INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Vergleich von Recommender Systemen

Marc Feger

Bachelorarbeit

Beginn der Arbeit:	20. September 2018
Abgabe der Arbeit:	20. Dezember 2018
Gutachter:	Prof. Dr. Stefan Conrad
	Prof. Dr. Stefan Harmeling

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 20. Dezember 2018

_____

Marc Feger

Zusammenfassung

Der heutige Einsatz von *Recommender Systemen* findet einen vermehrten und doch unbewussten Zugang in das alltägliche Leben. Immer mehr Lebensbereiche unterliegen somit einer stetigen Optimierung. Unternehmen wie *Amazon*, *Netflix* und *YouTube* passen ihre Produktvorschläge auf die einzelnen individuellen Wünsche ihrer Kunden an. Damit dies ermöglicht werden kann, kommen die verschiedenen Verfahren der *collaborative-filtering* und *content-based Recommender Systeme* zum Einsatz. Diese Bachelorarbeit beinhaltet den Vergleich der gemeinsam verwendeten Kategorien. Hierfür wurden drei verschiedene Datensätze aus dem Bereich der Musik- und Filmindustrie verwendet und analysiert. Um einen größeren Erkenntnisgewinn über die invarianten Algorithmen zu erzielen, wurde sich anders als im aktuellen Forschungsstand nicht nur auf einen der *MovieLens* Datensätze konzentriert, sondern vielmehr auch auf den *Webscope-R3* Datensatz von *Yahoo!-Research*.

Zunächst stand die Formalisierung des *Recommender-Problems* und der in dieser Bachelorarbeit verwendeten Algorithmen und Methoden im Fokus. Anschließend konnten bereits bestehende Evaluationsverfahren aus anderen Bereichen der Informatik so angepasst werden, dass diese zur Auswertung von *Recommender System* spezifischen Algorithmen genutzt werden können.
Die anschließende Implementierung trug dazu bei, dass bereits bestehende Verfahren wie *NDPM* verbessert werden konnten. Zudem wurde das einzige Entwicklungstool für entsprechende Messverfahren und Algorithmen erweitert, indem in die *Surprise-Library* die Fehlermessung *Mean-Squared-Error* und die Distanzfunktion *Spearman-Rank-Correlation* integriert wurden.

Die Analyse der *neighborhood-based* Verfahren ergab, dass keine Datensatz übergreifende Größe für eine optimale Nachbarschaft existiert. Dementsprechend ist die Größe der optimalen Nachbarschaften stets vom *User* zu *Item* Verhältnis des betrachteten Datensatzes abhängig. Diese Erkenntnis steht dem derzeitigen Wissensstand um solche Verfahren entgegen.

Im direkten Vergleich der *neighborhood-based* und *matrix-factorization* Verfahren stellte sich heraus, dass die *matrix-factorization* Verfahren geringfügig besser abschnitten. Unter wechselnder Verteilung der Eingabe zeigte sich, dass *neighborhood-based* Verfahren ein stetig gleiches Verhalten aufwiesen. Aufgrund der wechselnden Verteilung der Eingabe konnte bei dem *matrix-factorization* Verfahren *Funk-SVD* eine Verschlechterung aller Werte beobachtet werden. Diesem Hintergrund entsprechend bietet es sich an, *neighborhood-based* Verfahren aufgrund ihres soliden Verhaltens weiterzuentwickeln. Die kontinuierliche Veränderung des Datensatzes entspricht eher einer im Alltag befindlichen Situation. Demnach ist ein solides Verhalten der Algorithmen wünschenswert. Zukünftige Algorithmen für *Recommender Systeme* sollten daher als Grundlage *neighborhood-based* Verfahren verwenden. Mischformen mit *matrix-factorization* Verfahren sollten hiervon nicht ausgeschlossen werden.

Inhaltsverzeichnis

1 Recommender Systeme

Recommender Systeme (RS) wurden das erste Mal von Karlgren (1990) in seinem Paper: *An Algebra for Recommendations* als ein intelligentes Bücherregal vorgestellt. Dieses intelligente System sollte es dem Leser ermöglichen eine auf sich abgestimmte Büchersammlung zu erhalten. Trotz der fortschrittlichen Idee wurde Karlgrens Paper 1990 von Gutachtern des INTERACT-Komitees mit der Begründung, dass eine solche Art von System zu stark in die Privatsphäre und Integrität des Benutzers eingreifen würde, abgewiesen (Karlgren, 2017). Heute, knappe 30 Jahre nach Erscheinen des ersten Papers zu dem Thema *Recommender Systeme*, sind diese nicht mehr aus unserem Alltag wegzudenken. Immer mehr Unternehmen wie beispielsweise *Amazon*, *Netflix* und *YouTube* setzen erfolgreich das von Kalgren vorgestellte Konzept ein (Linden et al., 2003; Davidson et al., 2010; Gomez-Uribe und Hunt, 2015).

Im Nachfolgenden werden die zwei häufigsten Techniken der *Recommender Systeme* erläutert. Hierbei handelt es sich um das *collaborative-filtering System (CF)* und das *content-based System (CB)*. Dieses Kapitel soll dabei über Unterschiede und Zusammenhänge dieser beiden Techniken unterrichten. Vorrangig werden jedoch die Ansätze des *collborative-filtering Systems (CF)* dargestellt.

1.1 Formale Definition

Das *Recommender-Problem* besteht aus den Eingaben der Mengen $\mathcal{U}$ und $\mathcal{I}$, wobei $\mathcal{U}$ die Menge aller *User* und $\mathcal{I}$ die Menge aller *Items* abbildet. Jeder der *User* in $\mathcal{U}$ gibt *Ratings* aus einer Menge $\mathcal{S}$ an Möglichkeiten für die im System vorhandenen *Items* in $\mathcal{I}$ an. Die somit entstandene *Rating*-Matrix $\mathcal{R}$ setzt sich aus $\mathcal{R} = \mathcal{U} \times \mathcal{I}$ zusammen. Die Einträge in $\mathcal{R}$ geben die Bewertung von *User* u zu *Item* i an. Dieser Eintrag wird dann mit r_{ui} bezeichnet. Aufgrund unvollständiger *Item-Ratings* kann es dazu kommen, dass $\mathcal{R}$ ebenfalls unvollständig ist. Im weiteren Verlauf wird die Untermenge aller *User*, die ein bestimmtes *Item* i bewertet haben, als $\mathcal{U}_i$ bezeichnet. Analog bezeichnet $\mathcal{I}_u$ die Untermenge der *Items*, die von *User* u bewertet wurden. Ebenso bezeichnet $\mathcal{I}_{uv} = \mathcal{I}_u \cap \mathcal{I}_v$ die Untermenge der *Items* die sowohl von *User* u als auch von *User* v bewertet worden sind. Des Weiteren ergibt sich aus $\mathcal{U}_{ij} = \mathcal{U}_i \cap \mathcal{U}_j$ die Untermenge aller *User* die sowohl *Item* i als auch *Item* j bewertet haben. Da $\mathcal{R}$ nicht vollständig ausgefüllt ist, ergeben sich fehlende Werte für einige *User-Item* Relationen. Ziel des *Recommender Systems* ist es, die fehlenden *Ratings* $\hat{r}_{ui}$ durch eine *Prediction* $p(u, i)$ zu schätzen. Die *Prediction-Funktion* besteht aus $p : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{S}$ (Desrosiers und Karypis, 2011). Dabei ist $\mathcal{S}$ die Menge aller möglichen Bewertungen die ein *User* für ein *Item* abgeben kann.

Im weiteren Verlauf der Arbeit werden verschiedene Verfahren vorgestellt um $p(u, i)$ zu bestimmen. Insgesamt besteht das *Recommender-Problem* aus den folgenden drei *Basis-Schritten*:

1. *Inhalts-Analyse*
2. *Vorhersage der fehlenden Ratings*
3. *Vorschlagen von Objekten*

Der nachfolgende Algorithmus zeigt den strukturellen Ablauf des *Recommender-Problems* anhand der drei *Basis-Schritte*:

Algorithm 1: Recommender-Problem

Input:
Menge der *User* $\mathcal{U}$, Menge der *Items* $\mathcal{I}$, *Rating*-Matrix $\mathcal{R}$
Result: Vorschläge P

for *Inhalts-Analyse* **do**
  Reduziere die Eingabe auf die wichtigen Informationen
end
for *Vorhersage der fehlenden Ratings* **do**
  Erstelle eine Funktion $p : \mathcal{U} \times \mathcal{I} \rightarrow \mathcal{S}$
  Bestimme $\hat{r}_{ui} = p(u, i)$
end
for *Vorschlagen von Objekten* **do**
  Wähle Objekt/-e mit guter Bewertung
end

1.2 Content-based Recommender

Der folgende Abschnitt wird das Teilgebiet der Verfahren für *content-based Recommender Systeme* aufzeigen:

Content-based Recommender Systeme arbeiten in direkter Form mit *Feature-Vektoren*. Ein solcher *Feature-Vektor* kann beispielshalber ein *Benutzer-Profil* darstellen. Dieses Profil beinhaltet in diesem Fall Informationen über die Vorlieben wie *Genres, Autoren, usw.* Für *content-based Recommender* wird für je einen *User* lediglich ein spezifisches Modell trainiert. Somit können die Präferenzen des *Users* analysiert und die wichtigsten *Feature* extrahiert werden. Dabei beschreibt der *Feature-Vektor* x_u des betrachteten *Users* u die Präferenzen von u für *Items* in $\mathcal{I}_u$. Jedes *Item* i aus $\mathcal{I}$ wird durch einen *Feature-Vektor* x_i repräsentiert, welcher sich aus den Eigenschaften des *Items* i zusammensetzt. Solch entstandene Eigenschaften können den bereits oben genannten Vorlieben durchaus ähneln. Die somit generierten *Feature-Vektoren* können im Anschluss miteinander verglichen werden. Zur Veranschaulichung des Ablaufes kann hierzu der sogenannte *Rocchio-Algorithmus* betrachtet werden. Dieser Algorithmus wurde das erste Mal von Rocchio (1971) vorgestellt.

1.2.1 Rocchio-Algorithmus

Der *Rochhio-Algorithmus* ist ein klassischer *content-based Recommender* Algorithmus der den *Feature-Vektor* x_u des *Users* u aus den mit r_{ui} gewichteten *Feature-Vektoren* x_i der *Items* in $\mathcal{I}_u$ zusammensetzt (Desrosiers und Karypis, 2011). Der *Feature-Vektor* x_u wird bestimmt durch:

$$x_u = \sum_{i \in \mathcal{I}_u} r_{ui} x_i \tag{1}$$

Der generierte Vektor x_u wird anschließend auf seine Ähnlichkeiten mit den *Featur-*

Vektoren x_i verglichen. Dabei werden unter anderem die Funktionen *Cosine-Similarity* und *Pearson-Correlation* eingesetzt. Die fehlenden *Ratings* $\hat{r}_{ui}$ können bestimmt werden, indem für jede mögliche Bewertung $r \in \mathcal{S}$ ein *Vektor* $x_u^{(r)}$ erzeugt wird. Dabei setzt sich $x_u^{(r)}$ aus dem Durchschnitt der *Feature-Vektoren* der *Items* i, die von u mit r bewertet wurden, zusammen.

$$x_u^{(r)} = \frac{1}{|\mathcal{I}_u^{(r)}|} \sum_{i \in \mathcal{I}_u^{(r)}} x_i \tag{2}$$

Der fehlende Wert $\hat{r}_{ui}$ wird dadurch bestimmt, dass über jedes mögliche *Rating* r aus $\mathcal{S}$ iteriert wird. Der so entstandene Vektor $x_u^{(r)}$ wird dann mit dem *Feature-Vektor* des betrachteten *Items* i verglichen. Gewählt wird das r, für das $x_u^{(r)}$ die größte Ähnlichkeit zu x_i aufweist.

$$\hat{r}_{ui} = r \in \mathcal{S} : sim(x_i, x_u^{(r)}) \text{ ist maximal} \tag{3}$$

1.2.2 Eigenschaften der content-based Recommender

Für *contend-based Recommender* lässt sich der bis dato verwendete Algorithmus 1 so abwandeln, dass mehrere Vor- und Nachteile verdeutlicht werden können. Der daraus resultierende Algorithmus sieht dann wie folgt aus:

Algorithm 2: Recommender-Problem content-based

Input: Menge der *User* $\mathcal{U}$, Menge der *Items* $\mathcal{I}$,
Menge der *Feature* $\mathcal{F}$, Aufrufender *User* u
Result: Vorschläge P

$x_u \leftarrow 0^{1 \times |\mathcal{I}|}$
for $i \in \mathcal{I}_u$ **do**
  $x_{u,idx(i)} \leftarrow 1$
end
$\mathcal{FS} \leftarrow 0^{|\mathcal{I}| \times |\mathcal{F}|}$
for $i \in \{0, ..., |\mathcal{I}| - 1\}$ **do**
  for $f \in \{0, ..., |\mathcal{F}| - 1\}$ **do**
    if $\mathcal{I}_i$ *has* F_f **then**
      $\mathcal{FS}_{i,f} \leftarrow 1$
    end
  end
end
$model \leftarrow train(x_u, \mathcal{FS})$
$\mathcal{P} \leftarrow \emptyset$
for $i \in \mathcal{I} \setminus \mathcal{I}_u$ **do**
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{model.p(u, i)\}$
end
return $\mathcal{P}$

Der Algorithmus 2 erhält als Eingabe die Mengen $\mathcal{U}$ und $\mathcal{I}$ sowie die Menge aller zur Verfügung stehenden *Feature* $\mathcal{F}$. Dabei müssen die Mengen hinreichend gut mit Informa-

tionen versehen worden sein. Wichtig für die Laufzeit und die späteren Ergebnisse des Algorithmus ist eine verlässliche *Pre-Process Phase*. Häufig kommt es vor, dass nicht jedes *Feature* in den betrachteten *Feature-Vektoren* tatsächlich zum *Informationsgehalt* der Objekte beiträgt und somit unnötige Rechenkapazitäten in Anspruch nimmt. Deshalb wird in den meisten Fällen der *Feature-Space*, also der Raum aller *Feature-Paare*, auf die gehaltvollsten *Feature* reduziert. Häufig wird dabei das Verfahren *Principal Component Analysis (PCA)* angewandt (Barber, 2012). Mit den so gewonnen *Featur-Vektoren* können die Daten anschließend mittels *neuronalen Netzen*, *Bayes-Klassifizierer*, *Rochhio-Algorithmus*, *Distance-Measure*, *SVD*, usw. analysiert werden. Im Anschluss einer Analyse werden noch fehlende *Ratings* generiert und passende Vorschläge abgegeben. Zudem kann der Entwickler zwischen den folgenden zwei Darstellungsformen der Vorschläge wählen:

1. *Best-Item-Recommendation*
2. *Top-N-Recommendation*

Ersteres wählt hierbei lediglich das beste Objekt aus; die zweite Darstellungsform ergänzt diese Liste um $N - 1$ weitere.

1.2.3 Vor- und Nachteile der content-based Recommender

Die nachfolgende Tabelle führt die bekanntesten Vor- und Nachteile der *content-based Recommender* gegeneinander auf:

Name	Beschreibung	Pro	Contra
Unabhängigkeit	Dank eines auf den User abgestimmten Modells kann eine Manipulation durch einen andern User nicht erfolgen	- Keine Fremdeinwirkung - Personalisierte Modelle - Kein Cold-Start Problem - Kein Long-Tail Problem	- New-User Problem - Hoher Rechenaufwand für das gesamte System - Serendipity/Novelty nicht vorhanden
Transparenz	Die Wahl der Methodik kann dazu beitragen, dass der User das System als BlackBox wahrnimmt	- Vertrauen des Nutzers	- Meist nur durch labeln der Daten möglich - Misstrauen des Nutzers - Der Nutzer kann das Verfahren nicht 100% verstehen
Implementierung	Die Implementierung der Verfahren	- Verfahren sind gut erforscht - Verfahren sind gut implementiert	- Anpassung an speziellere Probleme notwendig - Viele Parameter die optimiert werden müssen
Effizienz	Die Effizienz des Systems im Gebrauch	- Trainierte Modelle sind schnell	- Modelle können zur Laufzeit trainiert werden - Modelle können nicht zur Laufzeit trainiert werden - Modelle können zeitverzögert wirken - Hoher Aufwand neue Modelle zu trainieren

Tabelle 1: Vor- und Nachteile der *content-based Recommender*

Das schwerste Problem für die Entwicklung eines *content-based Recommenders* besteht darin, die zugrundeliegenden *User-Modelle* zu optimieren. Für groß angelegte Systeme, wie beispielsweise *Netflix*, bedeutet der Einsatz von *content-based Recommendern* die Wahl zwischen hohem Rechenaufwand, um die *Modelle* zu trainieren, oder dem Problem, dass die *Modelle* nicht zur Laufzeit optimiert sind und somit die Vorschläge nicht aktuell genug an den Benutzer weitergegeben werden. Ungeachtet dessen unterliegt dieses System noch einem weiteren Problem: der *User* möchte nicht immer wieder die gleichen Vorschläge präsentiert bekommen. Vielmehr müssen *content-based Recommender* auch die Möglichkeit besitzen dem *User* unbekannte Objekte zu empfehlen die für ihn neu, jedoch nicht müßig oder gar nutzlos erscheinen. Dieser Aspekt wird als *Serendipity* bezeichnet.

Ein kurzes Beispiel: Ein *User* begibt sich zumeist in ein chinesisches Restaurant. Die ihm dort angebotenen Speisen sagen ihm stets zu. Nun sehnt sich dieser jedoch nach Veränderung und nutzt einen *content-based Recommender* um neue Vorschläge zu erhalten. Dieses System weist ihn auf ein japanisches Restaurant um die Ecke hin. Dort angekommen überrascht das Essen, wird jedoch eben so gerne verspiesen, wie das in dem alt bekannten Restaurant.
Somit vertraut der *User* dem System mehr, wenn es ihm auch neue und noch unbekannte Empfehlungen ausspricht, die von seinen normalen Vorlieben abweichen, ihm aber nicht fremd erscheinen. Die bekannten *Machine-Learning* Verfahren wie beispielsweise *Neuronale-Netze* und *PCA* arbeiten dem entgegen, indem sie versuchen, die relevanten *Feature* zu extrahieren. Somit können keine Objekte erfasst werden, die von der Präferenz des *Users* abweichen. Durch diese expliziten *User*-spezifisch ausgelegten Vorschläge kann es passieren, dass der Empfänger dank fehlender oder unbekannter Objekt-Vorschläge lediglich ein minderwertiges Vertrauen gegenüber dem System entwickelt. Dem entgegenwirkend existieren bereits seit einigen Jahren inkludierte Verfahren, welche eine bestimmt ausgelegte Zufälligkeit in die Auswahl der Vorschläge mit einbindet. Diese *Serendipity* kann somit stets auf einem hohen Niveau gehalten werden (Sheth und Maes, 1993). Zudem leiden *content-based Recommender* unter dem sogenannten *New-User* Problem. Das *New-User* Problem tritt dann auf, wenn ein neuer *User* für das System ersichtlich wird, aber noch keine Daten zur Modellierung des *User-Profils* vorliegen (Rubens et al., 2011).

Dennoch verfügen *content-based Recommender* auch über eine Vielzahl an positiven Aspekten. So stellt das sogenannte *Long-Tail* Problem keine Herausforderung für *content-based Recommender* dar. Hierbei werden einige *Items* so klassifiziert, dass diese in einen bestimmten Randgruppenbereich fallen und damit nur eine geringfügige Empfehlungsrate besitzen. Diese *Items* werden somit nur von wenigen *Usern* bedient und erhalten daher auch nur selten ein *Rating*. Da *content-based Recommender* nicht nur erhaltene *Ratings*, sondern vielmehr auch weitreichende Eigenschaften betrachten, wird dieses Problem bestmöglichst umgangen. Darüber hinaus können *content-based Recommender* dazu beitragen neue Trends zu entwickeln. Hierzu werden ebenso weniger bekannte *Items* zu den von einer großen Bandbreite an genutzter *Items* hinzugefügt (Bambini et al., 2011). Das *Cold-Start* Problem wird umgangen, indem die *Items* nicht nur nach ihren *Ratings*, sondern ebenso durch die verschiedensten Eigenschaften des *Feature-Vektors* beurteilt werden. Dabei kann davon ausgegangen werden, dass ein *Recommender-System* ohne hinreichend viele *Ratings* nicht in der Lage ist, seine Arbeit sinnvoll zu verrichten (Victor et al., 2011).

1.3 Collaborative-filtering Recommender

Anders als der *content-based Recommender* betrachtet der *collaborative-filtering Recommender* nicht nur einzelne *User* und *Feature-Vektoren*, sondern vielmehr eine gleichgesinnte Nachbarschaft des ursprünglichen *Users*. Fehlende *User-Ratings* können mittels dieser Nachbarschaft extrahiert und zu einer Gesamtheit vernetzt werden. Dabei wird angenommen, dass ein fehlendes *Rating* des betrachteten *Users* für ein unbekanntes *Item* i ähnlich des *Ratings* eines *Users* v ausfallen wird, sobald u und v einige *Items* ähnlich bewertet haben. Dabei bestimmt sich die Ähnlichkeit der *User* aus den gemeinschaftlichen *Ratings*. Insgesamt lässt sich ein *collaborative-filtering Recommender* in zwei Kategorien unterteilen:

1. *Neighborhood-based Recommender*
2. *Model-based Recommender*

Model-based Recommender arbeiten mit Lernalgorithmen. Diese Untergruppe der *collaborative-filtering Recommender* arbeitet mit vergleichbaren Modellen aus dem Bereich der *content-based Recommender*. Diese Modelle können beispielsweise aus lernfähigen Algorithmen zum Aufbau einer Nachbarschaft bestehen. Dem entgegen stehen die *neighborhood-based Recommender*. Diese Untergruppe arbeitet ohne die Fähigkeit der Abstraktion von Zusammenhängen. Vielmehr beschränken sich *neighborhood-based Recommender* auf gut abgestimmte iterative Verfahren wie *k-Nearest-Neighbors (k-NN)*.

Letztlich kann ihr Einsatzgebiet darüber Aufschluss geben, welche Kategorie tatsächlich für die Entwicklung des beabsichtigten *Recommender Systems* geeignet ist. Diese Entscheidung ist von dem spezifischen Anwendungsgebiet abhängig und kann nur bedingt festgelegt werden.

Der nachstehende Abschnitt soll über die Techniken der *Nachbarschafts-* und *Ähnlichkeits-*Bestimmung, sowie das *Normalisieren* von *Ratings* informieren. Zudem sollen Techniken der *model-based Recommender* vorgestellt werden, die sowohl im Bereich der *collaborative-filtering Recommender*, als auch im Bereich der *content-based Recommender* zum Einsatz kommen.

1.3.1 Neighborhood-based Recommender

Ein *neighborhood-based Recommender* kann aus zwei Problemstellungen betrachtet werden: Die erste und bekannteste Problemstellung ist die der sogenannten *user-based Prediction*. Dabei sollen die fehlenden *Ratings* eines betrachteten *Users* u aus seiner Nachbarschaft $\mathcal{N}_i(u)$ bestimmt werden. Hierbei wird die Annahme vertreten, dass gleichgesinnte *User* eine ähnliche Bewertung für unbekannte *Items* abgeben. Ein bekanntes *user-based-collaborative-filtering Recommender System* ist *MovieLens*. Die zweite Problemstellung ist die der sogenannten *item-based Prediction*. Diese Art der Problemstellung betrachtet die Nachbarschaft $\mathcal{N}_u(i)$ von *Items*. Als Grundlage hierfür verwendet der *User* einen auf ihn abgestimmten *Recommender*. Dieser *item-based Recommender* arbeitet mit der Annahme, dass der *User* u ein ihm unbekanntes *Item* i ähnlich den zuvor von ihm bewerteten *Items* beurteilen wird. Für die *neighborhood-based Recommender* kann ebenfalls der Algorithmus 1 aus Abschnitt 1.1 angewendet werden. Die Abwandlung kann aus dem unten stehenden Algorithmus 3 entnommen werden. Sowohl für *user-based-* als auch *item-based-Recommender* wird zur Bestimmung der *Nachbarschaft* $\mathcal{N}$ der sogenannte k*-NN* Algorithmus verwendet. An dieser Stelle muss bedacht werden, dass die dem *Recommender-System* spezifische Implementierung des k*-NN*-Algorithmus nicht mit dem k*-Means*-Algorithmus verwechselt werden darf. Im Kontext der *Recommender-Systeme* wählt der k*-NN*-Algorithmus die k Objekte mit der größten Ähnlichkeit aus und fasst diese unter dem Begriff einer Nachbarschaft zusammen. Die Ähnlichkeit zwischen den Objekten wird mittels der Distanzfunktionen wie beispielsweise der *Mean-Squared-Difference (MSD)*, der *Pearson-Correlation (PC)* oder der *Cosine-Similarity (CS)* bestimmt.

Algorithm 3: Recommender-Problem neighborhood-based

Input: Menge der *User* $\mathcal{U}$, Menge der *Items* $\mathcal{I}$,
Rating-Matrix $\mathcal{R}$, Distanzfunktion $d_{measure}$,
Anzahl der Nachbarn k, Aufrufender *User* u
Result: Vorschläge P

```
sim ← 𝟙^(|𝒰|×|ℐ|)
for i ∈ {1, ..., |𝒰| − 1} do
    for j ∈ {j, ..., |𝒰| − 1} do
        sim_{i,j} ← d_measure(u_i, u_j, ℛ)
        sim_{j,i} ← sim_{i,j}
    end
end
k-NN ← ∅
for i ∈ {0, ..., k − 1} do
    k-NN ← k-NN ∪{u_i ∈ 𝒰 \ {u}}
end
for û ∈ 𝒰 \ {u} do
    for ũ ∈ k-NN do
        if sim_{u,û} ≥ sim_{u,ũ} then
            k-NN ← k-NN \{ũ}
            k-NN ← k-NN ∪{û}
        end
    end
end
𝒫 ← ∅
for i ∈ ℐ \ ℐ_u do
    ℛ_{u,i} ← p(u, i)
    𝒫 ← 𝒫 ∪ {i}
end
return 𝒫
```

1.3.2 Distanzmessung

Der folgende Abschnitt soll über die am häufigsten verwendeten Methoden zur Bestimmung der Abstände informieren. Jede der nachfolgenden Funktionen kann als eine Methode für *user-based Recommender* angegeben werden. Diese Methoden können in *item-based*-fähige Funktionen mittels Mengenüberführung von $\mathcal{I}_{uv}$ zu $\mathcal{U}_{ij}$ umgewandelt werden. Die einfachste Methode zu Bestimmung der Ähnlichkeit ist die sogenannte *Mean-Squared-Difference (MSD)*. Diese Methode dient als eine Abwandlung des *Mean-Squared-Error (MSE)*.

$$MSD(u,v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} (r_{ui} - r_{vi})^2}{|\mathcal{I}_{uv}|} \tag{4}$$

Die *Mean-Squared-Difference* kann einen Wert zwischen 0 und n annehmen. Die 0 repräsentiert in diesem Fall einen sehr guten Wert und eine hohe Ähnlichkeit. Hingegen bedeutet eine hohe Abweichung eine ebenso große Unähnlichkeit.

