wenn ein Nutzer noch zu wenige Bewertungen abgegeben hat, um ihn sinnvoll in eine Nachbarschaft einordnen zu können, oder wenn für einen Artikel noch nicht genug Bewertungen vorliegen.

Wie weiter oben bereits ausgeführt, kommen demographische Ansätze ohnehin selten als alleiniger Empfehlungsansatz zum Einsatz. Wenn doch, setzt ihr erfolgreicher Einsatz zum einen natürlich das Vorliegen demographischer Informationen über die Nutzer voraus, zum anderen einen genügend großen Bestand an Bewertungs- bzw. Kaufdaten, der die statistische Auswertung anhand demographischer Kriterien erlaubt.

2.4 Stand der Technik – Wie arbeiten bekannte Recommendersysteme?

Genaue Informationen über die Arbeitsweise tatsächlich eingesetzter Recommendersysteme zu erhalten, ist naturgemäß schwierig, weil gute Empfehlungen einem Unternehmen im E-Commerce-Bereich einen Vorsprung vor der Konkurrenz verschaffen. Eine große Ausnahme bildet der Empfehlungsalgorithmus von Netflix. Wie bereits mehrfach erwähnt, wurde dieser im Rahmen eines Wettbewerbs entwickelt und es gibt eine Reihe von Veröffentlichungen der Wettbewerbsteilnehmer.

Amazon hat als erstes Unternehmen kollaborative Filter basierend auf den Nachbarschaftsbeziehungen von Artikeln eingesetzt, wozu die Entwickler einen Fachartikel veröffentlicht haben. Zusätzlich geben verschiedene Patentanmeldungen einen Überblick über die zum Einsatz kommenden Methoden, ohne jedoch Implementierungsdetails zu verraten.

Beide Recommendersysteme sollen im Folgenden vorgestellt werden, um einen groben Überblick über die tatsächlich in der Praxis verwendeten Algorithmen zu erhalten.

2.4.1 Filmempfehlungen bei Netflix

Der Netflix-Wettbewerb als Triebfeder für die Weiterentwicklung von Recommendersystemen im Bereich Film wurde bereits weiter oben erwähnt. Aus dem 2006 angelaufenen Wettbewerb ging tatsächlich ein Sieger hervor. Im Team *Bell-Kor's Pragmatic Chaos* schlossen sich mit *BellKor*, *Big Chaos* und *Pragmatic Theory* drei der bis dahin führenden Teams zusammen und stellten 2009 einen Algorithmus vor, dessen Prognosen für die Nutzerwertungen auf dem Testset eine mittlere quadratische Abweichung aufwiesen, die um 10 Prozent unter der des bis dahin von Netflix verwendeten *Cinematch*-Systems lag [9, 10, 11, 12]. In [12, S. 1] erläutern Bell, Koren und Volinsky recht anschaulich das Dilemma, vor dem sie und die anderen Teams standen:

> *Movies are complex beasts. Besides the most obvious characterization into genres, movies differ on countless dimensions describing setting, plot, characters, cast, and many more subtle features such as tone or style of the dialogue. The Movie Genome Project (www.jinni.*

> *com/movie-genome.html) reports using "thousands of possible genes". Consequently, any finite model is likely to miss some of the signal, or explanation, associated with people's ratings of movies. On the other hand, complex models are prone to over fitting, or matching small details rather than the big picture – especially where data are scarce.*

So überrascht es denn auch nicht, dass es sich bei dem letztlich erfolgreichen Ansatz nicht um einen einzelnen Algorithmus handelt, sondern um eine Kombination mehrerer bereits bekannter und mit Erfolg eingesetzter Algorithmen, die teilweise erweitert wurden. In [12, S. 29] geben die Autoren, die ursprünglich dem Team *BelKor* angehörten, an, dass ihr Beitrag am Ende des ersten Jahres bereits die Ergebnisse von 107 Algorithmen linear kombinierte, um das endgültige Ergebnis zu berechnen. Dabei kamen sowohl verschiedene Formen Nächster-Nachbar-Methoden zum Einsatz als auch Latente Variablenmodelle. Bei der Fusion mit den Gruppen *BigChaos* und *Pragmatic Theory* in den Jahren 2008 und 2009 kamen als weitere Ansätze unter anderem noch neuronale Netzwerke zur nichtlinearen Kombination von Ergebnissen und die Integration von Bewertungshäufigkeiten in das Modell des Nutzerverhaltens über die Zeit hinzu.

Bei der Durchsicht der Artikel [9, 10, 11] der Gewinnerteams stößt man auf viele bekannte Ansätze, die in allen möglichen Variationen und Verfeinerungen miteinander kombiniert werden. Sehr wichtig scheint die Integration der Zeitabhängigkeit zu sein, denn die zieht sich wie ein roter Faden durch alle Beiträge. Ebenfalls von Vorteil ist die Kombination nachbarschaftsbasierter Ansätze mit latenten Variablenmodellen verschiedener Couleur. Darüber hinaus ist es aber für Nicht-Experten sehr schwierig nachzuvollziehen, welches nun die entscheidenden Drehs und Kniffe sind, die letztendlich zum Erfolg geführt haben. Die Teammitglieder von *Pragmatic Theory* stellen in ihrem Paper [9] dann auch fest, dass natürlich theoretisches Wissen, passende Modellierung und die geschickte Auswahl von Ansätzen eine wichtige Rolle spielen, dass zum Auffinden der wirklich guten Lösungen aber auch eine gehörige Portion Pragmatismus ins Spiel kommen muss und nicht zuletzt, dass der Entwurf eines Recommendersystems für einen Wettbewerb definitiv etwas anderes ist als das Design von alltagstauglichen Lösungen [9, S. 3]:

> *As the name of our team implies, our strategy in this competition was to try anything and everything; to leave no stone unturned. Although we have always tried to choose our methods logically, some of the resulting predictors may not have a proper theoretical or psychological grounding;*

> *they were instead selected for their contribution to the blended prediction accuracy. Also, because of this pragmatic approach, not all of the concepts presented here will be usable in a real world recommender system. Still, we believe that our approach has allowed us to find original and creative ideas and to bring known algorithms to a new level, which should, in turn, allow for significant improvement to recommendation engines.*

Im Alltagsgeschäft wurde der Algorithmus, der den Netflix-Wettbewerb gewonnen hat, dann auch nie eingesetzt (vgl. [13]). Nach einem Jahr Wettbewerbslaufzeit übernahm Netflix vom aktuell führenden *BellKor*-Team, das den Fortschrittspreis gewann, zwei der besonders vielversprechenden Algorithmen, nämlich Matrixfaktorisierung und beschränkte Boltzmann-Maschinen. Die Lösung des Teams musste für die tatsächliche Verwendung noch etwas angepasst werden, zeigte dann aber gute Ergebnisse. Nach Abschluss des Wettbewerbs wurden einige der zusätzlich im Gewinnerbeitrag verwendeten Algorithmen getestet, der Genauigkeitszuwachs bei der Bewertungsvorhersage aber als zu gering eingestuft, um den Anpassungsaufwand für den Einsatz im Live-System zu rechtfertigen. Als weiterer Grund wird auf dem oben zitierten *Netflix Tech Blog* angeführt, dass sich durch den zunehmenden Ausbau des Streamingangebotes sowohl die Ansprüche an Empfehlungen als auch die Art ihrer Generierung während der Laufzeit des Wettbewerbs verändert haben.

2.4.2 Produktempfehlungen bei Amazon

Weil die Berechnung von Empfehlungen basierend auf Nutzerähnlichkeit mit steigender Nutzeranzahl immer aufwendiger wird, hatte Amazon als erfolgreiches E-Commerce-Unternehmen schon früh ein großes Interesse an alternativen Ansätzen. Amazon-Mitarbeiter haben deshalb vor mehr als 10 Jahren das artikelbasierte kollaborative Filtern entwickelt, das heute zu den Standardansätzen für Recommendersysteme gehört [5].[12]

Dieser Ansatz bildet nach wie vor die Basis für die Empfehlungsgenerierung bei Amazon und wurde sukzessive ausgebaut, wie zahlreiche Patente belegen. Ein Patent aus dem Jahr 2013 [15] nennt folgende Informationen, aus denen sich das Nutzerprofil zusammensetzt:

- Bewertungen, die der Nutzer abgegeben hat,

[12] Die Idee, Artikelähnlichkeiten für kollaborative Filter zu nutzen, findet sich auch schon in [14].

- Artikel, die der Nutzer gekauft hat,
- Artikel, die der Nutzer verschenkt hat,
- kürzlich betrachtete Artikel,
- Inhalt des Einkaufswagens,
- kürzlich aus dem Einkaufswagen entfernte Artikel sowie
- Wunschlisteninhalte.

Alle diese Informationen können einzeln oder in Kombination als Grundlage für Empfehlungen dienen.

Dazu werden, wie beim kollaborativen Filtern üblich, in regelmäßigen Abständen offline Ähnlichkeitstabellen erstellt, auf die später online zur Generierung von Empfehlungen zurückgegriffen werden kann. Die berechneten Ähnlichkeiten beruhen auf Korrelationen, die aus den oben genannten Informationsquellen gewonnen werden und das kollektive Interesse der Nutzer widerspiegeln. Der Ablauf ist im Prinzip immer gleich und soll daher im Folgenden am Beispiel von gekauften Artikeln erläutert werden.

In einem ersten Schritt werden aus den Nutzerhistorien zwei Listen generiert, die wir wie folgt bezeichnen wollen: $A(k)$ ist eine Liste, die von Nutzer k auf die Artikel verweist, die dieser gekauft hat. Mehrfache Käufe eines Artikels werden zu einem zusammengefasst. $K(i)$ ist eine Liste, die von Artikel i auf die Nutzer verweist, die diesen gekauft haben. Diese Liste hat die Länge n_i. Es bietet sich an, die dazu herangezogenen Daten zunächst zu bereinigen, indem zum Beispiel Artikel-IDs in Titel-IDs umgewandelt werden, um unterschiedliche Versionen desselben Artikels gleich zu behandeln, oder indem nur Käufe in einem bestimmten Zeitraum betrachtet werden.

Als Nächstes werden für alle beliebten Artikel, die von einer Mindestanzahl an Nutzern gekauft wurden, Listen mit ähnlichen Artikeln erstellt und für jeden dieser Artikel der Grad der Ähnlichkeit berechnet. Die Liste für Artikel i bezeichnen wir mit $S(i)$. Um nicht alle möglichen Artikelpaare vergleichen zu müssen und somit Rechenzeit zu sparen, wird wie folgt vorgegangen:

```
1   for i ∈ beliebte Artikel
2       for k ∈ K(i)
3           for j ∈ A(k)
4               n_ij ← n_ij + 1
5               add j to S(i)
6       for j ∈ S(i)
7           CI(i, j) = n_ij ÷ √(n_i · n_j)
```

Algorithmus 2.1: Berechne Artikelähnlichkeiten

In der letzten Zeile wird das Ähnlichkeitsmaß, der sogenannte *commonality index*, berechnet. Dieser nimmt dann seinen Maximalwert 1 an, wenn alle Nutzer, die i kaufen, auch j kaufen, und seinen Minimalwert, wenn i und j nie zusammen gekauft werden. Dieses Ähnlichkeitsmaß wird in Patent [15] erläutert. Natürlich können auch andere Maße verwendet werden, wie etwa die Kosinusähnlichkeit, wenn jeder Artikel als Kundenvektor repräsentiert wird (vgl. hierzu z.B. [5]). Die Listen werden nach CI sortiert und Artikel, die eine zu schwache Korrelation aufweisen, werden gelöscht. Für jeden beliebten Artikel werden, um Speicher zu sparen, maximal N ähnliche Artikel gespeichert und die Listen gegebenenfalls gekürzt. Zum Speichern sollte eine geeignete Datenstruktur gewählt werden, die das schnelle Auffinden der Listen ermöglicht. Häufig werden B-Bäume verwendet.

Aus den so entstandenen Listen mit ähnlichen Artikeln können anschließend wie folgt Empfehlungen generiert werden: Zunächst werden Artikel ausgewählt, an denen der Nutzer bekanntermaßen Interesse hat. Je nach Kontext, in dem die Empfehlungen generiert werden sollen, können dies gekaufte oder gut bewertete Artikel sein, aber auch kürzlich angesehene oder verschenkte. Für jeden dieser Artikel wird die Liste ähnlicher Artikel gewichtet mit einem Wert, der sich nach dem Interesse des Nutzers am Ausgangsartikel bemisst und jeder Artikel in der Liste erhält als Score das Produkt des ursprünglichen CI und des berechneten Gewichtsfaktors. Anschließend werden alle gewichteten Listen zusammengeführt. Dabei können entweder nur Artikel empfohlen werden, die in allen Listen ähnlicher Artikel auftauchen, oder die Scores von mehrfach auftretenden Artikeln werden addiert. Bevor dem Benutzer die Empfehlungen angezeigt werden, werden sie noch gefiltert, um unpassende Artikel zu entfernen, dann auf eine passende Länge gekürzt und gegebenenfalls vorher um einen weiteren Artikel ergänzt, der nach einem anderen Verfahren (z.B. inhaltsba-

siert) ausfindig gemacht wurde oder einer der vom Nutzer kürzlich betrachteten oder aus dem Einkaufswagen entfernten Artikel ist.

In Patent [15] werden drei verschiedene Einsatzmöglichkeiten für Empfehlungen genannt, die bei Amazon auch tatsächlich umgesetzt werden und bei deren Berechnung unterschiedliche Ähnlichkeitslisten zum Einsatz kommen:

1. **Sofort-Empfehlungen** (*instant recommendations*): Diese Empfehlungen werden aktiviert, wenn der Nutzer einem entsprechenden Link folgt. Die für ihn interessanten Artikel werden aus Käufen und Bewertungen abgeleitet. Die verwendete Ähnlichkeitstabelle ist in der Regel die, die auf Produktkäufen basiert. Der Nutzer kann diesen Empfehlungsservice nutzen, sobald er eine Mindestanzahl von beliebten Artikeln gekauft hat. Die Gewichtung der Listen ähnlicher Artikel erfolgt nach dem Kaufdatum: Je weniger Zeit seit dem Kauf des beliebten Artikels vergangen ist, desto höher wird die entsprechende Liste gewichtet. Der Nutzer kann die Empfehlungen verfeinern, indem er nach Kategorien filtert oder angibt, welche der empfohlenen Artikel er bereits besitzt.

2. **Einkaufswagen-Empfehlungen**: Sobald der Nutzer sich seinen Einkaufswagen anzeigen lässt, werden diese Empfehlungen generiert. Voraussetzung ist, dass sich mindestens ein beliebter Artikel im Einkaufswagen befindet, der als Grundlage verwendet werden kann. Ergänzend können außerdem kürzlich betrachtete oder aus dem Einkaufswagen entfernte Artikel herangezogen werden. Hat der Nutzer Suchanfragen ausgeführt, so können diese gegebenenfalls verwendet werden, um die generierte Empfehlungsliste inhaltsbasiert zu filtern. Die auf diese Weise berechneten Empfehlungen spiegeln stark die kurzzeitigen Interessen des Nutzers wieder.

3. **Session-Empfehlungen**: Die Artikel, die der Nutzer während einer Sitzung betrachtet, werden als Grundlage für Empfehlungen genutzt, wobei der Nutzer die Möglichkeit hat, einzelne Artikel auszuschließen. In der Regel wird die Ähnlichkeitstabelle verwendet, die unter Rückgriff auf Verlaufsdaten erstellt wurde. Das hat den Vorteil, dass dem Nutzer nicht nur ergänzende Artikel empfohlen werden („Staubsauger wird oft gekauft mit Beuteln“), sondern auch Alternativen („Wer den Samsung-Fernseher betrachtet hat, hat auch den LG-Fernseher angesehen“). Außerdem spiegeln Session-Empfehlungen stark die kurzzeitigen Nutzerinteressen wider, so dass etwa ein Nutzer, der abseits seiner sonstigen Vorlieben nach einem Geschenk sucht, sinnvolle Empfehlungen erhält.

3 Elasticsearch

Elasticsearch [24] ist eine Open-Source-Suchmaschine, gebaut für den Einsatz auf verteilten Systemen und die Echtzeitverarbeitung von großen Datenmengen, die sowohl zur Volltextsuche als auch zur Suche in strukturierten Daten und zu Analysezwecken eingesetzt wird. *Elasticsearch* ist in Java geschrieben und basiert auf der Bibliothek *Lucene* [25].

In den folgenden Abschnitten werden das Fundament von *Elasticsearch*, die bereitgestellten Funktionalitäten und die zur Verfügung gestellten Schnittstellen vorgestellt. Bei meiner Beschreibung orientiere ich mich an „Elasticsearch. The definitive Guide“ von Clinton Gormley und Zachary Tong [16] sowie an der API-Dokumentation [26].

3.1 Grundlagen

Elasticsearch nutzt als Grundlage *Lucene*, eine ursprünglich von Dough Cutting entwickelte Java-Bibliothek, die Funktionen zur Volltextsuche zur Verfügung stellt. *Lucene* gibt es seit 1997, inzwischen ist daraus ein Projekt der *Apache Software Foundation* geworden, zu dem es eine Menge Teilprojekte und Abspaltungen gibt. Neben *Elasticsearch* basiert zum Beispiel auch die Suchmaschine *Solr* [27] auf *Lucene*.

3.1.1 Anbindung und Schnittstellen

Erklärtes Ziel der Entwickler ist es, den Einstieg in *Elasticsearch* möglichst einfach zu gestalten. Damit *Elasticsearch* auf verteilten Systemen läuft, Suchanfragen in Echtzeit beantwortet werden können und Volltextsuche und Analyse möglich sind, sind im Hintergrund zwar komplexe Prozesse nötig, diese bleiben dem Nutzer aber weitgehend verborgen.

Elasticsearch stellt alle Funktionen über eine REST-Schnittstelle zur Verfügung. Die zu speichernden Daten werden in Form von JSON-Dokumenten beschrieben. Jedem Attribut des Objekts entspricht ein Feld im JSON-Dokument. Nimmt der Nutzer keine weiteren Einstellungen vor, wird die Struktur der Objekte von *Elasticsearch*

automatisch erkannt. Alle Felder werden auf die grundlegenden JSON-Typen abgebildet, wobei *String*-Felder grundsätzlich analysiert und damit für die Volltextsuche zugänglich gemacht werden. Dieses dynamische Mapping kann durch ein vom Nutzer definiertes ersetzt werden.

3.1.2 Datenhaltung und -verteilung

Alle Dokumente, die in *Elasticsearch* eingefügt werden, werden in einem Index gespeichert. Man spricht daher auch vom Indizieren von Dokumenten. In einem Index können unterschiedliche Typen gespeichert werden. Ein *Elasticsearch*-Typ besteht aus einem Namen sowie einem Mapping, das angibt, in welcher Form die Attribute der Objekte des Typs gespeichert werden sollen. Festgelegt wird, welchen Datentyp das dem jeweiligen Attribut entsprechende Feld haben soll, und außerdem, wie die Feldinhalte indiziert werden sollen (gar nicht, unanalysiert als feste Werte oder in analysierter Form (vgl. 3.2)).

Ein kurzes Beispiel mag dies verdeutlichen: Der Index *Rezeptsammlung* enthält Dokumente vom Typ *Rezept*. Die JSON-Darstellung eines Rezepts für Glühwein könnte wie folgt aussehen:

```
{
        "Titel": "Glühwein",
        "Zutaten": [
                "150 ml Wasser",
                "3 EL Zucker",
                "3 Stangen Zimt",
                "2 Anissterne",
                "4 Nelken",
                "750 ml trockener Rotwein"
        ],
        "Zubereitung": "Den Zucker im Wasser auflösen.
                Die restlichen Gewürze zugeben und eine gute
                Stunde bei geringer Hitze köcheln lassen.
                Anschließend den Wein hinzugeben und erwärmen."
}
```

Jedes Dokument vom Typ *Rezept* hat die Felder „Titel“, „Zutaten“ und „Zuberei-

tung“. Dass diese mit Strings bzw. String-Arrays belegt werden sollen, kann in einem Mapping explizit festgelegt werden und wird ansonsten von *Elasticsearch* beim Einfügen des ersten Rezepts automatisch erkannt. Beim Einfügen in den Index erhält jedes Dokument eine ID. Diese kann vorgegeben oder automatisch generiert werden. Das Dokument mit ID 15 kann über die Restschnittstelle wie folgt abgerufen werden:

```
GET /Rezeptsammlung/Rezept/15
```

Prinzipiell können die in *Elasticsearch* zu speichernden Objekte beliebig komplex sein und auch andere Objekte enthalten. Mehrdimensionale Objekte müssen, anders als bei klassischen Datenbanken, nicht explizit durch Normalisierung in eine passende relationale Form gebracht werden.[13]

Wie sich die *Elasticsearch*-Begrifflichkeiten auf die der klassischen relationalen Datenbanken abbilden lassen, verdeutlicht die folgende Tabelle, die Analogien aufzeigt:

Relationale Datenbank	*Elasticsearch*
Datenbank	Index
Tabelle	Typ
Zeile	Dokument
Spalte	Feld

Tabelle 3.1: Vergleich relationale Datenbank – *Elasticsearch*

Das oben gezeichnete Bild ist in Wirklichkeit noch etwas komplexer. Zwei wichtige Eigenschaften von *Elasticsearch* wurden nämlich bislang noch gar nicht angesprochen: Datenreplikation und Verteilung. Diese Mechanismen sorgen für Ausfallsicherheit und eine bessere Lastverteilung. Konkret bedeutet das, dass die Daten in einem Cluster vorgehalten werden, der aus mehreren Knoten besteht, die auf unterschiedlichen physikalischen Rechnern laufen können. Jedem Knoten entspricht eine Instanz von *Elasticsearch*. Zusammengehörige Knoten, die demselben Cluster angehören, finden sich automatisch und schließen sich zusammen. Dabei kann jeder *Elasticsearch*-Cluster mehrere Indizes beinhalten. Die Dokumente der einzelnen Indizes werden auf sogenannte *shards*, zu Deutsch Scherben, aufgeteilt, die auf die Knoten verteilt werden. Jeder *shard* ist eine eigene *Lucene*-Instanz.