Eine weitere häufig verwendete Funktion zur Bestimmung der Ähnlichkeit zwischen zwei Objekten ist die sogenannte *Cosine-Similarity (CS)*. Dabei bestimmt die *Cosine-Similarity* den Winkel zwischen den Vektoren x und y in einem *n-dimensionalen* Vektor-Raum $\mathbb{R}^n$.

$$\begin{aligned} cos(x, y) &= \frac{\sum_{i,j=1}^{n} x_i y_j}{\sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{j=1}^{n} y_j^2}} \\ &= \frac{\langle x, y \rangle}{\|x\| \|y\|} \end{aligned} \tag{5}$$

Bezogen auf *Recommender Systeme* wird ein *User* mittels *User-Vektor* $u \in \mathbb{R}^{|\mathcal{I}|}$ dargestellt. Diese Vektor gibt an, wie der Benutzer die *Items* $i \in \mathcal{I}$ bewertet hat. Dabei wird u_i und r_{ui} gleichgesetzt. Gesetz den Fall, dass r_{ui} nicht existiert, gilt folgendes: $u_i = 0$. Somit ergibt sich für *Recommender Systeme* die *Cosine-Similarity* aus:

$$\begin{aligned} CS(u, v) &= \frac{\sum_{i \in \mathcal{I}_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} r_{ui}^2} \sqrt{\sum_{j \in \mathcal{I}_{uv}} r_{vj}^2}} \\ &= cos(u, v) \end{aligned} \tag{6}$$

Die Werte der *Cosine-Similarity* liegen zwischen -1 und 1. Der Wert -1 gibt dabei eine absolute Unterschiedlichkeit an, wohingegen der Wert 1 eine absolute Ähnlichkeit oder Gleichheit angibt. Der Wert 0 wiederum zeigt die Orthogonalität der Vektoren an und kann als neutrale Beziehung zwischen den betrachteten Objekten verstanden werden. Sollte es wider Erwarten dazu kommen, dass der Nenner der cos-Funktion den Wert 0 annimmt, so ist $cos(x, y) = 0$. Dieser Sonderfall wurde von Hug (2017) bedacht und entsprechend in der *Surprise-Library* umgesetzt. Dieser Fall tritt genau dann ein, wenn eine der beiden Eingabegrößen x oder y dem Nullvektor entspricht. Bezogen auf das *Recommender-Problem* ist die neutrale Bewertung 0 durchaus sinnvoll, da die Eingabe eines Nullvektors mit dem Fehlen von *Ratings* gleichgesetzt werden kann. Demnach ist das Verhältnis zwischen einem *User* ohne *Ratings* und einem *User* mit *Ratings* stets neutral. Eine weitere bekannte Methode zur Bestimmung der Ähnlichkeit ist die sogenannte *Pearson-Correlation (PC)* als Invariante der *Cosine-Similarity*.

$$
\begin{aligned}
r(x,y) &= \frac{\sum_{i,j=1}^{n}(x_i-\overline{x})(y_j-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2}\sqrt{\sum_{j=1}^{n}(y_j-\overline{y})^2}} \\
&= \frac{\langle x-\overline{x}, y-\overline{y}\rangle}{\|x-\overline{x}\|\|y-\overline{y}\|} \\
&= cos(x-\overline{x}, y-\overline{y})
\end{aligned}
\tag{7}
$$

Die Besonderheit der *Pearson-Correlation* ist, dass die betrachteten Variablen x und y *z-standardisiert* wurden und somit einen Erwartungswert von 0 und eine Varianz von 1 aufweisen. Die *Pearson-Correlation* eignet sich somit für das Bestimmen des linearen Zusammenhangs zweier Variablen mit unterschiedlicher Streuung. Für *Recommender Systeme* ergibt sich daher die *Pearson-Correlation* aus:

$$
\begin{aligned}
PC(u,v) &= \frac{\sum_{i\in\mathcal{I}_{uv}}(r_{ui}-\overline{r}_u)(r_{vi}-\overline{r}_v)}{\sqrt{\sum_{i\in\mathcal{I}_{uv}}(r_{ui}-\overline{r}_u)^2}\sqrt{\sum_{j\in\mathcal{I}_{uv}}(r_{vj}-\overline{r}_v)^2}} \\
&= CS(u-\overline{u}, v-\overline{v})
\end{aligned}
\tag{8}
$$

Analog zur *Cosine-Similarity* gilt der Fall: Sollte der Nenner der *Pearson-Correlation* den Wert 0 annehmen, so ergibt sich $r(x,y) = 0$. Dieser Fall tritt im Gegensatz zur *Cosine-Similarity* genau dann auf, wenn jedes Element der Eingabegröße den gleichen Wert vorweist. Durch die kontinuierliche Angabe gleicher Werte in x oder y ergeben sich die entsprechenden Mittelwerte für $\overline{x}$ und $\overline{y}$. Dies hat zur Folge, dass die *Pearson-Correlation* der Eingabe eines Nullvektors in die *Cosine-Similarity* entspricht:
$r(x,y) = cos(x-\overline{x}, y-\overline{y}) = 0 \leftrightarrow \forall x_i \in x : x_i = \overline{x} \vee \forall y_i \in y : y_i = \overline{y}$. Die *Pearson-Correlation* bewertet das Verhältnis zweier Eingaben erst dann als neutral, wenn mindestens eine Größe durchgängig aus den selben Werten besteht. Bezogen auf das *Recommender-Problem* kann damit festgehalten werden, dass ein *User* mit durchgängig konstanten *Ratings* unter Verwendung der *Pearson-Correlation* eine ebenso neutrale Meinung vertritt, wie ein *User* der keine *Ratings* abgegeben hat. Es sollte daher vorweg bedacht werden, dass die *Cosine-Similarity* und die *Pearson-Correlation* durchaus unterschiedliche Aussagen über die Ähnlichkeit zweier Eingabegrößen produzieren. Als Beispiel seien dazu die Vektoren $x_1 = [1,1]^T$ und $x_2 = [5,5]^T$ gegeben. In diesem Beispiel sind nur Werte zwischen 0 und 5 möglich, wobei der Wert 0 das Fehlen eines *Ratings* darstellt. Betrachtet man lediglich die Richtung der Vektoren, so wären diese identisch. Bezogen auf das *Recommender-Problem* hätte dies zur Folge, dass die Meinung zweier *User* mit den Bewertungen x_1 und x_2 gleich ausfällt. Werden nur die Zahlenwerte betrachtet, so wären diese konträr zueinander. Werden hingegen die Distanzfunktionen *Cosine-Similarity* und *Pearson-Correlation* verwendet, so würde das Verhältnis der Vektoren als neutral eingestuft werden $r(x_1,x_2) = cos(x_1,x_2) = 0$. Wiederum würde die *Mean-Squard-Distance* die konträre Position bestätigen: $msd(x_1,x_2) = 16$. Dies liegt daran, dass die *Cosine-Similarity*

und die *Pearson-Correlation* die Winkel der Vektoren zueinander als Ähnlichkeitsmaß verwenden. Es wird jedoch nicht der relative Abstand der Zahlenwerte zueinander berücksichtigt. Daher muss vor Anwendung der *Pearson-Correaltion* und *Cosine-Similarity* darauf geachtet werden, wie ein entsprechendes Verhältnis zwischen zwei *Usern* definiert und bewertet werden soll.

1.3.3 Abschätzung der fehlenden Werte

Nachdem eine Nachbarschaft $N_i(u)$ für *User* u bestimmt worden ist, wird es für den *neighborhood-based Recommender* möglich, die fehlenden Werte $\hat{r}_{ui}$ zu bestimmen. Dieser Textabschnitt soll demnach über die in der Literatur (Desrosiers und Karypis, 2011; Wit, 2008) am häufigsten verwendeten *Prediction*-Funktionen aufklären und deren Unterschiede zueinander aufzeigen. Die nachfolgenden Funktionen sind für *user-based Recommender* definiert, können jedoch zu jeder Zeit in eine *Prediction*-Funktion für *item-based Recommender* überführt werden. Hierfür muss die Abänderung von $\mathcal{N}_i(u)$ zu $\mathcal{N}_u(i)$ erfolgen. Dabei bezeichnet $\mathcal{N}_i(u)$ die Untermenge der Nachbarschaft aller *User* die eine Ähnlichkeit zu u über das *Item* i aufweisen. Des Weiteren bezeichnet $\mathcal{N}_u(i)$ die Untermenge der von u bewerteten *Items*, welche eine Ähnlichkeit mit dem unbekannten *Item* i haben.

1.3.4 Average-Rating und Weighted-Average-Rating

Die einfachste und bekannteste Methode zur Bestimmung der fehlenden *Ratings* $\hat{r}_{ui}$ ist das sogenannte *Average-Rating*. Dabei wird der Durchschnitt aller *Ratings* in der Nachbarschaft $\mathcal{N}_i(u)$ als Schätzwert für $\hat{r}_{ui}$ genommen.

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in \mathcal{N}_i(u)} r_{vi}}{|\mathcal{N}_i(u)|} \tag{9}$$

Das *Average-Rating* gilt als ein sehr einfach zu implementierendes Verfahren. Ein *Rating* eines *Users* $v_1 \in \mathcal{N}_i(u)$ mit maximaler Distanz ist gleich bedeutend, wie das *Rating* eines *Users* $v_2 \in \mathcal{N}_i(u)$ mit minimaler Distanz zu u. *User* aus $\mathcal{N}_i(u)$ können eine Veränderung der *Prediction* vornehmen, wenngleich eine hohe oder geringfügige Ähnlichkeit zu u besteht. Sollten etwaige Ergebnisse mit einem negativen Outcome behaftet sein, so verliert der *User* unter Umständen das Vertrauen in das von ihm genutzte System. Um solchen potentiellen Ausreißern entgegen zu wirken wird häufig das sogenannte *Weighted-Average* Verfahren verwendet.

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{10}$$

Dabei betrachtet das *Weighted-Average* Verfahren den Durchschnitt der Nachbarschafts-*Ratings*. Die hierfür entscheidende Abhängigkeit ist folgende: Die Distanz w_{uv} von $v \in \mathcal{N}_i(u)$ zu u. Somit haben *User* mit einer großen Entfernung zu u weniger Einfluss auf die *Prediction* als *User* die einer geringen Entfernung zu u unterliegen. Zudem sollte bedacht

werden, dass je nach Wahl der *Distanzfunktion* der Wert von $\hat{r}_{ui}$ außerhalb von $\mathcal{S}$ liegen kann. Sollte $\hat{r}_{ui}$ nicht in $\mathcal{S}$ liegen, so wird $\hat{r}_{ui} = min(\mathcal{S})$ falls $\hat{r}_{ui} < min(\mathcal{S})$ und $\hat{r}_{ui} = max(\mathcal{S})$ falls $\hat{r}_{ui} > max(\mathcal{S})$.

1.3.5 Normalized-Rating

Das in Abschnitt 1.3.4 vorgestellte *Weighted-Average* Verfahren eignet sich für die *User*-spezifische Einflussnahme auf die *Prediction*. Trotzdem vernachlässigt *Weighted-Average* die Tatsache, dass nicht jeder *User* einem einheitlichen Bewertungsschema folgt. Demnach ist es anzunehmen, dass einige *User* kritischere *Ratings* abgeben als andere. Bei den von den *Usern* beurteilten Bewertungsschemata kann somit eine entsprechend große Streuung zwischen sehr negativen und sehr positiven Angaben erfolgen. Um solchen Unterschieden entgegenzuwirken werden die *Ratings* r_{vi} aus der Nachbarschaft $\mathcal{N}_i(u)$ durch eine h-Funktion normalisiert. Damit die *Prediction* die Urbildmenge an möglichen Bewertungen $\mathcal{S}$ nicht verlässt, wird die Umkehrfunktion h^{-1} von h auf $\hat{r}_{ui}$ angewendet. Dieser Schritt ist notwendig, da durch das Anwenden der h-Funktion der Zahlenbereich der Urbildmenge verlassen wird. Die fehlenden *Ratings* ergeben sich entsprechend dem angewandten Verfahren. Dies hat zur Folge, dass die resultierenden *Ratings* in einem anderen Wertebereich als die der *User* liegen. Dieser Wertebereich ist dem *User* nicht zugänglich. Um dieser unterschiedlichen Skalierung der *Ratings* entgegenzuwirken wird die Umkehrfunktion h^{-1} auf die durch h berechneten *Ratings* angewendet. Dadurch werden die generierten *Ratings* $\hat{r_{ui}}$ wieder in den selben Zahlenbereich transformiert in dem sich auch der *User* befindet. Entsprechend den zuvor genannten Kriterien ergibt sich die folgende Formel:

$$\hat{r}_{ui} = h^{-1}\left(\frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} h(r_{vi})}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}\right) \tag{11}$$

Auch können hier die für *user-based Recommender* definierten Funktionen für *item-based Recommender* überführt werden. Hier gilt es erneut $\mathcal{N}_i(u)$ durch $\mathcal{N}_u(i)$ zu ersetzen.

1.3.6 Mean-Centering

Ein Ansatz um die *Ratings* aus der Nachbarschaft zu normalisieren ist das sogenannte *Mean-Centering*. Dabei wird geprüft, ob ein *Rating* positiv oder negativ ausfällt und somit ober- oder unterhalb des Durchschnitts der Bewertungen aus der Nachbarschaft liegt. Dabei kann die h-Funktion aus Abschnitt 1.3.5 wie folgt angepasst werden:

$$\begin{aligned} \hat{r}_{ui} &= \overline{r}_u + \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv}(r_{vi} - \overline{r}_v)}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|} \\ &\text{mit } h(r_{ui}) = r_{ui} - \overline{r}_u \\ &\text{und } h^{-1}(r_{ui}) = r_{ui} + \overline{r}_u \end{aligned} \tag{12}$$

1.3.7 Z-Score-Normalization

Im Gegensatz zum *Mean-Centering*, welches die Abweichung der *Ratings* in der Nachbarschaft mit dem Durchschnitt der vergebenen *Ratings* in dieser betrachtet, werden bei der sogenannten *Z-Score-Normalization* auch die individuellen Streuungen der *Ratings* untersucht. An dieser Stelle kann das folgende Musterbeispiel betrachtet werden: Während *User* u_1 immer ein neutrales *Rating* vergibt, wählt *User* u_2 seine *Ratings* ganz beliebig aus. Somit wäre ein neutrales *Rating* von u_1 wahrscheinlicher als ein neutrales *Rating* von u_2. Für die Formel 11 ergibt sich mit der *Z-Score-Normalization*:

$$\begin{aligned} \hat{r}_{ui} &= \overline{r}_u + \sigma_u \left(\frac{\sum\limits_{v \in \mathcal{N}_i(u)} \frac{w_{uv}(r_{vi} - \overline{r}_v)}{\sigma_v}}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|} \right) \\ &\text{mit } h(r_{ui}) = \frac{r_{ui} - \overline{r}_u}{\sigma_u} \\ &\text{und } h^{-1}(r_{ui}) = \sigma_u r_{ui} + \overline{r}_u \end{aligned} \tag{13}$$

Die *Ratings* der einzelnen *User* werden dadurch vergleichbar, indem mit einem Vielfachen der Standardabweichung und nicht der originalen Maßeinheit gearbeitet wird. Die *Z-Score-Normalization* wird häufig als die Annäherung der Messdaten an eine Gleichverteilung missverstanden. Dieses Verfahren darf somit nicht als eine Veränderung der Verteilung der Eingabegröße verstanden werden. Die Verteilungen werden vielmehr miteinander verglichen, indem diese durch eine Veränderung der Mittelwerte und Standardabweichung gestreckt oder gestaucht werden. Dadurch werden die Eingabegrößen anhand ihrer zugrundeliegenden Verteilungen vergleichbar. Für die *Z-Score-Normalization* kann sinnbildlich der Begriff der *Autoskalierung* verwendet werden.

1.3.8 Matrix-factorization

Ähnlich wie die in Abschnitt 1.2 erwähnten *content-based Recommender* können auch *neighborhood-based Recommender* mit modellbasierten Verfahren wie *SVD* arbeiten. Ein entsprechendes Verfahren wurde bereits von Vozalis und Margaritis (2007) entwickelt. Der Unterschied von einem *content-based Recommender* zu einem *collaborative-filtering Recommender* liegt dann in der Wahl der *Dimensionsreduktion*. Hierbei nutzen *content-based Recommender* Verfahren wie *Principal Component Analysis (PCA)*, wohingegen *collaborative-filtering Recommender* Verfahren wie *k-Nearest Neighbours (k-NN)* nutzen.

1.3.9 Baseline-estimator

Neben den positiven, als auch negativen *User-Ratings* können ebenfalls *Genres* oder *Items* besser als andere bewertet werden. Dieses Problem ist unter dem Begriff *Long-Tail* zusammengefasst. Um diese *Seiteneffekte (bias)* zu berücksichtigen und tatsächlich relevante *Feature* zu extrahieren wird häufig der sogenannte *Baseline Estimator* verwendet (Koren und Bell, 2015):

$$\hat{r}_{ui} = \mu + b_u + b_i \tag{14}$$

Dabei gibt μ den Durchschnitt aller *Ratings*, sowie b_u respektive b_i, die Abweichung des *Users* u und des *Item* i von μ an. Um die fehlenden Parameter b_u und b_i zu schätzen, wird das folgende Minimierungsproblem mittels der *stochastic-gradient-decent*-Methode gelöst:

$$\begin{gathered}\min_{b_u,b_i} \sum_{(u,i)\in\mathcal{B}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(\sum_{u\in\mathcal{B}} b_u^2 + \sum_{i\in\mathcal{B}} b_i^2) \\ \text{mit } \mathcal{B} = \{(u,i)|r_{ui} \text{ ist bekannt}\}\end{gathered} \tag{15}$$

1.3.10 Funk-SVD

Bei der sogenannten *Single Value Decomposition (SVD)* nach Funk wird jedes *Item* i durch den Vektor $q_i \in \mathbb{R}^{1\times n}$ und jeder *User* u durch den Vektor $p_u \in \mathbb{R}^{1\times n}$ dargestellt. Dabei gibt $q_i^T p_u$ das Interesse von u zu i an. Ähnlich zu dem *Baseline Estimator* aus Abschnitt 1.3.9 berechnet sich $\hat{r}_{ui}$ aus:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{16}$$

Dabei werden die Parameter (b_u, b_i, p_u, q_i) durch das folgende Minimierungsproblem gelöst:

$$\begin{gathered}\min_{b_u,b_i,p_u,q_i} \sum_{(u,i)\in\mathcal{B}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2) \\ \text{mit } \mathcal{B} = \{(u,i)|r_{ui} \text{ ist bekannt}\}\end{gathered} \tag{17}$$

Ein einfacher Ansatz zur Lösung des Minimierungsproblems wurde von Simon Funk im Jahre 2006 während der *Netflix-Challenge* vorgestellt. Dieser zeigte mittels der *stochastic-gradient-decent*-Methode einen Optimierungsvorschlag des *collaborative-filtering Algorithmus* auf (Koren und Bell, 2015). Im Nachfolgenden ist diese Lösung beschrieben:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ q_i &\leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i) \\ p_u &\leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u) \\ &\text{mit } e_{ui} = r_{ui} - \hat{r}_{ui} \end{aligned} \tag{18}$$

Dabei ist der Parameter γ die *Lern-Rate* und λ der *Regulierungs-Parameter* zur Verhinderung von *Over-* oder *Underfitting*. Nach Funk (2006) sind diese Parameter frei wählbar. Dennoch optimierte Funk (2006) den von ihm vorgestellten Algorithmus im Rahmen der *Netflix-Challenge*. Dabei wurden die Parameter wie folgt festgelegt: $\gamma = 0.001$ und $\lambda = 0.02$. Nach Paterek (2007) können diese Parameter nur schwer optimiert werden und benötigen deshalb keinerlei Optimierung. Die Implementierung des *Funk-SVD*

durch Hug (2017) setzt die beiden Parameter auf: $\gamma = 0.005$ und $\lambda = 0.02$. Dieser setzt auch die optimale Anzahl der Lernepochen auf 20. Im weiteren Verlauf dieser Bachelorarbeit wird sich jedoch zeigen, dass die Parameter durch die Anwendung des *grid-search*-Verfahrens in der Nähe der von Funk (2006) festgelegten Werte für γ und λ optimierbar sind. Eine entsprechende *SGD*-Implementierung kann aus dem folgenden Algorithmus 4 entnommen werden:

Algorithm 4: SGD Funk-SVD

Input: Anzahl der zu trainierenden Epochen n_{epochs}, Menge der *User* $\mathcal{U}$, Menge der *Items* $\mathcal{I}$, Menge der *Feature* $\mathcal{F}$, *Rating*-Matrix $\mathcal{R}$, Lern-Rate γ, Regulierungs-Parameter λ
Result: b_u, b_i, p_u, q_i

$b_u \leftarrow 0^{1\times|\mathcal{U}|}$
$b_i \leftarrow 0^{1\times|\mathcal{I}|}$
$p_u \leftarrow \mathcal{N}(\mu, \sigma^2)^{|\mathcal{U}|\times|\mathcal{F}|}$
$q_i \leftarrow \mathcal{N}(\mu, \sigma^2)^{|\mathcal{I}|\times|\mathcal{F}|}$
$\mu \leftarrow \overline{\mathcal{R}}_{Train}$
for $epoch \in \{0, ..., n_{epochs} - 1\}$ **do**
  for $(u, i) \in \mathcal{R}_{Train}$ **do**
    $sim \leftarrow 0$
    for $f \in \{0, ..., |\mathcal{F}| - 1\}$ **do**
      $sim \leftarrow sim + q_i[i, f] \cdot p_u[u, f]$
    end
    $err \leftarrow r_{ui} - (\mu + b_u[u] + b_i[i] + sim)$
    $b_u[u] \leftarrow b_u[u] + \gamma(err - \lambda b_u[u])$
    $b_i[i] \leftarrow b_i[i] + \gamma(err - \lambda b_i[i])$
    for $f \in \{0, ..., |\mathcal{F}| - 1\}$ **do**
      $p_u[u, f] \leftarrow p_u[u, f] + \gamma(err \cdot q_i[i, f] - \lambda p_u[u, f])$
      $q_i[i, f] \leftarrow q_i[i, f] + \gamma(err \cdot p_u[u, f] - \lambda q_i[i, f])$
    end
  end
end
return b_u, b_i, p_u, q_i

1.3.11 Vor- und Nachteile der collaborative-filtering Recommender

Ein wesentlicher Vorteil des *collaborative-filtering Recommenders* ist es, die Beziehungen zwischen *Usern* und den entsprechenden *Items* aufzudecken. Diese Beziehungen gehen hierbei nicht offensichtlich aus den *Feature* der *Items* hervor. Des Weiteren findet ein Austausch zwischen den *Usern* großer Mengen mittels dem *collaborative-filtering Recommenders* statt. Der Informationsaustausch erfolgt ohne eine vorherige Beziehung zwischen den einzelnen Usern *(Virtual Community)*. Dadurch werden ebenfalls *Items* betrachtet die von der bisherigen Präferenz eines *Users* abweichen *(Novelty/-Serendipity)*, da der *Prediction*-Prozess eine stetige Veränderung der Nachbarschaften mit sich führt. Je länger ein solches *collaborative-filtering Recommender System* im Einsatz ist, desto genauer wird die

Treffsicherheit während der Bildung einer Nachbarschaft und der daraus resultierenden *Prediction*. Dies kann darauf zurückgeführt werden, dass die *Rating-Matrix* $\mathcal{R}$ mit fortschreitender Zeit- und *User*-Aktivität vervollständigt wird. Damit erhöht sich die Wahrscheinlichkeit eine solide Nachbarschaft aus gleichgesinnten *Usern* mit ausreichend vielen *Ratings* zu generieren. Dennoch muss beachtet werden, dass fehlende Werte in $\mathcal{R}$ auch gleichwohl ein Problem für *collaborative-filtering Recommender* mit sich führen *(Sparsity/-Density)*. Es ist für *collaborative-filtering Recommender* grundlegend erforderlich, dass eine Menge an *Usern* existiert von der jeder mindestens eine Bewertung abgegeben hat. Andernfalls werden *Items* vom System selbst nur schlecht oder gar nicht vernetzt. Dieses Problem wird als *Cold-Start* beziehungsweise als *Bootstrapping-Problem* bezeichnet. Hinzu kommt auch das sogenannte *Long-Tail Problem*. Hierbei werden weniger häufig bewertete *Items* nur selten oder gar nicht vorgeschlagen. Es resultiert somit ein generalisiertes Bewertungsproblem. Zudem werden Informationen, die für weitere Zusammenhänge verantwortlich sind, nicht betrachtet. Dadurch können nebst den korrekten auch falsch generierte Vorschläge den *User* erreichen. Der *User* kann durch solche Vorschläge das Vertrauen *(Trust)* in das System verlieren.

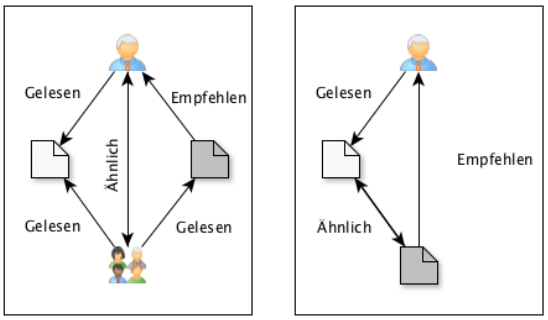
Name	Beschreibung	Pro	Contra
Abhängigkeit	Jeder User erhält Ratings aus seiner Nachbarschaft	- Erhöht die Serendipity - Erhöht die Novelty - Keine teuren Berechnungen nötig - Kein Domänen-Wissen nötig	- New-Item Problem (nicht bei item-based Recommendern) - New-User Problem - Sparsity - Cold-Start/Bootstrap-Problem - Long-Tail Problem
Transparenz	Die Wahl der Nachbarschaft von gleichgesinnten Usern	- Vertrauen des Nutzers durch Novelty - Nachbarschaften sind einsichtiger für User - Einfach zu berechnen - Einfach zu implementieren	- Schwer nachvollziehbar für fachfremde User - Folgen einer Änderung sind kaum abzusehen - Falsche Nachbarschaften verursachen Misstrauen des Users
Implementierung	Die Implementierung der Verfahren	- Verfahren sind gut erforscht - Verfahren sind gut implementiert - Verfahren sind einfach zu entwickeln - Verfahren sind kaum parametrisiert	- Speziellere Probleme bedürfen einer Parametrisierung - Parametrisierte Verfahren müssen einzeln untersucht werden
Effizienz	Die Effizienz des Systems im Gebrauch	- Einfache Verfahren - Es müssen keine Modelle gespeichert werden - Abwechslung durch stetige Anpassung ohne Modelle	- Scalability - Die Sparsity der R-Matrix führt zu fehlenden Werten - Rechenkapazität muss bereit gestellt werden

Tabelle 2: Vor- und Nachteile der *collaborative-filtering Recommender*

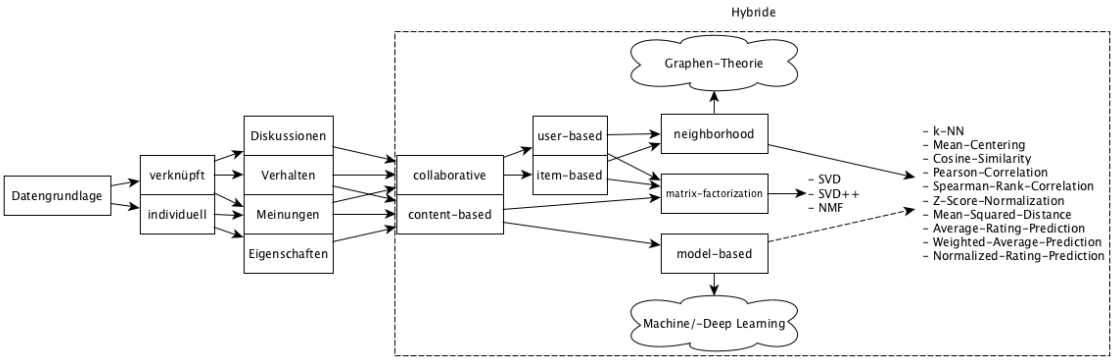
1.4 Résumé Recommender Systeme

Abschließend sollen die beiden Arten der *Recommender Systeme* miteinander verglichen werden. Dabei werden noch einmal die unterschiedlichen Betrachtungsweisen auf die entsprechenden Methoden herausgestellt. Darüber hinaus sollen die Zusammenhänge in der zugrundeliegenden Methodik verdeutlicht werden. Trotz unterschiedlicher Arbeitsweisen dieser Systeme wird im Nachfolgenden ebenso und vor allem über die Vielzahl gemeinsamer Methoden informiert.

Anders als herkömmliche Software-Systeme sind *Recommender Systeme* stark an den Kontext ihrer Anwendung gebunden. Dies hat zur Folge, dass es keine universell anwendbare Lösung für die Entwicklung eines solchen Systems gibt. Wer es sich zur Aufgabe macht ein solches *Recommender-System* zu entwickeln, der steht vor folgender Auswahl an Kategorien: *content-based* oder *collaborative-filtering*. Zusammengefasst bedeutet der Einsatz eines *collaborative-filtering Recommenders* nichts anderes als das Gruppieren von gleichgesinnten *Usern*. Dem entgegen steht der *content-based Recommender* der lediglich einzelne *User* und deren Vorlieben betrachtet. Diese Systematiken können den Abbildungen 1a und 1b entnommen werden:



(a) *Neighborhood* (b) *Content*



(c) Zusammenhänge von *content-based* und *collaborative-filtering* Verfahren

Abbildung 1: Übersicht der *Recommender Systeme*

Primär kann festgehalten werden, dass die beiden Arten der *Recommender Systeme* über ein Repertoire an gemeinsam genutzten Funktionen verfügen. Die Unterscheidung der beiden Arten ist demnach nicht ausschließlich durch die Betrachtung der verwendeten Methoden möglich. Vielmehr können die verwendeten Datengrundlagen tendenziell darüber entscheiden welche Art eines *Recommender Systems* verwendet wird. Unter der Annahme, dass *content-based* und *collaborative-filtering Recommender* keine Überschneidungen in ihrer Methodik aufweisen, wäre eine Kategorien-spezifische Unterscheidbarkeit möglich: *Collaborative-filtering Recommender* ordnen sich im Bereich der Graph-spezifischen Algorithmen ein. Die am häufigsten verwendeten Verfahren sind die Abwandlungen des *k-NN*. Im Fokus eines *collaborative-filtering Recommender* steht dann die Generierung fehlender Daten auf Grundlage von Netzwerken aus gleichgesinnten *Usern*. Im Kontrast dazu kann ein *content-based Recommender* als ein System aufgefasst werden, welches fehlende Werte einzig und allein auf den *Feature*-Informationen des aufrufenden *Users* generiert. Diese Art von *Recommender System* generiert demnach fehlende *Ratings* ohne die Einflussnahme von weiteren *Usern*. Es lässt sich daher nicht pauschal festhalten, dass es Arten-spezifische Anwendungsgebiete gibt. Jedoch ist die Art und Weise in welcher die fehlenden *Ratings* generiert werden maßgeblich für die Kategorie des *Recommender Systems* verantwortlich.

Eine Ebene darüber hinaus sollte die zugrundeliegende Methodik beider Sichtweisen betrachtet werden. Eine detaillierte Beschreibung der in Abbildung 1c gezeigten Methoden

kann aus den Abschnitten 1.2 und 1.3 entnommen werden. Sowohl für *content-based* als auch *collaborative-filtering Recommender* lassen sich spezifische Methoden-Gruppen identifizieren. Somit ist für *collaborative-filtering Recommender* die Gruppe der *neighborhood-based* Methoden zu ermitteln. Hierfür dient das Beispiel des k-*NN* Verfahrens. Zu ihnen können auch die Methoden aus der *Graphen-Theorie* gezählt werden. Solche *graph-based* Methoden wurden unter anderem von Desrosiers und Karypis (2011) beschrieben. Dem entgegen stehen die *model-based* Methoden für *content-based Recommender*. Diese erstrecken sich über das weite Gebiet *Machine-* und *Deep-Learning*. Dazu gehört unter anderem das Gebiet der *neuronalen-Netze* und *Klassifizierer*. Dennoch gibt es einige Überschneidungen in den Methoden. So teilen sich beide Systeme Methoden aus der Kategorie *matrix-factorization*. Zu diesen, in *Recommender Systemen* genutzten Methoden, zählen: *SVD*, *SVD++* und *NMF*. Auch können *content-based Recommender* mit *neighborhood-based*-spezifischen Verfahren arbeiten. Diese können zum Beispiel in *Pre-Process* Phasen genutzt werden. So können Abwandlungen des k-*NN* dazu genutzt werden ähnliche *Feature*-Vektoren miteinander zu gruppieren. Eine solche Phase hätte ähnlich zu der *PCA* eine Dimensionsreduktion innerhalb der Datenmenge zur Folge. Ein klassisches Beispiel hierfür ist der *Rocchio*-Algorithmus aus Abschnitt 1.2.1. Darüber hinaus können die Methoden aus der *neighborhood-based* Kategorie direkt in Nachbarschaftsmodelle aus dem Bereich der *content-based Recommender* überführt werden. Ein einfaches Beispiel hierfür ist der lernfähige k-*Means*-Algorithmus. Im Gegensatz zu k-*NN* bildet k-*Means* nicht die Nachbarschaft der k besten Objekte. Vielmehr werden eigenständig Nachbarschaften durch sogenannte *Cluster* erlernt.

Abschließend kann festgehalten werden, dass die beiden Arten der *Recommender Systeme* nicht eindeutig in der Methodik von einander unterscheidbar sind. Jedoch können sie anhand des Anwendungsfalls eindeutig einer der beiden Kategorien: *content-based* oder *collaborative-filtering* zugeordnet werden. Maßgeblich ist hierfür auch die Art der *Prediction*-Generierung. Für einen Vergleich der Vor- und Nachteile dieser Arten können die Tabellen 1 und 2 miteinander verglichen werden.

2 Evaluationsverfahren

Die Entwicklung eines *Recommender Systems* besteht nicht nur aus der perfekten mathematischen Modellierung von Algorithmen, sondern vielmehr auch aus deren Anwendbarkeit. Hierbei fordert das Einsatzgebiet dieser Systeme eine starke Interaktion mit dem jeweiligen *User*. Somit liegt nahe, dass die Evaluation von *Recommender Systemen* nicht ausschließlich mittels steriler Metriken erfolgen kann. Demnach darf der *User* als zentraler Faktor innerhalb der Entwicklung eines *Recommender Systems* zu keiner Zeit außer Acht gelassen werden. Es ist also nur wenig verwunderlich, dass zwei der insgesamt drei Entwicklungs- und Evaluations-Schritte den *User* als Indikator für die Güteklasse des Systems betrachten.

Im nachfolgenden Abschnitt werden diese drei Schritte beschrieben:

2.1 Analyse-Schritte

2.1.1 Offline-Analyse

Der erste Schritt in der Evaluations-Kette ist die sogenannte *Offline-Analyse*. Dieser Schritt beschäftigt sich einzig und allein mit dem Messen und Evaluieren der aus den Messdaten entwickelten Algorithmen. Dabei wird der *User* jedoch nicht mit in die Evaluation eingebunden. Dieser fundamentale Schritt erweist sich im Gegensatz zu den *User-Studien* und *Online-Analysen* als sehr kostengünstig. Als einzige Voraussetzung hierfür ist der Datensatz als solcher. Dieser Datensatz sollte dabei mindestens aus den folgenden Informationen bestehen:

| **ObjectID** | **UserID** | **Rating** | **Timestamp** |

Tabelle 3: Mindestanforderung an den Datensatz

Zusätzlich muss der zu analysierende Datensatz auf einen sogenannten Bias untersucht werden. Ein Bias kann sich beispielsweise in der Betrachtung von *Long-Tail-Diagrammen* oder *Rating*-Matrizen zeigen. Ein Datensatz wird als valide bezeichnet, wenn er mindestens die in Tabelle 3 gezeigten Schlüssel beinhaltet. Es kann jedoch auf den *Timestamp*-Schlüssel verzichtet werden. Dieser ist optional und ermöglicht eine detailliertere Analyse des Datensatzes. Sollte im Anschluss ein valider Datensatz vorliegen, so basiert die Evaluation auf den bekannten Methoden des *Machine-Learnings*. Hierfür wird der Algorithmus auf einen Trainingsdatensatz $\mathcal{R}_{Train}$ trainiert. Um anschließend die Ergebnisse des Erlernten zu vergleichen, kommt ein entsprechender Testdatensatz $\mathcal{R}_{Test}$ zum Einsatz. Dieser vergleicht somit die vom angelernten Modell generierten Daten. Um eine Evaluierung adäquat vernehmen zu können, wird zumeist die sogenannte *k-fold cross-validation* verwendet. Dabei wird der Datensatz in k gleich große Teile gespalten. Für eine 5-*fold cross-validation* wird also der gesamte Datensatz in 5 gleich große Teile zerlegt. Jeder Teil beinhaltet dann 20% der ursprünglichen Datenmenge. Das k-*fold cross-validation* Verfahren durchläuft dann in insgesamt k Iterationen jedes der k Teilstücke. In jedem Iterationsschritt wird einer der k Teile als Trainingsdatensatz $\mathcal{R}_{Train}$ verwendet. Anschließend werden die restlichen $k-1$ Teile als Testdatensatz $\mathcal{R}_{Test}$ verwendet. Dies wird für

jedes der k-möglichen Teilstücke wiederholt. Dadurch wird es dem Entwickler möglich bereits vor dem Einsatz des Systems die verschiedenen Algorithmen miteinander zu vergleichen. Hierbei lassen sich bereits erste Aussagen über die folgenden Kriterien treffen, beziehungsweise nicht treffen:

Möglich	**Unmöglich**
Information Retrieval	Trust
Accuracy	Serendipity
Performance	Diversity (teils)
Ranking	Novelty (teils)
Diversity (teils)	
Novelty (teils)	

Tabelle 4: Mögliche Messungen während der Offline Analyse

Somit muss vorab untersucht werden, welche Indikatoren relevant oder irrelevant für die Evaluation sind. Es darf ebenso nicht vernachlässigt werden, dass bestimme Indikatoren aus unterschiedlichen Phasen durchaus entgegengerichtet sein können. Als Beispiel können hier die Indikatoren *Accuracy* und *Diversity* betrachtet werden. Eine hohe *Accuracy* bedeutet zwangsläufig eine niedrige *Diversity*. Damit muss entschieden werden, welche Indikatoren tatsächlich zur Evaluation geeignet sind und welche in Phase zwei oder drei durch Tester untersucht werden sollten.

2.1.2 Benutzer-Studie

Als Weiterführung der *Offline-Analyse* kommen die sogenannten *Benutzer-Studien* zum Einsatz. Deren Ziel besteht darin, erste qualitative Daten für die quantitativen Ergebnisse aus den *Offline-Analysen* zu sammeln. Dieser Schritt kommt vor der eigentlichen Veröffentlichung des Systems. Hierbei kann demnach geprüft werden, wie die tatsächliche *User-System-Interaktion* ausfällt. Dazu werden *Tester* mit dem zu prüfenden System konfrontiert, indem sie eine bestimme Auswahl an Aufgaben mit diesem durchführen. Mittels dieser Methoden können die in der *Offline-Analyse* unmöglichen Messungen durchgeführt werden. Diese können aus Tabelle 4 entnommen werden. Für diese Studien sind nicht nur die auserwählten *Tester* von Relevanz, ebenso die für die Evaluation gewählten Aufgaben sind von enormer Bedeutung. Trotz des Ziels, mit der angewandten Methodik eine Verbesserung des Systems zu erreichen, muss eine entsprechende *Kosten-Nutzen-Abwägung* erfolgen, da insbesondere der oben beschriebene Evaluationsschritt einen enormen Kostenaufwand mit sich bringt.

2.1.3 Online-Analyse

Als abschließender Schritt wird das System auf Grundlage der in Schritt eins und zwei gewonnen Erkenntnisse veröffentlicht. Diese Erkenntnisse zählen zu den in der Evaluationskette aussagefähigsten Ergebnissen, da sie die tatsächlichen Effekte zwischen angewandter Methode und dem entsprechenden *User* aufzeigen. Dabei können die Indikatoren *Serendipity*, *Trust* und *Novelty* in Echtzeit erfasst werden. Des Weiteren ist diese Phase

der Evaluation für die Messung der *Scalability* interessant. Hierbei treffen echte Daten unkontrolliert auf das System, sodass diese zunächst noch verarbeitet werden müssen. Folgende Problematik ist hier ersichtlich: Der *User* muss in der Lage sein über die eingegangenen Daten ein entsprechendes *Feedback* zu vergeben. Anders als in den *Benutzer-Studien* ist dieses *Feedback* allerdings nur schwer zu erhalten. Somit wird festgehalten, dass *Recommender Systeme* innerhalb von drei Schritten evaluiert werden. Eine schematischer Ablauf kann aus Abbildung 2 entnommen werden.

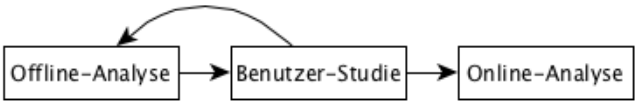


Abbildung 2: Evaluations-Schritte eines Recommender Systems

Der nachfolgende Textabschnitt befasst sich mit der *Offline-Analyse* und den damit verbundenen Messungen für die möglichen Aspekte aus Tabelle 4.

2.2 Accuracy Metriken

Wie bereits mehrfach erwähnt, kommen die verschiedensten Verfahren der *Recommender Systeme* aus den nachfolgenden Bereichen:

1. Machine-Learning
2. Regression
3. Neuronale-Netze
4. Matrix-factorization
5. Graphen-Theorie

Jedes dieser Verfahren muss für ein *Recommender System* entsprechende *Ratings* generieren können. Demzufolge müssen vom Verfahren generierte Vorschläge einen entsprechend hohen Datenbezug aufweisen. Dafür wird der betrachtete Datensatz in einen Trainingsdatensatz $\mathcal{R}_{Train}$ und einen Testdatensatz $\mathcal{R}_{Test}$ zerlegt. Der Trainingsdatensatz $\mathcal{R}_{Train}$ wird dabei für das Erlernen oder das Anwenden der *Prediction*-Funktion $p(u, i)$ verwendet. Anschließend wird das mit der Funktion Erlernte auf dem jeweiligen Testdatensatz $\mathcal{R}_{Test}$ geprüft. Untersucht wird dann der Fehler zwischen r_{ui} und $\hat{r}_{ui}$. Um etwaige Fehler ermessen zu können werden die folgenden drei Verfahren genutzt:

1. Mean Absolute Error (MAE)
2. Mean Squared Error (MSE)
3. Root Mean Squared Error (RMSE)

Als ein recht einfaches Verfahren wird der *Mean Absolute Error (MAE)* angesehen.

$$MAE = \frac{1}{\mathcal{R}_{Test}} \sum_{(u,i)\in\mathcal{R}_{Test}} |r_{ui} - \hat{r}_{ui}| \tag{19}$$

Diese Fehleranalyse unterliegt einem leichten Verständnis, da lediglich die absolute Distanz betrachtet wird. Dennoch weist auch dieses Verfahren ein Problem auf. Aufgrund der Betrachtung von absoluten Distanzen ist es demnach nicht ersichtlich aus welcher Richtung ein Fehler kommt.

Hier ein kurzes Beispiel: Ein *User* kauft erst dann ein *Item*, wenn es mit 3,5 oder weiteren Sternen bewertet worden ist. Nun kann folgender Fall betrachtet werden:

1. Fall: $r_{1,1}$ = 4 und $\hat{r}_{1,1} = 1$
2. Fall: $r_{1,1}$ = 4 und $\hat{r}_{1,1} = 7$

An dieser Stelle wäre in beiden Fällen der Fehler gleich zu bewerten, obwohl der zweite Fall einer eindeutig besseren Bewertung unterliegt. Diese Problemstellung wird von Carenini (2005) mit folgendem Lösungsvorschlag aufgegriffen: Um den Benutzer in das Zentrum des Fehlers zu stellen baut Carenini (2005) eine *user*-spezifische Schranke θ_i auf. Diese Schranke kann demnach angeben, ob ein *Item* für den jeweiligen *User* interessant sein könnte. Der Fehler berechnet sich dann wie folgt:

$$\begin{aligned} MUG &= \frac{1}{\mathcal{R}_{Test}} \sum_{(u,i)\in\mathcal{R}_{Test}} UG(u,i) \\ \text{mit } UG(u,i) &= \begin{cases} \hat{r}_{ui} - \theta_i & \text{wenn } \hat{r}_{ui} > \theta_i \\ \theta_i - \hat{r}_{ui} & \text{sonst} \end{cases} \end{aligned} \tag{20}$$

Trotz der *user*-berücksichtigenden Formel fällt es immer noch sehr schwer ein entsprechend gutes θ_i zu finden.

Des Weiteren wird ebenso die etwas weniger angewandte Methode *Mean Squared Error (MSE)* zur Fehlererkennung verwendet. Diese betrachtet die quadratische Abweichung zwischen r_{ui} und $\hat{r}_{ui}$. Um so größer die Distanz zwischen diesen beiden Werten ist, desto negativer fällt deren Bewertung aus.

$$MSE = \frac{1}{\mathcal{R}_{Test}} \sum_{(u,i)\in\mathcal{R}_{Test}} (r_{ui} - \hat{r}_{ui})^2 \tag{21}$$

Ebenso wie bei der vorherigen Methode liegt auch hier die Problemstellung bei der Richtungsbestimmung des Fehlers. Somit wird diese Methode überwiegend zur Betrachtung von Minimierungsproblemen verwendet.

Als dritte Methode zur Fehlerbetrachtung wird der *Root Mean Squared Error (RMSE)* verwendet. Diese Fehlermessung skaliert den *MSE* auf die gleiche Einheit der betrachteten Eingabe. Der *RMSE* ist somit leicht interpretierbar. Die Formel wird entsprechend aus der Wurzel des *MSE* generiert:

$$RMSE = \sqrt{MSE} \tag{22}$$

2.3 Information Retrieval

Neben der *Accuracy* werden häufig Algorithmen aus er Sicht des *Information Retrievals* betrachtet. Im Fokus steht hierbei die Frage, ob und als wie wichtig ein *User* die ihm vorgeschlagenen *Items* empfindet. Dem entsprechend ist es von enormer Relevanz zu prüfen, ob die Algorithmen, die zu dem *User* zugeordneten *Items* richtig klassifizieren. Die binäre Klassifikation eines Datensatzes gruppiert die vorgeschlagenen *Items* anhand der Eigenschaften *relevant* beziehungsweise *nicht relevant*. Hierfür müssen die Messdaten in die folgenden Kategorien unterteilt werden:

	Vorgeschlagen	Nicht Vorgeschlagen
Relevant	True Positiv (tp)	False Negativ (fn)
Nicht Relevant	False Positiv (fp)	True Negativ (tn)

Tabelle 5: Zerteilung des Datensatzes zur Berechnung von *Precision* und *Recall*

Aus der oben stehenden Tabelle 5 lassen sich somit die Werte für *Precision* und *Recall* bestimmen. An dieser Stelle erfolgt eine wichtige Anmerkung: Die *Precision* darf keineswegs mit der *Accuracy* verwechselt werden. Dieses Maß beschreibt das Verhältnis zwischen den ausgewählten *Items* die für den *User* relevant sein könnten und der Gesamtheit aller erwählten *Items*. Somit kann die *Precision* als ein Maß für die Güte der vorgeschlagenen Elemente genutzt werden. Die *Precision* ist folgendermaßen definiert:

$$\begin{aligned} Precision &= \frac{tp}{tp + fp} \\ &= \frac{\sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)(\hat{r}_{ui} \geq \theta_i)}{\sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)(\hat{r}_{ui} \geq \theta_i) + \sum\limits_{i \in \mathcal{P}} (r_{ui} < \theta_i)(\hat{r}_{ui} \geq \theta_i)} \\ &= \frac{\sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)(\hat{r}_{ui} \geq \theta_i)}{\sum\limits_{i \in \mathcal{P}} (\hat{r}_{ui} \geq \theta_i)} \end{aligned} \tag{23}$$

Entgegen dieser Formel steht der sogenannter *Recall*-Wert. Der *Recall* beschreibt das Verhältnis zwischen den vorgeschlagenen relevanten und den anderweitigen irrelevanten *Items*. Die entsprechende Formel ist dem nachfolgenden Abschnitt zu entnehmen:

$$
\begin{aligned}
Recall &= \frac{tp}{tp + fn} \\
&= \frac{\sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)(\hat{r}_{ui} \geq \theta_i)}{\sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)(\hat{r}_{ui} \geq \theta_i) + \sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)(\hat{r}_{ui} < \theta_i)} \\
&= \frac{\sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)(\hat{r}_{ui} \geq \theta_i)}{\sum\limits_{i \in \mathcal{P}} (r_{ui} \geq \theta_i)}
\end{aligned}
\tag{24}
$$

Der vom *Recall* generierte Wert kann daher als Kenngröße über die erreichte Bandbreite von Vorschlägen verstanden werden. Die aus den unterschiedlichen Verfahren entnommenen Kenngrößen stehen demnach für die jeweils unterschiedlichen Effekte. Trotz dieser Unterschiede kann folgendes Phänomen festgestellt werden: Eine Erhöhung der *Precision* hat beispielsweise eine Verminderung des *Recalls* zur Folge. Ebenso gilt die entgegengesetzte Richtung. Wie bereits zuvor von Carenini (2005) erwähnt, ist auch in diesem Fall nur erschwert eine entsprechende Schranke θ_i zu erwählen. Ob und in wie weit ein Objekt für den einzelnen *User* relevant ist, entscheidet immer noch der *User* selbst. Dennoch kann das individuelle θ_i mittels dem globalen Durchschnitt aller abgegebenen *Ratings* substituiert werden.

Um beide Kenngrößen zu kombinieren wird das $\mathcal{F}_1$-Maß betrachtet. Das $\mathcal{F}_1$-Maß erlaubt es zwei Gütemaße zusammenzufassen und ihre entsprechende Einflussnahme aufeinander zu untersuchen. Dieses Maß ist gegeben durch:

$$
\mathcal{F}_1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{25}
$$

Die Werte *Precision* und *Recall* können mittels dem sogenannten *Precision-Recall*-Graph analysiert werden. In diesem Graphen werden beide Werte gegeneinander aufgetragen. Eine Kurve im oberen rechten Rand bedeutet demnach auch entsprechend gute Werte, sowohl für die *Precision*, als auch für den *Recall*. Dem entgegengesetzt bedeutet eine Kurve im unteren linken Randbereich einen schlechten Algorithmus. Dennoch muss je nach Einsatzbereich des *Recommender Systems* die Wichtigkeit der Werte *Precision* und *Recall* separat betrachtet werden.

Darüber hinaus kann für den Vergleich von *Recommender-System*-Algorithmen zwischen drei Arten der *Precision-Recall*-Kurvendarstellung gewählt werden:

1. *Globaler-Precision-Recall (GPR)*: Diese Form der Darstellung eignet sich für die Betrachtung aller *User* in einem System. Dabei wird über alle *Precision*- und *Recall*-Werte der Durchschnitt gebildet, indem diese miteinander zu einer einzigen globalen Liste $\mathcal{G}$ vereint werden. Anschließend wird der sogenannte *Average-Precision (AP)*- und der *Average-Recall (AR)*-Wert ermittelt. Der AP ergibt sich dann aus: $AP = \frac{1}{|\mathcal{G}|} \sum_{u \in \mathcal{G}} precision(u)$. Der Wert für AR kann analog ermittelt werden, indem über den *Recall* von u in $\mathcal{G}$ summiert wird.

2. *Precision@k und Recall@k*: Bei dieser Darstellungsform wird die GPR-Kurve gebildet. Anders als im ersten Schritt werden an dieser Stelle jedoch nur einzelne der vorgeschlagenen *Items* betrachtet. Hierfür wird die Vorschlagsliste ab einem gewissen Indexpunkt k getrennt. Betrachtet werden somit ausschließlich die besten k Objekte. Diese Art der Darstellungsmethode bietet sich für den Vergleich von *Top-N Recommendern* an.

3. *Threshold-filtering*: Diese Art der Darstellung setzt sich aus den beiden obigen Methoden zusammen. Hierzu wird zunächst der *GPR* gebildet. Anders als zu dem klassischen *GPR* wird ein sogenannter *Threshold* festgelegt. Dieser *Threshold* kann zum Beispiel der Wert von k für die Größe einer Nachbarschaft sein. Auch kann der *Threshold* aus dem Wert θ_i bestehen. Für einen herkömmlichen *GPR* wäre θ_i mit dem Wert 0 versehen. Durch die Betrachtung der *Precision-* und *Recall*-Werte ist es möglich die Faktoren der *neighborhood-based* Verfahren genauer zu untersuchen. Eine Untersuchung der Algorithmen durch *threshold-filtering* ermöglicht es somit neben der Aussage über die Qualität, auch eine Aussage über die Größe der Vorschlagsliste $\mathcal{P}$ zu treffen.

2.4 Rank Accuracy Metriken

Als dritte und schwierigste Klasse der *Offline-Analyse-Methoden* gelten die *Rank Accuracy Metriken*. Das Ziel einer solchen Metrik besteht darin, den Algorithmus bezüglich seiner Elementanordnung innerhalb einer Vorschlagsliste zu überprüfen. Als Voraussetzung wird hierfür eine Referenzliste $\mathcal{R}_{ref}$ des *Users* benötigt. Eine solche Liste gibt an, ob und an welcher Stelle ein *User* das bevorzugte *Item* in einer Vorschlagsliste platzieren würde. Beide Listen werden anschließend miteinander verglichen. Diese *Evaluations-Metriken* sind somit aus folgendem Anlass nur schwer realisierbar: Zumeist liegt die benötigte Referenzliste für einen solchen Vergleich nicht vor. Nichtsdestoweniger existieren bereits Ansätze um solch künstliche Referenzlisten zu erzeugen. Ein Beispiel einer künstlichen Liste wäre ein Ausschnitt aus der Bestellliste eines *Users*. Doch auch dieser Ansatz unterliegt mehreren Schwierigkeiten. Zum einen wird angenommen, dass ein *Rating* für die Platzierung von *Items* in einer Liste ausschlaggebend ist und zum anderen wird vernachlässigt, dass ein einzelner *User* die Möglichkeit besitzt mehreren *Items* den selben Rang zuzuordnen. Eine künstlich erzeugte Referenzliste ist somit nicht in der Lage ausschlaggebende Ergebnisse zu liefern. Entweder der vorliegende Datensatz ist ausreichend, oder die folgenden Verfahren werden zur Durchführung einer *User-Studie* herangezogen:

Das erste Verfahren zu Analyse der Platzierung ist die *Spearman-Rank-Correlation*. Dieses Verfahren ist eine parameterlose Abwandlung der *Pearson-Correlation* und betrachtet nicht die *Ratings* und deren Verteilung, sondern vielmehr deren Rang (Agarwal und Chauhan, 2017). Somit ist die *Spearman-Rank-Correlation* gegenüber allen Ausreißern robust. Die *Spearman-Rank-Correlation* ist definiert durch:

$$\rho(x,y) = \frac{\sum_{i=1}^{n} \Big(rg(x_i) - \overline{rg}(x)\Big)\Big(rg(y_i) - \overline{rg}(y)\Big)}{\sqrt{\sum_{i=1}^{n} \Big(rg(x_i) - \overline{rg}(x)\Big)^2} \sqrt{\sum_{i=1}^{n} \Big(rg(y_i) - \overline{rg}(y)\Big)^2}} \tag{26}$$

Ein solcher Ansatz kann in dem von Hug (2017) entwickelten *Scikit-Library Surprise* nachvollzogen werden.