[13] Weil das zugrundeliegende *Lucene* allerdings nur eine einfache Liste von Feld-Wert-Paaren speichern kann, wird intern dennoch eine entsprechende Umwandlung vorgenommen, die dem Nutzer allerdings verborgen bleibt.

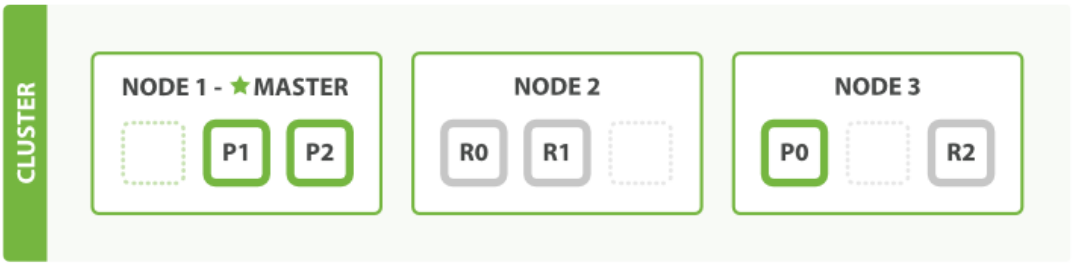


Abbildung 3.1: *Elasticsearch*-Cluster mit drei Knoten. Die Dokumente sind verteilt auf drei je einmal replizierte *shards*. Abbildung entnommen aus [16].

Bei der Indexerstellung wird festgelegt, auf wie viele primäre *shards* die Dokumente im Index verteilt werden sollen. Jeder dieser primären Container kann beliebig oft dupliziert werden, um durch das Vorhalten von Kopien auf anderen Knoten Ausfallsicherheit zu erreichen. Ob es solche *replica shards* gibt und wie häufig die primären *shards* repliziert werden, ist konfigurierbar und kann auch nach der Indexerstellung noch geändert werden.

Die obigen Erläuterungen machen deutlich, dass jeder Index im Grunde als eine Art logischer Namespace fungiert, der auf einen oder mehrere *shards* verweist.

Abbildung 3.1 zeigt, wie ein *Elasticsearch*-Cluster mit drei Knoten und drei *shards*, die jeweils einmal repliziert werden, aussehen könnte.

3.2 Suche

Um Dokumente durchsuchbar zu machen, legt *Elasticsearch* bzw. das zugrundeliegende *Lucene* eine sogenannte invertierte Datei (*inverted index*) an. In dieser wird für jeden Term im Index gespeichert, in welchen Dokumenten er vorkommt. Unter Term versteht man hierbei einen Textbestandteil. Dies kann eine Zeichenkette sein, die genau in dieser Form im Dokument vorkommt, aber auch eine Stammform des entsprechenden Wortes. In der invertierten Datei können sogar Terme auftauchen, die in keinem Dokument vorkommen, denn *Elasticsearch* erlaubt es auch, Synonyme abzuspeichern, so dass zum Beispiel eine Suche nach „Pasta“ auch Dokumente zurückliefert, in denen „Nudel“ oder „Nudeln“ vorkommt.

Was genau in der invertierten Datei enthalten ist, hängt von der Analyse ab, die

beim Einfügen eines neuen Dokuments in den Index ausgeführt wird. Eine solche Analyse verläuft in drei Schritten, deren Ausgestaltung jeweils konfiguriert werden kann:

1. **Zeichenfilter:** Ersetzung oder Entfernung einzelner Zeichen, um für einen einheitlichen Zeichensatz zu sorgen oder Probleme mit diakritischen Zeichen und Sonderzeichen zu vermeiden.

2. **Zerlegung in Tokens:** Die gefilterte Zeichenkette wird in ihre Bestandteile (Tokens) zerlegt. Trennzeichen für Tokens können zum Beispiel Leer- und Satzzeichen sein oder die Trennstellen können durch reguläre Ausdrücke vorgegeben werden.

3. **Normalisierung:** Im letzten Schritt werden die so erhaltenen Tokens noch normalisiert. Das heißt, es kommt ein weiterer Filter zum Einsatz, der zum Beispiel alle Groß- in Kleinbuchstaben umwandelt, sogenannte Stoppwörter entfernt oder sogar noch Terme hinzufügt. Stoppwörter sind Wörter, die häufig vorkommen, selbst aber wenig Information tragen, wie etwa „und“, „oder“, „sie“, „er“, „es“. Bei den zusätzlichen Termen kann es sich zum Beispiel um Synonyme handeln oder um sogenannte n-Gramme, also Buchstabenfolgen der Länge n, aus denen sich das ursprüngliche Token zusammensetzt („gestern“ etwa enthält die Trigramme „ges“, „est“, „ste“, „ter“ und “ern“). N-Gramme können zum Beispiel zur Suchwortergänzung genutzt werden.

Für jeden nach dem letzten Schritt erhaltenen Term wird beim Einfügen eines neuen Dokuments in der invertierten Datei eingetragen, dass er in diesem Dokument vorkommt.

Wird eine Suchanfrage zur Volltextsuche gestellt, dann wird die Sucheingabe ebenfalls analysiert. Dies geschieht entweder auf dieselbe Weise wie beim Einfügen des Dokuments oder in etwas abgewandelter Form. So könnte es zum Beispiel sinnvoll sein, beim Einfügen in den Index neben den eigentlichen Termen noch Synonyme zu speichern, beim Suchen aber auf das Hinzufügen von Synonymen zur Suchanfrage zu verzichten und nur die anderen Normalisierungsschritte auszuführen.

Weil *Elasticsearch* neben der Volltextsuche auch die sogenannte strukturierte Suche unterstützt, kann es sinnvoll sein, für gewisse Felder in der invertierten Datei auch deren Inhalt unverändert zu speichern. Mehr zu den grundsätzlichen Sucharten in den folgenden Abschnitten.

3.2.1 Strukturierte Suche

Die strukturierte Suche mit *Elasticsearch* ähnelt der Suche in einer klassischen relationalen Datenbank. Gesucht werden Dokumente, die der Anfrage exakt entsprechen. Es gibt keine mehr oder weniger passenden Ergebnisse und ein Scoring, also eine Bewertung der Ergebnisse, findet nicht statt. Zur strukturierten Suche werden Filter verwendet.

Gefiltert werden kann nach einem oder mehreren Termen in einem bestimmten Feld oder auch danach, ob Feldinhalte in einen bestimmten Bereich fallen (Zahlen, Daten und auch Strings, wobei Letztgenanntes aber gegebenenfalls mit Performanceeinbußen einhergehen kann). Um komplexere Anfragen zu bauen, können mehrere Filter kombiniert werden.

Wird eine Filteranfrage an *Elasticsearch* gesendet, dann werden intern drei Schritte ausgeführt:

1. Der Suchterm wird in der invertierten Datei gesucht und es wird eine Liste mit Dokumenten erstellt, die diesen Term im passenden Feld enthalten.
2. Auf Grundlage der Dokumentenliste wird dann ein Bitset erstellt, das beschreibt, welche Dokumente den gesuchten Term enthalten.
3. Bevor der Nutzer die Dokumentenliste erhält, wird das Bitset gecached. Für manche Filter, die zum Beispiel die aktuelle Uhrzeit oder eine aktuelle Position verwenden, ist das nicht möglich. Befindet sich ein Filter schon im Cache, dann wird die Dokumentenliste direkt zusammengestellt und zurückgegeben.

Aus dieser Beschreibung geht hervor, dass beim Filtern nicht nur Dokumente zurückgeliefert werden, die im durchsuchten Feld *ausschließlich* den gesuchten Term enthalten. Dies in der invertierten Datei zu überprüfen wäre sehr ineffizient. Wenn ein solches Verhalten gewünscht ist, müssen dazu zusätzliche Informationen gespeichert und genutzt werden, wie etwa die Länge der entsprechenden Felder.

Außerdem liefert die strukturierte Suche nur Dokumente zurück, für die der exakte Suchterm in der invertierten Datei auftaucht. War der Term zwar im ursprünglichen Dokument vorhanden, wurde beim Indizieren des Dokuments aber analysiert und nur in veränderter Form in die invertierte Datei aufgenommen, kommt es zu unerwünschten Ergebnissen. Es empfiehlt sich daher, Felder, die zur strukturierten Suche zur Verfügung stehen sollen, entweder nicht zu analysieren oder sowohl eine ana-

lysierte als auch eine unveränderte Version abzuspeichern und für die strukturierte Suche auf die unveränderten Feldinhalte zuzugreifen.

Filter können genutzt werden, um die Performanz von Volltextsuchen zu verbessern. Bei einer gefilterten Suchanfrage (*filtered query*) wird zunächst der Filter ausgeführt und das resultierende Bitset dann an die Suchanfrage übergeben, so dass für die Suche nur noch jene Dokumente beachtet werden, die den Filterkriterien entsprechen. Kommen mehrere Filter zum Einsatz, sollte in der Regel der spezifischste Filter zuerst ausgeführt werden, um die Dokumentenanzahl möglichst rasch zu reduzieren.

3.2.2 Volltextsuche

Anders als bei der strukturierten Suche gibt es bei der Volltextsuche nicht nur „passende“ und „unpassende“ Dokumente, sondern Dokumente, deren Inhalte der Suchanfrage mehr oder weniger entsprechen. Die Ergebnisse, die zurückgeliefert werden, werden deshalb mit einem Score versehen, der ihre Relevanz widerspiegelt. Mehr zur Berechnung von Scores in Abschnitt 3.2.3.

Zur Volltextsuche werden Queries eingesetzt. Diese gibt es in zwei Ausführungen[14], nämlich termbasiert und als echte Volltextsuche. Beide Varianten werden in den folgenden Abschnitten erläutert.

Termbasierte Queries

Termbasierte Queries arbeiten wie Filter: Sie analysieren die Suchanfrage nicht, sondern durchsuchen die invertierte Datei nach dem exakten Suchwort. Für jedes aufgefundene Dokument wird dann ein Score berechnet und die Ergebnisse werden nach Relevanz sortiert zurückgegeben.

Diese Queries werden selten direkt verwendet, dienen aber als Grundlage für die Volltextsuche.

[14] Genau genommen gibt es zwei Arten von Queries, die dem Absetzen von (Text)Suchanfragen dienen. Daneben existieren noch spezielle Queries, mit denen man zum Beispiel den Score verändern oder andere Queries verknüpfen kann.

Volltextsuche

Die Volltextsuche ist „intelligenter“ als Filter oder termbasierte Queries. Sie berücksichtigt das Mapping eines Feldes bei der Entscheidung darüber, wie Suchanfragen behandelt werden sollen:

- Wenn ein Datums- oder Zahlenfeld durchsucht werden soll, wird die Sucheingabe entsprechend interpretiert.
- Wird ein nicht analysiertes Feld durchsucht, wird die gesamte Suchanfrage als einzelne Zeichenkette behandelt und dann auf Übereinstimmungen geprüft.[15]
- Beim Durchsuchen eines analysierten Feldes wird die Sucheingabe selbst zunächst mit einem passenden Analysierer behandelt, so dass die produzierten Suchterme den (potentiellen) Feldinhalten entsprechen.

Die Suchergebnisse werden dann nach Wichtigkeit sortiert zurückgeliefert. Neben Einzeltermen können auch mehrere Wörter gesucht werden. Standardmäßig werden alle Dokumente zurückgeliefert, die mindestens eines der Wörter enthalten. Man kann aber auch eine Mindestanzahl an Übereinstimmungen festlegen oder erzwingen, dass alle Suchbegriffe enthalten sein müssen. Außerdem ist eine Suche über mehrere Felder und natürlich die Kombination von Suchanfragen möglich. Welche Kombination welcher Anfragen am besten zum Ziel führt, ist von den jeweils zu verarbeitenden Daten abhängig. Von diesen und den erwarteten Anfragen hängt auch ab, welche Felder beim Indizieren sinnvollerweise zusätzlich eingeführt werden sollten, wie die einzelnen Felder bei der Aufnahme des Dokuments in den Index und später bei der Suche analysiert werden sollten und welche Felder möglicherweise sogar mehrfach mit unterschiedlichen Analysierern behandelt und vorgehalten werden sollten.

3.2.3 Scoring

Um Dokumente zu finden, die der Suchanfrage entsprechen, verwendet *Elasticsearch* ein boolesches Modell. Die durch die einzelnen Bestandteile der Suchanfrage ausgedrückten Bedingungen werden durch die Operatoren „Und“, „Oder“ und „Nicht“ verbunden und passende Dokumente zurückgeliefert. Diese Dokumente sollen dem

[15] Im Endeffekt entspricht dies in der Regel der Funktion eines Filters, kann aber im Gegensatz zum Filter nicht gecached werden.

Benutzer nach Relevanz sortiert präsentiert werden. Zur Berechnung der Relevanz werden die von *Lucene* bereitgestellten Scoring-Algorithmen verwendet. Standardmäßig kommt TF-IDF zum Einsatz, alternativ kann auch Okapi BM25 verwendet werden.

In den folgenden Abschnitten wird zunächst der auf TF-IDF basierende Scoring-Algorithmus von *Elasticsearch* vorgestellt, darauf folgt ein kurzer Überblick über die Möglichkeiten, Einfluss auf die berechneten Scores zu nehmen.

Lucene-Scoringfunktion

Elasticsearch verwendet die Scoring-Funktion von *Lucene*, die auf TF-IDF aufbaut. Die Grundidee dieses Scoring-Ansatzes besteht darin, dass ein Dokument (im Falle von *Elasticsearch* eigentlich ein Feld) umso relevanter ist, je öfter es die gesuchten Terme enthält. Dies wird durch die *Term Frequency* (TF) beschrieben. Allerdings ist Häufigkeit allein noch nicht ausschlaggebend, denn es gibt Terme, die in allen durchsuchbaren Dokumenten, dem so genannten Korpus, sehr häufig vorkommen. Das wird durch die *Inverse Document Frequency* (IDF) gemessen. Solche häufigen Terme sollen sich entsprechend weniger stark auf die Relevanz eines Dokuments auswirken als Terme, die über alle Dokumente betrachtet selten auftauchen, im aktuellen Dokument aber sehr oft. In die Scoring-Funktion von *Lucene* fließt darüber hinaus noch die Länge des Feldes ein, in dem der gesuchte Term gefunden wurde. Damit soll eine Dominanz von langen Feldern verhindert werden. Außerdem kommen noch Koordinierungs- und Verstärkungsfaktoren hinzu.

Übereinstimmung von Dokument und Suchanfrage Um festzustellen, wie gut ein Dokument einer Suchanfrage entspricht, kommt ein Verfahren zum Einsatz, das uns schon an früherer Stelle begegnet ist: die Kosinusähnlichkeit (vgl. 2.2.2). Suchanfrage und Dokument werden als Vektoren dargestellt, die für jedes Suchwort einen Eintrag enthalten. Dieser ist ein Gewichtungsfaktor, der sich aus den oben genannten Bestandteilen zusammensetzt. Wie diese jeweils berechnet und schließlich kombiniert werden, ist das Thema der folgenden Abschnitte.

Term Frequency Zur Berechnung der TF gibt es verschiedene Ansätze. Die einfachste Herangehensweise ist, nur die tatsächliche Häufigkeit zu zählen. Allerdings werden die Unterschiede mit ansteigender Häufigkeit zunehmend unwichtiger. Sicher

ist ein Dokument, in dem der gesuchte Term fünfmal auftaucht, deutlich relevanter als eines, in dem er überhaupt nicht vorkommt. Ob aber ein Dokument mit 50 Vorkommen grundsätzlich relevanter ist als eines mit 30, darf bezweifelt werden. Im Allgemeinen werden daher Funktionen gewählt, deren Wachstum mit zunehmendem Argumentwert geringer wird.

Bei *Lucene* ist dies die Wurzelfunktion. Die TF von Term t in Dokument d wird berechnet als:

$$\mathrm{TF}(t, d) = \sqrt{freq(t, d)}.$$

Weil die TF von Term t nur für Dokumente berechnet wird, in denen t vorkommt, ist $\mathrm{TF}(t, d)$ stets größer-gleich Eins.

Inverse Document Frequency Der Berechnung der IDF liegen Ideen der Informationstheorie zugrunde: Nach Shannon ist der Informationsgehalt eines Zeichens x definiert als $\log \frac{1}{p(x)}$, wobei $p(x)$ die Auftrittswahrscheinlichkeit des Zeichens x ist. Je geringer die Auftrittswahrscheinlichkeit, desto höher der Informationsgehalt. Das gilt auch für die Suchterme: Je seltener sie im gesamten Korpus vorkommen, desto signifikanter ist ihr Auftreten in einem bestimmten Dokument.

Bezeichne K das gesamte Korpus an durchsuchbaren Dokumenten, T die Menge an Dokumenten, in denen Term t mindestens einmal vorkommt. Dann berechnet sich die IDF von Term t wie folgt:

$$\mathrm{IDF}(t) = 1 + \log \frac{|K|}{|T| + 1}.$$

$\frac{|K|}{|T|}$ ist im Endeffekt nichts anderes als $\frac{1}{p(t)}$, wobei $p(t)$ die Wahrscheinlichkeit ist, dass es sich bei einem zufällig herausgegriffenen Dokument um eines handelt, welches t enthält. Die beiden Einsen sind Korrekturfaktoren: Die Addition von Eins im Nenner verhindert eine Division durch Null, wenn t in keinem Dokument vorkommt, die vordere Eins verhindert, dass $\mathrm{IDF}(t)$ Null wird, wenn t in allen Dokumenten vorkommt. Letzteres wird deshalb zum Problem, weil zur Berechnung des endgültigen Scores mit dem IDF-Wert multipliziert wird.

Normierte Feldlänge Der dritte Bestandteil des Scores ist die normierte Feldlänge. Die Idee ist, kurze Felder höher zu gewichten als lange Felder, weil hier die

Wahrscheinlichkeit, dass der Term den Inhalt des Feldes beschreibt, höher ist als bei sehr langen Feldern, in denen der Term auch auftaucht. Weil auch in diesem Fall die Längenunterschiede kurzer Felder bedeutsamer sind als die längerer Felder, kommt wieder die Wurzelfunktion zum Einsatz und die normierte Feldlänge berechnet sich als

$$\text{Norm}(d) = \frac{1}{\sqrt{|d|}},$$

wobei $|d|$ die Länge des Feldes bezeichnet.

Kombination zu einem Score TF, IDF und Norm werden beim Indizieren von Dokumenten berechnet und gespeichert. Miteinander multipliziert bilden sie das Gewicht des jeweiligen Terms im Vektor, der das Dokument beziehungsweise die Suchanfrage repräsentiert.

Um die Ähnlichkeit von Suchanfrage und Dokument zu bestimmen, kommt eine abgewandelte Kosinusähnlichkeit zum Einsatz, bei der die Länge des Dokumentenvektors nicht mittels der euklidischen Norm berechnet wird. Stattdessen wird die oben bereits erwähnte Feldnorm verwendet. Dies hängt damit zusammen, dass bei der Division durch die euklidische Norm ein Einheitsvektor entsteht, der keinerlei Information über die Länge des Dokuments beziehungsweise des entsprechenden Feldes mehr enthält (vgl. hierzu sowie zu den folgenden Ausführungen auch [28]).

Als zusätzliche Parameter kommen noch hinzu:

- ein Koordinationsfaktor $\text{coord}(q, d)$, der dafür sorgt, dass Dokumente, in denen viele der gesuchten Terme vorkommen, stärker gewichtet werden als solche, in denen nur ein kleiner Teil der Suchworte enthalten ist. *Elasticsearch* verwendet hier die Anzahl der im Dokument enthaltenen Suchbegriffe geteilt durch die Gesamtzahl Suchbegriffe.
- ein Verstärkungsfaktor $\text{boost}(t)$, der pro Term in der Suchanfrage gesetzt werden kann. Dies ist eine der Möglichkeiten, auf den Score Einfluss zu nehmen und wird in Abschnitt 3.2.3 besprochen.

Der endgültige Score wird dann wie folgt berechnet:

$$\begin{aligned}\text{score}(q, d) =& \text{queryNorm}(q) \cdot \text{coord}(q, d) \\ & \cdot \sum_{t \in q} (\text{TF}(t, d) \cdot \text{IDF}(t)^2 \cdot \text{boost}(t) \cdot \text{norm}(t, d)).\end{aligned}$$

Dabei ist queryNorm(q) der Kehrwert der euklidischen Norm des Anfragevektors. Bei der Berechnung des Skalarprodukts werden alle Terme der Suchanfrage einzeln behandelt. Ihre TF wird auf Eins gesetzt und taucht daher in der obigen Gleichung nicht auf.

Einflussmöglichkeiten

Elasticsearch bietet an mehreren Stellen die Möglichkeit, Einfluss auf die Berechnung des Scores zu nehmen. Einen kurzen Überblick über die verschiedenen Optionen geben die folgenden Abschnitte.

Einflussnahme bei der Indizierung Prinzipiell ist es möglich, beim Indizieren eines Dokuments ein Feld mit einem *boost*-Wert zu verstärken. Dieser wird auf alle Terme des Felds angewandt und direkt mit der Feldnorm verrechnet, um Speicher zu sparen. Die Dokumentation von *Elasticsearch* rät allerdings davon ab, diese Möglichkeit der Einflussnahme auf den Score zu nutzen, weil sie mehrere Nachteile mit sich bringt:

- Die Feldnorm wird in einem einzigen Byte gespeichert. Die Multiplikation mit einem zusätzlichen Faktor führt zu Präzisionsverlust.
- Wenn Veränderungen an der Verstärkung vorgenommen werden sollen, müssen alle betroffenen Dokumente neu indiziert werden.
- Da der *boost*-Wert auf alle Terme des Feldes angewandt wird, erhalten lange Felder ein unverhältnismäßig hohes Gewicht.

Einflussnahme in Suchanfragen Die einfachste und wichtigste Möglichkeit, in der Suchanfrage Einfluss auf den Score zu nehmen, besteht in deren Strukturierung. *Elasticsearch* bietet die Möglichkeit, verschachtelte Suchanfragen zu erstellen, wobei Bestandteile auf derselben Ebene in gleichem Maße zum endgültigen Score der Dokumente beitragen. Indem wichtige Felder möglichst weit oben in der Hierarchie abgeprüft werden, lässt sich deren Beitrag positiv beeinflussen.

Darüber hinaus ist es auch möglich, verschiedene Teile der Suchanfrage explizit unterschiedlich zu gewichten. Beispielsweise kann es bei der Realisierung eines Bibliothekskatalogs sinnvoll sein, einen Treffer dann höher zu bewerten, wenn die Suchbegriffe im Titelfeld gefunden werden, aber dennoch auch Dokumente anzuzeigen, bei

denen die Suchbegriffe im Beschreibungsfeld enthalten sind. Die Gewichtung greift wiederum per Hierarchieebene.

Um die explizite Gewichtung umzusetzen, kann für jede Suchanfrage ein sogenannter *boost*-Parameter gesetzt werden, der die relative Wichtigkeit angibt. Ein Anfragebestandteil mit einem *boost*-Wert von Zwei ist doppelt so wichtig wie einer ohne explizit gesetzten Wert.[16]

Eine eigens dafür vorgesehene *boosting query* ermöglicht es, Suchtreffer nach bestimmten Kriterien ab- oder aufzuwerten, um bei einer Suche nach „ELK“ zum Beispiel auch englischsprachige Tierseiten anzuzeigen, aber mit niedrigerer Priorität als Dokumentationen des E(lasticsearch)L(ogstash)K(ibana)-Stacks.

Neben einzelnen Bestandteilen der Suchanfrage können auch ganze Indizes verstärkt werden, etwa um bei der Suche nach Logdateien neuere Logs zu bevorzugen.

In der Scoring-Funktion sind diese Parameter alle in $\text{boost}(t)$ enthalten.

Definition eigener Scoring-Funktionen Neben den oben genannten Möglichkeiten, bietet *Elasticsearch* auch die Option, die standardmäßig berechneten Scores zu verändern. Dies kann durch konstante Multiplikatoren geschehen, die die Scores all jener Dokumente verändern, die einer definierten Suchanfrage oder einem Filter entsprechen, oder durch die Verrechnung aller Scores mit einem bestimmten Feldwert. Außerdem ist es auch möglich, vordefinierte „Zerfallsfunktionen“ zu verwenden, die jene Dokumente am höchsten gewichten, die zu einem bestimmten Zeitpunkt erstellt wurden, mit einem bestimmten Ort befasst sind oder in denen ein Preis angegeben ist, der nahe an einem vorgegebenen Wert liegt.

Darüber hinaus können eigene Skripte eingebunden werden, die einen individuellen Score berechnen. Diese können den von *Elasticsearch* berechneten ursprünglichen Score berücksichtigen oder auch nicht.

[16] In der *Elasticsearch*-Dokumentation findet sich der Hinweis, dass die Wahl der richtigen Werte stark von der jeweiligen Anwendung abhängig ist und nicht durch einfache Formeln entschieden werden kann. Zu bedenken ist zum Beispiel, dass kurze Felder aufgrund der Feldnorm ohnehin eine „natürliche“ Verstärkung haben, so dass es in vielen Fällen gar nicht nötig ist, ihre Bedeutung noch einmal explizit durch einen hohen *boost*-Faktor zu erhöhen (s. Abschnitt „Query Time Boosting“ in [16]).

3.3 Aggregationen

Neben den in den vorausgehenden Abschnitten vorgestellten Suchfunktionalitäten bietet *Elasticsearch* auch Möglichkeiten, die indizierten Daten zu analysieren. Zu diesem Zweck können Aggregationen eingesetzt werden.

Bevor die für die vorliegende Arbeit wichtige Significant-Terms-Aggregation vorgestellt wird, zunächst ein paar grundlegende Erläuterungen zur Funktionsweise von Aggregationen in *Elasticsearch*.

3.3.1 Funktionsweise

Elasticsearch-Aggregationen basieren immer auf demselben Prinzip: Zunächst werden die Dokumente, deren Inhalte aggregiert werden sollen, in Gruppen, so genannten *Buckets*, zusammengefasst, anschließend kommt eine Metrik zur Anwendung, die pro Gruppe bestimmte Kennzahlen berechnet.

Die Eingruppierung kann über Filter erfolgen; es kann für jeden unterschiedlichen Term, der in einem bestimmten Feld gefunden wird, eine eigene Gruppe erstellt werden; es gibt vorgefertigte Gruppierungen für Histogramme und Zeitreihen, die nur noch parametriert werden müssen, sowie die Möglichkeit, anhand von Zeit- oder IP-Adress-Bereichen zu gruppieren oder für die besten Suchergebnisse Gruppen anhand von Feldwerten zu bilden. Eine Verschachtelung der *Buckets* ist ebenfalls möglich.

Auf jeder Hierarchieebene der Gruppierungen können Metriken angewandt werden. In der Regel sind dies einfache mathematische Operationen, die zum Beispiel den Minimal- oder Maximalwert auslesen, den Durchschnittswert oder andere statistische Kennzahlen berechnen oder Werte aufsummieren. Die verfügbaren Metriken sind in der *Elasticsearch*-Dokumentation [16], [26] aufgelistet und sollen mit Ausnahme der *Significant Terms* nicht näher besprochen werden.

3.3.2 Die Significant-Terms-Aggregation

Die Significant-Terms-Aggregation dient dem Auffinden signifikanter Terme. Dabei handelt es sich um Terme, deren Auftreten in einer Untermenge von Dokumenten ungewöhnlich und auffällig ist (*uncommonly common terms*). Um solche Wörter zu entdecken, muss man statistische Anomalien aufdecken. Es gilt, Wörter zu finden,

die in einigen Dokumenten signifikant häufiger auftauchen, als dies aufgrund der Gesamthäufigkeit ihres Auftauchens im Korpus zu erwarten wäre. Die Idee erinnert an die Berechnung der gewichteten Dokumentenvektoren per TF-IDF. Auch dort ist die Grundidee, einem Term dann ein höheres Gewicht zuzusprechen, wenn er im aktuellen Dokument häufig auftaucht, im Gesamtkorpus aber selten.

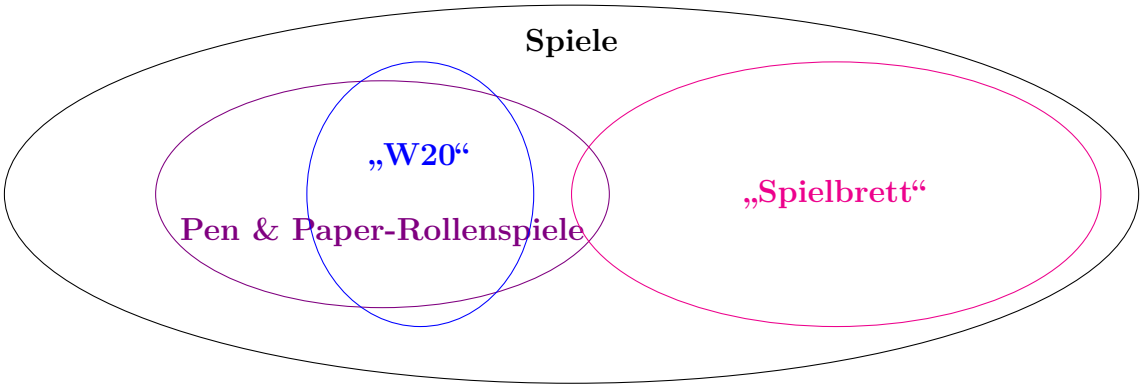


Abbildung 3.2: Schematische Darstellung eines Korpus zum Thema Spiele. „W20“ ist ein signifikanter Term für Dokumente, die Pen & Paper-Rollenspiele zum Thema haben, „Spielbrett“ nicht.

Abbildung 3.2 veranschaulicht die Grundidee der signifikanten Terme. In einem Korpus sind Dokumente zu Spielen verschiedener Art enthalten. Einige dieser Dokumente befassen sich mit Pen & Paper-Rollenspielen (violett gekennzeichnet). Außerdem gibt es eine Menge von Dokumenten, die den Ausdruck „W20“ enthalten, augenscheinlich also unter anderem von zwanzigseitigen Würfeln handeln, sowie eine Menge von Dokumenten, die den Ausdruck „Spielbrett“ enthalten. Um später leichter beispielhaft Signifikanzmaße berechnen zu können, nehmen wir folgende Anzahlen von Dokumenten an:

Menge	Kardinalität
Korpus (Spiele)	1000
Pen & Paper-Rollenspiele	200
Dokumente, die „W20“ enthalten	100
Pen & Paper-Dokumente, die „W20“ enthalten	80
Dokumente, die “Spielbrett“ enthalten	320
Pen & Paper-Dokumente, die „Spielbrett“ enthalten	10

Tabelle 3.2: Der Spiele-Korpus in Zahlen (frei erfunden)

Auffällig ist, dass insgesamt nur 10% der Dokumente im Korpus den Ausdruck „W20“ enthalten, von den Rollenspieldokumenten trifft dies auf 40% zu. „W20“ kommt in dieser Dokumentenklasse also deutlich häufiger vor, als eigentlich zu erwarten wäre, was den Schluss nahelegt, dass es sich bei „W20“ um einen signifikanten Term für diese Untermenge von Dokumenten handelt.

„Spielbrett“ hingegen kommt als signifikanter Term nicht in Frage. Während nämlich insgesamt 32% aller Dokumente diesen Ausdruck enthalten, ist dies bei nur 5% der Rollenspieldokumente der Fall.

Aus dem bisher Gesagten wird deutlich, dass sich signifikante Terme nur dann bestimmen lassen, wenn man die Menge von Dokumenten, für die man diese Terme bestimmen will, als Vordergrundmenge (*foreground set*) von einer Hintergrundmenge (*background set*) abgrenzen kann, die man zum Vergleich der Termhäufigkeiten heranzieht. Im obigen Beispiel ist die Vordergrundmenge die Menge der Dokumente zum Thema Pen & Paper-Rollenspiele, während die Hintergrundmenge das gesamte Korpus ist. *Elasticsearch* bietet aber auch die Möglichkeit, die Hintergrundmenge ebenfalls einzuschränken.

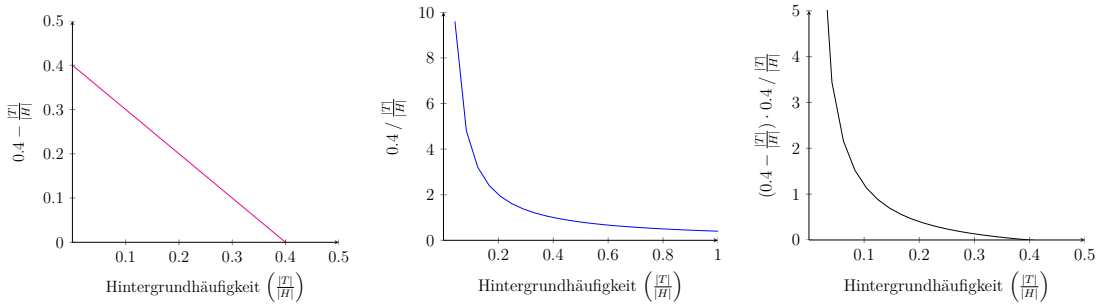
Bleibt die Frage, wie genau man signifikante Terme bestimmt. Die Dokumente der Vordergrundmenge enthalten schließlich viele unterschiedliche Terme, die prinzipiell in Frage kommen. Was noch fehlt, ist also ein Signifikanzmaß, das es erlaubt, die Signifikanz verschiedener Kandidatenterme zu vergleichen.

Berechnung des Scores

Für die Bewertung der Signifikanz eines Terms bietet *Elasticsearch* verschiedene Möglichkeiten an: *JLH-Score*, *Transinformation* (*mutual information*), *Chi-Quadrat* und die *normalisierte Google-Distanz*. Diese Ansätze werden in den folgenden Unterabschnitten beschrieben.

JLH-Score Standardmäßig verwendet *Elasticsearch* den JLH-Score zur Berechnung der Signifikanzbewertung. Es bezeichne H die Hintergrundmenge, V die Vordergrundmenge und T die Menge der Dokumente, die Term t enthalten. Dann wird der JLH-Score wie folgt berechnet:

$$\mathrm{JLH}(t) = \left(\frac{|T \cap V|}{|V|} - \frac{|T|}{|H|}\right) \cdot \left(\frac{|T \cap V|}{|V|} \,/\, \frac{|T|}{|H|}\right)$$



(a) Häufigkeitsdifferenz (b) Häufigkeitsrelation (c) Produkt

Abbildung 3.3: Veranschaulichung der Berechnung des JLH-Scores: Differenz und Relation von fester Vordergrundhäufigkeit (40%) und variabler Hintergrundhäufigkeit sowie das Produkt der beiden.

Zwei Bestandteile werden hier miteinander verrechnet: Absolute und relative Veränderung beim Wechsel von der Vordergrund- zur Hintergrundhäufigkeit. Beide liefern erwartungsgemäß hohe Werte, wenn t in der Vordergrundmenge prozentual gesehen häufiger auftaucht als in der Hintergrundmenge, der erste Bestandteil bestraft aber im gesamten Korpus häufige Terme deutlich weniger als der zweite, während letzterer seltene Terme sehr viel stärker durch hohe Werte belohnt. Abbildung 3.3 veranschaulicht das für $\frac{|W \cap R|}{|R|} = 0.4$ und eine sich ändernde Gesamthäufigkeit $\frac{|W|}{|S|}$.

Der JLH-Score ist der Versuch, die Vorteile beider Ansätze zu verbinden und damit einerseits eine möglichst hohe Trefferquote zu erzielen, ohne dass andererseits die Genauigkeit darunter leidet.

Angewandt auf unser Beispiel erhalten wir für die Terme „W20“ und „Spielbrett“ folgende JLH-Scores, wenn wir als Vordergrundmenge die Pen & Paper-Rollenspiele wählen:

$$JLH(\text{„W20“}) = \left(\frac{80}{200} - \frac{100}{1000}\right) \cdot \left(\frac{80}{200} \,/\, \frac{100}{1000}\right) = 1,2$$

$$JLH(\text{„Spielbrett“}) = \left(\frac{10}{200} - \frac{320}{1000}\right) \cdot \left(\frac{10}{200} \,/\, \frac{320}{1000}\right) \approx -0.042$$

Transinformation Bei der Transinformation oder gegenseitigen Information (engl. *mutual information*) besteht die Grundidee darin herauszufinden, wie viel Information eine Zufallsvariable über eine andere liefert. Dies wird zum Beispiel dazu genutzt, Textdokumente zu klassifizieren, indem Terme gesucht werden, deren Anwesenheit im Dokument Information über die Klassenzugehörigkeit desselben liefern (vgl. [17, S. 272 ff.]).

Auch unser obiges Beispiel, in dem wir signifikante Terme in der Menge der Pen & Paper-Rollenspiel-Dokumente finden wollen, lässt sich als Klassifizierungsproblem formulieren: Wir wollen wissen, wie viel Information die Anwesenheit des Terms t in einem Dokument über die Zugehörigkeit desselben zur Klasse der Rollenspieldokumente liefert, die Pen & Paper-Rollenspiele zum Thema haben. Später werden wir für die Terme „W20“ und „Spielbrett“ beispielhaft die Transinformation berechnen, zunächst betrachten wir jedoch die allgemeine Berechnungsvorschrift.

Zu diesem Zweck wechseln wir noch einmal den Blickwinkel und beschreiben unser Szenario als Zufallsexperiment: In unserem Korpus befinden sich 1000 Dokumente. Aus diesen wird zufällig ein Dokument ausgewählt und auf folgende Eigenschaften überprüft: Enthält es den Term t? Gehört es zur Klasse der Rollenspieldokumente? Wir führen zwei binäre Zufallsvariablen ein, um das Ergebnis unserer zufälligen Auswahl festzuhalten: Variable U nimmt den Wert $e_t = 1$ an, wenn Term t im Dokument vorkommt, ansonsten den Wert $e_t = 0$. Die Zufallsvariable C beschreibt die Klassenzugehörigkeit des Dokuments und nimmt den Wert $e_c = 1$ an, wenn das Dokument zur Klasse der Rollenspieldokumente gehört, ansonsten den Wert $e_c = 0$.

Was wir wissen wollen, ist, ob der Wert von U uns Informationen über den Wert von C liefert. Dazu berechnen wir die Transinformation wie folgt:

$$I(U;C) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} P(U = e_t, C = e_c) \log_2 \frac{P(U = e_t, C = e_c)}{P(U = e_t)P(C = e_c)}$$

Betrachten wir zunächst die Extremfälle: Wenn U und C statistisch unabhängig sind und die Belegung der einen Variablen nichts über die Belegung der anderen Variablen aussagt, dann gilt für alle möglichen Belegungen e_t und e_c von U und C:

$$P(U = e_t, C = e_c) = P(U = e_t)P(C = e_c)$$

Die gegenseitige Information ist in diesem Fall Null. Im anderen Extremfall bestimmt der Wert der einen Zufallsvariablen den der anderen vollständig. In diesem Fall liefert die Beobachtung von U genauso viel Information wie die von C und die Transinformation ist deshalb gleich der Entropie der Einzelvariablen.[17] In unserem oben beschriebenen Fall würde dieser Maximalwert der Transinformation dann erreicht, wenn Term t entweder genau dann in einem Dokument auftaucht, wenn dieses

[17] Die gegenseitige Information lässt sich auch herleiten als die Differenz der Entropie der Einzelvariablen und der bedingten Entropie. Vgl. hierzu zum Beispiel [18, S. 19 f.].

zur Klasse der Rollenspieldokumente gehört (oder genau dann, wenn das Dokument nicht zu dieser Klasse gehört).

In der Realität hat man es in der Regel mit Werten zwischen diesen Extremen zu tun. Beispielhaft wollen wir berechnen, welche Information die Anwesenheit der Terme „W20“ und „Spielbrett“ in einem Dokument jeweils über dessen Zugehörigkeit zur Klasse der Rollenspieldokumente liefert.

Dazu benötigen wir die Wahrscheinlichkeiten für das Vorhandensein des jeweiligen Terms und die Klassenzugehörigkeit. Diese schätzen wir durch Auszählung der Dokumente in unserem Korpus:

Term	Variablenbelegung	Wahrscheinlichkeit
„W20“	$e_t = e_c = 0$	$\frac{780}{1000} = 0,78$
	$e_t = 0,\ e_c = 1$	$\frac{120}{1000} = 0,12$
	$e_t = 1,\ e_c = 0$	$\frac{20}{1000} = 0,02$
	$e_t = e_c = 1$	$\frac{80}{1000} = 0,08$
„Spielbrett“	$e_t = e_c = 0$	$\frac{490}{1000} = 0,49$
	$e_t = 0,\ e_c = 1$	$\frac{190}{1000} = 0,19$
	$e_t = 1,\ e_c = 0$	$\frac{310}{1000} = 0,31$
	$e_t = e_c = 1$	$\frac{10}{1000} = 0,01$

Tabelle 3.3: Schätzung der Wahrscheinlichkeiten für das Auftreten einzelner Terme und die Zugehörigkeit zur Klasse der Rollenspieldokumente durch Auszählung der entsprechenden Dokumente im Korpus

Für den Term „W20“ und die Klasse der Rollenspieldokumente berechnet sich die gegenseitige Information dann wie folgt:

$$\begin{aligned} I(U;C) &= 0,78 \log_2 \frac{0,78}{(0,78+0,12)\cdot(0,78+0,02)} \\ &+ 0,12 \log_2 \frac{0,12}{(0,12+0,78)\cdot(0,12+0,08)} \\ &+ 0,02 \log_2 \frac{0,02}{(0,02+0,08)\cdot(0,02+0,78)} \\ &+ 0,08 \log_2 \frac{0,08}{(0,08+0,02)(0,08+0,12)} \\ &\approx 0,1399 \end{aligned}$$

Und für „Spielbrett“ ergibt sich folgende Transinformation:

$$
\begin{aligned}
I(U;C) = {} & 0,49 \log_2 \frac{0,49}{(0,49+0,19)\cdot(0,49+0,31)} \\
& + 0,19 \log_2 \frac{0,19}{(0,19+0,49)\cdot(0,19+0,01)} \\
& + 0,31 \log_2 \frac{0,31}{(0,31+0,01)\cdot(0,31+0,49)} \\
& + 0,01 \log_2 \frac{0,01}{(0,01+0,31)\cdot(0,01+0,19)} \\
& \approx 0,0766
\end{aligned}
$$

Wie zu erwarten liefert die Anwesenheit des Terms „W20“ in einem Dokument aus unserem Korpus mehr Information darüber, dass dieses in die Klasse der Rollenspieldokumente einzuordnen ist, als das Vorkommen von „Spielbrett“.