Im Rahmen dieser Bachelorarbeit wurde Formel 26 in Absprache mit Hug (2017) für die Anwendung mit *Recommender Systemen* entwickelt. Wenn die *Spearman-Rank-Correlation* dafür genutzt werden soll um Ähnlichkeiten zwischen Objekten zu bestimmen, so kann die nachfolgende Formel entsprechend verwendet werden:

$$\rho(u,v) = \frac{\sum_{i \in \mathcal{I}_{uv}} \Big(rg(r_{ui}) - \overline{rg}(r_u)\Big)\Big(rg(r_{vi}) - \overline{rg}(r_v)\Big)}{\sqrt{\sum_{i \in \mathcal{I}_{uv}} \Big(rg(r_{ui}) - \overline{rg}(r_u)\Big)^2} \sqrt{\sum_{i \in \mathcal{I}_{uv}} \Big(rg(r_{vi}) - \overline{rg}(r_v)\Big)^2}} \tag{27}$$

Bei einer solchen Benutzung dieser *Correlation* muss beachtet werden, dass die Formel auf die *user-* oder *item*-basierte Betrachtung des Algorithmus angepasst wird. Somit ist bei der Berechnung der *Spearman-Rank-Correlation* zu beachten, dass nur die gemeinsamen Elemente aus $\mathcal{U}_{ij}$ oder $\mathcal{I}_{uv}$ betrachtet werden dürfen. Um dies zu gewährleisten kann die allgemein gehaltene Formel 26 und 27 für zwei *User* konkretisiert werden:

$$\rho(u,v) = \frac{|\mathcal{I}_{uv}| \sum_{i \in \mathcal{I}_{uv}} rg(r_{ui}) rg(r_{vi}) - \sum_{i \in \mathcal{I}_{uv}} rg(r_{ui}) \sum_{i \in \mathcal{I}_{uv}} rg(r_{vi})}{\sqrt{|\mathcal{I}_{uv}| \sum_{i \in \mathcal{I}_{uv}} rg(r_{ui})^2 - \Big(\sum_{i \in \mathcal{I}_{uv}} rg(r_{ui})\Big)^2} \sqrt{|\mathcal{I}_{uv}| \sum_{i \in \mathcal{I}_{uv}} rg(r_{vi})^2 - \Big(\sum_{i \in \mathcal{I}_{uv}} rg(r_{vi})\Big)^2}} \tag{28}$$

Analog kann diese Formel auch für *item-based* Ansätze genutzt werden. Hierfür muss $\mathcal{I}_{uv}$ durch $\mathcal{U}_{ij}$ ersetzt werden.

Die *Spearman-Rank-Correlation* eignet sich besonders gut um Zufallsgrößen zu untersuchen die stark von einer Normalverteilung abweichen. Die Kernidee der *Spearman-Rank-Correlation* besteht darin, den Eingabewerten einen Rang zuzuordnen. Durch die Zuordnung von Werten zu Rängen entsteht eine Gleichverteilung. Diese Gleichverteilung deckt idealer Weise alle Werte zwischen 0 bis $n-1$ ab. Jedoch kann es auch passieren, dass eine Eingabe mehrere Elemente mit gleichen Werten beinhaltet. Diese Elemente müssen sich demnach einen Rang teilen. Eine standardmäßige Rangverteilung sieht keine Mehrfachbelegung eines Ranges vor. Dies liegt daran, dass die reine Rangzuweisung eine bijektive Abbildung ist. Um diesem Problem entgegen zu wirken werden die Ränge von Elementen mit gleichem Wert normiert. Sollte es zu dem Fall kommen, dass mehrere gleiche Werte vorliegen, so erhält jedes der Elemente zunächst einen eigenen Rang. Anschließend wird der Mittelwert der Ränge dieser gleichen Elemente berechnet. Die gleichen

Elemente erhalten den so ermittelten normierten Rang. Dies hat zur Folge, dass die interne Gleichverteilung der *Spearman-Rank-Correlation* Unregelmäßigkeiten aufweist. Dies bedeutet aber nicht, dass keine Gleichverteilung vorliegt, falls es zu Überschneidungen in den Werten kommt. Für die Berechnung der *Spearman-Rank-Correlation* in Bezug auf das *Recommender-Problem* ist es unumgänglich, dass die Ränge nur für die Elemente in $\mathcal{I}_{uv}$ beziehungsweise in $\mathcal{U}_{ij}$ der Eingabegrößen berechnet werden. Sollten wider Erwarten die Ränge über die gesamten Eingabegrößen errechnet werden, so kann es zu einer Verfälschung der tatsächlichen Ähnlichkeit kommen. Gemäß der *Pearson-Correlation* wird auch hier der Nenner der Rechenvorschrift 0, wenn die *Spearman-Rank-Correlation* Eingabegrößen mit durchgehend gleichen Werten erhält. Sollte dieser Fall auftreten, so wird $\rho(x, y) = 0$. Insgesamt können so die Werte der *Spearman-Rank-Correlation* im Wertebereich von -1 bis 1 liegen. Dabei gibt -1 eine absolute Unähnlichkeit und 1 eine absolute Ähnlichkeit der Eingabegrößen an.

Bei der Analyse von Objektplatzierungen ist ein weiterer Aspekt durchaus relevant: Um so tiefer ein *Item* in der Vorschlagsliste platziert ist, desto weniger Interesse zeigt ein *User* gegenüber diesem. Meistens werden diese *Items* beispielsweise in einer Art *Slider* oder Ähnlichem präsentiert. Die ersten Positionen eines solchen *Sliders* werden demnach von den, für den *User* interessanten, *Items* besetzt.

Um diese Problemstellung messbar zu machen vertritt Breese et al. (1998) die folgende These: Die Wahrscheinlichkeit, dass ein *User* ein entsprechendes *Item* interessant findet, steigt oder fällt exponentiell mit dessen Positionierung in der Vorschlagsliste $\mathcal{P}$. Hierfür stellt Breese et al. (1998) die folgende Formel zur Bestimmung der individuellen Relevanz auf:

$$\mathcal{R}_u = \sum_{i \in \mathcal{P}} \frac{utility(u, i)}{2^{\frac{idx(i)-1}{\alpha-1}}} \tag{29}$$

Hierbei ist α der sogenannte *half-life*-Parameter. Dieser Parameter gibt den Index des *Items* an, das eine 50%ige Wahrscheinlichkeit besitzt gesehen zu werden. Des Weiteren gibt idx den Index von i in $\mathcal{P}$ an. Breese et al. (1998) legt die $utility$-Funktion nicht fest, sondern lässt diese vielmehr frei wählbar und offen. Die gesamte Kenngröße ergibt sich dann aus der Betrachtung jedes *Users*.

$$\mathcal{R} = \frac{\sum\limits_{u \in \mathcal{U}} \mathcal{R}_u}{\sum\limits_{u \in \mathcal{U}} \mathcal{R}_u^{max}} \tag{30}$$

Die Summe des größt möglichen Nutzens wird mittels dem $\mathcal{R}_u^{max}$ dargestellt. Dies kann erreicht werden, indem die $utility$-Funktion für jedes *Item*, dass der *User* gesehen hat, einen Nützlichkeitswert angibt. Diese Werte können anschließend summiert werden.

Eine weitere Methode um die Platzierung der Elemente in einer Vorschlagsliste zu untersuchen ist die von Yao (1995) vorgestellte *Normalized Distance-Based Performance Measure (NDPM)*. Oft sind die betrachteten Listen partiell strukturiert. Dies ist erst dann der Fall, wenn es in der Liste mindestens ein *Item-Paar* gibt, welches sich eine gemeinsame Platzierung teilt. Dem entsprechend ist eine eindeutige Analyse dieser Vorschläge nicht

mehr gewährleistet. Um solch schwach strukturierte Listen miteinander vergleichen zu können stellt Yao (1995) das unten beschriebene Verfahren wie folgt vor:

$$NDPM = \frac{2C^- + C^u}{2C^i} \tag{31}$$

Die Vorschlags- und die Referenzlisten werden miteinander vergleichen, indem alle der $k = \frac{n(n-1)}{2}$ möglichen *Item*-Paarungen betrachtet werden. Ebenso wichtig ist für den Vergleich, dass in beiden Listen n viele Elemente vorhanden sind. Darüber hinaus beschreibt C^i die Anzahl der *Item*-Paare, für die in der Referenzliste eine eindeutige Struktur vorrausgesetzt wird. Zudem beschreibt C^- die Anzahl der Fehlstellungen in der Referenz- und Vorschlagsliste. Demgegenüber beschreibt C^+ die Anzahl der korrekt gebildeten Paare. Für den Parameter C^u gilt folgendes: An dieser Stelle werden die *Item*-Paare, welche in der Referenzliste einer fest zugeordneten Struktur unterliegen, gezählt. Diese gezählten Elemente besitzen jedoch keine Struktur in der Vorschlagsliste. Um die *Normalized Distance-Based Performance Measure (NDPM)* jedoch adäquat berechnen zu können, stellen Shani und Gunawardana (2011) eine entsprechende Rechenvorschrift zur Verfügung. Demnach sollte sich der Wert für *NDPM* aus $\frac{C^- + \frac{1}{2}C^u}{C^i}$ berechnen. Doch diese Vorschrift unterliegt einem Fehler: C^- berechnet nach Shani und Gunawardana (2011) nicht die Fehlstellungen, sondern vielmehr die negative Anzahl der richtigen *Item*-Paare. Im Rahmen dieser Bachelorarbeit wurde diese Rechenvorschrift verbessert und vereinfacht dargestellt. Damit es keine Überscheidungen mit denen von Yao (1995) festgelegten Variablennamen gibt, werden die folgenden Variablen mit einer Tilde gekennzeichnet. Diese entsprechen einer verbesserten Version der von Shani und Gunawardana (2011) erstellten Rechenvorschriften. Der Wert für *NDPM* kann dann wie folgt berechnet werden:

$$\begin{aligned}
\tilde{C}^- &= k + \sum_{i,j}^{k} sign(r_{ui} - r_{uj}) sign(\hat{r}_{uj} - \hat{r}_{ui}) \\
\tilde{C}^+ &= \sum_{i,j}^{k} sign(r_{ui} - r_{uj}) sign(\hat{r}_{ui} - \hat{r}_{uj}) \\
\tilde{C}^i &= \sum_{i,j}^{k} sign(r_{ui} - r_{uj})^2 \\
\tilde{C}^u &= \tilde{C}^i - (\tilde{C}^+ + \tilde{C}^-) \\
NDMP &= \frac{\tilde{C}^- + \frac{1}{2}\tilde{C}^u}{2\tilde{C}^i}
\end{aligned} \tag{32}$$

NDPM kann Werte zwischen 0 und 1 annehmen. Dabei gibt ein *NDPM*-Wert von 0 eine perfekt strukturierte Liste an. Der Wert 1 gibt an, dass die Vorschlagsliste in der invertierten Form der Referenzliste vorliegt. Sollten beide Listen keinerlei Zusammenhang aufweisen, so ist der Wert 0,5.

3 Datensatzanalyse

Zur Ausführung von *Offline-Analysen* werden vorhandene Datensätze als Grundlage vorausgesetzt. Die bekanntesten Herausgeber solcher Datensätze sind *Netflix*, *GroupLens*, *Jester*, *Book-Crossing* und viele weitere. Der nachfolgende Textabschnitt untersucht die bekanntesten Datensätze für *Recommender Systeme*. Dabei soll eine detaillierte Analyse der in dieser Arbeit verwendeten Datensätze erfolgen. Mittels dieser Analyse sollen anschließend die dafür hinreichenden Kriterien zur Auswahl eines Datensatzes erkannt werden. Diese Ergebnisse können ebenso für das Verständnis über das Verhalten solcher Algorithmen beitragen.

3.1 Struktur des Datensatzes

Es lässt sich wie bereits in Abschnitt 2.1.1 eine Mindestanforderung an den Datensatz stellen. Diese Mindestanforderungen sollten aus den folgenden Informationen bestehen: *userId*, *objectID*, *rating*, *timestamp*. Zumeist würde es jedoch ausreichen, einen Datensatz aus den beiden Schlüsseln *userId* und *objectId* aufzubauen. Diese Schlüssel können im Anschluss beispielsweise mit entsprechenden *Ratings* oder *Feature-Informationen* bestückt werden.

3.2 Aspekte einer Analyse

Da der Datensatz den Grundstein eines *Recommender Systems* darstellt, ist es umso wichtiger, diesen genauer zu untersuchen. Um aussagekräftige Analysepunkte ermitteln zu können, muss vor Beginn der Untersuchung noch einmal das *Recommender-Problem* aus Abschnitt 1.1 herangezogen werden. Im Mittelpunkt dieses Problems steht nach wie vor die Vorhersage von *Ratings* für die, dem *User*, noch unbekannten *Items*. Dieser Aspekt soll sowohl im *collaborative-filtering*, als auch im *content-based* Ansatz als Kernproblem verstanden und bestmöglich gelöst werden. Als erster Analysepunkt kann daher das Verhältnis zwischen den *Usern* und den zur Verfügung stehenden *Items* aufgelistet werden. Die Untersuchung dieses Merkmals kann unter Berücksichtigung folgender Begriffe stattfinden: *user-based* und *item-based Recommender*. Diese Merkmale können dahingehend von Bedeutung sein, als dass dieses Verhältnis maßgeblich am *Accuracy-Score* der Algorithmen beteiligt ist. Diese Problematik wurde bereits von Desrosiers und Karypis (2011) aufgegriffen. Aus Tabelle 6 können die Schätzfunktionen zur Bestimmung der durchschnittlichen Größe einer Nachbarschaft entnommen werden. Desrosiers und Karypis (2011) setzen für diese Schätzwerte voraus, dass die Anzahl der vergebenen *Ratings* für jeden *User* einer Gleichverteilung entspricht. Diese Gleichverteilung kann für einen Datensatz angenommen werden, wenn dieser eine Mindestanzahl an *Ratings* (*min.Ratings*) pro *User* voraussetzt. Zudem müssen die Parameter p und q ermittelt werden. Die durchschnittliche Anzahl von *Ratings* pro *Users* ist gegeben durch: $p = \frac{|\mathcal{R}|}{|U|}$. Auf der anderen Seite kann die Anzahl der durchschnittlich erhaltenen *Ratings* eines *Items* durch: $q = \frac{|\mathcal{R}|}{|I|}$ errechnet werden. Insgesamt lässt sich die Abschätzung dieser Nachbarschaft mittels der Darstellung des *Recommender-Problems* in Form eines bipartiten Graphen $k_{|U|,|I|}$ ermitteln.

Der bipartite Graph k ist definiert durch $k = \{\mathcal{U} \cup \mathcal{I}, R\}$. Hierbei ist $\mathcal{R}$ die Kantendarstellung $\{u, i\}$, wenn $u \in \mathcal{U}$ für $i \in \mathcal{I}$ ein *Rating* abgegeben hat. Um nun die Größe der durchschnittlichen Nachbarschaft eines *Users* zu bestimmen, muss die Wahrscheinlichkeit, dass zwei *User* mittels einer Kante über ein *Item* miteinander verbunden sind, gefunden werden. Für einen vollständigen bipartiten Graphen muss die Größe der maximalen Nachbarschaft eines *Users* $|\mathcal{U}| - 1$ sein. Die Wahrscheinlichkeit einer durch *Items* erzeugten Verbundenheit von *Usern* ergibt sich aus: $1 - \left(\frac{|\mathcal{I}|-p}{|\mathcal{I}|}\right)^p$.
Daher ergibt sich für die durchschnittliche Größe der Nachbarschaft eines *Users* die folgende Formel: $\big(|\mathcal{U}| - 1\big)\big(1 - \left(\frac{|\mathcal{I}|-p}{|\mathcal{I}|}\right)^p\big)$. Diese Formel kann analog für *item-based Recommender* angepasst werden. Es ist somit möglich, vorab sinnvolle Größen für solche Nachbarschafts-Algorithmen wie beispielsweise k-*NN* zu finden.

	Größe der Nachbarschaft
user-based	$\big(\lvert\mathcal{U}\rvert - 1\big)\left(1 - \left(\frac{\lvert\mathcal{I}\rvert-p}{\lvert\mathcal{I}\rvert}\right)^p\right)$
item-based	$\big(\lvert\mathcal{I}\rvert - 1\big)\left(1 - \left(\frac{\lvert\mathcal{U}\rvert-q}{\lvert\mathcal{U}\rvert}\right)^q\right)$

Tabelle 6: *User-* und *item-based* spezifische Schätzfunktionen der zu erwartenden Nachbarschaften

3.2.1 Daten Exploration

Des Weiteren sollte bei der Untersuchung eines Datensatzes die Verteilung der Informationsquellen betrachtet werden. Die Verteilung der zu betrachtenden Größen dient als Grundlage für die Wahl der *Prediction*-Funktion. Einige dieser Funktionen haben bestimmte Anforderungen an die Größenverteilung. Zu diesen Anforderungen zählt beispielsweise die *Pearson-Correlation*. Diese muss eine entsprechende Normalverteilung bei den zu betrachtenden Größen voraussetzen. Sollten jene Informationsgrößen, die stark von einer Normalverteilung abweichen betrachtet werden, so kann es durchaus sein, dass anstelle der *Pearson-Correlation* auf die sogenannte *Spearman-Rank-Correlation* zurückgegriffen werden muss. Um also interpretierbare Fehler wie *RMSE* oder *MAE* zu erhalten, ist die Untersuchung einer Verteilung von Informationsgrößen unumgänglich und von absoluter Notwendigkeit.

3.2.2 Gewichtung der Rating-Matrix

Ein weiterer Analyseschritt ist die Bestimmung der Dichte einer *Rating*-Matrix. Diese Dichte gibt an, wie viele der $|\mathcal{U}| \cdot |\mathcal{I}|$ möglichen *Ratings* tatsächlich angegeben wurden. Insgesamt lässt sich die Dichte des Datensatzes mittels: $\frac{|\mathcal{R}|}{|\mathcal{U}| \cdot |\mathcal{I}|}$ bestimmen. Die Dichte der *Rating*-Matrix ist neben dem Verhältnis zwischen den *Usern* und *Items* eine der wichtigsten Eigenschaften des Datensatzes. Wenn einmal die Verfahren wie *k-NN* und *SVD* betrachtet werden, so wird deutlich, dass es für diese Verfahren essentiell ist, wie dicht eine entsprechende *Rating*-Matrix ausfällt. Dies hat mehrere Auswirkungen auf die Algorithmen. So kann eine geringe Dichte als Herausforderung an einen Algorithmus angesehen werden. Diese Herausforderung kann eine Bewertung der Güte des Algorithmus bezüglich seiner *Performance* offenbaren. Da es bei einem Datensatz nicht unbedingt auf die Größe, sondern vielmehr auf die Dichte ankommt, kann durchaus auf einen kleineren Datensatz zurückgegriffen werden. Hierbei ist es jedoch wichtig, dass beide Datensätze eine vergleichbare Dichte und ein ähnliches *User* zu *Item* Verhältnis haben. Sollten zwei Datensätze mit gleichen Dichtewerten vorliegen, so benötigt der Algorithmus des kleineren Datensatzes für ein vergleichbares Ergebnis bedeutend weniger Zeitaufwand. Daher kann auf die Durchführung von Experimenten mit besonders großen Datensätzen verzichtet werden. Wichtig ist es jedoch, dass in beiden Datensätzen beispielsweise dem *User-Item* Verhältnis ähnliche Merkmale zugesprochen werden.

3.3 Datensätze

Die bekanntesten und am häufigsten verwendeten Datensätze wurden von *GroupLens* gesammelt und veröffentlicht. Hierfür wurden über verschiedene Zeiträume hinweg Filmbewertungen der *MovieLens*-Website zusammengetragen. Im nachfolgenden Textabschnitt werden die bekanntesten zwei *MovieLens* Datensätze analysiert und detailliert beschrieben.

3.3.1 MovieLens100k(old)

Der erste und zudem älteste Datensatz ist *MovieLens100k(old)* (GroupLens, 1998). Über einen Zeitraum von sieben Monaten wurden vom 19. September 1997 bis zum 22. April 1998 insgesamt 100.000 *Ratings* gesammelt. Diese *Ratings* wurden von 943 *Usern* für rund 1.682 *Movies* abgegeben. Die *Rating*-Skala, mit der ein *User* ein *Movie* bewerten konnte, lag zwischen einem und fünf Sternen. Es konnten jeweils nur vollständige Sterne abgegeben werden.

3.3.2 Dichte der Rating-Matrix

Als erster Analyseschritt kann die Dichte des Datensatzes betrachtet werden. Diese liegt mit $d = 100 \cdot \frac{|\mathcal{R}|}{|\mathcal{U}| \cdot |\mathcal{I}|} = \frac{100.000.000}{943 \cdot 1.682} \approx 6,3\%$. Eine Visualisierung dieser *Rating*-Matrix kann aus Abbildung 3 entnommen werden. Bei näherer Betrachtung der Abbildung 3 kann neben der Dichte der Matrix $\mathcal{R}$ auch ein *Long-Tail Problem* erkannt werden. Im linken Bereich der Matrix liegen die *Ratings* bedeutend dichter beieinander. Es gibt somit *Items* die häufiger bewertet wurden als andere. Demzufolge existieren *User*, welche häufiger eine Beurteilung gegenüber *Items* abgeben als andere.

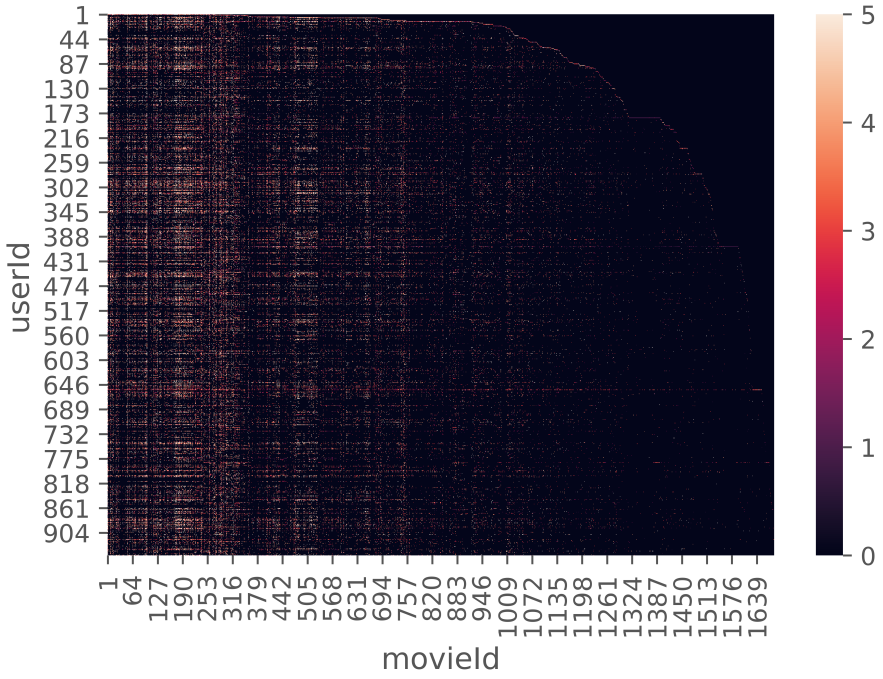
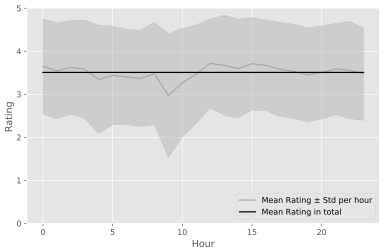


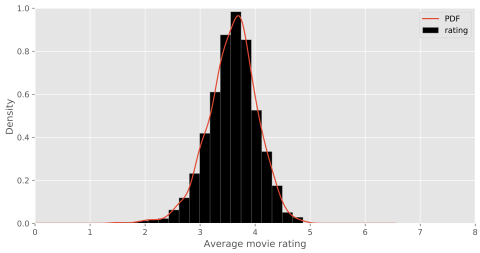
Abbildung 3: Darstellung der *Rating*-Matrix des *MovieLens100k(old)* Datensatzes

3.3.3 Verteilung der User-Ratings

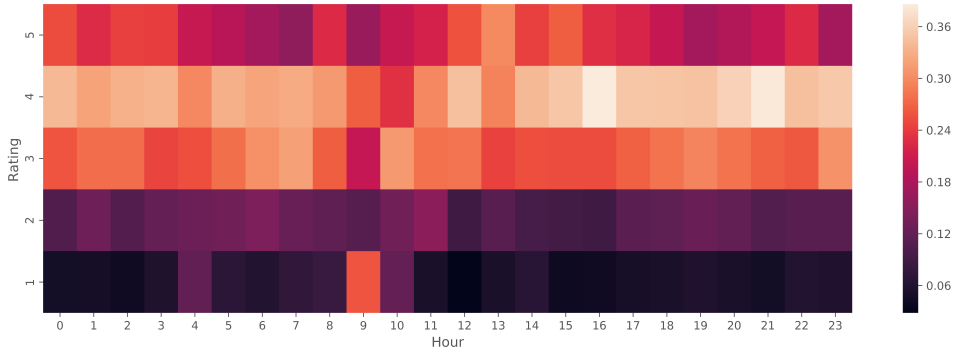
Als zweiter Analyseschritt sollte der Datensatz auf die Verteilung der *Ratings* untersucht werden. Diese Analyse sollte aus einem bestimmen Grund heraus immer erfolgen: Die *Pearson-Correlation* setzt voraus, dass die untersuchten Daten einer Normalverteilung entsprechen. Eine solche Analyse kann aus der unten stehenden Abbildung 4 entnommen werden. Es ist zu erkennen, dass die durchschnittliche Bewertung eines *Users* einer verschobenen Normalverteilung entspricht. Die durchschnittliche Bewertung kann auf $\overline{r} \approx 3,5$ geschätzt werden. Diese Schätzung von $\overline{r}$ ist über den betrachteten Zeitraum hinweg stabil und kann somit als Richtwert für θ_i genutzt werden.



(a) *Rating* pro Stunde



(b) Verteilung der *Ratings* pro *User*



(c) Heatmap *Ratings* pro Stunde

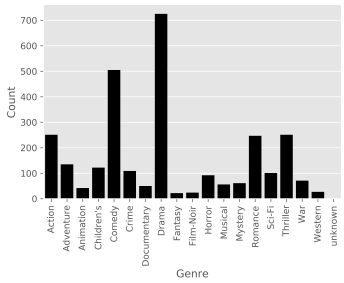
Abbildung 4: Analyse der *Rating*-Verteilung für *MovieLens100k(old)*

3.3.4 Verteilung der Feature-Informationen

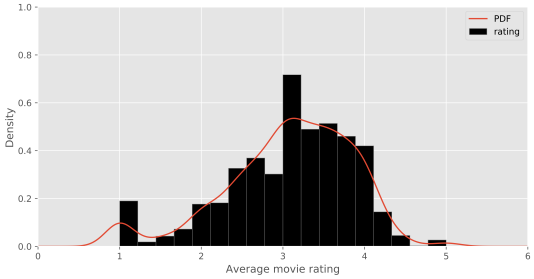
Im anschließenden Schritt sollten die *Feature-Informationen* untersucht werden. Diese können unter anderem für *content-based* oder auch *item-based Recommender* von enormer Wichtigkeit sein. Für den Datensatz *MovieLens100k(old)* sind diese *Feature-Informationen* beispielsweise mittels den *Genres* der Filme gegeben. Abbildung 5 zeigt die Häufigkeit dieser *Genres* sowie deren Verteilung in den *Ratings* Die Bewertung der *Items* unterliegt einer Normalverteilung. Darüber hinaus zeigt sich, dass auch die *Genre*-spezifischen Bewertungen einer Normalverteilung entsprechen. Eine Zusammenfassung der von *MovieLens100k(old)* erhaltenen Werte ist aus der eigens dazu erstellten Tabelle 7 zu entnehmen.

	User	Items	Ratings	Genres	Density	Ratio (I:U)	Intervall	Half Ratings	min.Ratings
MovieLens100k(old)	943	1.682	100.000	19	6,3%	2:1	[1, 5]	False	20

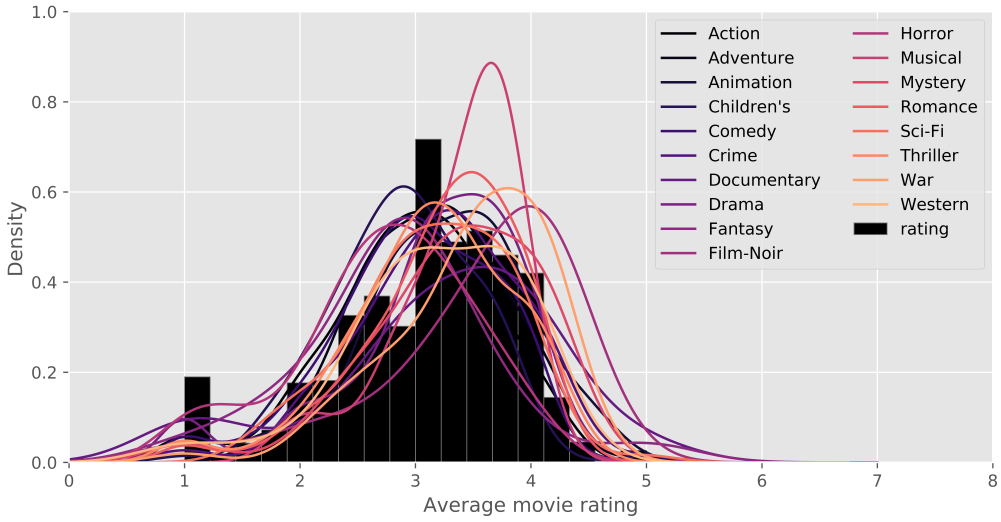
Tabelle 7: Datenübersicht *MovieLens100k(old)*



(a) *Ratings* pro *Genre*



(b) Verteilung der *Ratings* pro *Movie*



(c) Verteilung der *Ratings* pro *Movie* und *Genre*

Abbildung 5: Analyse der *Feature*-Verteilung für *MovieLens100k(old)*

3.3.5 MovieLens100k(latest)

Der zweite und ebenso bekannte Datensatz ist die neuere Auflage des *MovieLens100k(old)* (GroupLens, 2018). Dieser Datensatz besteht aus 100.836 *Ratings*, welche im Zeitraum vom 23. März 1996 bis zum 26. September 2018 gesammelt wurden. Insgesamt wurden von 610 *Usern* rund 9.742 *Movies* mit entsprechenden *Ratings* bewertet. Ebenso wie der *MovieLens100k(old)* Datensatz erlaubt auch *MovieLens100k(latest) Ratings* auf einer Skala von einem bis maximal fünf Sternen. Bei diesem Bewertungsschema ist es jedoch möglich nicht nur ganze sondern auch halbe Sterne zu vergeben. Da innerhalb einer zeitlichen Periode von 1996 bis zum Sommer 2002 nur ganze Sternen zu vergeben waren, mussten diese auch in den *MovieLens100k(latest)* Datensatz aufgenommen werden.

3.3.6 Dichte der Rating-Matrix

Die Dichte der *Rating*-Matrix lässt sich bei *MovieLens100k(latest)* analog zu Abschnitt 3.3.2 ermitteln. Die Dichte ergibt sich aus: $d = 100 \cdot \frac{|\mathcal{R}|}{|\mathcal{U}| \cdot |\mathcal{I}|} = \frac{100.836.000}{610 \cdot 9.742} \approx 1,7\%$. Abbildung 6 zeigt die *Rating-Matrix* des *MovieLens100k(lates)* Datensatzes.

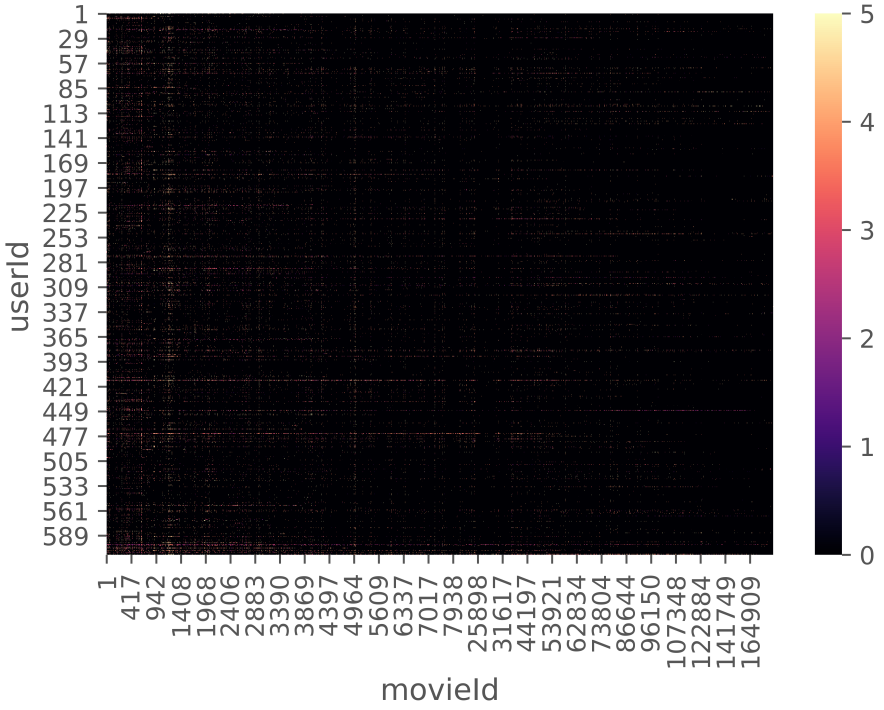
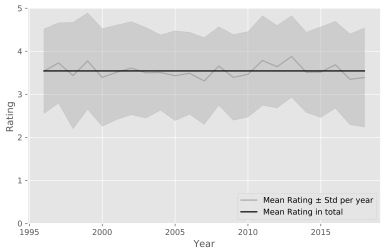


Abbildung 6: Darstellung der *Rating*-Matrix des *MovieLens100k(latest)* Datensatzes

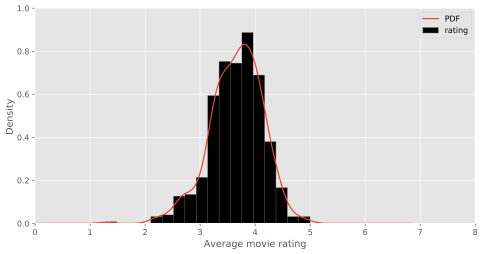
Ebenso besteht hier das *Long-Tail Problem*. Dieses Phänomen lässt sich im linken Abschnitt der Matrix identifizieren. Hier bilden häufig bewertete *Movies* waagerechte Linien. Wie bereits zuvor einmal beschrieben gibt es auch hier bestimmte *User*, die besonders viele *Movies* bewertet haben. Solche sind durch horizontale Linien in der Matrix erkennbar.

3.3.7 Verteilung der User-Ratings

Im Gegensatz zum *MovieLens100k(old)* Datensatz werden für die Übersicht Jahreszahlen anstelle von Stunden betrachtet. Aufgrund der äquivalenten Intervallbreite sind beide Skalen miteinander vergleichbar. Aus Abbildung 7 ist ersichtlich, dass auch hier die durchschnittliche Bewertung eines *Users* einer verschobenen Normalverteilung unterliegt. Des Weiteren kann entnommen werden, dass das durchschnittliche *Rating* mit $\overline{r} \approx 3,5$ versehen ist. Dieser Wert ist über die Jahre annähernd konstant geblieben. Somit kann $\overline{r}$ auch für diesen Datensatz als ein Richtwert für θ_i angesehen werden.



(a) *Rating* pro Jahr



(b) Verteilung der *Ratings* pro *User*

(c) Heatmap *Ratings* pro Jahr

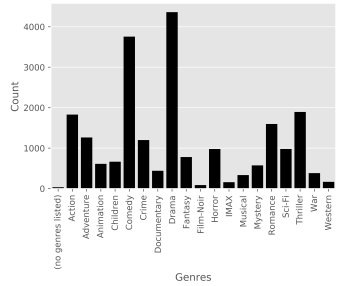
Abbildung 7: Analyse der *Rating*-Verteilung für *MovieLens100k(latest)*

3.3.8 Verteilung der Feature-Informationen

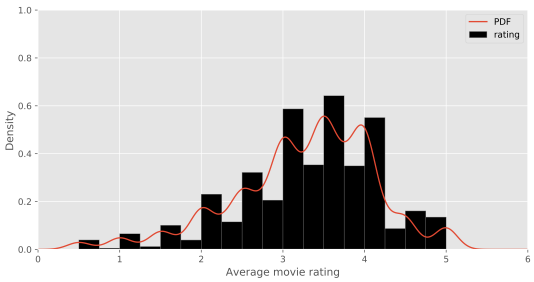
Auch für *MovieLens100k(latest)* müssen die *Genres* als wichtiger Bestandteil untersucht werden. Dazu wird erneut die Verteilung der Bewertungen aller *Genres* gegen die der einzelnen aufgetragen. Eine solche Darstellung kann der nachfolgenden Abbildung 8 entnommen werden. Der Großteil der *Genres* unterliegt einer eindeutigen Normalverteilung. Die meisten der Genres weisen jedoch auch leichte Unregelmäßigkeiten auf. Dies muss für den möglichen Einsatz der *Pearson-Correlation* bedacht werden. Folgende Merkmale lassen sich für den *MovieLens100k(latest)* Datensatz ermitteln:

	User	Items	Ratings	Genres	Density	Ratio (I:U)	Intervall	Half Ratings	min.Ratings
MovieLens100k(latest)	610	9.724	100.836	20	1,7%	16:1	[0.5, 5]	True	20

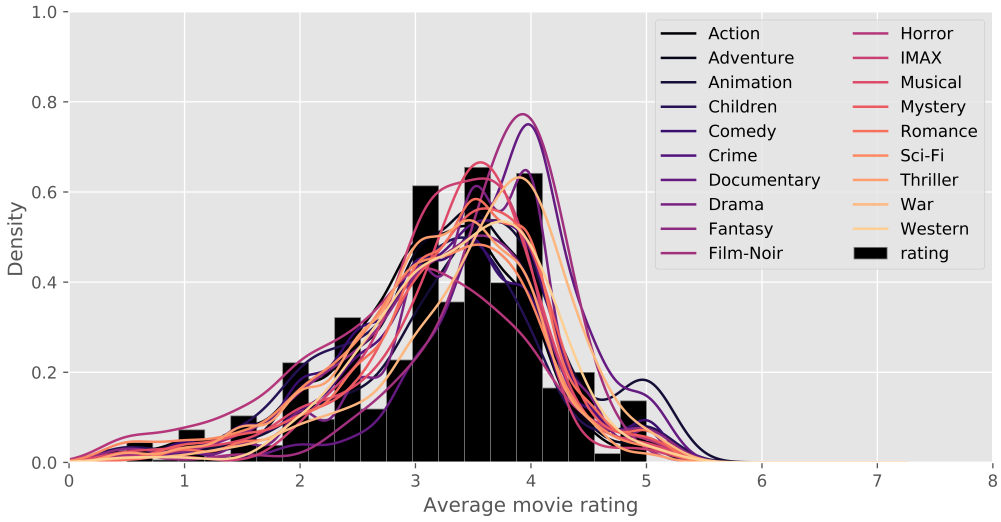
Tabelle 8: Datenübersicht *MovieLens100k(latest)*



(a) *Ratings* pro *Genre*



(b) Verteilung der *Ratings* pro *Movie*



(c) Verteilung der *Ratings* pro *Movie* und *Genre*

Abbildung 8: Analyse der *Feature*-Verteilung für *MovieLens100k(latest)*

3.3.9 Webscope-R3(yahoo!) Datensatz

Als dritter und letzter Datensatz wird in dieser Arbeit der *Webscope-R3(yahoo!)* Datensatz betrachtet. Dieser ist nicht frei erhältlich und kann lediglich nach einer Freigabe durch *Yahoo!-Research* untersucht werden (Yahoo!-Research, 2018). Diese Daten wurden zwischen dem 22. August 2006 und dem 07. September 2006 zufällig aus der *Yahoo!-Music* Datenbank ausgelesen. Somit wurden 311.704 *Ratings* von insgesamt 15.400 *Usern* gesammelt. Diese *User* konnten 1.000 Musikdateien auf einer *Rating*-Skala von einem bis maximal fünf Sternen bewerten. Zulässig waren auch hier nur ganze Sterne.

3.3.10 Dichte der Rating-Matrix

Die Dichte der *Rating*-Matrix für den *Webscope-R3(yahoo!)* Datensatz beträgt: $d = 100 \cdot \frac{|\mathcal{R}|}{|\mathcal{U}| \cdot |\mathcal{I}|} = \frac{311.704.000}{15.400 \cdot 1.000} \approx 2,0\%$. Die Visualisierung der *Rating*-Matrix kann aus der unten stehenden Abbildung entnommen werden.

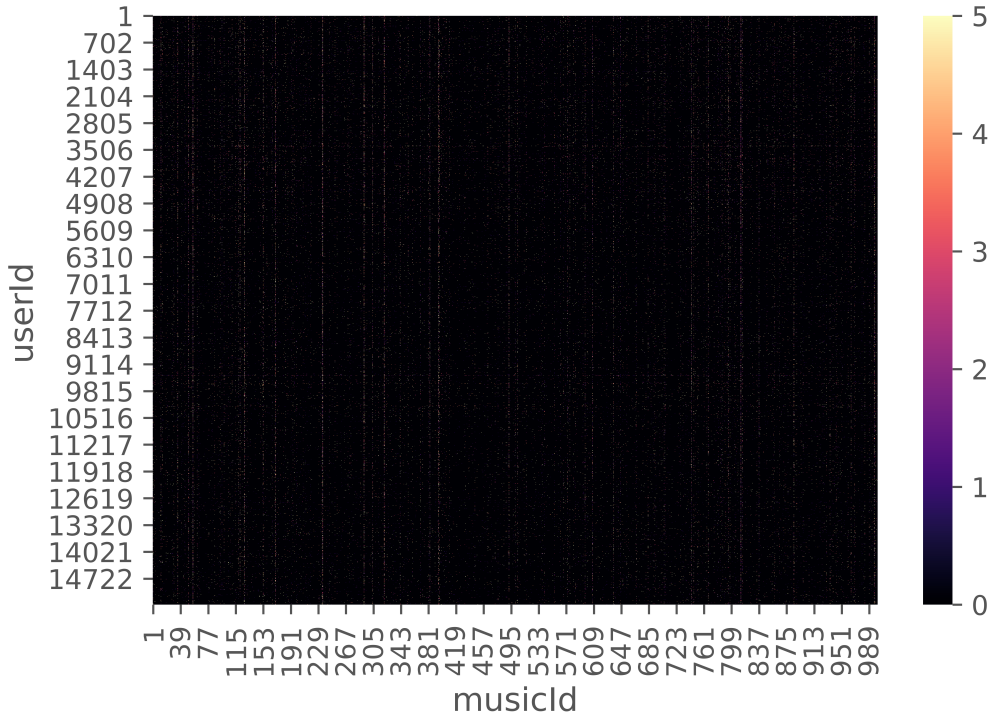
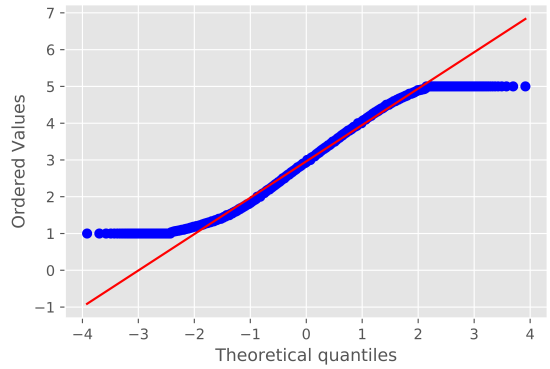


Abbildung 9: Darstellung der *Rating*-Matrix des *Webscope-R3(yahoo!)* Datensatzes

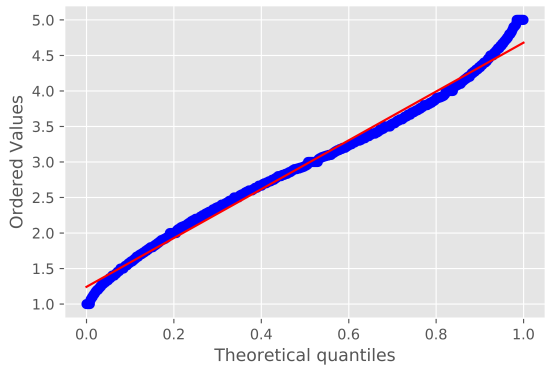
Ebenso wie bei den *MovieLens100k* Datensätzen kann hier ein *item*-spezifisches *Long-Tail Problem* identifiziert werden. Dieses Phänomen lässt sich anhand der ausgeprägten vertikalen Linien in der *Rating*-Matrix erkennen. Ein *user*-spezifisches Problem ist im Gegensatz zu den *MovieLens100k* Datensätzen nicht zu entnehmen.

3.3.11 Verteilung der User-Ratings

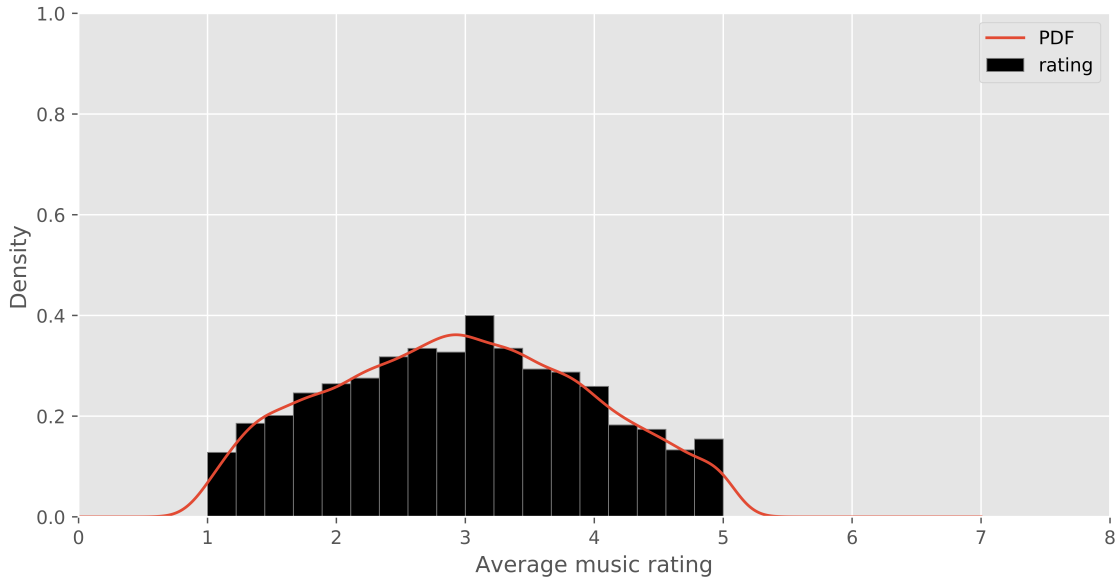
Auch bei der Verteilung der durchschnittlichen Bewertung von *Usern* unterscheidet sich *Webscope-R3(yahoo!)* von den *MovieLens100k* Datensätzen. Anders als bei den *MovieLens100k* Datensätzen zeigt die zugrundeliegende Verteilung eine starke Ähnlichkeit zu einer Gleichverteilung. Durch einen Blick auf die *QQ-Plots* aus der Abbildung 10 kann diese Ähnlichkeit verschärft werden. Es ist demnach zu erwarten, dass *user-based Recommender* im Vergleich zu *item-based Recommender* einen höheren Fehler für jede mögliche Größe der Nachbarschaft erzeugen werden. Die durchschnittliche Bewertung $\overline{r}$ kann auf $\overline{r} \approx 3,0$ festgelegt werden. Somit kann für den *Webscope-R3(yahoo!)* Datensatz die Schranke mit $\theta_i \geq 3,0$ geschätzt werden.



(a) QQ-Plot Normalverteilung



(b) QQ-Plot Gleichverteilung

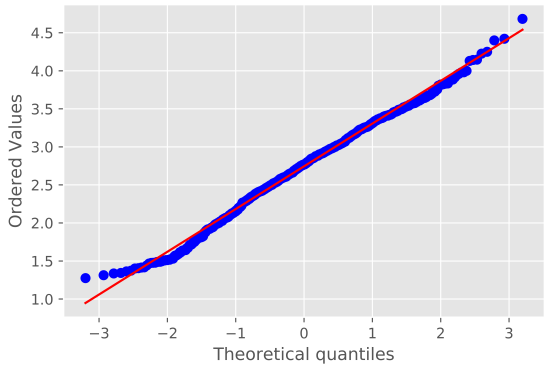


(c) Verteilung der *Ratings* pro *User*

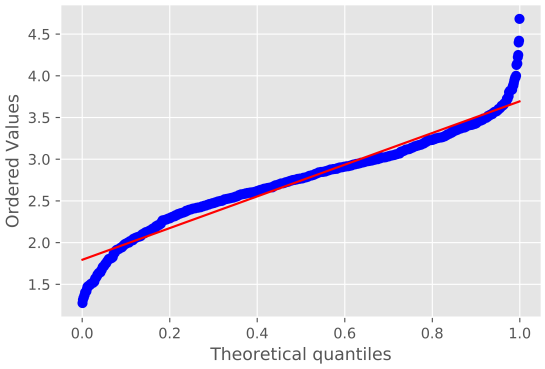
Abbildung 10: Analyse der *Rating*-Verteilung pro *User* für *Webscope-R3(yahoo!)*

3.3.12 Verteilung der Feature-Informationen

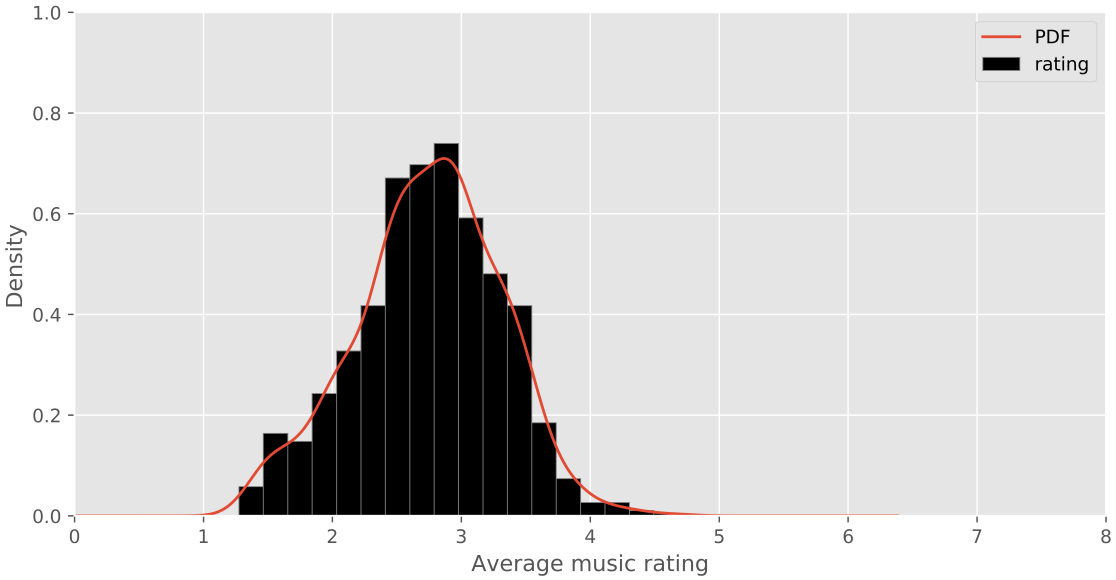
Als einziges *Feature* steht diesem Datensatz lediglich die *musicId* zur Verfügung. Die Analyse der Verteilung der durchschnittlichen Bewertung pro *Music-Track* kann für die Analyse eines *item-based Recommender* von Vorteil sein.



(a) QQ-Plot Normalverteilung



(b) QQ-Plot Gleichverteilung



(c) Verteilung der *Ratings* pro *Music-Track*

Abbildung 11: Analyse der *Rating*-Verteilung pro *Music-Track* für *Webscope-R3(yahoo!)*

Es zeigt sich, dass die Verteilung der durchschnittlichen Bewertung der *Music-Tracks* einer Normalverteilung entspricht. Der *Webscope-R3(yahoo!)* Datensatz ist somit besser für *item-based*, als für *user-based Recommender* geeignet. Insgesamt beinhaltet *Webscope-R3(yahoo!)* folgende Daten und Informationen:

	User	Items	Ratings	Genres	Density	Ratio (I:U)	Intervall	Half Ratings	min.Ratings
Webscope-R3(yahoo!)	15.400	1.000	311.704	None	2,0%	1:15	[1, 5]	False	10

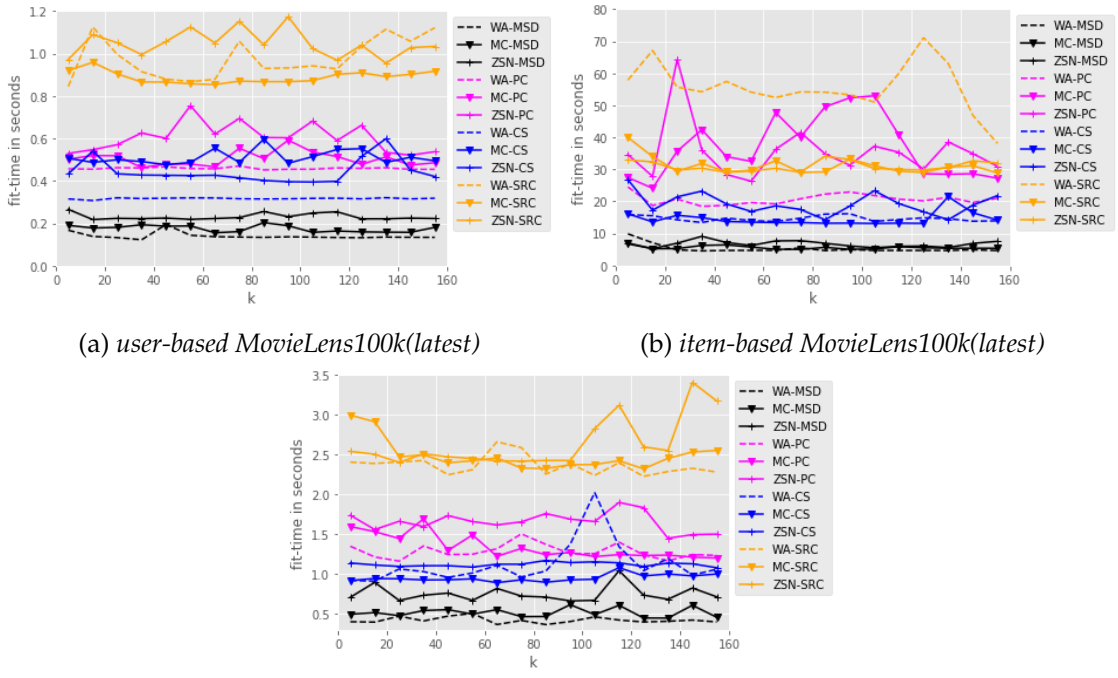
Tabelle 9: Datenübersicht *Webscope-R3(yahoo!)*

4 Accuracy-Analyse

Dieser Abschnitt der Bachelorarbeit untersucht die *Accuracy*-Metriken aus den in Abschnitt 1.3.1 und Abschnitt 1.3.8 vorgestellten *collaborative-filtering* und *content-based* Verfahren. Dabei sollen die folgenden Faktoren genauer betrachtet werden:

1. Genauigkeit (*Accuracy*)
2. Zeitaufwand (*Time*)

Zudem soll der Unterschied zwischen dem Einsatz von *item-* und *user-based* Strategien anhand der Datensätze *MovieLens100k(latest)* und *Webscope-R3(yahoo!)* veranschaulicht werden. Hierbei wird Bezug auf das *User* zu *Item* Verhältnis des betrachteten Datensatzes und dessen Auswirkung auf die *collaborative-filtering* Algorithmen eingegangen. Die hier zu verwendenden Datensätze eignen sich aufgrund der Verteilungen von *User* zu *Item Ratings* besonders gut. Hiermit können Unterschiede der Distanzfunktionen aufgezeigt werden. Des Weiteren wird für den *MovieLens100k(latest)* ein *user-based* und für den *Webscope-R3(yahoo!)* Datensatz ein *item-based* Ansatz gewählt. Diese Aufteilung wurde aufgrund des erhöhten Zeitaufwandes der einzelnen Berechnungsschritte vorgenommen. Ein Verglich der *Fit*-Zeiten kann aus der nachfolgenden Abbildung 12 entnommen werden.