Es kann allerdings vorkommen, dass die nach dieser Methode berechneten signifikanten Terme Wörter enthalten, die in der Vordergrundmenge seltener auftreten als im gesamten Korpus. Darauf weist die *Elasticsearch*-Dokumentation ebenso hin wie auf die Möglichkeit, dies durch das Setzen eines speziellen Parameters zu verhindern. Insgesamt werden häufig auftretende Terme bevorzugt, was den Nachteil haben kann, dass unter Umständen Stoppwörter zu signifikanten Termen erklärt werden.

Chi-Quadrat Mit dem χ^2-Test wird die Unabhängigkeit zweier statistischer Ereignisse gemessen. Die Ereignisse A und B sind, wie bereits erwähnt, dann statistisch unabhängig, wenn die Wahrscheinlichkeit ihres gemeinsamen Auftretens dem Produkt der Einzelwahrscheinlichkeiten entspricht ($P(A,B) = P(A)P(B)$). Dies ist gleichbedeutend damit, dass das Eintreten des Ereignisses B nichts an der Eintrittswahrscheinlichkeit für A ändert und anders herum ($P(A|B) = P(A)$, $P(B|A) = P(B)$). Auch dieser Test kann zur Klassifikation von Textdokumenten herangezogen werden [17, S. 275 ff.] oder in unserem Fall zum Auffinden signifikanter Terme.

Die Ereignisse, die beim Auffinden signifikanter Terme für unsere Rollenspieldokumente eintreten können, sind dieselben, die oben bereits beschrieben wurden: Ein zufällig ausgewähltes Dokument gehört entweder zur besagten Klasse oder nicht und es enthält entweder einen bestimmten Term t oder auch nicht.

Signifikant ist Term t für die Klasse der Rollenspieldokumente dann, wenn die Hypothese widerlegt werden kann, dass das Auftreten von t in einem Dokument und die Zugehörigkeit des besagten Dokuments zur Klasse statistisch unabhängige Ereignisse sind. Um dies zu überprüfen, werden tatsächliche (N) und bei statistischer Unabhängigkeit zu erwartende Häufigkeiten (E) miteinander zu folgender Kenngröße verrechnet:

$$X^2(K,t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}}$$

K ist das Korpus, t und c bezeichnen, wie oben auch, den aktuell auf Signifikanz untersuchten Term und die Klasse.

Sind die beiden untersuchten Ereignisse tatsächlich statistisch unabhängig, dann nimmt X^2 den Wert Null an. Je größer X^2 ist, desto höher die Sicherheit, dass die Hypothese statistischer Unabhängigkeit verworfen werden darf.

Für unser Beispiel sehen tatsächliche und erwartete Häufigkeiten wie folgt aus:

	$e_c = e_{\text{Rollenspiel}} = 0$	$e_c = e_{\text{Rollenspiel}} = 1$
$e_t = e_{\text{W20}} = 0$	$N_{00} = 780$, $E_{00} = 1000 \cdot 0,9 \cdot 0,8 = 720$	$N_{01} = 120$, $E_{01} = 1000 \cdot 0,9 \cdot 0,2 = 180$
$e_t = e_{\text{W20}} = 1$	$N_{10} = 20$, $E_{10} = 1000 \cdot 0,1 \cdot 0,8 = 80$	$N_{11} = 80$ $E_{11} = 1000 \cdot 0,1 \cdot 0,2 = 20$
$e_t = e_{\text{Spielbrett}} = 0$	$N_{00} = 490$, $E_{00} = 1000 \cdot 0,68 \cdot 0,8 = 544$	$N_{01} = 190$ $E_{01} = 1000 \cdot 0,68 \cdot 0,2 = 136$
$e_t = e_{\text{Spielbrett}} = 1$	$N_{10} = 310$ $E_{10} = 1000 \cdot 0,32 \cdot 0,8 = 256$	$N_{11} = 10$ $E_{11} = 1000 \cdot 0,32 \cdot 0,2 = 64$

Tabelle 3.4: Erwartete und tatsächliche Häufigkeiten für die Terme „W20“ und „Spielbrett“ und die Klasse der Pen & Paper-Rollenspieldokumente

Folglich erhalten wir folgende X^2-Werte:

$$X^2(\mathbb{D}, \text{„W20“}, \text{Rollenspiel}) = \frac{60^2}{720} + \frac{60^2}{180} + \frac{(-60)^2}{80} + \frac{60^2}{20} = 250$$

$$X^2(\mathbb{D}, \text{„Spielbrett“}, \text{Rollenspiel}) = \frac{(-54)^2}{544} + \frac{54^2}{136} + \frac{54^2}{256} + \frac{(-54)^2}{64} \approx 83,75$$

Beide Werte sind recht hoch, so dass sich sowohl „W20“ als auch „Spielbrett“ als signifikante Terme für unsere Dokumentenklasse qualifizieren.[18] Es zeigt sich hier das Problem, das bei der Transinformation bereits angesprochen wurde: Dass zwei Ereignisse statistisch abhängig sind, muss nicht zwangsläufig heißen, dass sie auffällig häufig gemeinsam eintreten. Auch das Gegenteil kann der Fall sein, wie etwa in unserem Beispiel beim Term „Spielbrett“, der in Dokumenten der betrachteten Klasse nicht signifikant häufiger, sondern seltener vorkommt, als eigentlich zu erwarten wäre. Da solche Terme in der Regel nicht als signifikante Terme gewertet werden sollen, bietet *Elasticsearch* die Möglichkeit, sie durch das Setzen eines Parameters auszuschließen.

Google-Distanz Das letzte von *Elasticsearch* angebotene Maß für die Bewertung der Signifikanz ist die normalisierte Google-Distanz, die ausführlich in der Arbeit [19] von Cilibrasi und Vitányi beschrieben wird. Dieses Maß sticht etwas heraus, weil es im Gegensatz zu den bislang betrachteten Maßen nicht misst, ob das Auftreten eines Terms in einer Menge von Dokumenten in irgendeiner Weise auffällig ist. Stattdessen ist es ein Maß für die semantische Ähnlichkeit zweier Terme.

Um diese Ähnlichkeit zu bestimmen, nutzen die beiden Autoren Konzepte der algorithmischen Informationstheorie. Die Grundidee besteht darin, die Terme t und r, deren Ähnlichkeit bestimmt werden soll, über ihren semantischen Gehalt darzustellen, der sich aus den Dokumenten ergibt, in denen der jeweilige Term vorkommt. Anschließend wird ermittelt, wie weit diese semantischen Inhalte voneinander entfernt sind. Dies geschieht mittels einer Näherungsfunktion für die Informationsdistanz (*information distance*).

[18] Für die χ^2-Funktion gibt es kritische Werte, deren Überschreitung garantiert, dass die Ausgangshypothese der statistischen Unabhängigkeit mit einer gewissen Sicherheit (90%, 95%, 99%, ...) verworfen werden darf.

Die Informationsdistanz wird unter Rückgriff auf Kolmogorov-Komplexitäten definiert. Die Kolmogorov-Komplexität einer Zeichenkette ist die Länge (in Bits) des minimalen Programms, das diese Zeichenkette erzeugt – oder anders ausgedrückt: die Länge der minimalen Beschreibung oder der besten Komprimierung der Zeichenkette. Will man die Ähnlichkeit zweier Zeichenketten bestimmen, dann untersucht man, wie schwierig es ist, sie ineinander umzuwandeln. Die Länge des kürzesten Programms, das diese Umwandlung vornehmen kann, wird als Informationsdistanz E bezeichnet. Es lässt sich zeigen, dass eine zulässige Definition, die nur minimal von der tatsächlichen Informationsdistanz abweicht, wie folgt aussieht [19, S. 3]:

$$E(t,r) = K(t,r) - \min\{K(t), K(r)\}$$

$K(t,r)$ ist dabei die Kolmogorov-Komplexität des kürzesten Programmes, das das Paar (t,r) erzeugt und eine Möglichkeit zur Unterscheidung von t und r bietet. Um die Informationsdistanz als Metrik verwenden zu können, wird sie noch normalisiert:

$$NID(t,r) = \frac{K(t,r) - \min\{K(t), K(r)\}}{\max\{K(t), K(r)\}}$$

Weil die Informationsdistanz nicht berechenbar ist [20], werden für die praktische Anwendung Näherungen verwendet, und zwar in der Regel in Form der normalisierten Kompressionsdistanz. Dazu werden die Kolmogorov-Komplexitäten durch die Längen der Ausgabe eines Kompressionsalgorithmus approximiert, der auf t, r und tr angesetzt wird.

Auch die Google-Distanz ist eine solche Näherung der Informationsdistanz. Weil sie aber mit der semantischen Ähnlichkeit von Termen befasst ist, werden nicht die Terme selbst komprimiert, sondern deren semantischer Gehalt. Um die Länge der entsprechenden Codewörter zu berechnen, treffen Cilibrasi und Vitányi zunächst ein paar Annahmen: Für jedes der M im gesamten Korpus von Google enthaltenen Dokumente nehmen sie eine Wahrscheinlichkeit von $\frac{1}{M}$ an, dass dieses von einer Suchanfrage zurückgeliefert wird. Die Suchanfrage nach Term t liefert die Dokumentenmenge T zurück. Dabei handelt es sich, so die Argumentation der Autoren, um ein Zufallsereignis, dem die Wahrscheinlichkeit $p(T) = \frac{|T|}{M}$ zugeordnet werden kann. In diesem Ereignis sind alle semantischen Informationen zu t enthalten und da die Wahrscheinlichkeit seines Auftretens angegeben werden kann, kann auch der Informationsgehalt berechnet werden als $\log \frac{1}{p(T)}$. Der Informationsgehalt wiederum

gibt eine untere Schranke für die Länge einer Kodierung des Ereignisses an. Um sicherzustellen, dass die Kodierung eindeutig ist, muss allerdings die so genannte Kraft'sche Ungleichung erfüllt sein und das ist nur dann der Fall, wenn sich die Wahrscheinlichkeiten zu Eins summieren (s. z. B. [18, Kap. 5, S. 82 ff.]). Das ist zunächst nicht gegeben, weil sich die Ereignisse überlappen, kann aber durch Normierung erreicht werden, indem alle Wahrscheinlichkeiten mit $\frac{M}{N}$ multipliziert werden. Um den Normierungsfaktor N zu berechnen, müssten eigentlich die Kardinalitäten aller möglichen Einzelereignisse (also aller zurückgelieferten Dokumentmengen für alle möglichen Suchterme) sowie aller möglichen Schnittmengen derselben addiert werden. In der Praxis wird aber häufig M verwendet.

Für die Terme t und r und deren zugehörige Dokumentenmengen T und R berechnet sich die Google-Distanz dann wie folgt:

$$NGD(t,r) = \frac{G(t,r) - \min\{G(t), G(r)\}}{\max\{G(t), G(r)\}}$$

Die (syntaktische) Ähnlichkeit zur Berechnung der normierten Informationsdistanz ist offensichtlich, statt der Kolmogorov-Komplexität K wird aber der Google-Code G verwendet, der den Informationsgehalt des semantischen Gehalts der betreffenden Terme kapselt und wie folgt definiert ist:

$$G(t) = G(t,t), \quad G(t,r) = log(1/g(t,r))$$

Die Verteilung g beschreibt die oben erwähnten normierten Wahrscheinlichkeiten der Zufallsereignisse und es gilt:

$$g(t) = g(t,t), g(t,r) = \frac{|T \cap R|}{M} \cdot \frac{M}{N} = \frac{|T \cap R|}{N}$$

Einsetzen in die obige Formel ergibt

$$NGD(t,r) = \frac{\log \frac{1}{\frac{|T \cap R|}{N}} - \min\{\log \frac{1}{\frac{|T|}{N}}, \log \frac{1}{\frac{|R|}{N}}\}}{\max\{\log \frac{1}{\frac{|T|}{N}}, \log \frac{1}{\frac{|R|}{N}}\}}$$

und Umformen führt zur endgültigen Formel

$$NGD(t,r) = \frac{\max\{\log|T|, \log|R|\} - \log|T \cap R|}{\log N - \min\{\log|T|, \log|R|\}}$$

Kommen die Terme t und r ausschließlich in denselben Dokumenten vor, beträgt ihre Google-Distanz 0. Eine große Google-Distanz weist also auf unähnliche Terme hin, und um signifikante Terme zu finden, die häufig gemeinsam auftreten, sollten solche mit kleiner Google-Distanz gewählt werden.

Abschließend wollen wir beispielhaft Google-Distanzen für unser Spiele-Korpus berechnen. Dazu nehmen wir an, dass unsere Rollenspiel-Dokumente alle den Term „Pen & Paper“ enthalten und dass sie die einzigen Dokumente im Korpus sind, auf die das zutrifft. Für die Terme „Pen & Paper“ und „W20“ ergibt sich dann folgende Google-Distanz:

$$NGD(\text{„Pen \& Paper“}, \text{„W20“}) = \frac{\max\{\log 200, \log 100\} - \log 80}{\log 1000 - \min\{\log 200, \log 100\}} \approx 0,398$$

Für “Pen & Paper“ und “Spielbrett“ hingegen:

$$NGD(\text{„Pen \& Paper“}, \text{„Spielbrett“}) = \frac{\max\{\log 200, \log 320\} - \log 10}{\log 1000 - \min\{\log 200, \log 320\}} \approx 2,153$$

„W20“ und „Spielbrett“ schließlich weisen folgende Google-Distanz auf:

$$NGD(\text{„W20“}, \text{„Spielbrett“}) = \frac{\max\{\log 100, \log 320\} - \log 0}{\log 1000 - \min\{\log 100, \log 320\}} = \infty$$

Genau genommen ist $\log 0$ nicht definiert, für gegen 0 strebende Argumente geht die Logarithmus-Funktion aber gegen $-\infty$, was in diesem Fall dann als Funktionswert angenommen wird (vgl. [19, S. 6]). In unserem Beispiel weisen also „W20“ und „Spielbrett“ keinerlei semantische Ähnlichkeit auf und „W20“ und „Pen & Paper“ sind sich semantisch deutlich näher als „Pen & Paper“ und „Spielbrett“.

Generell gilt, so auch der Hinweis in der *Elasticsearch*-Dokumentation, dass Terme mit hoher Kookkurrenz von der Google-Distanz bevorzugt, Stoppwörter hingegen nur mit geringer Wahrscheinlichkeit ausgewählt werden.

Verwendung der Aggregation

Im *Elasticsearch*-Blog[19] nennt Mark Harwood ganz unterschiedliche mögliche Anwendungsfälle für die Significant-Terms-Aggregation:

[19] `http://www.elasticsearch.org/blog/significant-terms-aggregation/`

- Geographischer Überblick über ungewöhnliche Häufungen bestimmter Arten von Straftaten,
- Analyse der Ursache für Fehlfunktionen von Autos durch die Auswertung von Freitextbeschreibungen in Fehlerberichten,
- Trainieren eines Klassifikators für Filme, der anhand der Filmbeschreibung eine passende Kategorie vorschlägt,
- Auffinden von falsch kategorisierten Filmen in einer Datenbank,
- Aufdecken von Kreditkartenbetrug,
- Umsetzung eines einfachen Recommendersystems.[20]

Allen diesen Anwendungen ist gemeinsam, dass in irgendeiner Form Anomalien aufgedeckt und genutzt werden. Für die vorliegende Arbeit interessant ist natürlich vor allem das zuletzt genannte Anwendungsszenario.

[20] Näheres dazu auch in der Präsentation von Britta Weber vor der Schweizer *Elasticsearch* User Group [21].

4 Umsetzung eines einfachen Recommender-Plugins für Elasticsearch

Im Rahmen der Masterthesis entstand ein Plugin, das die REST-Schnittstelle von *Elasticsearch* erweitert und Empfehlungen entweder für einen vorgegebenen Nutzer oder basierend auf einem Artikel generieren kann. Für die Entwicklung dieses Plugins gab es zwei hauptsächliche Anreize: Zum einen sollte eine Idee zur einfachen Generierung von Empfehlungen mittels *Elasticsearch* auf ihre Umsetzbarkeit überprüft werden (vgl. hierzu auch den folgenden Abschnitt 4.1), zum anderen sollte untersucht werden, wie leicht sich eine Plug-and-Play-Lösung entwickeln lässt, die nicht auf einen speziellen Datensatz zugeschnitten ist, sondern die Generierung von Empfehlungen für ganz unterschiedliche Datensätze erlaubt.

4.1 Grundidee

Die Idee für die im Plugin umgesetzte Art der Empfehlungsgenerierung stammt aus einem Vortrag [21], den Britta Weber in Zürich gehalten hat. Die Grundannahmen sind dieselben wie beim kollaborativen Filtern auch: Wir gehen davon aus, dass Nutzer Interesse an Artikeln haben, die denjenigen ähneln, die sie bereits positiv bewertet haben und/oder Nutzern gefallen, die einen ähnlichen Geschmack haben. Dabei wird die Ähnlichkeit von Artikeln nicht mit inhaltsbasierten Ansätzen bestimmt, sondern daran festgemacht, dass zwei Artikel häufig gemeinsam in Positivlisten auftauchen. Im Interesse des Nutzers ist es dabei, wenn unter den Vorschlägen möglichst wenige Artikel sind, die ohnehin „jeder“ mag und auf die der Nutzer daher aller Wahrscheinlichkeit nach auch ohne Recommendersystem aufmerksam geworden wäre, und möglichst viele, die auf seine Bedürfnisse beziehungsweise die seiner „Peer Group“ zugeschnitten sind.

Die Funktionalität der bereits vorgestellten Significant Terms-Aggregation geht genau in die gewünschte Richtung: Gegeben ein Subset von Dokumenten und ein zu

durchsuchendes Feld, findet sie „uncommonly common terms“, also jene Terme, deren gemeinsames Auftreten im gegebenen Feld statistisch auffällig ist.

Dies kann man für die Generierung von Empfehlungen wie folgt nutzen: Angenommen, wir haben für jeden Nutzer eine Liste mit Artikeln indiziert, die dieser positiv bewertet hat. Dann kann man für einen gegebenen Artikel mittels der Significant Terms-Aggregation in diesem Feld nach anderen Artikeln suchen, die auffällig häufig zusammen mit diesem auftreten. Die zu durchsuchende Untermenge sind dann all jene Dokumente, in denen der Ausgangsartikel in der Positivliste auftaucht, und empfohlen werden die n Artikel mit der höchsten Signifikanzbewertung. Sollen Empfehlungen für einen bestimmten Nutzer generiert werden, dann wird der soeben beschriebene Schritt für jeden der Artikel, die er positiv bewertet hat, durchgeführt. Von allen auf diese Weise aufgespürten Artikeln werden dann diejenigen empfohlen, die die höchsten Scores erhalten haben und dem Nutzer noch nicht bekannt sind. Dabei wird ein Artikel dann als bekannt angenommen, wenn der Nutzer ihn (positiv oder negativ) bewertet hat. Taucht ein Artikel in mehreren Listen signifikanter Terme auf, dann werden die einzelnen Scores addiert, um diesem Artikel mehr Gewicht zu verleihen.

Das Plugin kapselt diese Funktionalität und stellt sie über eine REST-Schnittstelle zur Verfügung. Es kann auf allen Datensätzen arbeiten, die bestimmte Voraussetzungen erfüllen: Für die artikelbasierte Empfehlungsgenerierung müssen Felder vorhanden sein, die Artikel-IDs enthalten und diese in einen Zusammenhang bringen, der sich zur Auswertung mit der (Significant) Terms-Aggregation eignet. Bei diesen Feldern kann es sich zum Beispiel um Positivlisten einzelner Nutzer handeln, aber auch um „Warenkörbe“ von Artikeln, die zusammen gekauft werden, oder um Listen von Artikeln, die nacheinander betrachtet werden etc. Für den nutzerbasierten Ansatz muss für jeden Nutzer auf jeden Fall eine Positivliste vorhanden sein, also eine Aufzählung von IDs von Artikeln, die er gekauft/gut bewertet/konsumiert hat. Im Idealfall bietet der Datensatz darüber hinaus Informationen darüber, welche Artikel dem Nutzer bereits bekannt sind, um die Empfehlung dieser irrelevanten Artikel zu vermeiden. Mehr zur Funktionsweise des Plugins in Abschnitt 4.2.