(a) *user-based MovieLens100k(latest)*

(b) *item-based MovieLens100k(latest)*

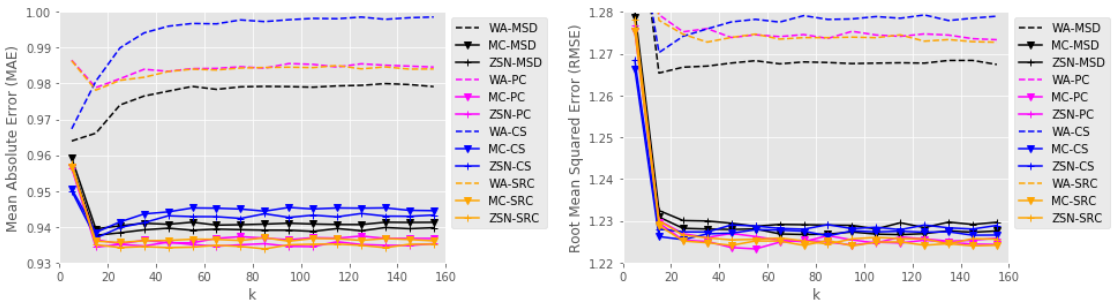
(c) *item-based Webscope-R3(yahoo!)*

Abbildung 12: Messung der *Fit*-Zeit

Die großen Zeitunterschiede können auf die Wahl zwischen dem *user-* und dem *item-based* Ansatz zurückgeführt werden. Um genauer zu verstehen, wie die Zeitunterschiede verursacht werden, können erneut die von Desrosiers und Karypis (2011) aufgeführten Nachbarschafts-Schätzungen betrachtet werden. Diese können dazu genutzt werden um vorab zu untersuchen, ob ein *user-* oder *item-based* Ansatz geeignet ist. Für den *MovieLens100k(latest)* Datensatz ergibt sich eine durchschnittliche Anzahl an *Ratings* pro *User* von $p = \frac{|\mathcal{R}|}{|\mathcal{U}|} = \frac{100.836}{610} \approx 165$. Diese Zahl kann für die Schätzfunktion eines *user-based* Ansatzes verwendet werden. Die durchschnittliche Größe einer zu erwartenden Nachbarschaft ergibt sich dann aus: $|\mathcal{N}_u| = 609\big(1 - \big(\frac{9.724-p}{9.724}\big)^p\big) \approx 572$. Demnach ergibt sich für jeden *User* eine Nachbarschaft, die rund 93% aller *User* abdeckt. Entsprechend dieser *User*-Abdeckung wird in kürzester Zeit eine stabile Nachbarschaft gefunden. Für diesen Datensatz ergibt sich für die durchschnittliche Anzahl an *Ratings* pro *Item*: $q = \frac{|\mathcal{R}|}{|\mathcal{I}|} = \frac{100.836}{9.724} \approx 10$. Daraus kann die Größe der durchschnittlichen Nachbarschaft eines *Items* mittels der nachfolgenden Formel geschätzt werden: $|\mathcal{N}_i| = 9.723\big(1 - \big(\frac{610-q}{610}\big)^q\big) \approx 1.481$. Somit deckt die Größe einer solchen *Item*-Nachbarschaft lediglich rund 15% aller möglichen *Items* ab. Folgende Erkenntnis konnte demnach erzielt werden: Ein *user-based* Ansatz ist somit besser für den *MovieLens100k(latest)* Datensatz geeignet. Analog können die obigen Berechnungen für *Webscope-R3(yahoo!)* durchgeführt werden. Für einen *item-based* Ansatz ergibt sich dann eine Nachbarschaft von $|\mathcal{N}_i| \approx 997$ und eine *Item*-Abdeckung von rund 99%. Im Gegensatz dazu ergibt sich für einen *user-based* Ansatz eine durchschnittliche Nachbarschaft von $|\mathcal{N}_u| \approx 5.118$ *Usern* und eine Abdeckung von 33%. Dementsprechend ist für den *Webscope-R3(yahoo!)* Datensatz eine *item-based* Ansatz durchaus geeignet. Die Zeitunterschiede in den Ansätzen können daher durch die unterschiedlichen Größen der Nachbarschaften zurückgeführt werden. Von Desrosiers und Karypis (2011) wurde folgendes Phänomen beobachtet: Es kann bereits vorab festgehalten werden, dass sollte auf ein ungeeignetes Verfahren zurückgegriffen werden, eine entsprechend schlechte, dem geringen Informationsgehalt verschuldete *Accuracy* entsteht.

4.1 Item-based Verfahrensanalyse

In diesem Experiment wird die *Accuracy* des *item-based* Ansatzes untersucht. Hierfür wird der *Webscope-R3(yahoo!)* Datensatz verwendet. Eine detaillierte Analyse kann aus Abschnitt 3.3.9 entnommen werden.

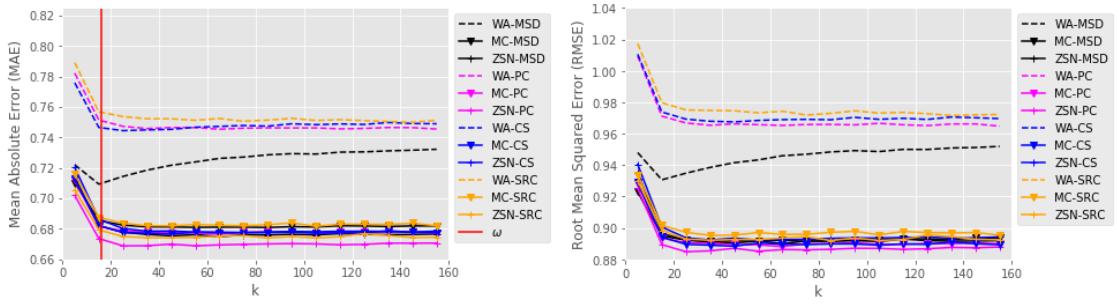


(a) *MAE* (b) *RMSE*

Abbildung 13: *Accuracy* Messung auf dem Datensatz *Webscope-R3(yahoo!)*

Abbildung 13 zeigt das Verhalten der in Abschnitt 1.3 erwähnten Verfahren. Zu sehen ist in Abbildung 13(a) der *MAE* im Verhältnis zu der Größe k der Nachbarschaft. Abbildung 13(b) zeigt den identischen Zusammenhang unter der Berücksichtigung des *RMSE*. Zu Beginn fällt auf, dass der Kurvenverlauf beider Graphen durchaus ähnlich erscheint. Dementsprechend wird im weiteren Textabschnitt auf den *MAE* eingegangen. Es ist zu erkennen, dass das *Weighted-Average (WA)* mit jeder Distanzfunktion schlechter abschneidet als die anderweitigen Verfahren. Auch fällt auf, dass die Kurvenverläufe all jener Verfahren, die mit der *Spearman-Rank-Correlation (SRC)* und *Pearson-Correlation (CS)* arbeiten nahezu identische Fehlerwerte erzeugen. Dieser Zusammenhang ist dadurch erklärbar, dass die *Spearman-Rank-Correlation* eine Erweiterung der *Pearson-Correlation* ist. Ungeachtet dessen fällt auf, dass die *Spearman-Rank-Correlation* minimal kleinere Fehler als die *Pearson-Correlation* erzeugt. Dieser Zusammenhang ist dadurch erklärbar, dass die Eingabegrößen beider Messeverfahren im *Webscope-R3(yahoo!)* Datensatz einer Gleichverteilung entsprechen. Im Gegensatz zu der *Pearson-Correlation* ist die *Spearman-Rank-Correlation* auch für Gleichverteilungen ausgelegt, da sie intern durch die Rangvergabe eine solche erzeugt. Die *Pearson-Correlation* hingegen setzt Eingabegrößen, die annähernd einer Normalverteilung entsprechen, voraus. Daher kann hier der leichte Unterschied durch die, den Eingabegrößen, zugrundeliegende Verteilung begründet werden. Eine Ebene darüber hinaus zeichnet sich sogar eine hierarchische Struktur der Verfahren ab. Jedes der gezeigten Verfahren, welches die *Cosine-Similarity* als Distanzfunktion verwendet, schneidet schlechter ab als jene, die die *Spearman-Rank-* beziehungsweise *Pearson-Correlation* verwenden. Auch zeigt sich, dass *Mean-Centering (MC)* immer schlechter als *Z-Score-Normalization (ZSN)* abschneidet, wenn sich beide Verfahren eine gemeinsame Distanzfunktion zu Nutzen machen. Insgesamt zeigt sich so für den *Webscope-R3(yahoo!)* Datensatz, dass das *Weighted-Average* Verfahren in jeder Kombination mit einer der Distanzfunktionen ungeeignet ist. Verfahren die sich nicht auf die gewichteten Distanzen, sondern auf die Verteilung der Eingabegrößen konzentrieren, schneiden besser ab. Besonders gut schneiden hier Kombinationen aus den Verfahren *Mean-Centering* und *Z-Score-Normalization* mit den Distanzfunktionen *Pearson-Correlation* und *Spearman-Rank-Correlation* ab.

4.2 User-based Verfahrensanalyse



(a) *MAE* (b) *RMSE*

Abbildung 14: *Accuracy* Messung auf dem Datensatz *MovieLens100k(latest)*

Dieses Experiment untersucht die *Accuracy* des *user-based* Ansatzes. Es wird für die Ausführung der Verfahren der *MovieLens100k(latest)* Datensatz verwendet. Eine detaillierte Analyse kann aus Abschnitt 3.3.5 entnommen werden. Diese Betrachtung beschränkt sich auf die Darstellung des *MAE*, da der Kurvenverlauf des *RMSE* nahezu identisch ist. Hier fällt ebenfalls auf, dass *Weighted-Average* mit jeder der zur Verfügung stehenden Distanzfunktionen schlechter abschneidet. Ähnlich zu den *item-based* Ansätzen zeigt sich für die *user-based* Ansätze, dass die Verfahren *Mean-Centering* und *Z-Score-Normalization* mit jeder Distanzfunktion besser als *Weighted-Average* abschneiden. Hierfür sind alle Verfahren, die die *Spearman-Rank-Correlation* als Distanzfunktion verwenden, ausgeschlossen. Es zeigt sich eindeutig, dass die *Spearman-Rank-Correlation* dazu beiträgt, dass die Werte von *MAE* und *RMSE* größer ausfallen. Eine mögliche Ursache dafür ist die Verteilung der *Ratings*. Wie aus Abbildung 7 ersichtlich wird, entspricht diese einer Normalverteilung. Es kann demnach schneller zu einer Verknüpfung der berechneten *Rankings* kommen. Dadurch wird die interne, mittels der Vergabe von Rängen erzeugte, Gleichverteilung der *Spearman-Rank-Correlation* stark verändert. Als erster Vergleich der beiden Ansätze kann demnach festgehalten werden: Sollte keine Normalverteilung vorliegen, so schneidet die *Pearson-Correlation* eindeutig besser als die *Spearman-Rank-Correlation* ab. Im Gegensatz zu Abbildung 13 geht aus den in Abbildung 14 gezeigten Kurvenverläufen keine eindeutige Struktur hervor. Die Fehlerwerte der Verfahren liegen dafür zu nahe beieinander. Dennoch stellt sich die Kombination aus der *Z-Score-Normalization* mit der *Pearson-Correlation* als besonders geeignet heraus. Diese Paarung fällt durch die große Abweichung von $\delta \approx 0.01$ im Vergleich zu den Fehlerwerten der anderen Kombinationen besonders auf. Eine Begründung dieses Zusammenhangs kann darauf zurückgeführt werden, dass die Verfahren als Eingabe eine normalverteilte Größe erwarten. Eine detaillierte Beschreibung zur Arbeitsweise der *Z-Score-Normalization* kann aus Abschnitt 1.3.7 entnommen werden. Insgesamt kann auch hier festgehalten werden, dass *Weighted-Average* im Vergleich zu den anderen Verfahren deutlich schlechter abschneidet. Hingegen scheint eine Kombination aus den Verfahren *Z-Score-Normalization* und *Mean-Centering* mit den Distanzfunktionen *Mean-Squared-Distance*, *Cosine-Similarity*, *Spearman-Rank-* und *Pearson-Correlation* besonders geringe Fehler zu erzeugen. Aus Tabelle 10 können die durchschnittlichen Fehler- und Zeit-Messungen der in diesem Abschnitt untersuchten Verfahren entnommen werden. Dabei wurde der Durchschnitt über die 16 verschiedenen Größen der Nachbarschaften $k \in [5, 155]$ gebildet.

	MAE	RMSE	Fit-Zeit
WA-MSD	0.7254	0.9456	0.1402
WA-CS	0.7493	0.9721	0.3168
WA-PC	0.7488	0.9689	0.4588
WA-SRC	0.7543	0.9765	0.8188
MC-MSD	0.6836	0.8944	0.1752
MC-CS	0.6800	0.8924	0.5109
MC-PC	0.6801	0.8925	0.5051
MC-SRC	0.6851	0.8992	0.8479
ZS-MSD	0.6790	0.8946	0.2314
ZS-CS	0.6815	0.8968	0.4440
ZS-PC	0.6721	0.8891	0.6048
ZS-SRC	0.6771	0.8958	0.8975

(a) *MovieLens100k(latest)*

	MAE	RMSE	Fit-Zeit
WA-MSD	0.8083	1.0356	0.6390
WA-CS	0.8601	1.0972	1.9347
WA-PC	0.8613	1.0941	2.6386
WA-SRC	0.8643	1.0972	1.4397
MC-MSD	0.8030	1.0196	0.5017
MC-CS	0.8148	1.0312	2.0985
MC-PC	0.8045	1.0269	2.3298
MC-SRC	0.8037	1.0266	1.5321
ZS-MSD	0.8075	1.0324	1.0142
ZS-CS	0.8201	1.0451	1.0955
ZS-PC	0.7989	1.0253	2.3488
ZS-SRC	0.7984	1.0250	1.4643

(b) *MovieLens100k(old)*

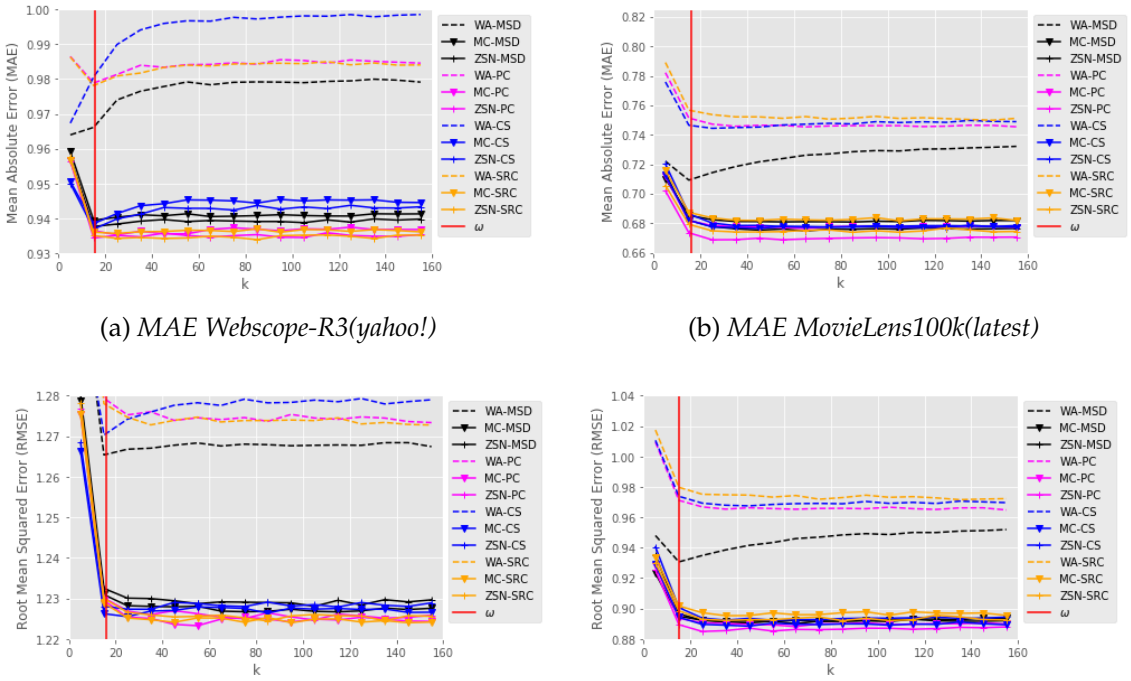
	MAE	RMSE	Fit-Zeit
WA-MSD	0.9769	1.2703	0.4197
WA-CS	0.9940	1.2791	1.1272
WA-PC	0.9842	1.2783	1.2854
WA-SRC	0.9836	1.2776	2.5164
MC-MSD	0.9419	1.2308	0.5145
MC-CS	0.9446	1.2295	0.9507
MC-PC	0.9377	1.2281	1.3417
MC-SRC	0.9376	1.2280	2.5337
ZS-MSD	0.9402	1.2326	0.7512
ZS-CS	0.9427	1.2308	1.1215
ZS-PC	0.9363	1.2292	1.6574
ZS-SRC	0.9361	1.2291	4.2916

(c) *Webscope-R3(yahoo!)*

Tabelle 10: Durchschnittliche Fehler- und Zeit-Werte der *k-NN* Verfahren

4.3 Auswirkung des User zu Item Verhältnis

Dieser Textabschnitt beschäftigt sich mit der Auswirkung des *User* zu *Item* Verhältnisses auf die *Accuracy*. Sowohl in Abbildung 13 als auch in Abbildung 14 zeigt sich im Intervall von 0 bis 20, dass jedes dieser Verfahren eine rapide Absenkung der Fehler mit sich bringt. Des Weiteren zeigt sich, dass es einen gemeinsamen Punkt ω gibt. An diesem Punkt konsolidiert oder verstärkt sich der Fehler. Dieses Phänomen kann in Abbildung 15 betrachtet werden:



(a) *MAE Webscope-R3(yahoo!)*

(b) *MAE MovieLens100k(latest)*

(c) *RMSE Webscope-R3(yahoo!)*

(d) *RMSE MovieLens100k(latest)*

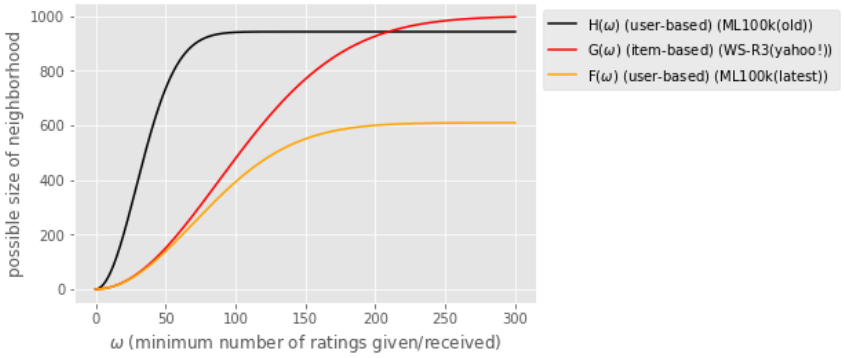
Abbildung 15: Zusammenhang der *neighborhood-based* Verfahren in ω

Um diesen Zusammenhang zu verstehen muss noch einmal die von Desrosiers und Karypis (2011) vorgestellte Abschätzung der durchschnittlichen Nachbarschaft betrachtet werden. Die *user-* oder *item-based* spezifische Funktion kann aus Tabelle 11 entnommen werden. Wie zuvor erwähnt liegt diesen Funktionen ein bipartiter Graph zugrunde über den die durchschnittliche Größe der Nachbarschaften geschätzt werden kann. Wie im vorhergehenden Abschnitt 4 erwähnt, können diese Funktionen dazu genutzt werden, um zwischen einem *user-* oder einem *item-based* Ansatz zu wählen. Diese Wahl beruht auf den Parametern p und q. Hierbei ist p die Anzahl der *Ratings*, die ein *User* im Schnitt gegeben hat, und q die durchschnittliche Anzahl an erhaltenen *Ratings* pro *Item*. Sowohl p, als auch q, können durch eine gemeinsame Variable ω ersetzt werden. Diese Variable gibt dann nicht mehr die durchschnittlich verteilten *Ratings* pro *User* oder *Item* an, sondern vielmehr die Anzahl der *Ratings*, die ein *User* gegeben, oder ein *Item* erhalten hat. Damit ergeben sich die Formeln:

	Größe Nachbarschaft
MovieLens100k(latest)	$F(\omega) = 609\left(1 - \left(\frac{9.724-\omega}{9.724}\right)^{\omega}\right)$
MovieLens100k(old)	$H(\omega) = 942\left(1 - \left(\frac{1.682-\omega}{1.682}\right)^{\omega}\right)$
Webscope-R3(yahoo!)	$G(\omega) = 999\left(1 - \left(\frac{15.400-\omega}{15.400}\right)^{\omega}\right)$

Tabelle 11: Datensatz-spezifische Schätzung der Größe einer Nachbarschaft basierend auf dem verfahrenstypischen Ansatz

Durch die Parametrisierung der Schätzfunktionen ist eine genauere Untersuchung der Größe einer Nachbarschaft in Abhängigkeit der Anzahl von *Ratings* möglich. Die unten stehende Abbildung 16 zeigt die spezifischen Schätzfunktionen für die in Abschnitt 3 betrachteten Datensätze:



(a) Nachbarschafts-Schätzung

Abbildung 16: Nachbarschafts-Schätzung in Abhängigkeit von ω in Bezug auf die Verfahrens-spezifischen Ansätze

Es ist zu sehen, dass jede der abgebildeten Kurven zu Beginn langsam anwächst und im Anschluss drastisch steigt. Werden diese Punkte genauer betrachtet, so fällt auf, dass an diesen Stellen die Anzahl der *Ratings* identisch mit der Größe der potentiellen Nachbarschaft ist. Bezogen auf den bipartiten Graphen, welcher dem *Recommender-Problem* zugrunde liegt, bedeutet der Punkt ω eine deutliche Erhöhung der Knoten-Abdeckung durch die verbindenden Kanten. Darüber hinaus kann ω auch als Punkt, ab dem sich die *Accuracy* eindeutig interpretieren lässt, verstanden werden. Demnach gibt ω die Anzahl der *Ratings* an, die mindestens pro *User* beziehungsweise pro *Item* abgegeben werden müssen, um einen soliden Informationsgehalt für die Nachbarschafts-Algorithmen zu generieren. Ab diesem Punkt steht für die Algorithmen eine potentiell größere Menge an Nachbarn zu Verfügung. Die tatsächliche Menge an abgegebenen *Ratings* hat ab die-

sem Bereich eine immense Auswirkung auf die Vernetzung der Knoten. Demnach lohnt sich eine genauere Untersuchung der ω-Punkte. Um diese Punkte bestimmen zu können muss das ω gefunden werden, an dem die betrachtete Schätzfunktion einen identischen Funktionswert annimmt. Entsprechend wird die Schätzfunktion auf die Existenz von Fixpunkten untersucht. Diese Fixpunkte ergeben sich aus den Nullstellen der folgenden Differenzfunktion: $\alpha(\omega) = \omega - f(\omega)$. In der nachfolgenden Abbildung 17 können die Schätzfunktionen und die dazugehörigen Differenzfunktionen der in Abbildung 15 untersuchten Verfahren betrachtet werden.

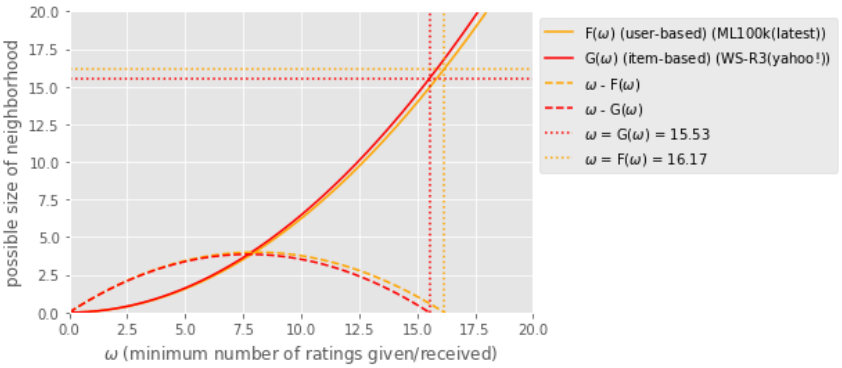
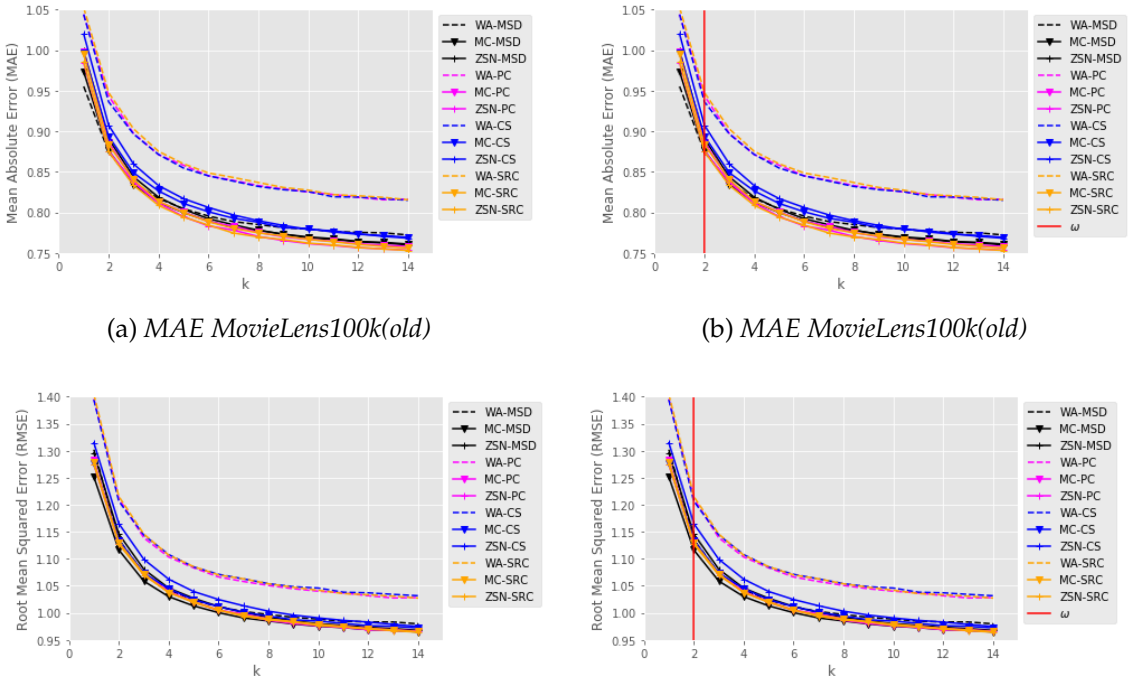


Abbildung 17: Fixpunkte der Differenzfunktionen: $\omega - F(\omega)$ und $\omega - G(\omega)$

Werden die Nullstellen der Differenzfunktionen ermittelt, so ergeben sich die folgenden Werte: Für den *user-based* Ansatz in Kombination mit dem *MovieLens100k(latest)* Datensatz ergibt sich die Nullstelle $\omega_1 \approx 16,17$. Die Nullstelle der Differenzfunktion für den *item-based* Ansatz in Kombination mit dem *Webscope-R3(yahoo!)* Datensatz ergibt sich aus $\omega_2 \approx 15,53$. Aus der Betrachtung dieser Werte wird ersichtlich, dass sich an genau diesen ω-Werten ein Einbruch des Fehlers ergibt. Damit liegt nahe, dass die Mindestanzahl ω an abgegebenen *Ratings* einen direkten Zusammenhang mit der Größe k der für die k-*NN* verwendeten Nachbarschaften besitzt. Der Zusammenhang zwischen den *Ratings* und der Größe der Nachbarschaft wird ersichtlich, wenn noch einmal erneut die Nachbarschaftsbildung für *collaborative-filtering Recommender* betrachtet wird. Die Nachbarschaften zwischen Knoten in einem bipartiten Graphen kommen erst dann zustande, wenn ein oder mehrere Knoten der gleichen Partition über mindestens einen Knoten der anderen Partition miteinander verbunden sind. Die verbindenden Kanten zwischen benachbarten Elementen entsprechen dann den *Ratings*. Ferner existieren in einer Nachbarschaft der Größe n, die aus einem bipartiten Graphen entnommen wurde, mindestens ebenso viele Kanten. Demnach besteht ein direkter Zusammenhang der Größe k einer Nachbarschaft und der Mindestanzahl ω an darin enthaltenen *Ratings*. Ab diesen ω-Werten steht dem k-*NN* Algorithmus eine hinreichend große Nachbarschaft zu Verfügung. Eine Erhöhung der betrachteten Nachbarschaft bringt für die meisten Verfahren keine Änderung der *Accuracy* mit sich. Da dieses Phänomen bei allen zur Verfügung stehenden Verfahren betrachtet werden kann, liegt der Schluss nahe, dass es sich dabei um ein k-*NN* typisches Verhalten handelt. Dieser Zusammenhang wird verstärkt, wenn man die *User* zu *Item* Verhältnisse der Datensätze betrachtet. Es fällt auf, dass diese Verhältnisse nahe an den errechneten ω-Werten liegen. Für den *MovieLens100k(latest)* ergibt sich ein tatsächliches *User* zu *Item* Verhältnis von: $(\mathcal{U} : \mathcal{I}) = 1 : 15,94 \approx 1 : 16$. Des Weiteren ist dieses

Verhältnis für den *Webscope-R3(yahoo!)* mittels $(\mathcal{U} : \mathcal{I}) = 15,4 : 1 \approx 15 : 1$ gegeben. Das tatsächliche Verhältnis liegt durchaus nahe an den errechneten ω-Werten und den beobachteten Einbrüchen in den Fehlern.