In Abbildung 4.1 finden sich beispielhaft Empfehlungen ähnlicher Filme für den Film „Toy Story“, die einmal mittels der Significant Terms-Aggregation generiert wurden und einmal mittels der Term-Aggregation. Als Datengrundlage wurde der *Movielens*-Datensatz verwendet.

```
{
  "recommendations": [
    {
      "_id": "3114",
      "title": "Toy Story 2 (1999)"
    },
    {
      "_id": "588",
      "title": "Aladdin (1992)"
    },
    {
      "_id": "1265",
      "title": "Groundhog Day (1993)"
    },
    {
      "_id": "2355",
      "title": "Bug's Life, A (1998)"
    },
    {
      "_id": "1196",
      "title": "Star Wars: Episode V - The Empire Strikes Back (1980)"
    },
    {
      "_id": "1270",
      "title": "Back to the Future (1985)"
    },
    {
      "_id": "2571",
      "title": "Matrix, The (1999)"
    },
    {
      "_id": "318",
      "title": "Shawshank Redemption, The (1994)"
    },
    {
      "_id": "1197",
      "title": "Princess Bride, The (1987)"
    },
    {
      "_id": "364",
      "title": "Lion King, The (1994)"
    }
  ]
}
```


(a) Significant Terms-Empfehlungen basierend auf der positiven Bewertung des Films „Toy Story“


```
{
  "recommendations": [
    {
      "_id": "1196",
      "title": "Star Wars: Episode V - The Empire Strikes Back (1980)"
    },
    {
      "_id": "260",
      "title": "Star Wars: Episode IV - A New Hope (1977)"
    },
    {
      "_id": "2858",
      "title": "American Beauty (1999)"
    },
    {
      "_id": "2571",
      "title": "Matrix, The (1999)"
    },
    {
      "_id": "1198",
      "title": "Raiders of the Lost Ark (1981)"
    },
    {
      "_id": "318",
      "title": "Shawshank Redemption, The (1994)"
    },
    {
      "_id": "3114",
      "title": "Toy Story 2 (1999)"
    },
    {
      "_id": "1270",
      "title": "Back to the Future (1985)"
    },
    {
      "_id": "593",
      "title": "Silence of the Lambs, The (1991)"
    },
    {
      "_id": "2762",
      "title": "Sixth Sense, The (1999)"
    }
  ]
}
```


(b) Terms-Empfehlungen basierend auf der positiven Bewertung des Films „Toy Story“

Abbildung 4.1: Screenshots des Recommender-Plugins

4.2 Aufbau des Plugins

Um die Ideen zur Empfehlungsgenerierung zu testen, ist ein *Elasticsearch*-Plugin entstanden. Der besondere Anspruch an dieses Plugin war, dass es sich um eine Plug-and-Play-Lösung handeln sollte, mit deren Hilfe sich möglichst unkompliziert Empfehlungen auf verschiedenen Datensätzen generieren lassen, die im Optimalfall eine Auswahl an Recommendern bereitstellt und außerdem möglichst leicht erweiterbar ist.

Entstanden ist ein Plugin, das drei zusätzliche *Elasticsearch*-REST-Schnittstellen zur Verfügung stellt. Die erste der beiden dient zur Konfiguration von Recommendern, die zweite zum Anzeigen vorhandener Recommender und die dritte Schnittstelle erlaubt es, GET-Anfragen an *Elasticsearch* zu schicken, um von einem registrierten Recommender Empfehlungen generieren zu lassen. Alle Schnittstellen werden in den folgenden Unterabschnitten vorgestellt.

4.2.1 Konfiguration von Recommendern

Das Plugin definiert eine abstrakte Recommenderklasse, von der alle konkreten Implementierungen erben müssen. Jede Implementierung muss außerdem ihren Typ und die zugehörige Klasse zentral registrieren, damit im Bedarfsfall basierend auf der vom Nutzer angelegten Konfiguration automatisch eine Recommenderinstanz erzeugt werden kann.

Konfigurationen werden vom Nutzer per POST-Request an die `/_recommender`-Schnittstelle des Plugins gesendet und dann in einem separaten Index in *Elasticsearch* gespeichert. Die Grundstruktur der Konfiguration sieht wie folgt aus:

```
{
        "id": "<id>",
        "type": "<type>",
        "output": {
                "index": "<index>",
                "type": "<type>",
                "fields": ["<field1>", "<field2>", ...]
        },
        "params": {
```

```
                "<key1>": "<value1>",
                "<key2>": "<value2>"
        }
}
```

Bei `id` und `type` handelt es sich um verpflichtende Parameter, weil sie verwendet werden, um einen Recommender vom entsprechenden Typ mit der vorgegebenen ID anzulegen und später abzurufen beziehungsweise zu instantiieren. Aktuell gibt es zwei Typen von Recommendern, die unterstützt werden: `Item` und `User`. Ersterer ist ein artikelbasierter Recommender, der zu einem vorgegebenen Artikel passende Artikel findet und empfiehlt, letzterer ist ein nutzerbasierter Recommender, der basierend auf den Vorlieben eines Nutzers weitere Artikel findet, die diesem gefallen könnten.

Jeder Recommender erzeugt eine Liste mit IDs von empfohlenen Artikeln. Diese kann noch um zusätzliche Informationen angereichert werden. Welche das sein sollen, kann im optionalen `output`-Parameter angegeben werden. Dieser beinhaltet den Index, in dem die zusätzlichen Artikelinformationen gespeichert sind, den Typ der Artikel und die Namen der Felder, die zusätzlich zur ID zurückgegeben werden sollen. Vorausgesetzt wird dabei, dass die Artikel-IDs in der Empfehlungsliste denjenigen entsprechen, mit denen man im angegebenen Index unter Verwendung des angegebenen Typs auf die passenden Felder zugreifen kann.

In `params` werden die Parameter gekapselt, die der jeweilige Recommendertyp zur Erstellung von Empfehlungen benötigt. Welche dies sind, legt jede Recommenderklasse selbst fest. Für Recommender vom Typ `Item` sieht das `params`-Objekt folgendermaßen aus:

```
{
        "rec_info" : {
                "index": "<index>",
                "type": "<type>",
                "field": "<field>"
        },
        "aggregation_type": "<agg_type>"
}
```

Um Empfehlungen generieren zu können, ist für den artikelbasierten Recommender nur die Information nötig, welcher Aggregationstyp verwendet werden soll („terms“

oder „significant_terms“) und auf welchem Feld die Aggregation angewandt werden soll. Sollen zum Beispiel Artikel basierend auf den Positivlisten von Nutzern empfohlen werden, dann muss `rec_info` die Information beinhalten, wo diese Listen zu finden sind.

Ein Recommender vom Typ `User` benötigt etwas mehr Informationen, weil er für die Empfehlungsgenerierung Informationen über die Vorlieben des einzelnen Nutzers sowie die diesem bereits bekannten Artikel benötigt. Das entsprechende Parameter-Objekt sieht so aus:

```
{
        "user_info": {
                "index": "<index>",
                "type": "<type>",
                "preferred_items": "<field>",
                "known_items": ["<field1>", "<field2>", ...]
        },
        "aggregation_type": "<agg_type>"
}
```

Das `user_info`-Objekt enthält die erforderlichen Informationen über den Nutzer. Auch hier wird wieder vorausgesetzt, dass das Feld `preferred_items`, das zur Generierung der Empfehlungen verwendet wird, IDs enthält, die gegebenenfalls verwendet werden können, um die Ausgabe um die zusätzlichen Felder anzureichern, die in `output` angegeben sind.

Handelt es sich bei den Daten, die der Nutzer mit dem POST-Request an das Plugin schickt, um eine gültige Konfiguration, dann wird diese gespeichert und der entsprechende Recommender kann zur Generierung von Empfehlungen verwendet werden.

4.2.2 Anzeige von Recommendern

Welche Recommender aktuell registriert sind und wie deren Konfiguration aussieht, kann per GET-Request an `/_recommender/<recommender_id1,recommender_id2, ...>` in Erfahrung gebracht werden. Die Angabe von einer oder mehreren IDs ist optional. Werden keine Recommender-IDs angegeben, wird eine Liste aller regis-

trierten Recommender zurückgegeben, andernfalls eine Liste der Recommender mit den angegebenen IDs.

4.2.3 Abfrage von Empfehlungen

Für das Anstoßen der Empfehlungsgenerierung ist die dritte Schnittstelle verantwortlich, die per GET-Request an `/_recommendations/<recommender_id>/<id>` zu erreichen ist. Die übergebenen Parameter sind zunächst die ID des Recommenders, der verwendet werden soll, um die Empfehlungen zu berechnen, und außerdem eine ID, die den Artikel oder Nutzer identifiziert, für den Empfehlungen erstellt werden sollen.

Das Plugin überprüft dann, ob Informationen über einen Recommender mit dieser ID vorliegen. Ist das der Fall, dann wird dieser auf die übergebene Artikel- oder Nutzer-ID angesetzt und liefert Empfehlungen zurück.

4.3 Skalierbarkeit

Das Plugin ist für den Einsatz auf mehreren *Elasticsearch*-Knoten geeignet. Um alle Knoten über Updates von Recommender-Konfigurationen zu informieren, wird der `TransportService` von *Elasticsearch* verwendet.

Auf diese Weise ist dafür gesorgt, dass jede Plugin-Instanz lokal Recommenderobjekte wiederverwenden kann, solange sich deren Konfiguration nicht ändert. Nimmt der Nutzer über die REST-Schnittstelle Änderungen an einem der Recommender vor, werden alle Plugin-Instanzen informiert und invalidieren ihre lokale Recommenderinstanz.

5 Evaluationsgrundlagen

Ziel der Thesis war es nicht nur, ein Empfehlungsplugin für *Elasticsearch* zu schreiben, sondern natürlich auch, dieses zu evaluieren. Dazu ist zunächst einmal zu klären, ob das Plugin überhaupt tut, was es soll, nämlich den Nutzern Filme empfehlen, die diese noch nicht kennen, aber im Idealfall später gut bewerten.

In einem weiteren Schritt stellt sich dann die Frage, wie gut die generierten Empfehlungen im Vergleich zu solchen sind, die mit anderen Werkzeugen und/oder Methoden generiert wurden. In der vorliegenden Arbeit werden zum Vergleich zum einen Empfehlungen herangezogen, die mittels der Terms-Aggregation erstellt wurden, zum anderen diejenigen, die das Taste-Plugin [29] von Shinsuke Sugaya generiert, das die Funktionen von Apache Mahout Taste [30] für *Elasticsearch* zur Verfügung stellt.

Dieses Kapitel stellt zunächst den Datensatz und die evaluierten Methoden zur Empfehlungsgenerierung vor. Darauf folgt ein kurzer Überblick über die Maße, die zur Qualitätsbewertung herangezogen werden. Die eigentliche Bewertung der Ansätze erfolgt dann in Kapitel 6.

5.1 Datensätze

Grundlage für die Empfehlungsgenerierung bildet der *Movielens-1M*-Datensatz [31], dessen Daten angepasst an die vorliegende Aufgabenstellung indiziert wurden. Aus dem ursprünglichen Datensatz wurden zwei Sätze mit Trainings- und Testdaten erstellt, um Empfehlungen generieren und testen zu können.

5.1.1 Ausgangsdatensatz

Im Movielens-1M-Datensatz sind 1.000.209 Filmbewertungen enthalten, die von 6.040 Nutzern abgegeben wurden, und außerdem 3.883 Filme, von denen die Nutzer 3.706 bewertet haben. Einen Überblick über die Filme gibt Tabelle 5.1.

Anzahl Filme insgesamt	Anzahl bewerteter Filme	Anzahl Filme mit positiver Bewertung	Anzahl Filme mit negativer Bewertung
3.883	3.706	3.533	3.651

Tabelle 5.1: Überblick über die Filme im Movielens-1M-Datensatz

Art der Bewertung	Anzahl insgesamt (alle Nutzer)	min. Anz. pro Nutzer	max. Anz. pro Nutzer	durchschnittl. Anz. pro Nutzer
alle	1.000.209	20	2.314	165,6
pos	575.281	0	1.435	45,2
neg	424.928	0	1.220	70,4

Tabelle 5.2: Überblick über die Bewertungen im Movielens-1M-Datensatz

Pro Film konnte jeder Nutzer zwischen einem und fünf Sternen zur Bewertung vergeben, wobei fünf Sterne die bestmögliche Wertung darstellen und ein Stern die schlechteste. Es konnten nur ganze Sterne vergeben werden. Der Datensatz enthält nur Nutzer, die mindestens 20 Bewertungen abgegeben haben. Tabelle 5.2 gibt einen Überblick über die Verteilung der Bewertungen im Datensatz.

Für jeden Nutzer wurde auf Grundlage der von ihm im Datensatz enthaltenen Bewertungen eine Positiv- und eine Negativliste erstellt und in *Elasticsearch* indiziert. Wie im Vortrag [21] von Britta Weber vorgeschlagen, wurden dabei Filme, die mit 4 oder mehr Sternen bewertet wurden, in die Positivliste des bewertenden Nutzers aufgenommen, solche mit weniger Sternen in die entsprechende Negativliste.

5.1.2 Testdatensatz mit Aufteilung 80-20

Der erste Testdatensatz nimmt die Aufteilung der Ausgangsdaten wie folgt vor: Für jeden Nutzer wurden 80% seiner Bewertungen (zufällig ausgewählt) in den Trainingsdatensatz übernommen und 20% für Tests aufgespart. Aufgrund der zufälligen Aufteilung liegen nur für 3.688 Filme Bewertungen im Trainingsdatensatz vor. 18 Filme, die im Ausgangsdatensatz bewertet wurden, können auf Grundlage des Testdatensatzes also nicht empfohlen werden. Einen Überblick gibt Tabelle 5.3.

Wie sich die Bewertungen im Trainingsdatensatz verteilen, zeigt Tabelle 5.4.

Anzahl bewerteter Filme	Anzahl Filme mit positiver Bewertung	Anzahl Filme mit negativer Bewertung
3.688	3.506	3.621

Tabelle 5.3: Überblick über die Filme im Trainingsdatensatz

Art der Bewertung	Anzahl insgesamt (alle Nutzer)	min. Anz. pro Nutzer	max. Anz. pro Nutzer	durchschnittl. Anz. pro Nutzer
alle	800.193	16	1.851	132,5
pos	460.635	0	1.138	76,3
neg	339.558	0	976	56,2

Tabelle 5.4: Überblick über die Bewertungen im Trainingsdatensatz

5.1.3 All-But-20-Datensatz

Für die Generierung des zweiten Testdatensatzes wurden nur Nutzer berücksichtigt, die mehr als 20 positive Filmbewertungen abgegeben haben. Für jeden Nutzer wurden dann 20 positive Bewertungen in den Testdatensatz übernommen, alle anderen Bewertungen dieses Nutzers wanderten in den Trainingsdatensatz.

Der so zusammengestellte Trainingsdatensatz enthält Nutzerwertungen für 3.676 Filme, wie Tabelle 5.5 zeigt.

Die Bewertungen wurden von 5.084 Nutzern abgegeben. Für 956 Nutzer lagen maximal 20 positive Filmbewertungen vor, weshalb sie nicht in den Trainingsdatensatz übernommen wurden. Die Bewertungen verteilen sich im Trainingsdatensatz wie folgt:

Anzahl bewerteter Filme	Anzahl Filme mit positiver Bewertung	Anzahl Filme mit negativer Bewertung
3.676	3.452	3.635

Tabelle 5.5: Überblick über die Filme im Trainingsdatensatz

Art der Bewertung	Anzahl insgesamt (alle Nutzer)	min. Anz. pro Nutzer	max. Anz. pro Nutzer	durchschnittl. Anz. pro Nutzer
alle	870.099	2	2294	171,14
pos	459.200	1	1.415	90.32
neg	410.899	0	1.220	80.82

Tabelle 5.6: Überblick über die Bewertungen im Trainingsdatensatz

5.2 Zu vergleichende Ansätze

Für die vorliegende Arbeit wurden drei Ansätze ausgewählt, um deren generierte Empfehlungen miteinander zu vergleichen. Zwei davon sind selbst im Rahmen der Masterthesis in einem *Elasticsearch*-Plugins implementiert worden, der dritte wird ebenfalls durch ein *Elasticsearch*-Plugin bereitgestellt, dessen Quellcode auf github erhältlich ist.

5.2.1 Significant Terms

Der Significant-Terms-Recommender ist das *Elasticsearch*-Plugin, das im Rahmen der Masterthesis umgesetzt wurde. Die Funktionsweise wurde bereits in Abschnitt 4.1 erläutert. An dieser Stelle soll nur noch einmal ins Gedächtnis gerufen werden, dass es um das Auffinden von ungewöhnlichen Gemeinsamkeiten der Positivlisten einer bestimmten Untergruppe von Nutzern geht.

5.2.2 Terms

Das während dieser Thesis erstellte Plugin kann statt der Significant-Terms-Aggregation auch die Terms-Aggregation nutzen, um Empfehlungen zu generieren. Das Vorgehen ist dabei im Prinzip dasselbe wie bei der Significant-Terms-Aggregation auch. Der Fokus liegt auch in diesem Fall auf dem Auffinden von Gemeinsamkeiten, allerdings handelt es sich dabei nicht um ungewöhnliche Gemeinsamkeiten. Dies legt die Vermutung nahe, dass Empfehlungen, die mittels der Terms-Aggregation generiert werden, generischer und weniger auf die Eigenheiten einer bestimmten Nutzergruppe zugeschnitten sind.

Ob das so ist und wie der Terms-Recommender im Vergleich zum Significant-Terms-Recommender abschneidet, soll untersucht werden.

5.2.3 Taste-Plugin

Das Taste-Plugin [29] für *Elasticsearch* stammt von Shinsuke Sugaya und stellt die Funktionen von *Apache Mahout Taste* [30] für *Elasticsearch* zur Verfügung. Wie das während der Thesis entstandene Plugin auch, erlaubt es sowohl die Generierung von Listen ähnlicher Artikel als auch von Empfehlungen für einen bestimmten Nutzer.

Zum Einsatz kommen dabei kollaborative Filter. Um Nutzer beziehungsweise Artikel vergleichen zu können, kann aus einer Reihe von Ähnlichkeitsmaßen gewählt werden, standardmäßig kommt die Log-Likelihood-Ähnlichkeit zum Einsatz. Der kollaborative Filter, um Empfehlungen für einen bestimmten Nutzer zu generieren, ist nutzerbasiert und schätzt die Artikelbewertungen des Nutzers auf Basis der Artikelbewertung ähnlicher Nutzer, die als Nachbarn zur Empfehlungsgenerierung herangezogen werden. Die Anzahl der zu betrachtenden Nachbarn kann konfiguriert werden. Standardmäßig werden die 10 nächsten Nachbarn betrachtet, die eine Mindestähnlichkeit von 0.9 zum aktuellen Nutzer aufweisen.

5.3 Qualitätsmaße

Die folgenden Abschnitte sind Maßen gewidmet, die man heranziehen kann, um die Qualität der generierten Empfehlungen zu beurteilen.

Ideal ist es natürlich, Empfehlungen direkt von der Zielgruppe, also den Nutzern, bewerten zu lassen. Sei es explizit durch Befragungen oder implizit, indem man zum Beispiel A/B-Tests durchführt und das Verhalten von Nutzern, die auf unterschiedliche Weise generierte Empfehlungen präsentiert bekommen, vergleicht. Wenn die Nutzer auf die gegebenen Empfehlungen reagieren können, dann schaltet das auch eine Unsicherheit aus, die bei Offlinetests immer vorhanden ist: Wenn ein Nutzer einen bestimmten Artikel nicht kauft/betrachtet/bewertet, liegt das dann daran, dass er an diesem Artikel tatsächlich kein Interesse hat, oder hängt es vielleicht eher damit zusammen, dass er nichts von der Existenz des besagten Artikels wusste?

Weil Onlinetests nicht immer möglich sind, haben sich eine Reihe von Qualitätsmaßen für die Offline-Evaluation von Recommendersystemen etabliert, von denen einige auch in der vorliegenden Arbeit zum Einsatz kommen. Für einen Überblick sei auf den Beitrag von Guy Shani und Asela Gunawardana in [2, Kap. 8, S. 274 ff.] verwiesen.

5.3.1 Datenlage

Für jeden Benutzer werden 20 Empfehlungen generiert, basierend auf Informationen aus dem Trainingsdatenset. Das Trainingsdatenset enthält für jeden Benutzer eine Liste mit positiv und eine mit negativ bewerteten Filmen. Zur Evaluation steht ein Testset zur Verfügung. Im Falle des 80-20-Testsets enthält dieses ebenfalls für jeden Nutzer Positiv- und Negativlisten, im Falle des All-But-20-Testsets sind für jeden Nutzer genau 20 positiv bewertete Filme im Testset vorhanden.

Für einen gegebenen Nutzer u und Film i können folgende Fälle auftreten:

- i wird empfohlen und ist in der Positivliste von u im Testset enthalten.
- i wird empfohlen und ist in der Negativliste von u im Testet enthalten.
- i wird empfohlen und ist in keiner der beiden Listen enthalten.
- i wird nicht empfohlen, ist aber in der Positivliste von u im Testset enthalten.
- i wird nicht empfohlen und ist in der Negativliste von u im Testset enthalten.
- i wird nicht empfohlen und taucht in keiner der beiden Listen auf.

Wie oft welcher der Fälle pro Nutzer auftritt, können wir auszählen und aus den so erhaltenen Zahlen Kenngrößen berechnen, die etwas über die Qualität der generierten Empfehlungen aussagen.

Die Maße, die wir verwenden, wurden zur Beurteilung der Qualität binärer Klassifikatoren entwickelt. Für unseren Fall heißt das, dass es zwei Klassen von Artikeln gibt: Solche, die dem Nutzer gefallen und solche, auf die das nicht zutrifft. Zur letztgenannten Gruppe von Artikel gehören damit sowohl diejenigen, die der Nutzer negativ bewertet hat, als auch diejenigen, für die keine Bewertung des betreffenden Nutzers vorliegt.

	i wird empfohlen	i wird nicht empfohlen
i gehört P_u an	TP	FN
i gehört nicht P_u an	FP	TN

Tabelle 5.7: Klassifikation der möglichen Ergebnisse der Empfehlung eines Artikels für einen Nutzer

Diese Zuordnung führt zu vier möglichen Fällen, die bei der Empfehlung von Artikeln auftreten können. Diese sind in Tabelle 5.7 dargestellt. P_u bezeichnet die Positivliste des Nutzers u im Testset.

Die Tabelle ist wie folgt zu lesen: Unter den *True Positives (TP)* versteht man all jene Filme in der Empfehlungsliste, die dem Nutzer tatsächlich gefallen, *False Positives (FP)* sind all jene Empfehlungen, die dem Nutzer nicht gefallen. Das umfasst, wie bereits erwähnt, auch all jene Fälle, in denen die Nutzermeinung nicht bekannt ist. Die *False Negatives (FN)* sind nicht empfohlene Filme, für die im Testset positive Bewertungen vorliegen, *True Negatives (TN)* sind all jene Artikel, die zu Recht nicht empfohlen werden.

5.3.2 Precision

Unter der Präzision (*precision*) versteht man den Anteil der korrekten Empfehlungen an allen generierten Empfehlungen. Sie berechnet sich wie folgt:

$$\text{Precision} = \frac{\#TP}{\#TP + \#FP}$$

5.3.3 Recall

Als Recall bezeichnet man den Anteil an für den Nutzern relevanten Artikeln, der ihm tatsächlich empfohlen wird:

$$\text{Recall} = \frac{\#TP}{\#TP + \#FN}$$

5.3.4 False Positive Rate

Das letzte Maß im bekannten Trio ist die *False Positive Rate*, die den Anteil der nicht passenden Empfehlungen an allen Artikeln, die dem Nutzer nicht gefallen, wiedergibt:

$$\text{FPR} = \frac{\#FP}{\#FP + \#TN}$$

Für unseren Anwendungsfall ist dieses Maß aber eher nicht geeignet, weil die Anzahl der FP nach oben durch 20 beschränkt ist, die Menge der für den Nutzer nicht relevanten Filme aber selbst für Nutzer, von denen sehr viele positive Bewertungen im Testset vorliegen, mehr als 3000 Filme enthält, so dass die FPR in jedem Fall maximal im Promillebereich liegt. Dieses Maß ist dort sinnvoller anzuwenden, wo die Trefferanzahl nicht beschränkt ist.

5.3.5 Weitere Maße

Zumindest für den 80-20-Datensatz gibt es einige Fragen, die mit den oben genannten Maßen nicht zufriedenstellend beantwortet werden können. Zu ihrer Beantwortung werden Maße herangezogen, die den oben genannten ähneln, aber mit diesen nicht deckungsgleich sind. Für diese Maße gehen wir auch wieder von der binären Klassifikation weg, die alle Filme, die der Nutzer nicht positiv bewertet hat, als implizit negativ bewertet ansieht, und betrachten wieder die sechs in Abschnitt 5.3.1 genannten Fälle.

Anteil „echter“ False Positives

Die erste der noch offenen Fragen ist die nach der Anzahl der „echten“ False Positives, also danach, von wie vielen der empfohlenen Filme wir definitiv wissen, dass sie später (im Testset) vom Nutzer negativ bewertet wurden. Bezeichnen wir die Empfehlung eines Films, der im Testset in der Negativliste des Nutzers enthalten ist, mit FP', dann berechnet sich der Anteil der „echten“ FPs an den Empfehlungen wie folgt:

$$\text{Anteil FP'} = \frac{\#FP'}{\#TP + \#FP}$$

Anteil bekannter Filme an den Empfehlungen

In eine ähnliche Richtung geht die Frage nach der Anzahl der dem Nutzer bekannten Filme unter den Empfehlungen. Als bekannt werden dabei alle Filme angenommen, die im Testdatenset mit positiver oder negativer Bewertung des Nutzers vorhanden sind. Hintergrund ist die Überlegung, dass das Auftauchen des Films in einer der Bewertungslisten des Nutzers zumindest ein Hinweis darauf ist, dass der Nutzer den Film für interessant genug hielt, um ihn anzuschauen und zu bewerten.

Diesen Anteil kann man wie folgt berechnen:

$$\text{Anteil bekannter Filme} = \frac{\#FP' + \#TP}{\#TP + \#FP}$$

„Korrigierte“ Precision

Ein Problem des 80-20-Datensatzes ist, dass nicht für alle Nutzer 20 positive Bewertungen im Testset vorliegen. Aus diesem Grund könnte auch der beste Empfehlungsalgorithmus für einige Nutzer keine Präzision von 1.0 erreichen. Um ein Gespür dafür zu bekommen, wie nah der Empfehlungsalgorithmus am bestmöglichen Ergebnis ist, korrigieren wir die Berechnung der Präzision wie folgt:

$$\text{Precision'} = \frac{\#TP}{\max(20, |P_u|)}$$

6 Evaluation

Das verwendete Datenset ist nun ebenso bekannt wie die Ansätze zur Empfehlungsgenerierung und mögliche Qualitätsmaße. Bleibt nur noch, die generierten Empfehlungen auch auszuwerten. Dies ist das Ziel dieses Kapitels, das wie folgt gegliedert ist:

In den ersten zwei Teilkapiteln werden die unterschiedlichen Ansätze auf die beiden generierten Trainings- und Testdatensätze angewandt und die Empfehlungen, die auf dieser Grundlage für einzelne Nutzer erstellt werden, mithilfe der im vorigen Kapitel vorgestellten Qualitätsmaße verglichen und bewertet. Es folgt ein Teilkapitel, das dem Vergleich der Listen ähnlicher Filme gewidmet ist, die mittels der getesteten Empfehlungsalgorithmen basierend auf einem „Ausgangsfilm“ erstellt werden. Die Zielsetzung ist, einen Überblick zu erhalten, wie groß die Übereinstimmungen zwischen den einzelnen Ansätzen sind und wie sich die Variation von Parametern auswirkt. Den Abschluss bildet eine kurze Zusammenfassung der Ergebnisse

6.1 80-20-Testset

Für das 80-20-Testset ist es sinnvoll, alle im vorhergehenden Kapitel beschriebenen Maße zu berechnen. Dabei wurden für das Taste-Plugin unterschiedliche Nachbarschaftsgrößen und Ähnlichkeitsmaße für die Empfehlungsgenerierung vorgegeben, die jeweils unter der entsprechenden Grafik vermerkt sind. „NN“ steht dabei für „nächste Nachbarn“, „LL“ für „Log-Likelihood-Ähnlichkeit“, „Pear.“ für die Pearson-Ähnlichkeit und „F.“ steht für „Film(e)“.

Die einzelnen Maße wurden pro Nutzer berechnet und dann aggregiert, so dass sich eine Verteilung der möglichen Werte der Qualitätsmaße (in 0,05-er Schritten) auf die Nutzer ergibt.

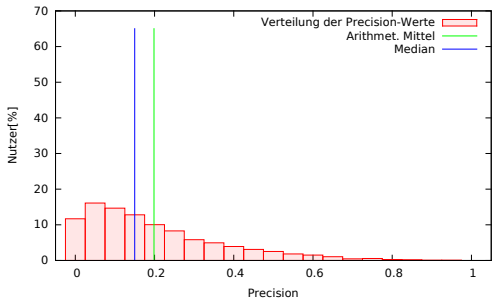
6.1.1 Precision

Abbildung 6.1 zeigt die Verteilung der Präzisionswerte für die unterschiedlichen Empfehlungsansätze, Tabelle 6.1 gibt einen Überblick über statistische Kennzahlen.

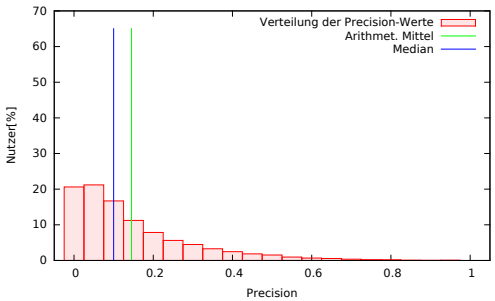
	min	max	arithmet. Mittel	Median
Significant Terms	0,0 (0 F.)	0,95 (20 F.)	0,199 (3,9 F.)	0,15 (3 F.)
Terms	0,0 (0 F.)	0,95 (19 F.)	0,145 (2,9 F.)	0,1 (2 F.)
Taste (5 NN, LL)	0,0 (0 F.)	0,8 (16 F.)	0,095 (1,9 F.)	0,05 (1 F.)
Taste (10 NN, LL)	0,0 (0 F.)	0,7 (14 F.)	0,059 (1,18 F.)	0,05 (1 F.)
Taste (20 NN, LL)	0,0 (0 F.)	0,6 (12 F.)	0,036 (0,72 F.)	0,0 (0 F.)
Taste (10 NN, Pear.)	0,0 (0 F.)	0,35 (7 F.)	0,039 (0,78 F.)	0,0 (0 F.)

Tabelle 6.1: Statistische Kennzahlen Precision

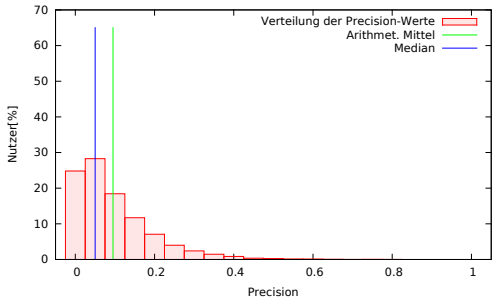
Es zeigt sich, dass keiner der Ansätze auch nur in die Nähe einer Präzision von 1,0 kommt. Im Mittel liefert der Significant-Terms-Ansatz einen Film mehr zurück als der Terms-Ansatz und dieser wiederum einen Film mehr als der beste der Taste-Ansätze. Von den Taste-Plugin-Konfigurationen sorgt die mit der kleinsten Nachbarschaft für die beste Präzision.



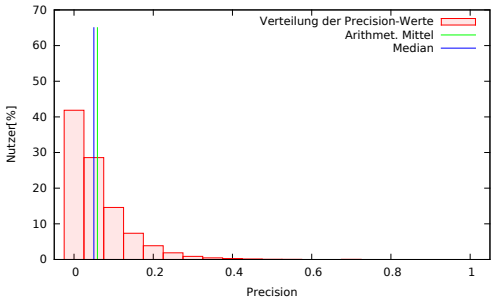
(a) Significant Terms



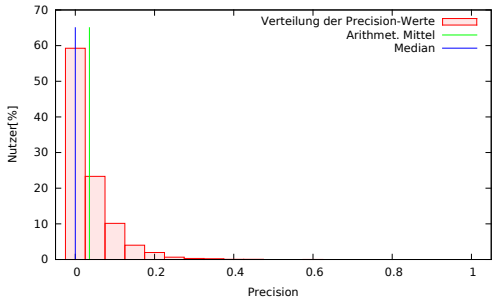
(b) Terms



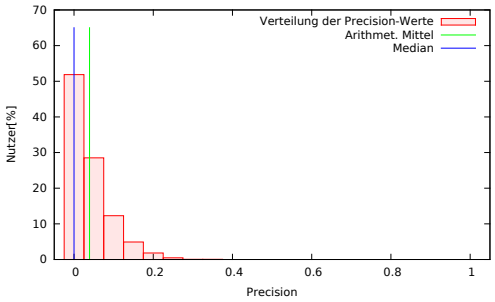
(c) Taste (5 NN, Log Likelihood)



(d) Taste (10 NN, Log Likelihood)



(e) Taste (20 NN, Log Likelihood)



(f) Taste (10 NN, Pearson)

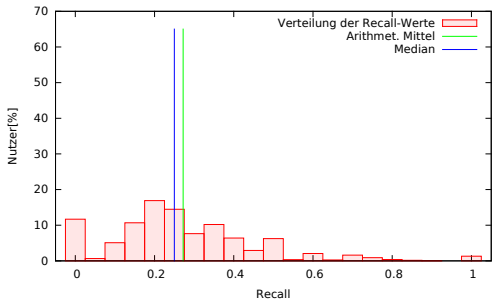
Abbildung 6.1: Auswertung der Präzision der generierten Empfehlungen

6.1.2 Recall

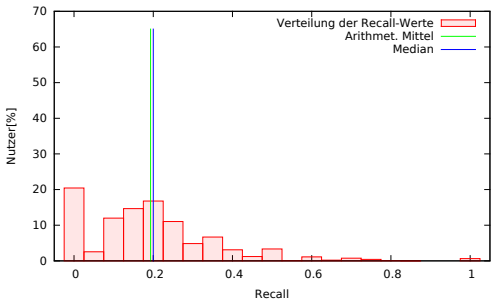
Auch für den Recall wurden statistische Kennzahlen berechnet und die aggregierten Ergebnisse geplottet. Die Kennzahlen finden sich in Tabelle 6.2, die Diagramme in Abbildung 6.2. Die Kennzahlen in Filme umzurechnen ist in diesem Fall nicht sinnvoll, weil die Summe der Anzahl von TP und FN für alle Nutzer unterschiedlich ist. Die „Rangfolge“ ist ganz ähnlich wie bei der Präzision.

	min	max	arithmet. Mittel	Median
Significant Terms	0,0	1,0	0,272	0,25
Terms	0,0	1,0	0,192	0,2
Taste (5 NN, LL)	0,0	1,0	0,159	0,1
Taste (10 NN, LL)	0,0	1,0	0,095	0,05
Taste (20 NN, LL)	0,0	1,0	0,051	0,0
Taste (10 NN, Pear.)	0,0	1,0	0,066	0,0

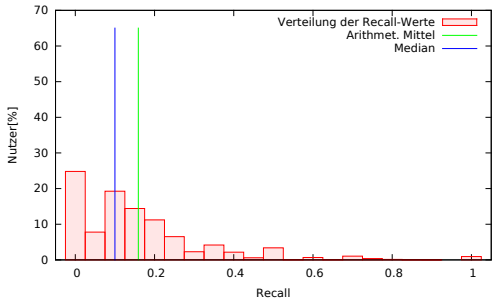
Tabelle 6.2: Statistische Kennzahlen Recall



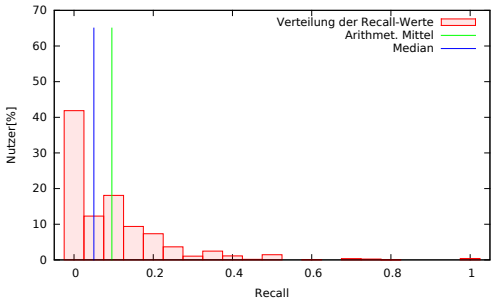
(a) Significant Terms



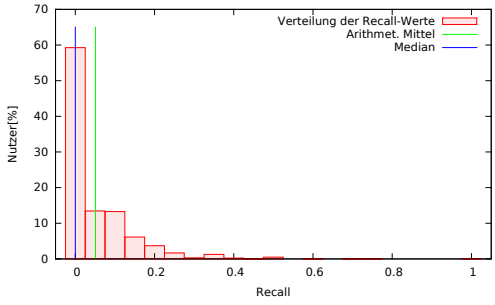
(b) Terms



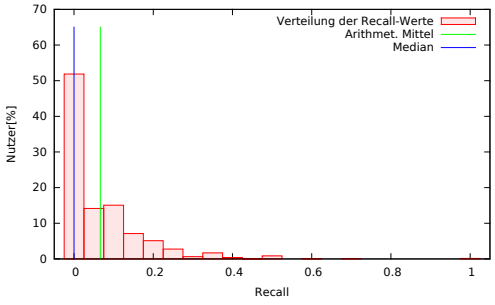
(c) Taste (5 NN, Log Likelihood)



(d) Taste (10 NN, Log Likelihood)



(e) Taste (20 NN, Log Likelihood)



(f) Taste (10 NN, Pearson)

Abbildung 6.2: Auswertung des Recalls der generierten Empfehlungen

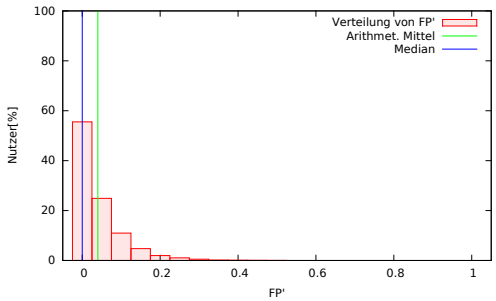
6.1.3 Echte False Positives

Bei den „echten“ False Positives, also bei dem Anteil der Empfehlungen, der sich auf der Negativliste im Testset des Nutzers befindet, sieht die Rangfolge anders aus.

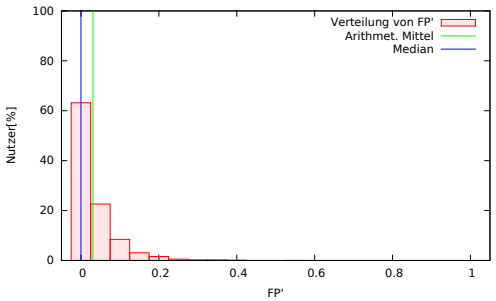
	min	max	arithmet. Mittel	Median
Significant Terms	0,0 (0 F.)	0,5 (10 F.)	0,040 (0,8 F.)	0,0 (0 F.)
Terms	0,0 (0 F.)	0,55 (11 F.)	0,031 (0,62 F.)	0,0 (0 F.)
Taste (5 NN, LL)	0,0 (0 F.)	0,5 (10 F.)	0,024 (0,48 F.)	0,0 (0 F.)
Taste (10 NN, LL)	0,0 (0 F.)	0,3 (6 F.)	0,011 (0,22 F.)	0,0 (0 F.)
Taste (20 NN, LL)	0,0 (0 F.)	0,3 (6 F.)	0,006 (0,12 F.)	0,0 (0 F.)
Taste (10 NN, Pear.)	0,0 (0 F.)	0,25 (5 F.)	0,011 (0,22 F.)	0,0 (0 F.)

Tabelle 6.3: Statistische Kennzahlen für die „echten“ False Positives

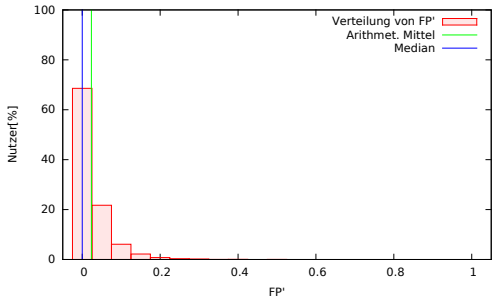
Hier schneiden die Significant-Terms-Empfehlungen am schlechtesten ab. Positiv ist allerdings, dass der Median für alle Ansätze bei 0 FP’ liegt und selbst beim schlechtesten Ansatz im Schnitt weniger als ein definitiv falscher Film empfohlen wird.



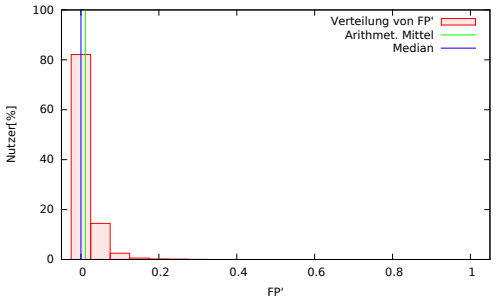
(a) Significant Terms



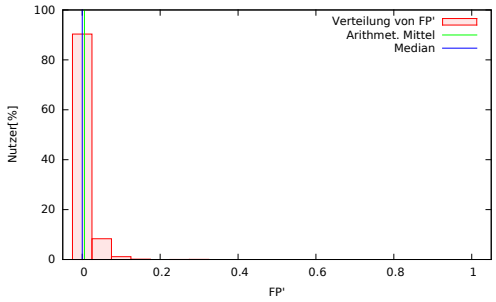
(b) Terms



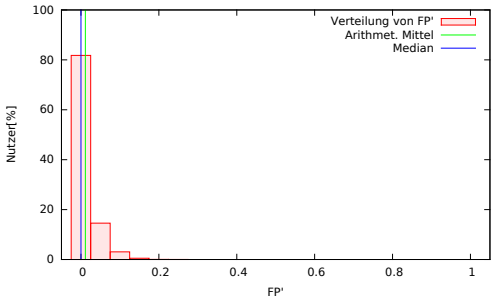
(c) Taste (5 NN, Log Likelihood)



(d) Taste (10 NN, Log Likelihood)



(e) Taste (20 NN, Log Likelihood)



(f) Taste (10 NN, Pearson)

Abbildung 6.3: Auswertung der „echten“ False Positives

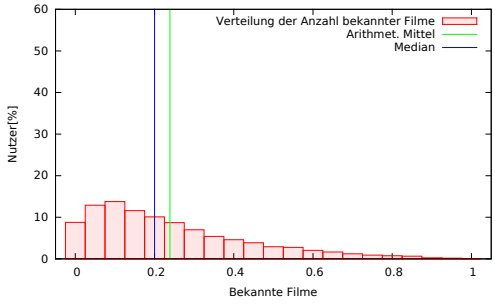
6.1.4 Anteil bekannter Filme

Statistische Kenngrößen für den Anteil bekannter Filme unter den Empfehlungen sind in Tabelle 6.4 zusammengefasst, die Plots findet man in Abbildung 6.4.

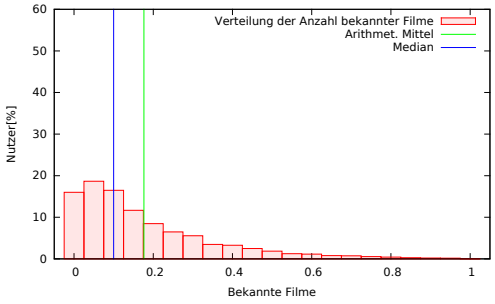
Das Bild, das sich bietet, ist wieder ähnlich wie bei der Präzision oder beim Recall. Keiner der Ansätze liefert ausschließlich bekannte Filme zurück. Die Significant-Terms-Empfehlungen schneiden am besten ab, gefolgt von Terms. Bei den Taste-Empfehlungen schneiden wieder die Empfehlungen am besten ab, die unter Rückgriff auf die wenigsten Nachbarn generiert wurden.

	min	max	arithmet. Mittel	Median
Significant Terms	0,0 (0 F.)	1,0 (20 F.)	0,239 (4,78 F.)	0,2 (4 F.)
Terms	0,0 (0 F.)	1,0 (20 F.)	0,176 (3,52 F.)	0,1 (2 F.)
Taste (5 NN, LL)	0,0 (0 F.)	0,9 (18 F.)	0,119 (2,38 F.)	0,1 (2 F.)
Taste (10 NN, LL)	0,0 (0 F.)	0,8 (16 F.)	0,070 (1,4 F.)	0,05 (1 F.)
Taste (20 NN, LL)	0,0 (0 F.)	0,6 (12 F.)	0,041 (0,82 F.)	0,0 (0 F.)
Taste (10 NN, Pear.)	0,0 (0 F.)	0,5 (10 F.)	0,050 (1 F.)	0,05 (1 F.)