Die k-*NN* Verfahren stehen in direktem Zusammenhang mit den betrachteten Datensätzen. Diese Vermutung bestätigt sich durch eine Betrachtung der Verfahren in Kombination mit dem *MovieLens100k(old)* Datensatz. Der *MAE* und *RMSE* der k-*NN* Verfahren kann aus Abbildung 18 entnommen werden:



(a) *MAE MovieLens100k(old)*

(b) *MAE MovieLens100k(old)*

(c) *RMSE MovieLens100k(old)*

(d) *RMSE MovieLens100k(old)*

Abbildung 18: Zusammenhang der *neighborhood-based* Verfahren in ω. Es wird der *MovieLens100k(old)* Datensatz betrachtet.

Zu sehen ist das Intervall von einem bis zu zwanzig *Ratings*. Das tatsächliche *User* zu *Item* Verhältnis beträgt hier: $(\mathcal{U} : \mathcal{I}) = 1 : 1,79 \approx 1 : 2$. Die Nullstelle der Differenzfunktion liegt bei $\omega_3 \approx 1,78$. Die Abbildung 19 zeigt diesen Zusammenhang.

Es zeigt sich an dieser Stelle, dass alle Verfahren bei ω einen immensen Einbruch der Fehler aufweisen. Demnach ergibt sich ein starker Zusammenhang zwischen dem tatsächlichen *User* zu *Item* Verhältnis und der *Accuracy* der k-*NN* Verfahren.

Zusammenfassend kann gesagt werden, dass ω die Anzahl der mindestens im System vorhandenen *Ratings* angibt. Dieser Wert ist darüber hinaus die obere Grenze des *User* zu *Item* Verhältnisses. Eine Erhöhung des ω-Wertes hat zur Folge, dass die Wahrscheinlichkeit, dass zwei Knoten miteinander verbunden sind, drastisch ansteigt. Umso höher ω gewählt wird, desto vollständiger wird der dem *Recommender-Problem* zugrundeliegende bipartite Graph. Dies hat zur Folge, dass auch die Nachbarschafts-Algorithmen ihr volles Potential erreichen. Eine mit Sinn behaftete Abbildung dieser Erkenntnis kann aus dem nachfolgenden Bildabschnitt 20 entnommen werden.

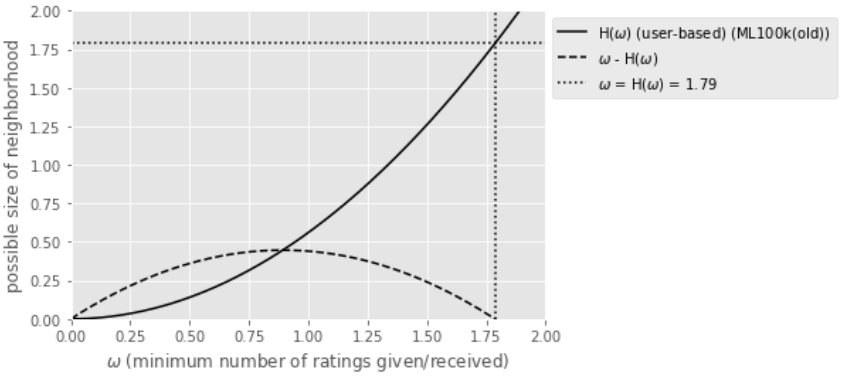


Abbildung 19: Fixpunkte der Differenzfunktion: $\omega - H(\omega)$

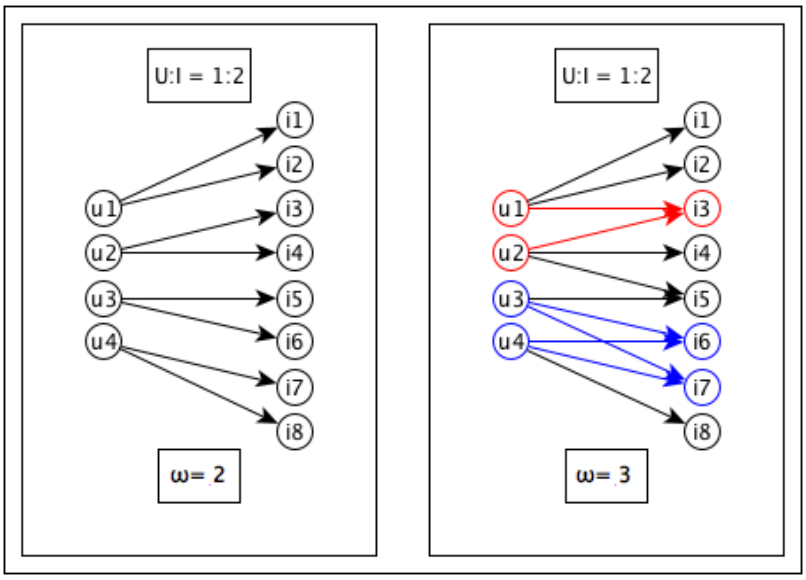


Abbildung 20: Entwicklung der möglichen Nachbarschaften mit steigendem ω

Diese Erkenntnis ermöglicht es für die Entwicklung eines *Recommender Systems*, eine Mindestanzahl an *Ratings* von einem spezifischen *User* zu fordern, um sicher zu stellen, dass das System einen nahezu konstanten Fehler erzeugt. Solche Anforderungen sind unter anderem in dem *Recommender System* von *Netflix* integriert. Auch könnte dieser Zusammenhang dazu genutzt werden, sogenannte *Cold-Start* oder *Long-Tail* Probleme von *collaborative-filtering Recommender* zu verbessern. Damit könnten künstliche *Ratings* für neue Objekte erzeugt werden, die dazu beitragen, dass das neue Objekt von k-*NN* Verfahren berücksichtigt wird. Darüber hinaus ist es durch diesen Zusammenhang möglich zu verstehen, dass es keine allgemeine Grenze gibt, ab der einer der beobachteten Fehler minimiert wird. Dies widerspricht der von Herlocker et al. (2002) aufgestellten These, dass die ideale Größe k einer Nachbarschaft für die meisten Fälle zwischen 20 und 50 liegt.

4.4 Matrix-factorization Verfahrensanalyse

Dieses Experiment betrachtet die drei am häufigsten verwendeten Verfahren der *matrix-factorization*. Zu ihnen zählen, im Bezug auf *Recommender Systeme*, die folgenden Verfahren:

1. *Single-Value-Decomposition (Funk-SVD)* (Paterek, 2007; Vozalis und Margaritis, 2007)
2. *Single-Value-Decomposition-Plus-Plus (SVD++)* (Koren, 2008)
3. *Non-negative-matrix-factorization (NMF)* (Zhang et al., 2006)

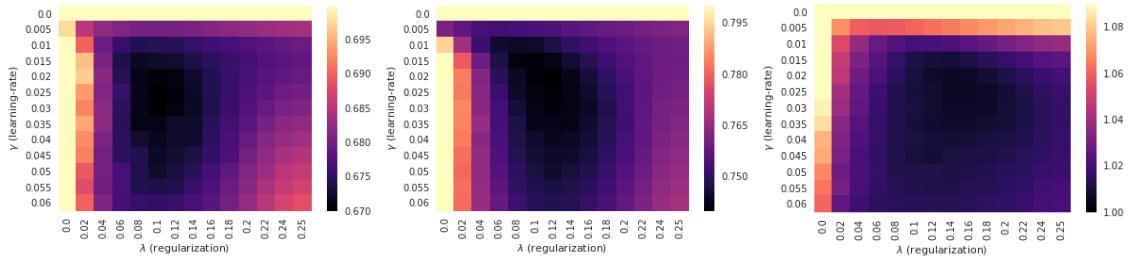
Die Kernidee der *matrix-factorization* beruht auf dem Grundsatz, die *Rating*-Matrix $\mathcal{R}$ in kleinere Matrizen $\mathcal{U}$ und $\mathcal{V}$ mit geringeren Dimensionalitäten zu zerteilen. Aus den so entstandenen Sub-Matrizen kann zu gegebenem Anlass die ursprüngliche Matrix $\mathcal{R}$ reproduziert werden: $\mathcal{R} = \mathcal{U}\Sigma\mathcal{V}^T$. Es ist jedoch wichtig zu verstehen, dass für das *Recommender-Problem* die klassische *Single-Value-Decomposition* nicht zur Vervollständigung von $\mathcal{R}$ beitragen kann. Die Verfahren *Funk-SVD*, *SVD++* und *NMF* können somit nicht als klassische *SVD*-Verfahren verstanden werden. Sie gelten vielmehr als *SVD*-ähnliche Verfahren. Dabei wird die Matrix $\mathcal{R} = \mathcal{U} \times \mathcal{I}$ in *User*- und *Item*-spezifische Vektoren p_u und q_i zerlegt. Es können dann die fehlenden *Ratings* mittels den *matrix-factorization*-Verfahren bestimmt werden. Das bekannteste der drei Verfahren ist das von Funk (2006) veröffentlichte *Funk-SVD*. Eine detaillierte Darstellung der Lösungsidee kann aus Abschnitt 1.3.10 entnommen werden. Ebenso kann die *stochastic-gradient-descent*-Lösung *(SGD)* aus dem dort festgehaltenen Algorithmus 4 entnommen werden. Die Veranschaulichung der *SGD*-Lösung dient dem idealen Verständnis, weshalb die in Abschnitt 1.2 und Abschnitt 1.3 genannten Arten der *Recommender Systeme* durch eine gemeinsame Methodengrundlage miteinander verbunden sind. Durch die Darstellung der *User*-spezifischen Vektoren p_u in Form einer Matrix, ist es möglich, einzelne *User*-Profile zu betrachten. Ein *collaborative-filtering* typisches Verfahren kann demnach in ein *content-based* Verfahren überführt werden. Dies gelingt, indem eben diese *user*-spezifischen Vektoren einzeln betrachtet werden.

Eine weitere bekannte Methode ist die *SVD++*. Dieses Verfahren gilt als eine Erweiterung des *Funk-SVD* (Koren, 2008). Im Gegensatz zu *Funk-SVD* werden von *SVD++* die impliziten *Ratings* der *User* berücksichtigt. Als Beispiele für implizite *Ratings* gelten: Anzahl Klicks pro Artikel, Dauer des Besuchs, Anzahl der gekauften Artikel. Für das *SVD++* Verfahren sind die impliziten *Ratings* durch die Anzahl der von *User* u bewerteten *Items* i in $\mathcal{I}_u$. Dazu wird für *SVD++* der *User*-Vektor p_u durch $p_u + |\mathcal{I}_u|^{-\frac{1}{2}} \sum_{i \in \mathcal{I}_u} y_i$ ersetzt. Hierbei ist y_i die implizite Aussage, ob ein *User* u *Item* i bewertet hat. Somit kann *SVD++* als eine *user-based* Variante des *Funk-SVD* verstanden werden.

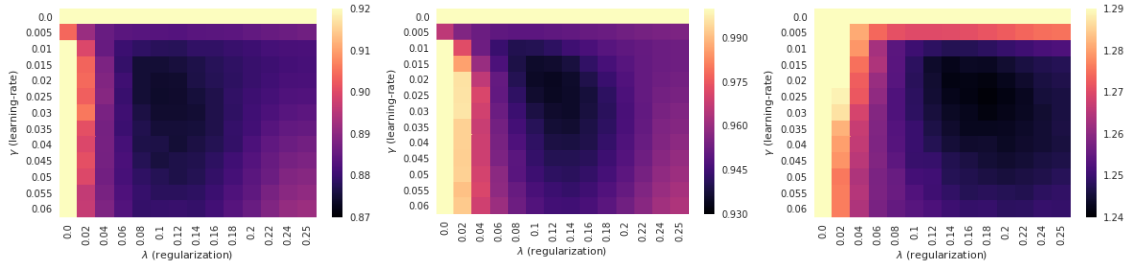
Die dritte und weniger häufig verwendete Methode ist die sogenannte *non-negative-matrix-factorization (NMF)*. Eine entsprechende Analyse wurde bereits von Zhang et al. (2006) verfasst.

4.5 Parameteroptimierung Funk-SVD

Der folgende Textabschnitt befasst sich mit dem Optimieren der Lern- und Regulierungsparameter γ und λ. Hierbei wird hauptsächlich auf das *Funk-SVD* Verfahren eingegangen. Anderweitige Verfahren wie beispielsweise *SVD++* können aufgrund ihrer hohen Laufzeit nicht verwendet werden. Die geringe Verbesserung der *Accuracy* steht in keinem Verhältnis mit dem damit verbundenen Zeitaufwand. Ein entsprechendes *Benchmark* für die Verfahren *SVD++* und *NMF* wurden bereits von Hug (2017) für die entsprechenden Datensätze *MovieLens100k(old)* und *MovieLens1M* durchgeführt. Das *Funk-SVD* Verfahren wird im Nachfolgenden in Kombination mit dem *MovieLens100k(old)*, *MovieLens100k(latest)* und dem *Webscope-R3(yahoo!)* Datensatz optimiert. Diese Datensätze eignen sich optimal, um die Auswirkung des *User* zu *Item* Verhältnisses zu untersuchen. Somit können ebenfalls Vergleiche zwischen den *item-* und user-based Verfahrensanalysen aus Abschnitt 4.1 und Abschnitt 4.2 gemacht werden. Ein wichtiger Aspekt des *Funk-SVD* ist dessen Parametrisierung. Für die Optimierung der Parameter γ und λ kann das sogenannte *grid-search* Verfahren verwendet werden. Dabei wird ein Gitter an möglichen Parametern vor der Analyse festgelegt. Dieses Gitter besteht aus den Mengen Λ und Γ. Das *grid-search* Verfahren trainiert dann den zu betrachtenden Algorithmus mit jedem möglichen Paar $(\lambda \in \Lambda, \gamma \in \Gamma)$. Die so trainierten Modelle werden anschließend mittels einer *k-fold cross-validation* geprüft. Dadurch entsteht eine Fehlermessung in Abhängigkeit von den Parametern γ und λ. Es können im Anschluss jene Parameter gewählt werden, für die der beobachtete Fehler minimal wird. In Abbildung 21 kann das Ergebnis einer Durchführung von *grid-search* in Kombination mit einer *5-fold cross-validation* betrachtet werden.



(a) *MAE MovieLens100k(latest)* (b) *MAE MovieLens100k(old)* (c) *MAE Webscope-R3(yahoo!)*



(d) *RMSE MovieLens100k(latest)* (e) *RMSE MovieLens100k(old)* (f) *RMSE Webscope-R3(yahoo!)*

Abbildung 21: Parameteroptimierung für *Funk-SVD*

In Abbildung 21 können optimale Intervalle für Parameter γ und λ des *Funk-SVD* beobachtet werden. Diese Intervalle sind durch ihre dunkle Färbung erkennbar. Es werden die überlappenden Intervalle gesucht, in denen sowohl der *MAE* als auch der *RMSE* minimiert wird. Diese Parameterintervalle garantieren eine Minimierung des *MAE* und des *RMSE* für den *Funk-SVD*. Aus Tabelle 12 können die Parameterintervalle und die dazugehörigen Fehlerwerte entnommen werden:

	γ	λ	MAE	RMSE	Fit-Zeit
MovieLens100k(old)	$0.015 \leq \gamma \leq 0.025$	$0.10 \leq \lambda \leq 0.16$	0.7288	0.9195	4.36
MovieLens100k(latest)	$0.015 \leq \gamma \leq 0.030$	$0.08 \leq \lambda \leq 0.14$	0.6569	0.8554	4.54
Webscope-R3(yahoo!)	$0.020 \leq \gamma \leq 0.035$	$0.12 \leq \lambda \leq 0.18$	0.9682	1.2059	13.60

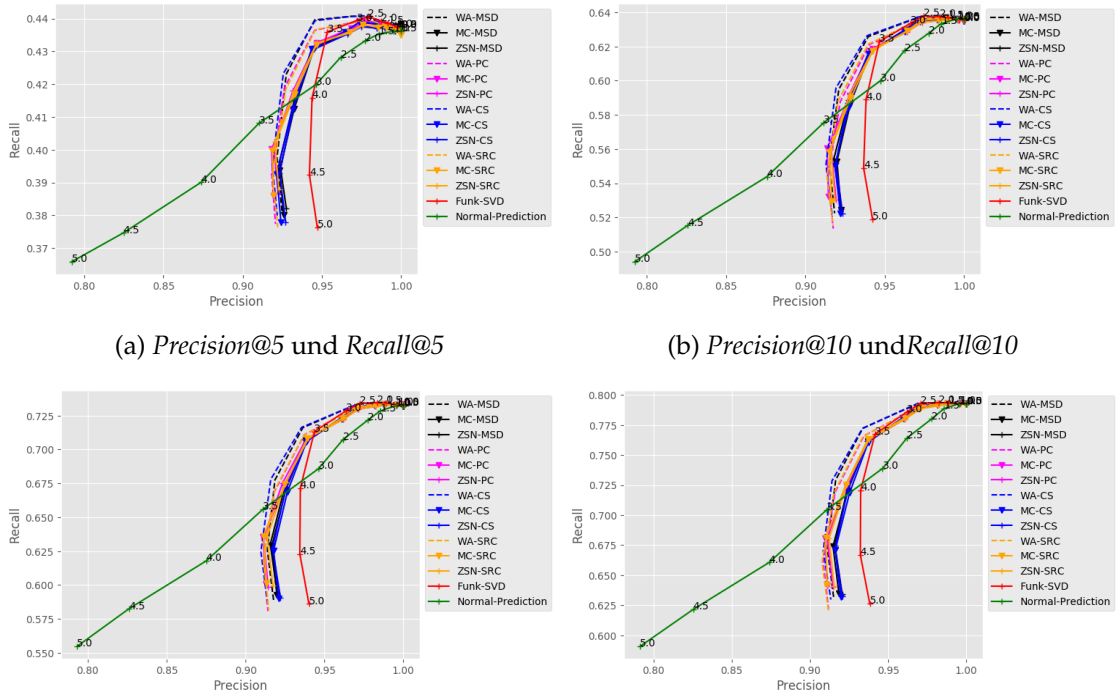
Tabelle 12: Optimierung der *Funk-SVD* Parameter

Werden die für *MovieLens100k(old)* und *MovieLens100k(latest)* erzielten Fehlerwerte aus Tabelle 10 und Tabelle 12 miteinander verglichen, so fällt folgendes auf: Das *Funk-SVD*-Verfahren schneidet, im Vergleich zu den *collaborative-filtering* Verfahren, deutlich besser ab. Jedoch zeigt sich auch, dass dieses Ergebnis nicht auf eine Anwendung des *Funk-SVD* auf dem *Webscope-R3(yahoo!)* Datensatz zutrifft. Vielmehr schneidet *Funk-SVD* deutlich schlechter als die *collaborative-filtering* Verfahren ab. Die erhöhten Fehlerwerte lassen sich durch die Initialisierung der Vektoren p_u und q_i erklären. Diese werden, wie aus Algorithmus 4 hervorgeht, aus einer Normalverteilung generiert. Auch die von Hug (2017) implementierte Lösung sieht diese Initialisierung vor. Wie bereits mehrfach erwähnt, entspricht die Verteilung der *Ratings* für den *Webscope-R3(yahoo!)* Datensatz annähernd einer Gleichverteilung. Die interne Verwendung einer Normalverteilung, zur Initialisierung der Vektoren, hat anscheinend zur Folge, dass Eingabegrößen, die keiner Normalverteilung entsprechen, vergleichsweise größere Fehler erzeugen. Hingegen lässt sich der erhöhte Zeitaufwand nicht auf dieses Problem zurückführen. Vielmehr ist dieser durch die dreimal so hohe Datenmenge des *Webscope-R3(yahoo!)* Datensatzes erklärbar. Vergleicht man die Zeiten aus Tabelle 12 miteinander, so ergibt sich auch ein dreimal so hoher Zeitaufwand für den, auf dem *Webscope-R3(yahoo!)* Datensatz, angewendeten *Funk-SVD*. Daher lässt sich das vergleichsweise schlechte Ergebnis des *Funk-SVD* lediglich durch die Eingabe einer gleichverteilten Zufallsgröße erklären.

4.6 Auswirkung der Accuracy auf die Precision- und Recall-Werte

Dieser Abschnitt untersucht die Abhängigkeit zwischen den *Precision*- und den *Recall*-Werten in Bezug auf die erzielte *Accuracy*. Es wird der Einfluss der θ_i-Schranken und deren Auswirkungen auf die obigen Werte betrachtet. Für dieses Experiment werden die in Abschnitt 4.1 und Abschnitt 4.2 untersuchten Verfahren in Kombination mit dem *MovieLens100k(latest)* Datensatz verwendet. Die Messwerte für die *Precision*- und *Recall*-Werte wurden mittels einer *5-fold cross-validation* erzeugt.

Die Abbildung 22 zeigt die Darstellung eines *Precision-Recall-Graphen*. Hierfür wurden die in Abschnitt 2.3 vorgestellten Darstellungsarten *GPR*, *Precision@k* und *Recall@k* , sowie *threshold-filtering* verwendet. Durch diese Form der Darstellungen können mit nur



(a) *Precision@5* und *Recall@5*

(b) *Precision@10* und*Recall@10*

(c) *Precision@15* und *Recall@15*

(d) *Precision@20* und *Recall@20*

Abbildung 22: *Precison-* und *Recall*-Graph

einem Blick mehrere Abhängigkeiten untersucht werden. Der erste und auch offensichtlichste Aspekt ist die Auswirkung der *Accuracy* auf die *Precison-* und *Recall*-Werte für jede θ_i-Schranke. Es zeigt sich, dass die Verfahren mit einer genaueren *Accuracy* bedeutend besser in den jeweiligen *Precision-* und *Recall*-Werten abschneiden. Diese Werte zeigen ebenfalls eine ähnliche Hierarchie. Somit schneidet das *Weighted-Average* Verfahren in Kombination mit jeder der betrachteten Distanzfunktionen schlechter ab, als die anderweitig zu betrachtenden Verfahren. Aus diesem Grund scheidet *Weighted-Average* als ein, für *Recommender Systeme*, ungeeignetes Verfahren aus. Zudem erzielt das *Mean-Centering* leicht bessere Werte. Werden die entsprechenden Fehler aus Tabelle 10 und Tabelle 12 mit den in Abbildung 22 gezeigten Kurven verglichen, so finden sich eindeutige Zusammenhänge zwischen der *Accuracy* und den *Precision-* und *Recall*-Werten. Mittels anschließender Betrachtung der *Precision-* und *Recall*-Kurven für die einzelnen *Funk-SVD*-Werte, zeigt sich auch hier die zuvor genannte Vermutung. Dieser Zusammenhang ist dadurch erklärbar, dass eine starke Abweichung bezüglich der $\hat{r}_{ui}$- zu r_{ui}-Werte vorliegt. Diese Werte lassen sich in der Messung der *Precision-* und *Recall*-Komponente aus Tabelle 5 wiederfinden. Zur Berechnung dieser Komponenten wird eine θ_i-Schranke vorausgesetzt. Diese Schranke dient zur Klassifikation der Relevanz. Ein Algorithmus mit entsprechend schlechter *Accuracy* erzeugt demnach eine hohe Abweichung zu den tatsächlichen Werten. Diese hohen Abweichungen kommen auch während der Messung der *Precision-* und *Recall*-Werte zustande.

Der zweite Aspekt behandelt die Untersuchung der θ_i-Schranke auf die *Precision-* und *Recall*-Werte. Wie aus der Abbildung 7 zu entnehmen liegt die durchschnittliche Bewertung

im *MovieLens100k(latest)* Datensatz bei $\overline{r} \approx 3,5$. Werden nun die Graphen in Abbildung 22 betrachtet, so zeigen diese, dass der *Recall*, also der Werte der ungeeigneten Vorschläge, bei θ_i drastisch abfällt. Der *Precision*-Wert $\theta_i = 3,5$ bleibt annähernd konstant. Ab dem Wert $\theta_i = 4,5$ stellt sich durchaus eine leichte Verbesserung ein. Dieses Phänomen lässt sich mittels der *Accuracy* erklären. Wie sich bei Abbildung 7 bereits erwiesen hat, liegt $\overline{r}$ bei ungefähr $3,5$. Die Algorithmen arbeiten demnach häufiger mit Werten aus diesem Zahlenbereich. Die Fehlerquote garantiert eine geringe Abweichung von $\overline{r}$. Demnach zeigt sich eine Verbesserung der *Precision* und eine Verschlechterung des *Recalls* bei diesem θ_i-Wert. Es kann festgehalten werden, dass $\overline{r}$ ein annehmbarer Richtwert für θ_i und demnach auch ein positiver Wert für den *Threshold* der Vorschlagsliste ist.