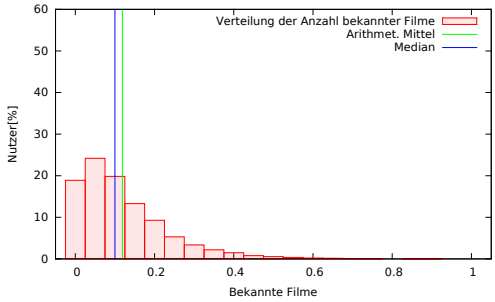
Tabelle 6.4: Statistische Kennzahlen für die Anzahl bekannter Filme



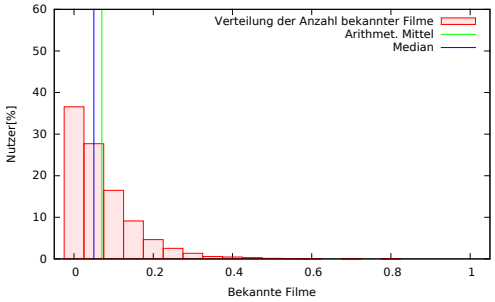
(a) Significant Terms



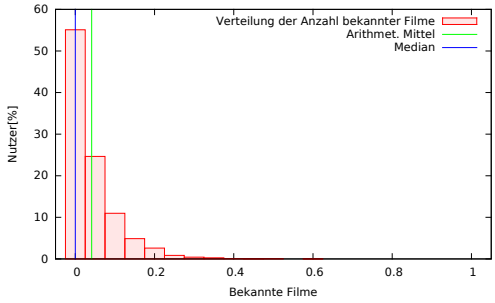
(b) Terms



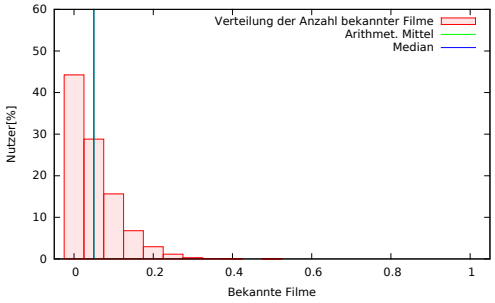
(c) Taste (5 NN, Log Likelihood)



(d) Taste (10 NN, Log Likelihood)



(e) Taste (20 NN, Log Likelihood)



(f) Taste (10 NN, Pearson)

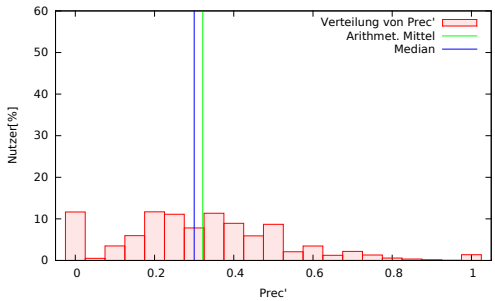
Abbildung 6.4: Anteil bekannter Filme unter den generierten Empfehlungen

6.1.5 Korrigierte Präzision

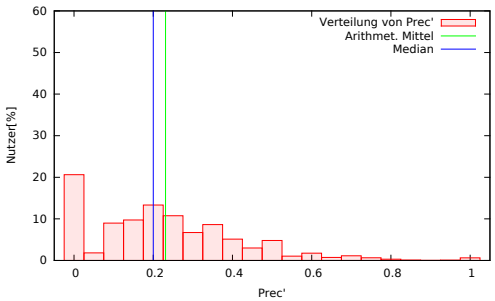
Auch für die „korrigierte“ Präzision gibt es eine Übersichtstabelle und Grafiken zur Veranschaulichung. Auch hier liegen die Empfehlungen mit Significant Terms vorne. Eine Umrechnung in Filme ist hier wieder nicht sinnvoll, weil pro Nutzer unterschiedlich viele Empfehlungen berücksichtigt wurden.

	min	max	arithmet. Mittel	Median
Significant Terms	0,0	1,0	0,322	0,3
Terms	0,0	1,0	0,231	0,2
Taste (5 NN, LL)	0,0	1,0	0,179	0,15
Taste (10 NN, LL)	0,0	1,0	0,107	0,05
Taste (20 NN, LL)	0,0	1,0	0,058	0,0
Taste (10 NN, Pear.)	0,0	1,0	0,071	0,0

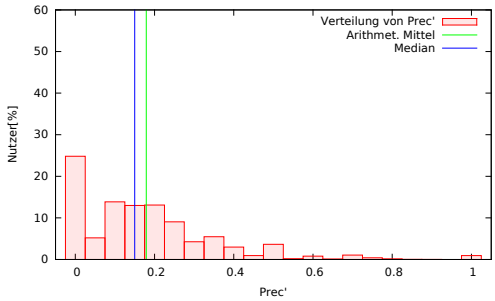
Tabelle 6.5: Statistische Kennzahlen für die „korrigierte“ Präzision.



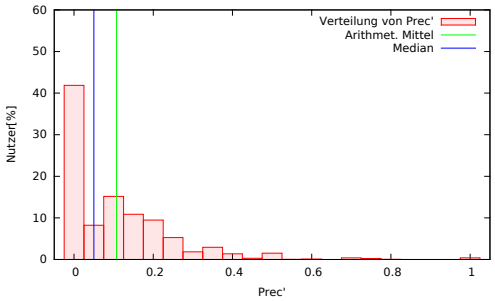
(a) Significant Terms



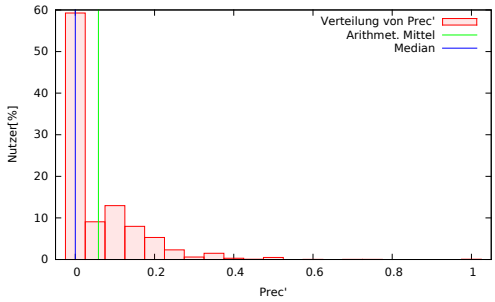
(b) Terms



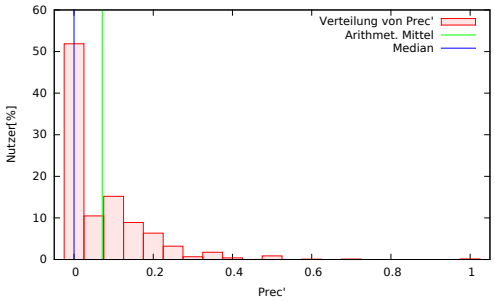
(c) Taste (5 NN, Log Likelihood)



(d) Taste (10 NN, Log Likelihood)



(e) Taste (20 NN, Log Likelihood)



(f) Taste (10 NN, Pearson)

Abbildung 6.5: Auswertung der „korrigierten“ Präzision

6.2 All-But-20-Testset

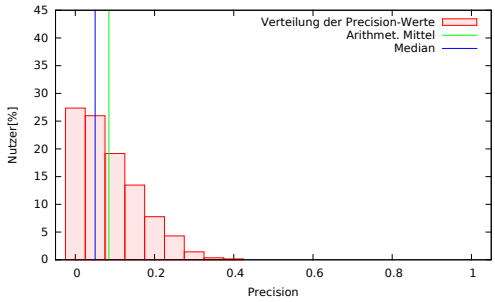
Für das All-But-20-Testset fallen Präzision und Recall zusammen. Korrekturmöglichkeiten ergeben sich nicht, da für jeden Nutzer genau 20 positive Bewertungen im Testset vorhanden sind.

Wie sich die Präzision für die einzelnen Ansätze verteilt, zeigt Tabelle 6.6, die Plots sind in Abbildung 6.6 zu sehen. Für die Taste-Empfehlungen, die die Log-Likelihood-Ähnlichkeit verwenden, hatte die Wahl der Nachbarschaftsgröße (5, 10 oder 20 NN) in diesem Fall keinen Einfluss auf die Präzision, weshalb in der Übersicht nur LL mit 10 NN auftaucht.

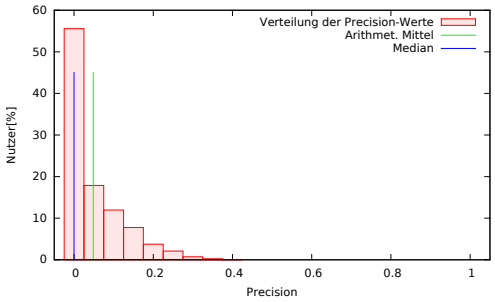
Bei diesem Testset schneidet keiner der getesteten Ansätze wirklich gut ab. Die Significant-Terms-Empfehlungen schneiden etwas besser ab als die Taste-Empfehlungen, die unter Rückgriff auf die Log-Likelihood-Ähnlichkeit erstellt wurden. Besonders schlecht schneiden die Taste-Empfehlungen unter Verwendung der Pearson-Ähnlichkeit ab, gefolgt von den Terms-Empfehlungen.

	min	max	arithmet. Mittel	Median
Significant Terms	0,0 (0 F.)	0,4 (8 F.)	0,085 (1,7 F.)	0,05 (1 F.)
Terms	0,0 (0 F.)	0,4 (8 F.)	0,048 (0,96 F.)	0,0 (0 F.)
Taste (10 NN, LL)	0,0 (0 F.)	0,35 (7 F.)	0,054 (1,08 F.)	0,05 (1 F.)
Taste (10 NN, Pear.)	0,0 (0 F.)	0,3 (6 F.)	0,019 (0,38 F.)	0,0 (0 F.)

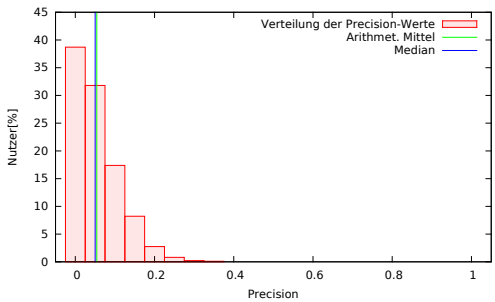
Tabelle 6.6: Statistische Kennzahlen für die Präzision



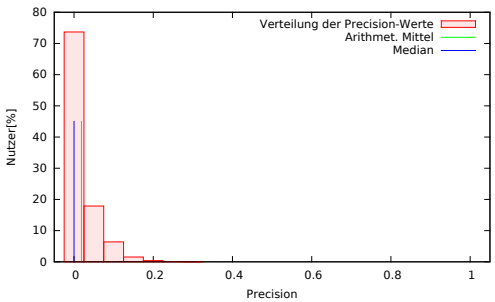
(a) Significant Terms



(b) Terms



(c) Taste (10 NN, Log Likelihood)



(d) Taste (10 NN, Pearson)

Abbildung 6.6: Auswertung der Präzision der generierten Empfehlungen

6.3 Vergleich der Listen ähnlicher Filme

Weiter vorne wurde bereits angemerkt, dass die getesteten Recommender nicht nur Empfehlungen für einzelne Nutzer generieren, sondern auch Filme empfehlen können, die zu einem anderen Film „passen“. In diesem Abschnitt soll untersucht werden, wie stark sich die generierten Empfehlungen ähneln. Um zwei Ansätze zu vergleichen, wurden für alle Filme, für die im Trainingsset des 80-20-Datensatzes positive Bewertungen (ggf. korrigiert, s.u.) enthalten sind, jeweils 20 Empfehlungen generiert. Filme, für die diese Anzahl von Empfehlungen aufgrund der im Trainingsset vorhandenen Informationen nicht erreicht werden konnte, wurden von der Berechnung ausgeschlossen.[21]

Auf diese Weise werden zwar nicht alle Filme berücksichtigt, aber es wird sichergestellt, dass nur vollständige Empfehlungslisten verglichen werden.

6.3.1 Empfehlungsgenerierung

Filme, die zu einem vorgegebenen Film passen, findet das Taste-Plugin, indem es für diesen Film und alle möglichen „Kandidaten“ ein Ähnlichkeitsmaß berechnet und dann die Filme empfiehlt, für die die höchsten Werte zurückgeliefert werden. Welches Ähnlichkeitsmaß verwendet werden soll, kann man bei der Erstellung der Empfehlungen angeben. Für die Vergleiche in den folgenden Abschnitten wurde die Log-Likelihood-Ähnlichkeit gewählt, was der Default-Einstellung des Plugins entspricht.

Zur Generierung der Empfehlungen mittels der Terms- oder Significant-Terms-Aggregation können, wie bereits erwähnt, verschiedene Felder verwendet werden. Zum Einsatz kamen bei diesem Test neben dem Feld, das für jeden Nutzer die Filme enthält, die er im Trainingsset positiv (mehr als 3 Sterne) bewertet hat, noch zwei weitere Felder: Sie enthalten ebenfalls alle vom Nutzer im Trainingsset positiv bewerteten Filme, allerdings bereinigt um Verzerrungseffekte, die dadurch zustande kommen, dass einige Nutzer systematisch besser oder schlechter bewerten als andere und dass einige Filme systematisch besser oder schlechter bewertet werden als

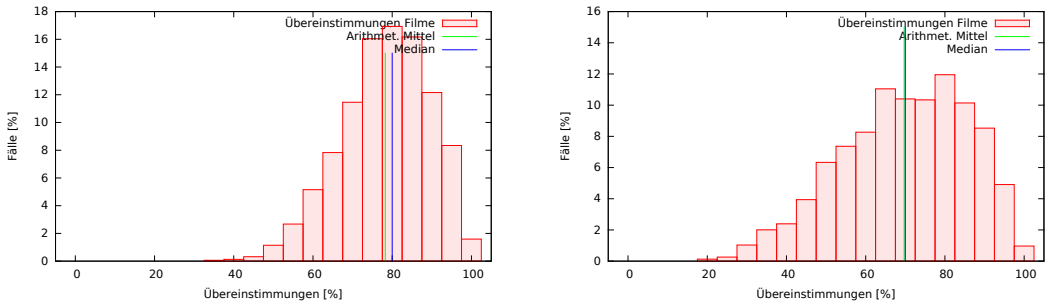
[21] Zu weniger als 20 generierten Empfehlungen kommt es, wenn entweder für den Film selbst weniger als 10 positive Bewertungen im Testset vorhanden sind, oder wenn er mit vielen Filmen zusammen auftaucht, die von weniger als 10 der Nutzer, die den Ausgangsfilm mögen, positiv bewertet wurden.

andere. Bei kollaborativen Filtern werden diese systematischen Verzerrungen in den *baseline predictors* berücksichtigt, die uns bereits in Abschnitt 2.2.3 begegnet sind.

Um die systematischen Verzerrungen herauszurechnen, wurde ein solcher *baseline predictor* für jeden Nutzer und jeden Film durch Bildung des arithmetischen Mittels über alle zugehörigen Bewertungen berechnet. In die um den nutzerspezifischen Verzerrungsfaktor bereinigte Positivliste wurden dann nur diejenigen Bewertungen des Nutzers aufgenommen, die nach Abzug des Faktors noch positiv waren. Analog wurden, um systematische Verzerrungen für die einzelnen Filme herauszurechnen, nur diejenigen Filme in die entsprechende Positivliste übernommen, deren Bewertungen durch den Nutzer auch nach Abzug des *baseline predictors* für den entsprechenden Film noch positiv waren.[22]

6.3.2 Korrekturauswirkungen Significant-Terms-Empfehlungen

Um einen Eindruck zu bekommen, welche Auswirkungen die gerade erwähnten Korrekturen auf die generierten Empfehlungen haben, untersuchen wir, wie viele der generierten Empfehlungen übereinstimmen, wenn einmal die normale Positivliste des jeweiligen Nutzers zur Empfehlungsgenerierung verwendet wird und alternativ dazu die zwei korrigierten Positivlisten. Es zeigt sich, dass beide Korrekturen durchaus einen Einfluss auf die generierten Empfehlungen haben. Die Empfehlungslisten sind in der Regel nicht identisch, wobei die Abweichungen im Schnitt größer sind, wenn um artikelspezifische Verzerrungsfaktoren korrigiert wird.



(a) Übereinstimmung unkorrigiert und um Nutzerbias korrigiert

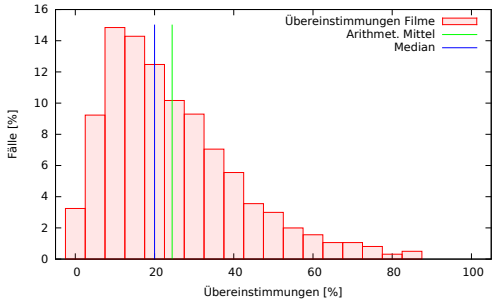
(b) Übereinstimmung unkorrigiert sowie um Artikelbias korrigiert

Abbildung 6.7: Übereinstimmung der unkorrigierten Significant-Terms-Empfehlungen mit den korrigierten

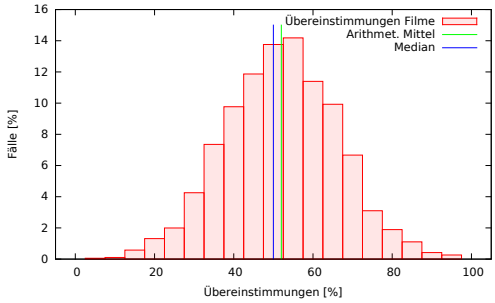
[22] Da wir nur ganzzahlige Bewertungen haben, sind es immer alle Filme in einer „Sternekategorie“, die damit entweder neu in die Positivliste aufgenommen oder aber aus dieser entfernt werden.

6.3.3 Vergleich von Terms- und Significant-Terms-Empfehlungen

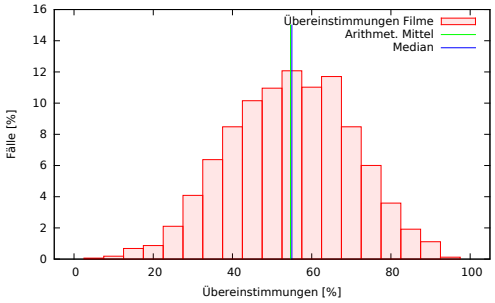
Weiter oben wurde bereits angemerkt, dass die Verwendung der Significant-Terms-Aggregation einerseits und der Terms-Aggregation andererseits in vielen Fällen zu sehr unterschiedlichen Empfehlungen führen wird, weil erstere nicht nur die Häufigkeit des gemeinsamen Auftretens zweier Filme in einer Positivliste berücksichtigt, sondern auch dessen statistische Signifikanz. Dass sich die vorgeschlagenen Filme in der Regel tatsächlich stark unterscheiden, veranschaulicht Abbildung 6.8. Neben den „normalen“ Trainingsdaten wurden auch die um den jeweiligen Nutzer- bzw. Artikelbias korrigierten Daten verwendet. Es zeigt sich, dass die Korrekturen zu besserer Übereinstimmung führen. Während ohne Berücksichtigung der Verzerrungsfaktoren im Schnitt nur etwa 20% der Empfehlungen übereinstimmen, sind es bei den korrigierten Empfehlungen etwa 50%.



(a) Ohne Korrektur



(b) Korrektur um Artikelbias



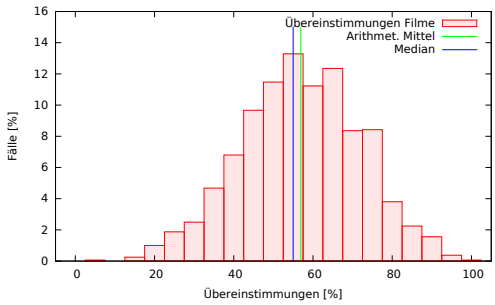
(c) Korrektur um Nutzerbias

Abbildung 6.8: Übereinstimmung Significant Terms und Terms

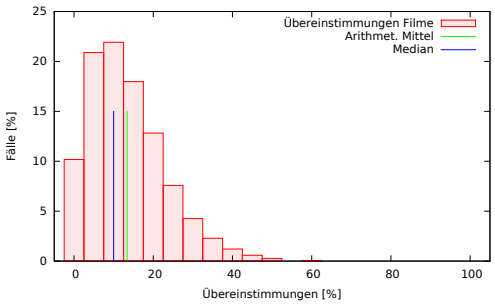
6.3.4 Vergleich mit Taste-Empfehlungen

Abbildung 6.9 zeigt, wie gut die Empfehlungen, die unter Verwendung der Significant-Terms- bzw. Terms-Aggregation anhand der unveränderten Positivlisten

der Nutzer erstellt wurden, mit denen übereinstimmen, die das Taste-Plugin generiert. Die Übereinstimmung zwischen Taste- und Significant-Terms-Empfehlungen ist deutlich besser als die zwischen Taste- und Terms-Empfehlungen.



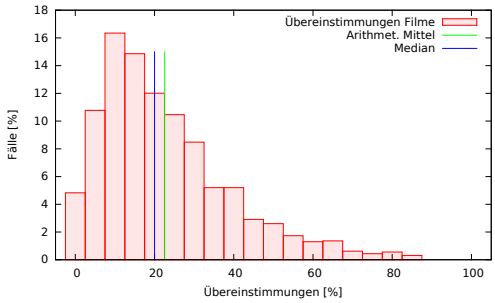
(a) Significant Terms



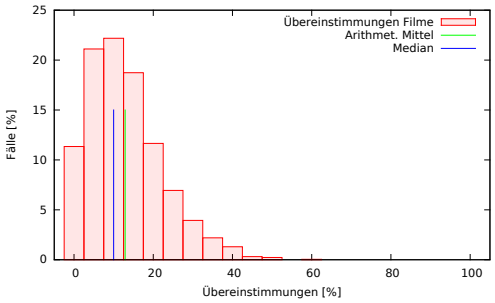
(b) Terms

Abbildung 6.9: Übereinstimmung zwischen (Significant-) Terms- und Taste-Empfehlungen

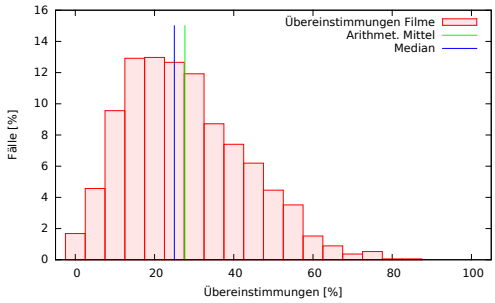
In Abbildung 6.10 sieht man, wie sich das Ergebnis ändert, wenn die systematischen Verzerrungen für Nutzer bzw. Filme herausgerechnet werden.



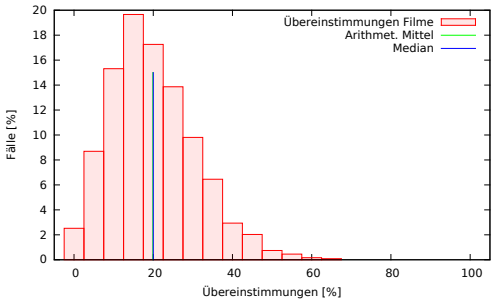
(a) Sig. Terms korrigiert um Nutzerbias



(b) Terms korrigiert um Nutzerbias



(c) Sig. Terms korrigiert um Artikelbias



(d) Terms korrigiert um Artikelbias

Abbildung 6.10: Übereinstimmung der um Verzerrungen korrigierten (Significant-) Terms-Empfehlungen mit denen, die das Taste-Plugins anhand der LL-Ähnlichkeit berechnet hat.

Während sich die Übereinstimmungen zwischen Taste- und Terms-Empfehlungen

nur geringfügig erhöhen, wenn der Artikelbias berücksichtigt wird, haben die Korrekturfaktoren im Falle der Significant-Terms-Empfehlungen sogar einen negativen Einfluss auf deren Übereinstimmung mit den Taste-Empfehlungen.

6.4 Zusammenfassung der Ergebnisse

Die Auswertung der generierten Empfehlungen zeigt, dass sich alle drei getesteten Ansätze auf einem ähnlichen Niveau bewegen. Die Significant-Terms-Empfehlungen schneiden etwas besser ab als die anderen Ansätze, was Precision, Recall sowie den Anteil der bekannten Filme unter den Empfehlungen angeht, produzieren aber im Schnitt auch etwas mehr „echte“ False Positives.

Betrachtet man die Listen ähnlicher Filme, so sind die Unterschiede zwischen den Terms- und Significant-Terms-Empfehlungen wie erwartet recht groß. Die Übereinstimmungen zwischen den Significant-Terms- und Taste-Empfehlungen sind etwas größer.

Weiterhin zeigt sich, dass die Berücksichtigung von systematischen Verzerrungen einen deutlichen Einfluss auf die generierten Empfehlungen hat. Die Auswirkung auf die Empfehlungen für einzelne Nutzer wurde im Rahmen der Arbeit zwar nicht systematisch untersucht, stellt aber einen interessanten Aspekt für weitergehende Analysen dar.

7 Fazit und Ausblick

Ziel der Arbeit war es, das Forschungsgebiet Recommendersysteme mit der aktuell im industriellen Umfeld vielbeachteten und -genutzten Open-Source-Suchmaschine *Elasticsearch* zu verknüpfen und zu evaluieren, ob sich ein Empfehlungssystem mit den Bordmitteln von *Elasticsearch* als Plug-and-Play-Lösung umsetzen lässt.

In den folgenden Abschnitten sollen die Ergebnisse der Arbeit sowohl auf ihre Anwendbarkeit als auch auf die Qualität der generierten Empfehlungen hin überprüft werden. Zum Vergleich ziehen wir nochmals die entsprechenden Resultate des in Abschnitt 5.2.3 vorgestellten Taste-Plugins heran, das die Funktionen von *Apache Mahout Taste* für *Elasticsearch* zur Verfügung stellt. Abschnitt 7.4 gibt einen Ausblick auf mögliche Anknüpfungspunkte für nachfolgende Arbeiten.

7.1 Anwendbarkeit des Empfehlungsplugins

Das Ziel, eine Plug-and-Play-Lösung zur Generierung von Empfehlungen mittels *Elasticsearch* umzusetzen, ist durch das in der Masterthesis entstandene Plugin auf jeden Fall erreicht worden. Das Plugin erweitert die REST-Schnittstelle von *Elasticsearch* und erlaubt es dem Nutzer, für beliebige Indizes verschiedene Arten von Recommendern zu registrieren, die dann zur Empfehlungsgenerierung verwendet werden können. Die einzige Voraussetzung ist, dass die Daten in einer Form vorliegen, die zur Verwendung von (Significant) Terms zur Empfehlungsgenerierung geeignet ist. Das heißt, dass Listen mit ähnlich bewerteten/bevorzugten/gemeinsam gekauften/... Artikeln vorliegen sollten. Sollen Empfehlungen für einzelne Nutzer generiert werden, müssen diese Vorlieben den jeweiligen Nutzern zugeordnet werden können. Sollen weitere Informationen über die zu empfehlenden Artikel ausgegeben werden, dann sollten die „Positivlisten“ Artikel-IDs enthalten, die verwendet werden können, um diese weiteren Informationen in *Elasticsearch* abzufragen.

Das Empfehlungsplugin ist nicht nur einfach anzuwenden, es lässt sich auch leicht um weitere Arten von Recommendern erweitern, so dass andere Ansätze zur Empfehlungsgenerierung verwendet werden können. Neue Recommender müssen lediglich

eine abstrakte Recommenderklasse erweitern und anschließend ihren Typ und ihre Klasse registrieren, um instanziiert werden zu können.

Für Szenarien, in denen „mal eben schnell“ unterschiedliche Empfehlungsmethoden für einige Nutzer getestet werden sollen, hat das im Rahmen der Thesis entstandene Plugin den Vorteil, dass keine Offlineberechnungen nötig sind, sondern die Empfehlungen für jeden Nutzer zumindest auf den verwendeten Testdatensätzen innerhalb weniger Sekunden berechnet werden. Das Taste-Plugin hingegen bietet nur die Möglichkeit, eine bestimmte Anzahl von Empfehlungen für alle Nutzer zu berechnen. Diese werden dann allerdings auch gespeichert und müssen nicht jedesmal neu generiert werden. Ihre Berechnung dauert allerdings je nach gewählter Nachbarschaftsgröße und verwendetem Ähnlichkeitsmaß zwischen mehreren Minuten und mehreren Tagen.

7.2 Qualität der generierten Empfehlungen

Die Qualität der Empfehlungen, die nur mit Bordmitteln von *Elasticsearch* realisiert werden, ist zumindest auf den verwendeten Testsets nicht schlechter als die von Empfehlungen, die mittels des Taste-Plugins realisiert wurden, das etablierte Empfehlungsmethoden für *Elasticsearch* zur Verfügung stellt.

Im Gegenteil: Insbesondere die Significant-Terms-Empfehlungen schnitten durchweg besser ab und dies bei allen verwendeten Qualitätsmaßen außer der Rate der „echten“ False Positives (also der dem Nutzer nicht zusagenden Empfehlungen), bei denen der Schnitt allerdings auch für die Significant-Terms-Empfehlungen bei weniger als einem falsch empfohlenen Film pro Nutzer lag.

Bislang werden für die Generierung der Empfehlungen nur die von den Nutzern abgegebenen Bewertungen herangezogen und dies auch nur in sehr vereinfachter Form („Nutzer A mag ...“, „Nutzer B mag ... nicht“). Für die Generierung von Empfehlungen im professionellen Einsatz werden häufig verschiedene Methoden kombiniert und ganz unterschiedliche Daten verwendet. Hier könnten die Empfehlungen, die mittels Significant Terms generiert wurden, einen Teil des Ganzen bilden. Denkbar ist auch, diesen Empfehlungsansatz dort anzuwenden, wo vorher noch gar keine Empfehlungen für die Nutzer zur Verfügung gestellt wurden, *Elasticsearch* im Einsatz ist und eine Möglichkeit gesucht wird, Recommendations ohne großen Aufwand zu testen.

7.3 Einfluss von Parametern auf die generierten Empfehlungen

Während bei den Empfehlungen, die das Taste-Plugin generiert, die gewählte Nachbarschaftsgröße und das Ähnlichkeitsmaß eine entscheidende Rolle dafür spielen, welche Artikel einem Nutzer empfohlen werden, wirkt sich bei den (Significant-) Terms-Empfehlungen die gewählte Empfehlungsanzahl auf die Zusammensetzung der Liste empfohlener Artikel aus.

Das liegt daran, dass zur Generierung der Empfehlungsliste für alle Artikel, die der Nutzer mag, (signifikante) Terme gesucht werden und von diesen diejenigen mit den höchsten (Signifikanz-)Scores empfohlen werden, wobei Scores addiert werden, wenn ein Artikel in mehr als einer Liste auftaucht. Dieses Vorgehen führt dazu, dass die Berücksichtigung von $n + k$ statt n Artikeln eine Verschiebung der Reihenfolge in den Top n nach sich ziehen kann und gegebenenfalls Artikel, die in vielen Listen auftauchen, allerdings nicht immer unter den Top n sind, es durch die Addition der Scores bei $n + k$ betrachteten Artikeln doch weit nach vorne schaffen.

Ein vereinfachtes Beispiel mag dies verdeutlichen: Für einen Nutzer sollen Empfehlungen generiert werden, basierend auf seiner Präferenz für die Artikel X und Y. Bei n zu erstellenden Empfehlungen werden sowohl für Artikel X als auch für Artikel Y n (signifikante) Terme berechnet. Diese Listen werden dann zusammengeführt und nach Scores sortiert, wobei die Scores von mehrfach auftauchenden Artikeln addiert werden.

Nehmen wir an, bei drei zu empfehlenden Artikeln sehen die Listen ähnlicher Artikel (bereinigt um bereits bekannte Artikel) wie in Tabelle 7.1 aus. Dann enthält die Empfehlungsliste, wie in dieser Tabelle dargestellt, die Artikel A, B und C. Sollen nun fünf statt drei Artikel empfohlen werden, trägt jeder der „Basisartikel“ X und Y entsprechend fünf Empfehlungskandidaten zur endgültigen Liste bei und die Addition von Scores sorgt dafür, dass sich auch in den Top 3 Veränderungen ergeben. Dies ist in Tabelle 7.2 dargstellt.

Ein weiterer Parameter, der Einfluss auf die generierten (Significant-) Terms-Empfehlungen hat, ist die Angabe der minimalen Anzahl von Dokumenten, in denen ein Term auftreten muss, um als (signifikanter) Term zurückgeliefert zu werden. Je höher man diesen Parameter wählt, auf desto breiterer Basis stehen die Empfehlungen, allerdings können selten bewertete Filme dann nicht mehr empfohlen werden. In der

Liste X mit Scores	Liste Y mit Scores	Empfehlungen
A (21)	B (20)	A (21)
C (15)	D (14)	B (20)
E (13)	C (4)	C (19)

Tabelle 7.1: Generierung von drei Empfehlungen basierend auf Präferenz für X und Y (vereinfachte Darstellung)

Liste X mit Scores	Liste Y mit Scores	Empfehlungen
A (21)	B (20)	D (25)
C (15)	D (14)	A (22)
E (13)	C (4)	B (21)
D (11)	E (3)	C (19)
B (1)	A (1)	E (16)

Tabelle 7.2: Generierung von fünf Empfehlungen basierend auf Präferenz für X und Y (vereinfachte Darstellung)

vorliegenden Arbeit wurde dieser sogenannte `min_doc_count` auf 10 gesetzt.

7.4 Ausblick

Ein einfacher Recommender hat den Vorteil, dass er leicht zu verstehen und zu warten ist. Allerdings ist die Qualität der generierten Empfehlungen gegebenenfalls weniger hoch als bei einem komplexeren System. Die Abwägung zwischen Komplexität und Qualität muss daher für jedes Recommendersystem getroffen werden (vgl. hierzu z. B. [22]). Weiterführende Fragen, die im Rahmen der vorliegenden Arbeit nicht mehr geklärt werden konnten, sind deshalb einerseits, ob sich die Qualität durch eine Erweiterung des Algorithmus erhöhen lässt, andererseits auch, ob sich die Datenbasis durch die Hinzunahme zusätzlicher Informationen erweitern und verbessern lässt. Diese zusätzlichen Daten können einerseits die zu empfehlenden Artikel betreffen, indem zum Beispiel Inhalte oder von Nutzern vergebene Tags mit Hilfe der Significant-Terms-Aggregation ausgewertet werden. Andererseits könnten auch soziodemographische Nutzerdaten zur Bildung von Vordergrundmengen herangezogen oder Artikel von der Empfehlung ausgeschlossen werden, wenn sie nicht nur

signifikant häufig in den Positivlisten von Nutzern mit ähnlichem Geschmack auftauchen, sondern auch zu den signifikanten Termen gehören, die man erhält, wenn man die „Negativliste“ des entsprechenden Nutzers untersucht.

Interessant wäre es außerdem zu untersuchen, wie sich die generierten Empfehlungen ändern, wenn das Signifikanzmaß der Significant-Terms-Aggregation anders gewählt wird. Die dokumentierten Ergebnisse wurden alle mit dem *JLH*-Score berechnet. Wie in Abschnitt 3.3.2 erwähnt, können alternativ auch andere Maße gewählt werden.

Auch die Variation des `min_doc_count`-Parameters in den verwendeten Aggregationen könnte systematisch untersucht werden.

Elasticsearch als Technologie kann auch dort bei der Empfehlungsgenerierung von Nutzen sein, wo nicht auf Bordmittel der Suchmaschine, sondern auf externe Ansätze zurückgegriffen wird. In diesem Falle kann *Elasticsearch* als verteilter Datenspeicher verwendet werden, um ähnliche Artikel schnell auffindbar abzulegen. Die Zielsetzung ist in diesem Fall natürlich eine andere als in der vorliegenden Arbeit, wo es um die Generierung von Empfehlungen mittels *Elasticsearch* ging und nicht nur um die Nutzung desselben als Datencontainer. Unter [23] findet sich ein solcher Ansatz. Statt *Elasticsearch* wird *Solr* verwendet, die Umsetzung mit *Elasticsearch* würde aber analog funktionieren.

Literatur

[1] Robin Burke: *Hybrid Web Recommender Systems*. In: Brusilovsky, Peter, Alfred Kobsa und Wolfgang Nejdl (Herausgeber): *The Adaptive Web*, Seiten 377–408. Springer-Verlag, Berlin, Heidelberg, 2007, ISBN 978-3-540-72078-2. http://dl.acm.org/citation.cfm?id=1768197.1768211.

[2] Francesco Ricci, Lior Rokach, Bracha Shapira und Paul B. Kantor (Herausgeber): *Recommender Systems Handbook*. Springer, 2011. http://www.springerlink.com/content/978-0-387-85819-7.

[3] Joseph A. Konstan und Michael D. Ekstrand: *Introduction to Recommender Systems*. *Coursera* Onlinekurs, Videos abrufbar unter https://class.coursera.org/recsys-001/lecture/preview, Januar 2014.

[4] Dietmar Jannach, Markus Zanker, Alexander Felfernig und Gerhard Friedrich: *Recommender Systems: An Introduction*. Cambridge University Press, 2011, ISBN ISBN: 978-0-521-49336-9.

[5] Greg Linden, Brent Smith und Jeremy York: *Amazon.Com Recommendations: Item-to-Item Collaborative Filtering*. IEEE Internet Computing, 7(1):76–80, Januar 2003, ISSN 1089-7801. http://dx.doi.org/10.1109/MIC.2003.1167344.

[6] Derek Bridge, Mehmet H. Göker, Lorraine McGinty und Barry Smyth: *Case-based recommender systems*. The Knowledge Engineering Review, 20(3):315–320, 2006.

[7] Chong Wang und David M. Blei: *Collaborative Topic Modeling for Recommending Scientific Articles*. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, Seiten 448–456, New York, NY, USA, 2011. ACM, ISBN 978-1-4503-0813-7. http://doi.acm.org/10.1145/2020408.2020480.

[8] Gayatree Ganu, Yogesh Kakodkar und Amélie Marian: *Improving the Quality of Predictions Using Textual Information in Online User Reviews*. Inf. Syst.,

38(1):1–15, März 2013, ISSN 0306-4379. http://dx.doi.org/10.1016/j.is.2012.03.001.

[9] Martin Piotte und Martin Chabbert: *The Pragmatic Theory solution to the Netflix Grand Prize*. http://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf, August 2009.

[10] Andreas Töscher, Michael Jahrer und Robert M. Bell: *The BigChaos Solution to the Netflix Grand Prize*. Abrufbar unter http://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf, September 2009.

[11] Yehuda Koren: *The BellKor Solution to the Netflix Grand Prize*. Abrufbar unter http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf, August 2009.

[12] Robert M. Bell, Yehuda Koren und Chris Volinsky: *All together now: A perspective on the NETFLIX PRIZE*. CHANCE, 23(1):24–29, 2010, ISSN 0933-2480. http://dx.doi.org/10.1007/s00144-010-0005-2.

[13] Xavier Amatriain und Justin Basilico: *Netflix Recommendations: Beyond the 5 stars*. Online abrufbar unter http://techblog.netflix.com/2012/04/netflix-recommendations-beyond-5-stars.html, April 2012.

[14] Slobodan Vucetic und Zoran Obradovic: *A Regression-Based Approach for Scaling-Up Personalized Recommender Systems in E-Commerce, Workshop on Web Mining for E-Commerce*. Online abrufbar unter http://www.ist.temple.edu/~vucetic/documents/webmining00.ps, 2000.

[15] G. D. Linden, B. R. Smith und N. K. Zada: *Recommendations based on items viewed during a current browsing session*. Online abrufbar unter http://www.google.com/patents/US8620767, Dezember 2013. US Patent 8,620,767.

[16] Clinton Gormley und Zachary Tong: *Elasticsearch: The Definitive Guide*. Erhältlich bei O'Reilly oder online unter http://www.elasticsearch.org/guide/en/elasticsearch/guide/current/index.html, 2014.

[17] Christopher D. Manning, Prabhakar Raghavan und Hinrich Schütze: *An Introduction to Information Retrieval*. Cambridge University Press, 2009. http://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf.

[18] Thomas M. Cover und Joy A. Thomas: *Elements of Information Theory*. John Wiley & Sons, Inc., 1991, ISBN 0-471-06259-6.

[19] Rudi L. Cilibrasi und Paul M. B. Vitány: *The Google Similarity Distance*. IEEE Transactions on Knowledge and Data Engineering, 19(3):370–383, März 2007.

[20] Sebastiaan A. Terwijn, Leen Torenvliet und Paul M.B. Vitányi: *Nonapproximability of the normalized information distance*. Journal of Computer and System Sciences, 77(4):738 – 742, 2011, ISSN 0022-0000. `http://www.sciencedirect.com/science/article/pii/S0022000010001029`, {JCSS} {IEEE} {AINA} 2009.

[21] Britta Weber: *The Significant Terms Aggregation*. Präsentiert beim Treffen der Schweizer *Elasticsearch* User Group. Abrufbar unter: `https://speakerdeck.com/elasticsearch/the-significant-terms-aggregation`, Juni 2014.

[22] Ted Dunning und Ellen Friedman: *Practical Machine Learning. Innovations in Recommendation*. O'Reilly Media, Inc., 1. Auflage, Januar 2014, ISBN ISBN: 978-1-491-95038-8.

[23] Apache Mahout: *Intro to Cooccurrence Recommenders with Spark*. Online abrufbar unter `http://mahout.apache.org/users/recommender/intro-cooccurrence-spark.html`.

Software

[24] Elasticsearch: *Suchmaschine mit Analysefunktionen auf Basis von Lucene.* Download und Dokumentation: `http://www.elasticsearch.org`, Quellcode auf github unter `https://github.com/elasticsearch/elasticsearch`.

[25] Lucene: *Java-Bibliothek mit Methoden zur Volltextsuche.* Download und Dokumentation unter `http://lucene.apache.org/core/`.

[26] Elasticsearch: *Vollständige API-Dokumentation.* Abrufbar im Internet unter `http://www.elasticsearch.org/guide/en/elasticsearch/reference/1.4/index.html`, 2014.

[27] Solr: *Suchmaschine auf Basis von Lucene.* Download und Dokumentation unter `http://lucene.apache.org/solr/`.

[28] Lucene: *Dokumentation zur Scoring-Funktion von Lucene.* Online abrufbar unter `https://lucene.apache.org/core/4_3_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`, November 2014.

[29] Sugaya, Shinsuke: *Elasticsearch-Taste.* Sourcecode auf github unter `https://github.com/codelibs/elasticsearch-taste`.

[30] Mahout: *Bibliothek innerhalb des Apache Mahout-Projektes für nicht-verteilte, nicht-Hadoop-basierte Recommendations. Ursprünglich eigenständiges Projekt namens „Taste“.* Dokumentation unter `https://mahout.apache.org/users/recommender/recommender-documentation.html`.

[31] GroupLens: *MovieLens Datensätze mit Filmbewertungen.* Online abrufbar unter `http://grouplens.org/datasets/movielens/`.