Im nächsten Abschnitt wird die Größe k der Vorschlagsliste untersucht werden. Diese Größe gilt als ein interessanter Analysepunkt, da nicht jeder Vorschlag an den *User* übermittelt wird. Häufig wird nur eine bestimmte Anzahl der *Top-N Recommendations* an den *User* vorgestellt. Laut Shani und Gunawardana (2011) gilt die Analyse der Größe k einer Vorschlagsliste als der wichtigste Aspekt in Bezug auf das *Information Retrieval*. In Abbildung 22 ist ersichtlich, dass sich die *Precision*- und *Recall*-Werte mit der Größe k einer Vorschlagsliste verändern. Ein solches Verhalten ist der *Precision-Recall*-Kurve der *Normal-Prediction* zu entnehmen. Der hier verwendete Algorithmus schätzt demnach die fehlenden Werte mittels Zufallszahlen. Diese Zahlen werden mit einer Normalverteilung $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ generiert. Der Mittelwert $\hat{\mu}$ und die Standartabweichung $\hat{\sigma}$ ergeben sich aus dem Trainingsdatensatz. Dabei ist $\hat{\mu} = \frac{1}{|\mathcal{R}_{train}|} \sum_{(u,i) \in \mathcal{R}_{train}} r_{ui}$ und $\hat{\sigma} = \sqrt{\sum_{(u,i) \in \mathcal{R}_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|\mathcal{R}_{train}|}}$. Der *Normal-Predictor* eignet sich besonders gut um Algorithmen und andere Einflüsse zu vergleichen und zu beobachten. Dies liegt daran, dass dieser *Normal-Predictor* annähernd neutrale *Precision*- und *Recall*-Werte erzeugt. Dieser Zusammenhang zeigt sich in Abbildung 22. Hier liegen die Werte im neutralen Bereich. Mit steigender Größe k verschlechtert sich der *Recall*, der Wert für die *Precision* bleibt dagegen annähernd konstant oder unterliegt einem leichten Anstieg. Ein solches Phänomen wurde bereits von Shani und Gunawardana (2011) erwähnt. Die Untersuchung der Größe k unterstreicht die Zusammenhänge aus den zuvor beschriebenen Analysepunkten. Das dort beschriebenen Verhalten findet sich in den Größen der Vorschlagslisten. Hierdurch kann die Konsistenz der Zusammenhänge positiv verstärkt werden. Es kann somit abschließend festgehalten werden, dass auch die Größe der Vorschlagsliste einen Einfluss auf die *Precision*- und *Recall*-Werte nimmt. Die Vergrößerung der Vorschlagsliste führt somit zu einer Verschlechterung der *Recall*-Werte. Dadurch kann es zu einer leichten Verbesserung der *Precision* kommen.

4.6.1 Résumé Accuracy

In diesem Kapitel wurden sowohl *neighborhood-based*, als auch *matrix-factorization* Ansätze in Bezug auf die von ihnen erzielte *Accuracy* und die damit einhergehenden Eigenschaften untersucht. Es zeigt sich für die *neighborhood-based* Verfahren, dass *Weighted-Average* in jeder möglichen Kombination mit den Distanzfunktionen am schlechtesten abschneidet. Es zeigt sich ebenfalls, dass die Distanzfunktionen einen geringen Einfluss auf die erzielte *Accuracy* der *Mean-Centering*- und der *Z-Score-Normalization* Verfahren haben. Dennoch kann festgehalten werden, dass die *Z-Score-Normalization* zu jeder Zeit leicht besser

abschneidet als das *Mean-Centering* Verfahren. Die Bedingung hierfür ist, dass beide Verfahren die gleiche Distanzfunktion haben. Sollte der verwendete Datensatz eine starke Abwandlung der Normalverteilung in den betrachteten Größen beinhalten, so werden, durch die Verwendung der *Spearman-Rank-Correlation*, die Fehler der beiden zuvor erwähnten Verfahren minimiert. Ein weiterer, durchaus erwähnenswerter Aspekt dieses Kapitels ist die Betrachtung des *Recommender-Problems* in Form eines bipartiten Graphen. Der bipartite Graph $k_{|\mathcal{U}|,|\mathcal{I}|} = (\mathcal{U} \cup \mathcal{I}, R)$ ermöglicht es, die Ergebnisse und Arbeitsweisen der k-*NN* Verfahren besser zu verstehen. Dank ihm kann der Zusammenhang zwischen der *Accuracy* und dem *User* zu *Item* Verhältnis erklärt werden. Es zeigt sich, dass dieses Verhältnis maßgeblich an der *Accuracy* beteiligt ist. Das Verhältnis bestimmt die minimale Anzahl derer *Ratings*, ab denen sich eine, für die Generierung fehlender *Ratings* wichtige und solide Nachbarschaften bildet. Mit diesem Zusammenhang lassen sich Voraussetzungen für den erfolgreichen Einsatz von *collaborative-filtering* und *content-based Recommendern* festlegen. Eine Voraussetzung die aus dem Graphen abgeleitet werden kann, ist eine Mindestanzahl an *Ratings*, die ein *User* vor der Nutzung des *Recommender Systems* abgeben muss, damit dieser adäquat arbeiten kann. Dieser Schwellenwert kann mit ω bezeichnet werden. Der Wert ω muss überschritten werden, um eine hinreichend gute Vernetzung und eine damit solide Nachbarschaft des *Users* zu gewährleisten. Ein solcher Ansatz findet sich beispielsweise auch bei *Netflix* und *Jester*. Im Vergleich mit den *matrix-factorization* Verfahren, wie dem bekannten *Funk-SVD*, schneiden die *neighborhood-based* Verfahren auf den *MovieLens100k*-Datensätzen schlechter ab. Werden jedoch die auf dem *Webscope-R3(yahoo!)* gewonnenen Fehlerwerte miteinander verglichen, so schneidet *Funk-SVD* für den MAE wesentlich schlechter als die *neighborhood-based* Verfahren ab. Demnach ist ein weiterer, bedenkenswerter Aspekt die Parameteroptimierung der vorgestellten Verfahren. In Tabelle 12 wurden die Fehler *MAE* und *RMSE* des *Funk-SVD* anhand der überlappenden Intervalle minimiert. Dies hat zur Folge, dass sowohl *MAE* als auch *RMSE* optimiert und minimiert werden. Es kann jedoch auch passieren, dass sich einer der beiden Fehler entgegengesetzt verhält. Demnach ist die Parametrisierung der *matrix-factorization* Verfahren problematisch. Beide Parameter γ und λ können an unterschiedlichen Stellen ihre tatsächlichen Minima erreichen. Daher sollte vor der Analyse geklärt werden, welcher der beiden Fehler für das spezifische *Recommender System* relevant ist. Zusätzlich hängen die Ergebnisse des *Funk-SVD* stark von dessen Implementierung ab. Im Gegensatz zu den *neighborhood-based* Verfahren kann es passieren, dass die Verteilung der Eingabegröße einen negativen Einfluss auf die so entstehenden Fehlerwerte nimmt und zu dessen Verschlechterung beiträgt. Dies kann beispielsweise dann eintreten, wenn die Vektoren p_u und q_i mittels einer Normalverteilung initialisiert werden. Das *Funk-SVD* erhält jedoch eine gleichverteilte Eingabegröße. Ein solcher Effekt kann bei genauerer Betrachtung des *Webscope-R3(yahoo!)* Datensatzes aus Tabelle 10 und Tabelle 12 entnommen werden. Demnach kann sich die Initialisierung und Optimierung der *matrix-factorization* Verfahren als durchaus problematisch herausstellen. Im Gegensatz dazu sind die *neighborhood-based* Verfahren durch die geringe Anzahl der zu optimierenden Parameter weniger anfällig für wechselnde Bedingungen. Für *neighborhood-based* Verfahren sollte jedoch nicht *Weighted-Average* verwendet werden. Diese *Prediction*-Generierung schneidet in jedem Fall deutlich schlechter als *Mean-Centering* und *Z-Score-Normalization* ab. Daher kann gesagt werden: Die Kombination aus den Verfahren *Mean-Centering* und *Z-Score-Normalization* mit den Distanzfunktionen *Mean-Squared-Distance*, *Cosine-Similarity* und *Pearson-* beziehungsweise *Spearman-Rank-Correlation* können in der erzielten *Accura-*

cy durchaus mit *Funk-SVD* gleichziehen. Vielmehr sind sie gegenüber den im Datensatz enthaltenen Verteilungen robust. Daher sind *neighborhood-based* Verfahren aufgrund ihrer Simplizität bezüglich der Nachvollziehbarkeit, sowie dem immer gleichen Verhalten der Fehler, unter wechselnder Eingabe, den *matrix-factorization* Verfahren, überlegen. Damit sollte eine Weiterentwicklung und Anwendung der *neighborhood-based* Verfahren in Erwägung gezogen werden.

5 Fazit

Die derzeitige Forschung befasst sich mit den unterschiedlichsten Verfahren aus dem Bereich der *collaborative-filtering* und *content-based Recommender*. Dieser Grundlagengebung entsprechend wurde sich auf die gemeinsamen Methodiken beider Bereiche konzentriert. Hierzu zählen *Weighted-Average*, *Mean-Centering*, *Z-Score-Normalization* und *Funk-SVD*. Der Fokus liegt jedoch nicht allein auf der Anwendung der oben genannten Verfahren, sondern vielmehr auch auf dem Formalisieren des *Recommender-Problems* und der dazugehörigen Messverfahren. Zur Untersuchung der für *Recommender Systeme* spezifischen Verfahren steht innerhalb des nicht-kommerziellen Sektors nur die von Hug (2017) entwickelte *Surprise-Library* zur Verfügung. *Surprise* wird innerhalb der nicht-kommerziellen Forschung durchgängig verwendet. Bekannte Entwickler wie Kane (2018) empfehlen diese *Library* explizit. Kane (2018) arbeitete 9 Jahre lang für *Amazon.com* und *IMDb.com* an der Entwicklung und Erforschung von *Recommender Systemen*. Um in diesen Bereichen den Wissensstand konstruktiv zu verbessern, wurde im Rahmen dieser Bachelorarbeit die *Surprise-Library* erweitert. Dabei wurde in *Surpise* die Fehlermessung *Mean-Squared-Error (MSE)* und die Distanzmessung *Spearman-Rank-Correlation (SRC)* integriert. Auch wurde für die *Surpise-Library* ein spezifisches *Dockerimage* entwickelt. Dadurch soll die Entwicklung der *Library* vereinfacht werden.

Das zweite Ergebnis dieser Bachelorarbeit umfasst die Verbesserung der von Shani und Gunawardana (2011) vorgestellte Formel zur Berechnung der von Yao (1995) vorgestellten *Normalized Distance-based Performance Measure (NDPM)*. Um diese Verbesserung erzielen zu können, wurde die von Shani und Gunawardana (2011) aufgestellte Rechenvorschrift implementiert und überprüft. Hierbei fiel auf, dass die berechnete Anzahl der Fehlstellungen C^- in den Referenz- und Vorschlagslisten falsch berechnet wurden. Zudem fiel der errechnete Wert für *NDPM* im Vergleich mit dem tatsächlichen Wert immer doppelt so groß aus. Dies lag daran, dass die von Shani und Gunawardana (2011) aufgestellte Rechenvorschrift nicht exakt in die von Yao (1995) vorgestellte Formel überführt wurde. Das von Yao (1995) vorgestellte Verfahren betrachtet im Nenner den Wert $2 \cdot C^i$, Shani und Gunawardana (2011) ignorieren jedoch diesen Faktor 2. Die so verbesserte Rechenvorschrift kann somit im Sinne des von Yao (1995) vorgestellten *NDPM* berechnet werden.

Um konkrete Ergebnisse zu erhalten, wurden in dieser Arbeit die drei Datensätze *MovieLens100k(old)*, *MovieLens100k(latest)* und der *Webscope-R3(yahoo!)* verwendet und analysiert. Folgendes viel hierbei auf: Die *MovieLens*-Datensätze eignen sich für *user-based* Ansätze und der *Webscope-R3* Datensatz bietet sich zur Untersuchung der *item-based* Verfahren an. Grundlage hierfür war die von Desrosiers und Karypis (2011) aufgestellte Schätzfunktion zur Berechnung der zu erwartenden Größe einer Nachbarschaft unter der Eingabe einer durchschnittlichen Anzahl an vergebenen *Ratings* pro *User*. Diese Anzahl konnte aus den Datensätzen entnommen werden.

Weitere Analyseschritte der *collaborative-filtering* Verfahren zeigten mittels der zuvor genannten Methodiken eine übergreifende Gemeinsamkeit. Bis zu einem bestimmten ω-Wert schienen die Fehler *MAE* und *RMSE* zu fallen. Eine Überschreitung der Werte hatte zur Folge, dass eine Stabilisierung der Fehler erkennbar wurde. Die Untersuchung dieser Stabilisierung zeigte, dass ab diesem ω die Nachbarschaft eine optimale Größe erreicht

hat und den Algorithmen diesbezüglich eine ausreichend große Bewertungsgrundlage zur Verfügung steht.

Das *Recommender-Problem* wurde im Anschluss an diese Erkenntnis in einen bipartiten Graphen überführt um eine entsprechende Schätzfunktion der Nachbarschaft zu erzeugen. Die Schätzfunktion ermöglicht es, den Zusammenhang zwischen der optimalen Größe k einer Nachbarschaft und dem *User* zu *Item* Verhältnis des jeweiligen Datensatzes herzustellen. Das Phänomen, dass alle *collaborative-filtering* Verfahren bis zu einem bestimmen Punkt ω fallen, wurde bereits von Herlocker et al. (2002) beobachtet. Auch er beschrieb das Phänomen und schlussfolgerte, dass eine optimale Größe einer Nachbarschaft stets zwischen 20 und 50 Objekten beinhalte. Herlocker et al. (2002) unterzog seinem Ergebnis jedoch keiner weiteren Analyse um etwaige Zusammenhänge zu beantworten. Die hier vorliegende Bachelorarbeit knüpft an diese ersten Erkenntnisfundamente an und erweitert diese.

In dieser Bachelorarbeit wurde festgestellt, dass das tatsächliche Optimum der Nachbarschaft bereits nach Erreichen des *User* zu *Items* Verhältnisses vorliegt. Hierfür erfolgte eine Untersuchung der Schätzfunktion auf Fixpunkte. Die errechneten Werte stimmten mit den Werten des *User* zu *Item* Verhältnis der spezifischen Datensätze überein. Durch eine Betrachtung des bipartiten Graphen konnte dieser Zusammenhang nachvollzogen werden. Es gibt demnach keine allgemeingültige Grenze für die optimale Nachbarschaft eines *collabortive-filtering Recommender Systems*. Vielmehr ist diese stets von dem *User* zu *Item* Verhältnis des betrachteten Datensatzes abhängig.

Da sich nach dem Erreichen der ω-Werte die Fehler stabilisieren, kann eine Untersuchung der *Accuracy* ermöglicht werden. Es zeigt sich, dass *Weighted-Average* in jedem Fall schlechter, als die Verfahren *Mean-Centering* und *Z-Score-Normalization*, abschneidet. Besonders gut fielen die *Accuracy*-Werte für die Kombinationen aus den Verfahren *Mean-Centering* und *Z-Score-Normalization* in Kombination mit den Distanzfunktionen *Mean-Squared-Distance*, *Cosine-Similarity* und *Pearson-* beziehungsweise *Spearman-Rank-Correlation* aus. Diese erzielten im direkten Vergleich mit den vom *Funk-SVD* erzeugten Fehlern ähnliche Ergebnisse. Obwohl das *Funk-SVD* Verfahren leicht besser abschnitt, konnten mittels der Eingabe von gleichverteilten Zufallsgrößen die resultierenden Fehler erhöht werden. Für weitere Untersuchungen bietet sich demnach die Weiterentwicklung der *collaborative-filtering* Verfahren *Mean-Centering* und *Z-Score-Normalization* an. Diese sind im Gegensatz zu *Funk-SVD* robust gegenüber den unterschiedlichen Verteilungen der Eingaben. Eine Kombination aus *matrix-factorization* und *collaborative-filtering* Verfahren kann die Vorteile beider Bereiche vereinen. Das *Weighted-Average* Verfahren sollte hiervon jedoch ausgeschlossen werden.

6 Future Work

Im letzten Abschnitt dieser Bachelorarbeit wird auf die zukünftigen Entwicklungsschritte verwiesen. Im Fokus dieser Betrachtung stehen die, in dieser Bachelorarbeit gewonnenen Erkenntnisse:

Zunächst wird der Blick auf den Bereich der Hybridisierung von *neighborhood-based* und *matrix-fatorization* Verfahren gerichtet. Hierzu ist es durchaus denkbar, dass die Verfahren *Z-Score-Normalization* und *Mean-Centering* in einer *Pre-Process-Phase* die wichtigsten Elemente aus einem Datensatz extrahieren und für *Funk-SVD* bereitstellen. Es könnten ebenso verschiedene Implementierungen des *Funk-SVD* untersucht werden. Insbesondere sollte hier der Fokus auf die Initialisierung der Vektoren p_u und q_i gelegt werden. Für einen solchen Initialisierungsschritt sollten verschiedene Verteilungen ausprobiert und in Betracht gezogen werden. Mittels solcher Untersuchungen kann somit festgestellt werden, ob und in wie weit *Funk-SVD* gegenüber wechselnder Verteilungen von Eingabegrößen robust ist.

Des Weiteren sollte an der Weiterentwicklung der *Suprise-Library* gearbeitet werden. Die aktuelle Version umfasst viele Aspekte der *collaborative-filtering* Algorithmen, jedoch fehlt es an spezielleren Algorithmen aus dem Bereich der *model-based Recommender*. Es wäre hier durchaus denkbar, dass zusätzliche lernfähige Verfahren aus dem Bereich *Machine-Learning* implementiert werden. Einige Beispiele hierfür sind:

1. *Restricted Boltzmann Machines (RBM)* (Salakhutdinov et al., 2007)
2. *Simple Bayesian Classifier* (Miyahara und Pazzani, 2000)
3. *Support Vector Machines* (Xia et al., 2006)

Unter anderem stellt die *Suprise-Library* die grundlegenden Messtechniken wie *RMSE* und *MAE* zur Verfügung. Um künftige Forschungsarbeiten mit dieser *Libary* vielseitiger zu gestalten, sollte die *Suprise-Library* mit Verfahren aus den Bereichen *Information-Retrieval* und *Accuracy-Measure* erweitert werden. Dazu könnten die folgenden Verfahren implementiert werden (Chen, 2017):

1. *Mean Reciprocal Rank (MRR)*
2. *Normalized Discounted Cumulative Gain (nDCG)*
3. *Precision* und *Recall*
4. *F-Measure*

Aufgrund meiner zweijährigen Arbeit an dem dialogbasierten Argumentationssystem *D-BAS*, welches an dem Lehrstuhl für *Technik sozialer Netzwerke* der Heinrich-Heine Universität Düsseldorf entwickelt wird, soll diese Arbeit dazu beitragen ein entsprechendes *neighborhood-based Recommender System* zu implementieren. Im Vergleich zu anderweitigen Diskussionsplattformen besitzt *D-BAS* den Vorteil etwaige Auseinandersetzungen in

Form eines Diskussions-Graphen darzustellen. Zudem vertreten die Entwickler die Meinung, dass die Mitglieder dieser *Community* einer Selbstverwaltung unterliegen sollten (Krauthoff et al., 2018). Um die Mitglieder dieser Vereinigung miteinander zu vernetzen und Diskussionen zu gestalten, muss eine Vielzahl an auserwählten Vorschlägen an die *User* herangetragen werden. Ein früherer Teil meiner Arbeit beschäftigte sich mit der semantischen Textanalyse der in *D-BAS* enthaltenen Texte und den daraus resultierenden Vorschlägen. Dies erfolgte mittels *Elasticsearch* (Elasticsearch, 2018). Da die, in dieser Bachelorarbeit vorgestellten *collaborative-filtering* Verfahren mit virtuellen *Communitys* arbeiten und sich Graphstrukturen zu Nutze machen, liegt es nahe, solche Verfahren auch für die Entwicklung eines *Recommender Systems* zu verwenden. Dieser *collaborative-filtering Recommender* richtet sich dann nach denen von *D-BAS* vorgesehen Strukturen. Demnach sollte für das dialogbasierte Argumentationssystem *D-BAS* ein *collaborative-filtering Recommender* entwickelt werden. Geeignete Entwicklungstools wären hierfür *Recombee* oder die graphenbasierte Datenbank *Neo4j* (Recombee, 2018; Neo4j, 2018). Da in *D-BAS* keine expliziten *Ratings* vergeben werden können, muss hierfür auf implizite *Ratings* (semantische Ähnlichkeiten/Meinungen) zurückgegriffen werden. Diese Ähnlichkeiten können dafür genutzt werden, um *neighborhood-based* Verfahren zu verwenden. Um die semantische Analyse zu verbessern, können die von Liebeck et al. (2017) untersuchten Verfahren zur Bestimmung von semantischen Ähnlichkeiten und Meinungsextraktionen verwendet werden.

Ein doch sehr wichtiger und oftmals vergessener Aspekt ist folgender: Der *User* steht stets im Mittelpunkt des *Recommender-Problems*. Durch die rein mathematische Betrachtung und Evaluierung der Algorithmen kann keine entsprechende Aussage darüber getroffen werden, wie und vor allem ob ein *User* mit dem resultierenden System zufrieden ist. Zukünftige Forschungsarbeiten sollten sich daher auch mit der Auswirkung eines verwendeten Verfahrens bezüglich der Wahrnehmung des *Users* auseinandersetzen.

References

Ajay Agarwal und Minakshi Chauhan (2017). „Similarity Measures used in Recommender Systems: A Study“. In: *International Journal of Engineering Technology Science and Research*. IJETSR, S. 619–626.

Riccardo Bambini, Paolo Cremonesi und Roberto Turrin (2011). „A Recommender System for an IPTV Service Provider: a Real Large-Scale Production Environment“. In: *Recommender Systems Handbook*. Hrsg. von P.B. Kantor, F. Ricci, L. Rokach und B. Shapira. Springer, S. 299–331.

David Barber (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press, S. 291–292.

John S. Breese, David Heckerman und Carl Kadie (1998). *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*. Morgan Kaufmann Publishers Inc., S. 43–52.

Giuseppe Carenini (2005). „User-Specific Decision-Theoretic Accuracy Metrics for Collaborative Filtering“. In: *Workshop: Beyond Personalization*. University of Minnesota.

Mingang Chen (2017). „Performance Evaluation of Recommender Systems“. In: *International Journal of Performability Engineering*. Bd. 13. ELSEVIER, S. 1246–1256.

James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston und Dasarathi Sampath (2010). „The YouTube Video Recommendation System“. In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. ACM, S. 293–296.

Christian Desrosiers und George Karypis (2011). „A Comprehensive Survey of Neighborhood-Based Recommendation Methods“. In: *Recommender Systems Handbook*. Hrsg. von P.B. Kantor, F. Ricci, L. Rokach und B. Shapira. Springer, S. 107–144.

Elasticsearch (2018). *About Elasticsearch*. https://www.elastic.co/. (Besucht am 18. 12. 2018).

Simon Funk (2006). *Netflix Update: Try This at Home*. https://sifter.org/simon/journal/20061211.html. (Besucht am 18. 12. 2018).

Carlos A. Gomez-Uribe und Neil Hunt (2015). „The Netflix Recommender System: Algorithms, Business Value, and Innovation“. In: *ACM Trans. Manage. Inf. Syst.* Bd. 6. 4. ACM, 13:1–13:19.

GroupLens (1998). *MovieLens100k(old)*. https://grouplens.org/datasets/movielens/100k/. (Besucht am 18. 12. 2018).

GroupLens (2018). *MovieLens100k(latest)*. https://grouplens.org/datasets/movielens/100k/. (Besucht am 18. 12. 2018).

Jon Herlocker, Joseph Konstan und John Riedl (2002). „An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms“. In: *Information Retrieval*. Bd. 5. DBLP, Universität Trier, S. 287–310.

Nicolas Hug (2017). *Surprise, a Python library for recommender systems*. http://surpriselib.com. (Besucht am 18. 12. 2018).

Frank Kane (2018). *About Frank Kane*. https://sundog-education.com/about/. (Besucht am 18. 12. 2018).

Jussi Karlgren (1990). *An algebra for recommendations : Using reader data as a basis for measuring document proximity*. SYSLAB technical reports 179. Department of Computer und Systems Sciences, Stockholm University, S. 1–11.

Jussi Karlgren (2017). *A digital bookshelf: original work on recommender systems*. URL: https://jussikarlgren.wordpress.com/2017/10/01/a-digital-bookshelf-original-work-on-recommender-systems/ (besucht am 20. 10. 2018).

Yehuda Koren (2008). „Factorization meets the neighborhood: A multifaceted collaborative filtering model“. In: *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'08)*. ACM, S. 426–434.

Yehuda Koren und Robert Bell (2015). „Advances in Collaborative Filtering“. In: *Recommender Systems Handbook*. Hrsg. von P.B. Kantor, F. Ricci, L. Rokach und B. Shapira. Springer, S. 145–184.

T. Krauthoff, C. Meter, M. Baurmann, G. Betz und M. Mauve (2018). „D-BAS - A dialog-based online argumentation system“. In: *7th International Conference on Computational Models of Argument, COMMA 2018; Warsaw; Poland; 12 September 2018 through 14 September 2018*. Bd. 305. IOS Press, S. 325–336.

Matthias Liebeck, Katharina Esau und Stefan Conrad (2017). „Text Mining für Online-Partizipationsverfahren: Die Notwendigkeit einer maschinell unterstützten Auswertung“. In: *HMD Praxis der Wirtschaftsinformatik*. Bd. 54. Springer, S. 544–562.

G. Linden, B. Smith und J. York (2003). „Amazon. com recommendations: Item-to-item collaborative filtering“. In: *Internet Computing*. Bd. 7. IEEE, S. 76–80.

Koji Miyahara und Michael J. Pazzani (2000). „Collaborative Filtering with the Simple Bayesian Classifier“. In: *Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence*. PRICAI'00. Springer, S. 679–689.

Neo4j (2018). *About Neo4j*. https://neo4j.com. (Besucht am 18. 12. 2018).

Arkadiusz Paterek (2007). „Improving regularized singular value decomposition for collaborative filtering“. In: *Proceedings of KDD Cup and Workshop*. ACM, S. 39–42.

Recombee (2018). *About Recombee*. https://www.recombee.com/about-us.html. (Besucht am 18. 12. 2018).

J Rocchio (1971). „Relevance Feedback in Information Retrieval“. In: *The SMART Retrieval System: Experiments in Automated Document Processing*. Prentice-Hall Inc., S. 313–323.

Neil Rubens, Dain Kaplan und Masashi Sugiyama (2011). „Active Learning in Recommender Systems“. In: *Recommender Systems Handbook*. Hrsg. von P.B. Kantor, F. Ricci, L. Rokach und B. Shapira. Springer, S. 809–846.

Ruslan Salakhutdinov, Andriy Mnih und Geoffrey Hinton (2007). „Restricted Boltzmann Machines for Collaborative Filtering“. In: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. ACM, S. 791–798.

G. Shani und A. Gunawardana (2011). „Evaluating Recommendation Systems.“ In: *Recommender Systems Handbook*. Hrsg. von P.B. Kantor, F. Ricci, L. Rokach und B. Shapira. Springer, S. 257–294.

B. Sheth und P. Maes (1993). „Evolving agents for personalized information filtering“. In: *Proceedings of 9th IEEE Conference on Artificial Intelligence for Applications*. IEEE Internet Computing, S. 345–352.

Patricia Victor, Martine De Cock und Chris Cornelis (2011). „Trust and Recommendations“. In: *Recommender Systems Handbook*. Hrsg. von P.B. Kantor, F. Ricci, L. Rokach und B. Shapira. Springer, S. 645–675.

M.G. Vozalis und K.G. Margaritis (2007). „Using SVD and demographic data for the enhancement of generalized Collaborative Filtering“. In: *Information Sciences*. Bd. 177. 15. ELSEVIER, S. 3017–3037.

Joost de Wit (2008). *Evaluating recommender systems : an evaluation framework to predict user satisfaction for recommender systems in an electronic programme guide context*. TNO Information und Communication Technology, University of Twente, S. 25–55.

Zhonghang Xia, Yulin Dong und Guangming Xing (2006). „Support Vector Machines for Collaborative Filtering“. In: *Proceedings of the 44th Annual Southeast Regional Conference*. ACM-SE 44. ACM, S. 169–174.

Yahoo!-Research (2018). *Yahoo! Webscope dataset ydata-ymusic-user-artist-ratings-v1*$_0$. https://webscope.sandbox.yahoo.com/. (Besucht am 18. 12. 2018).

Y. Y. Yao (1995). „Measuring retrieval effectiveness based on user preference of documents“. In: *Journal of the American Society for Information Science*. Bd. 46. 2. Wiley-Blackwell, Hoboken, S. 133–145.

Sheng Zhang, Weihong Wang, James Ford und Fillia Makedon (2006). „Learning from Incomplete Ratings Using Non-negative Matrix Factorization“. In: *Proceedings of the Sixth SIAM International Conference on Data Mining*. Academic Press Inc., S. 548–552.

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis