Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Vorschlagen von Datenvalidierungsmethoden für IoT-Daten mithilfe von Kontextmodellen

Alexander Aßenmacher

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. rer. nat. habil. Holger Schwarz
Betreuer/in:	Daniel Del Gaudio, M.Sc.
Beginn am:	12. Juni 2023
Beendet am:	12. Dezember 2023

Kurzfassung

Das Internet of Things wird mittlerweile in vielen Bereichen wie Smart Homes, Smart Hospitals oder der Industrie 4.0 eingesetzt. Mit Hilfe von modellgetriebener Entwicklung können auch technisch unerfahrene Anwender eigene IoT-Anwendungen erstellen. Die von den IoT-Geräten dieser Anwendungen generierten Daten können jedoch aufgrund von Defekten oder Übertragungsfehlern fehlerhafte Werte enthalten. Da es aufgrund der großen Datenmengen von IoT-Anwendungen mühsam sein kann, die Daten manuell zu überprüfen, wurden viele Methoden zur automatischen Datenbereinigung entwickelt. Technisch unerfahrene Anwender sind jedoch nicht immer in der Lage, geeignete Datenvalidierungsmethoden auszuwählen. Kontextmodelle können verwendet werden, um diese IoT-Anwendungen zu modellieren und ihren Aufbau darzustellen. Ziel dieser Arbeit ist es daher, geeignete Datenvalidierungsmethoden auf Basis von Kontextmodellen vorzuschlagen. Dazu wird untersucht, wie ein Kontextmodell von Recommender Systemen genutzt werden kann und welche Recommender Systeme sich für den Vorschlag von Datenvalidierungsmethoden eignen. Die Recommender Systeme werden prototypisch implementiert und mithilfe dieses Prototyps wird die Performance der Recommender Systeme an einem Datensatz und verschiedenen Kontextmodellen gemessen. Die Evaluierung zeigt, dass die Vorschläge der Recommender Systeme besser sind als die eines Benutzers, der aufgrund seiner Unerfahrenheit zufällige Methoden auswählt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

In den letzten Jahren ist die Anzahl der IoT-Geräte kontinuierlich gestiegen. Die Anwendungsbereiche des Internet of Things (IoT) sind vielfältig und die Technologie wird unter anderem für Smart Mobility, Smart Homes, im Gesundheitswesen [KK20] oder in der Industrie 4.0 [GKT19] eingesetzt. Eine IoT-Anwendung könnte beispielsweise aus mehreren Temperatur- und Feuchtigkeitssensoren in einer Wohnung bestehen. Die Messdaten der Sensoren können dann zur Analyse von Schimmelbildung genutzt werden.

Mit Hilfe von grafischen Benutzeroberflächen können nun auch technisch unerfahrene Benutzer ohne große Vorkenntnisse modellgetriebene IoT-Anwendungen erstellen. Die erstellten IoT-Anwendungen können mit einem Kontextmodell wie [DABS22] dargestellt werden. Für die Analyse der Daten, z.B. bei Schimmelbildung, ist es notwendig, dass die Daten so wenig Fehler wie möglich enthalten. Die von den Sensoren gemessenen Daten können jedoch Fehler enthalten, die z.B. durch defekte Sensoren oder Fehler bei der Datenübertragung entstehen. Durch die breite Anwendung des Internet of Things entstehen zudem immer größere Datenmengen [MNG+17]. Aufgrund der großen Datenmengen ist die manuelle Fehlererkennung sehr aufwändig. Um die Fehlererkennung zu automatisieren, wurden viele verschiedene Methoden entwickelt, die unterschiedliche Fehlertypen wie z.B. Ausreißer erkennen können [Smi20]. Technisch unerfahrene Anwender sind jedoch oft nicht in der Lage, geeignete Methoden zur Datenvalidierung auszuwählen. Zudem gibt es keinen Algorithmus zur Fehlererkennung, der für alle Datensätze gut funktioniert. Die Auswahl eines geeigneten Algorithmus erfordert ein gutes Verständnis der Domäne, ein Verständnis der vorliegenden Daten und ein gutes Wissen über die verfügbaren Methoden und ihre Anwendungsgebiete [CYW+23], was nicht alle Benutzer besitzen.

Recommender Systeme versuchen, eine kleine Menge passender Produkte für einen Nutzer auszuwählen, um ihm die Entscheidung zu erleichtern [WZW+18]. Dabei gibt es Recommender Systeme für verschiedene Anwendungsgebiete wie z.B. E-Commerce, Online-Nachrichten und Social Media [HLZ+17]. Die Recommender Systeme können dabei auf unterschiedlichen Daten arbeiten, wie z.B. Bewertungen [KBV09], Knowledge Graphen [WZW+18] oder Textbeschreibungen [MMIP14]. Das Kontextmodell enthält Informationen über die IoT-Anwendung, die für die Auswahl einer geeigneten Datenvalidierungsmethode relevant sein können.

Ziel dieser Arbeit ist es, auf Basis von Kontextmodellen [DABS22], welche die IoT-Umgebungen repräsentieren, geeignete Datenvalidierungsmethoden für IoT-Anwendungen vorzuschlagen. Dazu soll zunächst ein Konzept entwickelt werden, wie das Kontextmodell als Input für Recommender Systeme dienen kann. Anschließend sollen geeignete Recommender Systeme zur Vorhersage von Datenvalidierungsmethoden vorgestellt und prototypisch implementiert werden. Abschließend sollen die vorgestellten Recommender Systeme an einem Datensatz evaluiert werden.

Die Arbeit ist wie folgt aufgebaut. Zunächst werden in Kapitel 2 die Grundlagen zu Kotextmodellen, Recommender Systemen und anderen relevanten Konzepten dieser Arbeit gegeben. Anschließend werden in Kapitel 3 verwandte Arbeiten vorgestellt, die ähnliche Ziele wie diese Arbeit verfolgen. In

Kapitel 4 wird das Konzept dieser Arbeit vorgestellt. Dabei wird unter anderem beschrieben, wie die Kontextmodelle als Grundlage zum Vorschlagen von Datenvalidierungsmethoden verwendet werden. Außerdem wird beschrieben, welche Art von Recommender Systemen verwendet wird und wie das für diese Arbeit erstellte System aussieht. In Kapitel 5 wird dann die prototypische Implementierung des im Konzept erarbeiteten Systems beschrieben. Das prototypisch implementierte System wird dann in Kapitel 6 evaluiert. Dabei werden die verwendeten Metriken und die Performance des Systems vorgestellt. Zuletzt wird eine Zusammenfassung in Kapitel 7 gegeben, gefolgt von einem Ausblick auf mögliche Erweiterungen und Weiterentwicklungen.

2. Grundlagen

In diesem Kapitel werden die Grundlagen dieser Arbeit vorgestellt. Zunächst wird in Abschnitt 2.1 erklärt, was das Internet of Things ist. In Abschnitt 2.2 werden die Grundlagen von Knowledge Graphen und RDF erläutert. Anschließend wird in Abschnitt 2.3 beschrieben, was ein Kontextmodell ist. Die Grundlagen von Datenvalidierungsmethoden werden in Abschnitt 2.4 vorgestellt. Im darauf folgenden Abschnitt 2.5 werden die Grundlagen von Recommender Systemen vorgestellt, die dazu verwendet werden, geeignete Datenvalidierungsmethoden vorzuschlagen. Abschließend werden in Abschnitt 2.6 verschiedene Word Embedding Methoden vorgestellt, die zur Umwandlung von Wörtern in Vektoren verwendet werden.

2.1. Internet of Things

Der Begriff Internet of Things wurde zum ersten Mal von Kevin Ashton verwendet, um ein System zu beschreiben, in dem das Internet durch ubiquitäre Sensoren mit der physischen Welt verbunden ist [GBB18]. Das Internet of Things (IoT) ist ein Konzept, das Umgebungen beschreibt, in denen verschiedene Objekte/Dinge über verschiedene Arten von Verbindungen kommunizieren und so neue Anwendungen ermöglichen. Das Ziel des IoT ist es, dass sich beliebige Dinge zu jeder Zeit miteinander verbinden und miteinander kommunizieren können [VF13].

Genauer gesagt beschreibt das Internet of Things ein Netzwerk, das aus verschiedenen physischen Objekten wie Geräten, Messgeräten, Fahrzeugen oder anderen Gegenständen mit eingebauter Elektronik besteht und es diesen Geräten ermöglicht, untereinander Daten auszutauschen oder Daten zu sammeln. Darüber hinaus bietet das Internet of Things die Möglichkeit, Daten von den angeschlossenen Geräten zu empfangen und diese fernzusteuern. Die Kommunikation kann dabei über bereits bestehende Netzwerkinfrastrukturen erfolgen [GBB18]. Mögliche Übertragungswege sind beispielsweise Radio Frequency Identification (RFID) oder Wireless Sensor Networks (WSNs) [SL11]. Sie ermöglichen eine direkte Interaktion zwischen der physischen Welt und verschiedenen Computersystemen. In einem IoT-System sind die einzelnen Geräte eindeutig identifizierbar [GBB18]. Das Internet of Things wird in vielen Bereichen eingesetzt. Einige dieser Anwendungsgebiete sind beispielsweise das Gesundheitswesen, das Transportwesen, die Automobilindustrie [GBB18], Smart Traffic Systems, Smart Homes oder Smart Agriculture [FWM+15]. Im Bereich der Smart Homes können die Benutzer besipielsweise ihre Heizung oder das Licht über ihr Handy steuern, wobei die Kommunikation häufig über WLAN erfolgt.

2.2. Knowledge Graphen

Knowledge Graphen verwenden Graphen, um unterschiedliches Wissen für Anwendungen verfügbar zu machen. [HBC+21] definiert einen Knowledge Graphen als einen Graphen aus Daten, der Wissen aus der realen Welt widerspiegelt. Um dieses Wissen abzubilden, wird ein heterogener Graph verwendet [CSY+21; GZQ+20]. Das bedeutet, dass die Knoten und Kanten aus verschiedenen Typen bestehen können. Die Knoten repräsentieren die Entitäten und die Kanten die Beziehungen zwischen diesen Entitäten [CSY+21; HBC+21; SFBM23]. Der Knowledge Graph enthält somit eine starke Ausdruckskraft, da er verschiedene Attribute einer Entität enthalten kann. Diese können durch das Verfolgen der ausgehenden Kanten einer Entität gewonnen werden. Darüber hinaus ist es möglich, durch einen Knowledge Graphen auch den Zusammenhang einer Entität zu anderen Entitäten zu erhalten [GZQ+20]. Ein Modell zur Beschreibung von Knowledge Graphen ist das Resource Description Framework (RDF) [HBC+21].

RDF ist ein Standard des W3C, mit dem semantische Netze erstellt werden können. Da RDF es nicht erlaubt, eigene Konzepte zu definieren, kann es mit Hilfe der Web Ontology Language (OWL) erweitert werden, die es dem Benutzer erlaubt, eigene Konzepte zu definieren [GPVW17]. RDF ist eine Knowledge Representation Sprache für das Semantic Web und wird verwendet, um Wissen über verschiedene Dinge auszudrücken. RDF ist ein domänen-neutrales Framework, um Wissen auszudrücken, das an eine bestimmte Anwendung angepasst werden kann. Dabei kann RDF durch bestimmte Ontologien an eine bestimmte Domäne angepasst werden [GS09]. Eine Anpassung für das Internet of Things ist beispielsweise durch die IoT-Lite Ontologie [BEBT16] möglich. Die dargestellten Informationen des Semantic Web können als ein gerichteter Graph mit Beschreibungen aufgefasst werden, in dem die Knoten Dinge/Werte darstellen und die Kanten als Eigenschaften aufgefasst werden, die die Knoten verbinden. Jede Information besteht aus einem Tripel, das aus zwei Knoten und einer Kante besteht. Das Tripel besteht aus einem Subjekt, einem Prädikat und einem Objekt. Das Subjekt ist der Knoten, von dem die Kante ausgeht. Das Objekt ist der Knoten, zu dem die Kante geht. Das Prädikat ist der Typ der Kante. Das Subjekt hat also die Eigenschaft (Prädikat) des Objekts. Die Knoten können aus einem URI (Uniform Resource Identifier), Literalen oder einem leeren Knoten bestehen. URIs bezeichnen dabei Objekte aus der Ontologie. Zum Beispiel beschreibt 'http://purl.oclc.org/NET/UNIS/fiware/iot-lite#ActuatingDevice' ein Actuating Device in der IoT-Lite Ontologie. Literale repräsentieren Werte wie Strings oder Zahlen [GS09].

2.3. Kontextmodelle

Ein Kontextmodell [DABS22] beschreibt die Struktur einer IoT-Anwendung. Dabei umfasst das Kontextmodell den aktuellen Zustand der Komponenten, der sich aus statischen Informationen, wie z.B. Sensornamen, und Live-Informationen, wie z.B. die an einem Sensor anliegenden Messwerte, zusammensetzt.

Das Kontextmodell soll helfen, den Kontext der durch IoT-Anwendungen erzeugten Daten zu verstehen. Beispielsweise macht es einen Unterschied, ob sich ein Temperatursensor in einem Raum oder im Freien befindet. So können Sensoren durch Defekte falsche Werte liefern, welche ohne den Kontext in dem sich der Sensor befindet, schwer zu erkennen sind. Ein Beispiel hierfür wäre ein Temperatursensor, der andere Werte liefert als andere Temperatursensoren. Ist der Kontext, in dem die Sensoren zueinander stehen, nicht bekannt, so ist es nicht ohne weiteres möglich festzustellen,

ob die vom Sensor gelieferten Werte fehlerhaft sind oder nicht. Wenn der Kontext jedoch bekannt ist, z.B. alle Sensoren zusammen in einem Raum, kann auf einen Defekt des Sensors geschlossen werden.

Die IoT-Anwendung wird durch das Kontextmodell mit Hilfe von Ontologien beschrieben. Durch die Ontologien können die Komponenten des Systems und die Beziehungen zwischen diesen dargestellt werden. Das Kontextmodell nach Del Gaudio et al. [DABS22] erweitert die IoT-Lite Ontologie um Klassen und Relationen, die für Messdaten verwendet werden können. Die IoT-Lite Ontologie bietet die Möglichkeit IoT Konzepte zu beschreiben [BEBT16] Die erstellten Kontextmodelle können mittels RDF beschrieben werden.

2.4. Datenvalidierungsmethoden

Die meisten Datensätze, die reale Daten enthalten, enthalten verschiedene Fehler, wie fehlende Werte, Ausreißer oder Duplikate [AHS23]. Aus diesem Grund ist es wichtig, fehlerhafte Daten zu erkennen und zu reparieren, da die Daten sonst zu ungenauen und unzuverlässigen Ergebnissen führen können. Um Fehler in den Daten zu erkennen, wurden viele Datenvalidierungsmethoden entwickelt. Dabei werden oft statistische Methoden, Constraints oder Regeln und Patterns verwendet, um bestimmte Fehler zu erkennen. Nachdem die Fehler erkannt wurden, werden sie korrigiert. Datenvalidierungsmethoden bestehen also aus zwei Teilen: Zuerst wird versucht, die Fehler zu finden, und anschließend werden diese Fehler repariert [CIKW16]. Datenvalidierungsmethoden können dadurch die Datenqualität verbessern [RD+00]. In dieser Arbeit werden dabei hauptsächlich Outlier Detection Methoden verwendet.

Outlier sind Datenpunkte, die sich stark von anderen Datenpunkten unterscheiden und nicht in das erwartete Verhaltensmuster passen. Dabei unterscheiden sich Outlier in der Regel in ihren Eigenschaften von der Norm und treten im Vergleich zu den normalen Datenpunkten im Datensatz nur selten auf [BZA20]. Outlier können durch verschiedene Umstände entstehen, wie z.B. Mess- oder Aufzeichnungsfehler, ungewöhnliche aber wahre Werte, Falschangaben oder Stichprobenfehler. Um diese Ausreißer zu finden wurden Outlier Detection Methoden entwickelt. Dabei können die Methoden in verschiedene Funktionsweisen unterteilt werden: Statistische Methoden, Distanzbasierte Methoden, Clusteringbasierte Methoden und Dichtebasierte Methoden [Smi20]. Die Outlier Detection Methoden haben viele Anwendungsgebiete, wie z.B: Intrusion Detection Systeme, Fraud Detection, Erkennung von Fehlern oder Defekten in industriellen Anlagen oder Anomaly Detection in Wireless Sensor Networks [BZA20].

2.5. Recommender Systeme

Recommender Systeme sollen helfen, die Informationsüberflutung der Nutzer zu vermeiden, indem sie eine kleine Menge relevanter Produkte vorschlagen [HLZ+17; WZW+18]. Die Empfehlungen basieren in der Regel auf früheren Interaktionen des Nutzers mit verschiedenen Produkten. Dadurch basieren die Empfehlungen in der Regel auf den bisherigen Interessen des Nutzers. Beispielsweise wird eine Person, die in der Vergangenheit hauptsächlich Marvel-Filme angesehen hat, auch in Zukunft wahrscheinlich eher an Marvel-Filmen als an Dokumentationen interessiert sein. In Recommender Systemen wird die Entität, der die Empfehlungen vorgeschlagen werden, als Nutzer

bezeichnet und die Dinge, die vorgeschlagen werden, als Produkte. Recommender Systeme versuchen in der Regel entweder die Bewertung eines Nutzers für ein Produkt vorherzusagen oder die k Produkte vorzuschlagen, die am besten zum Benutzer passen. Das Vorschlagen der besten k Produkte ist jedoch das häufigere Problem und wird auch in dieser Arbeit versucht zu lösen [Agg+16].

Recommender Systeme verwenden in der Regel zwei Arten von Daten, um relevante Produkte vorzuschlagen: 1) Die Interaktionen zwischen den Nutzern und den Produkten, welche z.B. Bewertungen oder auch ein einfacher Klick auf das Produkt sein können, oder 2) die Attribute der Nutzer und der Produkte, wie z.B. Genres bei Filmen. Auf dieser Basis werden Recommender Systeme üblicherweise in zwei Typen unterteilt: Collaborative Filtering und Content-based. Collaborative Filtering nutzt die Interaktionen zwischen Nutzern und Produkten, während Content-based Methoden auf Attributen/Produkteigenschaften basieren. Eine weitere Art von Recommender Systemen, die in letzter Zeit vermehrt eingesetzt wird, sind die Knowledge-based Recommender Systeme, die meist auf Knowledge Graphen basieren. Oft werden die verschiedenen Arten von Recommender Systemen auch kombiniert, was dann als hybride Recommender Systeme bezeichnet wird [Agg+16]. Recommender Systeme können in vielen Bereichen eingesetzt werden, wie z.B. E-Commerce, Online-Nachrichten oder Social Media [HLZ+17].

2.5.1. Collaborative Filtering

Bei Collaborative Filtering Methoden werden die Bewertungen mehrerer Nutzer mit ähnlichen Interessen verwendet, um eine Empfehlung für ein Produkt zu erstellen. Die beiden Hauptprobleme von Collaborative Filtering sind das Cold-Start-Problem und die dünne Datenlage, da die einzelnen Nutzer in der Regel bisher nur mit wenigen Produkten interagiert haben. Das Cold Start Problem besteht darin, dass ein Nutzer erst eine bestimmte Anzahl von Interaktionen durchgeführt haben muss, bevor andere Nutzer mit ähnlichen Interessen gefunden werden können. Die Grundidee des Collaborative Filtering ist, dass Nutzer, die in der Vergangenheit ähnliche Interessen hatten, diese auch bei anderen Produkten teilen [Agg+16].

Ein vereinfachtes Beispiel findet sich in Abbildung 2.1. Hier kann die Bewertung für das Fragezeichen beim dritten Nutzer durch den ersten Nutzer impliziert werden, da sich die Nutzer auch bei anderen Produkten ähnlich waren. In diesem Fall würde das Recommender System für das Fragezeichen einen Daumen nach unten vorschlagen.

2.5.2. Content-based

In Content-based Recommender Systemen werden Attribute/Eigenschaften von Produkten verwendet, um Empfehlungen zu geben. Im Gegensatz zu Collaborative Filtering Verfahren können Content-based Recommender Systeme auch dann Empfehlungen aussprechen, wenn keine anderen Nutzer zum Vergleich zur Verfügung stehen [Agg+16].

Ein vereinfachtes Beispiel findet sich in Abbildung 2.2. Dem Benutzer wird ein Film mit ähnlichen Eigenschaften vorgeschlagen, wie der Film, den er bereits gesehen/bewertet hat. In diesem Fall sind die gemeinsamen Eigenschaften „Actionfilm“ und „Bösewicht“.

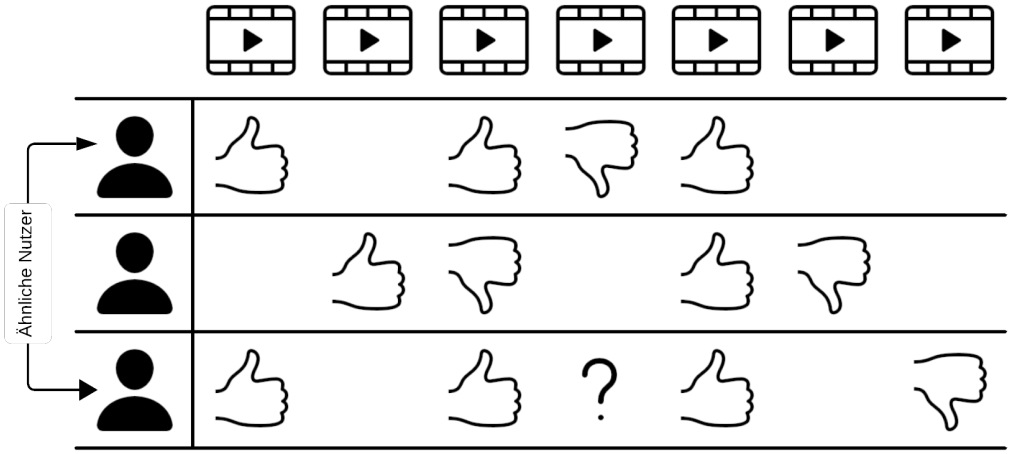


Abbildung 2.1.: Beispiel für die Funktionsweise des Collaborative Filtering

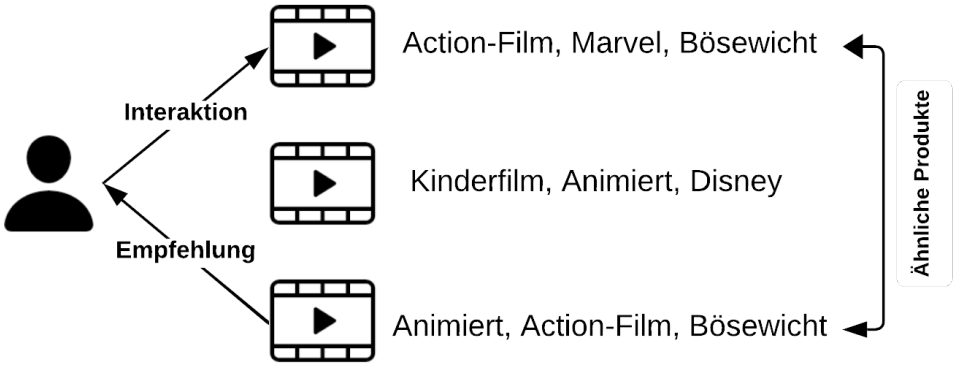


Abbildung 2.2.: Beispiel für die Funktionsweise eines Content-based Recommender Systems

2.5.3. Knowledge-based

Knowledge-based Recommender Systeme sind eng verwandt mit Content-based Recommender Systemen, da auch diese teilweise auf Basis von Produkteigenschaften Empfehlungen geben. Im Gegensatz zu Content-based Methoden berücksichtigen diese aber nicht nur die direkten Eigenschaften eines Nutzers/Produkts sondern auch die Relationen zwischen diesen Eigenschaften und es können Verbindungen über mehrere Eigenschaften geknüpft werden. Knowledge-based Recommender Systeme sind vor allem dann sinnvoll, wenn nur wenige Interaktionen/Bewertungen eines Nutzers vorliegen. Bei den Knowledge-based Recommender Systemen werden nicht nur die direkten Interaktionen des Nutzers mit einem Produkt für eine Empfehlung genutzt, sondern es werden zusätzlich die verschiedenen Beziehungen zwischen verschiedenen Attributen berücksichtigt [Agg+16].

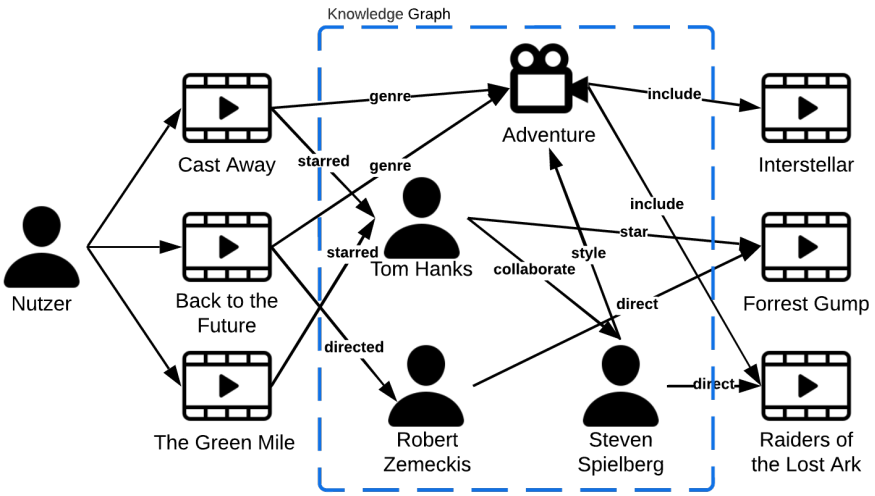


Abbildung 2.3.: Beispiel von einem Knowledge-based Recommender System nach [WZW+18]

Knowledge-based Recommender Systeme können dabei das Cold-Start-Problem und das Problem der dünnen Datenlage von Collaborative Filtering Methoden überwinden, indem sie verschiedene Attribute von Nutzern und Produkten nutzen, um zusätzliche Informationen zur Verfügung zu haben [WZX+19]. Knowledge-based Methoden verwenden in der Regel einen Knowledge Graph, um das Wissen darzustellen.

In Abbildung 2.3 wird ein Beispiel nach [WZW+18] gegeben. Dabei werden anhand verschiedener Beziehungen zwischen verschiedenen Eigenschaften mit Hilfe eines Knowledge Graphen passende Filme vorgeschlagen. Durch den Knowledge Graph können auch Produkte vorgeschlagen werden, die nicht direkt mit den Filmen verwandt sind, die der Nutzer bereits gesehen hat, sondern auch Filme, die durch mehrere Relationen mit diesen verbunden sind. Beispielsweise kann in dem Beispiel dem Nutzer der Film „Raiders of the Lost Ark“ empfohlen werden, da in dem Knowledge Graph die folgende Verbinndung existiert: Cast Away $\xrightarrow{starred}$ Tom Hanks $\xrightarrow{collaborate}$ Steven Spielberg $\xrightarrow{direct}$ Raiders of the Lost Ark.

2.6. Word Embedding Methoden

Word Embeddings ermöglichen die Darstellung von Wörtern als Vektoren. Im Folgenden werden die Funktionsweisen der folgenden Word Embedding Modelle vorgestellt: Word2Vec [MCCD13], GloVe [PSM14] und fastText [BGJM17].

Word2Vec Embeddings

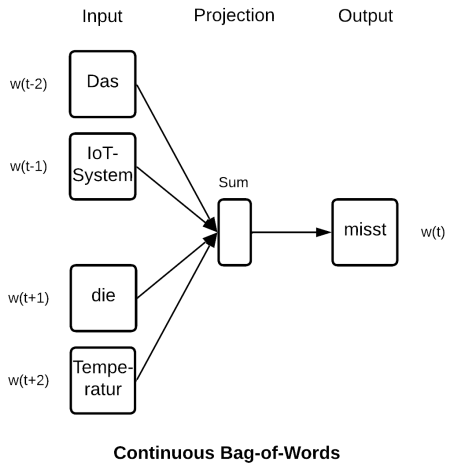
Das Word2Vec Modell wurde von Mikolov et al. bei Google [MCCD13] entwickelt. Dabei wird ein einfaches neuronales Netz trainiert, um eine Vektorrepräsentation der Wörter zu erhalten. Die resultierenden Vektoren haben die Eigenschaft, dass ähnliche Wörter nahe beieinander liegen. Beispielsweise haben Wörter der Kategorie Früchte einen kleineren Abstand zueinander als zu Wörtern der Kategorie Tiere. Außerdem können Beziehungen zwischen Wörtern dargestellt werden. Zum Beispiel kann man ein Wort berechnen, das in derselben Beziehung zu „small“ steht wie „big“ zu „biggest“, indem man das ähnlichste Wort zum Vektor X = „biggest“ − „big“ + „small“ berechnet [MCCD13].

Für die Umwandlung von Wörtern in Embeddings stellen die Autoren zwei Architekturen vor: Continuous Bag-of-Words und Continuous Skip-Gram. Die Continuous Bag-of-Words (CBOW) Architektur versucht, ein Wort basierend auf dem Kontext vorherzusagen. Dabei werden Wörter vor und nach dem eigentlichen Wort verwendet, um das Zielwort vorherzusagen. Beispielsweise würden bei dem Satz „Das IoT-System misst die Temperatur“ und dem Zielwort „misst“ die Wörter „Das“, „IoT-System“, „die“ und „Temperatur“ verwendet werden, um „misst‘ vorherzusagen. Die Architektur dieses Modells ist in Abbildung 2.4a dargestellt. Im Gegensatz zu CBOW versucht die Continuous Skip-Gram Architektur, die Wörter um ein Wort herum vorherzusagen. Dabei wird ein Wort als Eingabe verwendet und es wird versucht, wie in Abbildung 2.4b dargestellt, die Wörter vorherzusagen, die in einem Satz vor und nach diesem Wort stehen würden. Bei dem Satz „Das IoT-System misst die Temperatur“, würde also bei der Eingabe „misst“ versucht werden, die Wörter „Das“, „IoT-System“, „die“ und „Temperatur“ vorherzusagen. In beiden Architekturen wird der Satz durchlaufen, wobei jedes Wort nacheinander als Eingabe (Ausgabe bei Skip-Gram) verwendet wird. So würde bei dem Satz „Das IoT-System misst die Temperatur“ zuerst „Das“ als Eingabe (Ausgabe bei Skip-Gram) verwendet werden und dann die Wörter „IoT-System“, „misst“, „die“ und „Temperatur“. Die Autoren kommen zu dem Schluss, dass die CBOW-Architektur besser für syntaktische Aufgaben funktioniert, während die Skip-gram-Architektur besser für semantische Aufgaben geeignet ist. Bei den syntaktischen Aufgaben wird getestet, ob verschiedene Formen desselben Wortes nahe beieinander liegen, z.B. „System“ und „Systeme“. Bei semantischen Aufgaben geht es dagegen darum, die Beziehungen zwischen Wörtern zu testen. Ein Beispiel dafür wäre es ausgehend von der Relation „Berlin“ - „Deutschland“ die Relation „Athen“ - „Griechenland“ vorherzusagen.

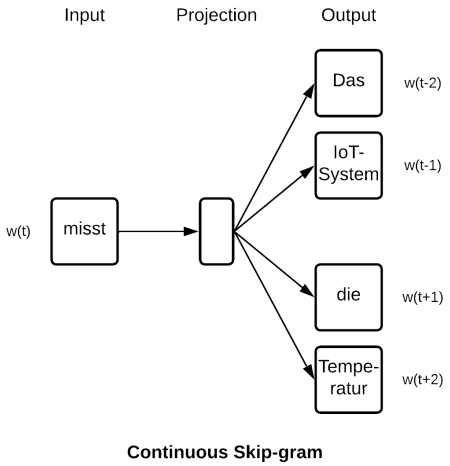
GloVe Embeddings

Das GloVe Modell [PSM14] wurde von Pennington et al. entwickelt und steht für „Global Vectors for Word Representation“. Die Embeddings des GloVe Modells haben ähnliche Eigenschaften wie die des Word2Vec Modells. Auch beim GloVe Modell liegen die Vektoren ähnlicher Wörter nahe beieinander und die Beziehungen zwischen den Wörtern können ebenfalls durch Vektoraddition ausgedrückt werden.

Im Gegensatz zum Word2Vec-Modell, das nur lokale Informationen der einzelnen Sätze verwendet, nutzt GloVe globale Informationen mit Hilfe von Co-Occurrence-Vorkommen. Dadurch berücksichtigt das GloVe Modell aufgrund der Co-Occurrence-Vorkommen die Zusammenhänge zwischen Wörtern explizit, wohingegen das Word2Vec Modell die Zusammenhänge nur durch gemeinsame Vorkommen in einzelnen Sätzen lernt. Die Co-Occurrence-Vorkommen geben an, wie oft ein Wort



(a) Continous Bag-of-Words Architektur



(b) Skip-gram Architektur

Abbildung 2.4.: Architekturen für Word2Vec nach [MCCD13] mit dem Satz „Das IoT-System misst die Temperatur“

Wahrscheinlichkeit und Verhältnis	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	$1{,}9 \cdot 10^{-4}$	$6{,}6 \cdot 10^{-5}$	$3{,}0 \cdot 10^{-3}$	$1{,}7 \cdot 10^{-5}$
$P(k\|steam)$	$2{,}2 \cdot 10^{-5}$	$7{,}8 \cdot 10^{-4}$	$2{,}2 \cdot 10^{-3}$	$1{,}8 \cdot 10^{-5}$
$P(k\|ice)/P(k\|steam)$	8,9	$8{,}5 \cdot 10^{-2}$	1,36	0,96

Tabelle 2.1.: Co-occurrence Vorkommen von „ice“ und „steam“ mit verschiedenen Wörtern. Die Tabelle wurde aus [PSM14] übernommen

im Kontext mit einem anderen Wort vorkommt, und beziehen sich auf den gesamte Korpus. Aus diesen Co-Occurrence-Vorkommen wird dann auf die Beziehungen zwischen den einzelnen Wörtern geschlossen. Ein Beispiel ist in Tabelle 2.1 zu finden. In der Tabelle wird in den ersten beiden Zeilen die Wahrscheinlichkeit angegeben, mit der ein bestimmtes Wort zusammen mit den Wörtern „ice“ oder „steam“ auftritt. Die letzte Zeile gibt das Verhältnis dieser Wahrscheinlichkeiten an. Es ist zu erkennen, dass das Verhältnis bei Wörtern, die häufiger mit „ice“ als mit „steam“ vorkommen, groß ist, wie beispielsweise bei dem Wort „solid‘. Wenn Wörter häufiger mit „steam“ vorkommen, wie es bei dem Wort „gas“ der Fall ist, ist das Verhältnis relativ klein. Anhand der Verhältnisse lässt sich also feststellen, dass „ice“ mit „solid“ und „steam“ mit „gas“ verwandt ist. Wenn das Wort entweder mit beiden Wörtern verwandt ist, wie bei „water“, oder mit keinem von beiden, wie bei „fashion“, ist das Verhältnis ungefähr eins. Mit dieser Idee trainieren die Autoren ein neuronales Netz, das auf einer Co-Occurrence Matrix basiert, die angibt, wie oft Wörter zusammen vorkommen.

fastText Embeddings

Das fastText Modell [BGJM17] wurde von Bojanowski et al. bei Facebook entwickelt. Im Vergleich zu den beiden anderen beiden Methoden hat diese Methode den Vorteil, dass mit diesem Modell auch Embeddings für Wörter erstellt werden können, die nicht im Training vorkamen. Somit kann auch für Wörter mit Rechtschreibfehlern ein Embedding erstellt werden. Außerdem kann mit diesem Modell im Gegensatz zu Word2Vec und GloVe für jedes Wort ein Embedding erzeugt werden. Wenn beispielsweise das Wort „IoT“ nicht im Modell vorkommt, kann dennoch ein Embedding erzeugt werden. Die Autoren verwenden die gleiche Architektur wie das Word2Vec Modell, die in Abbildung 2.4 beschrieben ist. Für ihr vortrainiertes Modell verwenden die Autoren die Skip-gram Architektur, wobei das Modell auch mit der Continous Bag-of-Words Architektur trainiert werden kann. Um Embeddings für Wörter erstellen zu können, die nicht im Trainingskorpus vorkommen, verwenden die Autoren ein Subword Modell, bei dem nicht nur die einzelnen Wörter zum Lernen der Embeddings verwendet werden, sondern zusätzlich auch deren n-grams. Die n-grams sind dabei Folgen von n Buchstaben des Wortes. Beispielsweise geben die Autoren für das Wort „where“ und n=3 folgende n-grams an: <wh, whe, her, ere, re>. Die Zeichen < und > kennzeichnen den Anfang bzw. das Ende des Wortes. Für die Eingabe in die Skip-gram Architektur wird dann sowohl das ganze Wort als auch die generierten n-grams verwendet. Tritt bei der Erzeugung der Embeddings ein Wort auf, das nicht im Trainingskorpus vorkommt und für das das Modell keine direkte Vektorrepräsentation gelernt hat, wird das Wort in seine n-grams zerlegt und aus diesen ein Embedding erzeugt.

3. Verwandte Arbeiten

In diesem Kapitel werden Arbeiten vorgestellt, die ein ähnliches Ziel wie diese Arbeit verfolgen, und es werden sowohl die Unterschiede als auch die Gemeinsamkeiten mit dieser Arbeit erläutert.

3.1. DEKR: Vorschlagen von Machine Learning Methoden

In „DEKR: Description Enhanced Knowledge Graph for Machine Learning Method Recommendation “ [CSY+21] stellen Cao et al. eine Methode vor, um für Datensätze geeignete Machine Learning Methoden vorzuschlagen. Laut den Autoren gibt es mittlerweile eine sehr große Anzahl von Machine Learning Methoden, die auf einen Datensatz angewendet werden können. Um die Suche nach einer passenden Methode zu erleichtern, stellen die Autoren ein Recommender System vor, um eine Vorhersage zu treffen, ob eine Machine Learning Methode mit einem bestimmten Datensatz funktionieren wird. Das Recommender System besteht dabei aus zwei Komponenten und ist ein hybrides System. Die beiden Komponenten sind das Knowledge Graph Neural Network (KGNN) und die Text-based Collaborative Filtering (TextCF) Komponente. Die KGNN-Komponente verwendet einen Knowledge Graph, um eine Vorhersage zu treffen und die TextCF-Komponente verwendet die Textbeschreibungen der Datensätze und Methoden. Beide Komponenten berechnen eine Wahrscheinlichkeit, wie wahrscheinlich eine Machine Learning Methode zu einem Datensatz passt. Für die endgültige Empfehlung wird dann der Durchschnitt aus beiden Komponenten gebildet.

Die Autoren verfolgen ein ähnliches Ziel wie diese Arbeit. Anstatt eine geeignete Datenvalidierungsmethode für ein Kontextmodell vorzuschlagen, versuchen die Autoren eine geeignete Machine Learning Methode für einen Datensatz vorzuschlagen. Zudem wird in dieser Arbeit auch die TextCF-Komponente der DEKR-Methode verwendet. Die KGNN Komponente wird jedoch nicht verwendet, da es aufgrund der geringen Datenmenge nicht möglich war einen Knowledge Graph zu generieren. Zusätzlich werden in dieser Arbeit noch weitere Methoden getestet, wie die Textbeschreibungen zur Vorhersage verwendet werden können. Zudem wird in dieser Arbeit die Textbeschreibung mithilfe eines Kontextmodells generiert, anstatt das diese fest vorgegeben ist.

3.2. Automatische Datenbereinigung

Zur automatischen Erstellung einer Outlier Detection Methode stellen Cao et al. in „AutoOD: Automatic Outlier Detection“ [CYW+23] eine Methode vor, die verschiedene existierende Outlier Detection Methoden kombiniert, um die Ausreißer zu klassifizieren. Dabei benötigt die Methode keinen Zugriff auf die Labels, um zu entscheiden, ob ein Datenpunkt ein Outlier ist oder nicht. Stattdessen werden verschiedene unsupervised Methoden kombiniert, um die Ausreißer zu finden, da eine einzelne Methode nicht immer in der Lage ist, die Ausreißer optimal zu klassifizieren. Dazu

verwenden die Autoren zunächst viele verschiedene Instanziierungen der unsupervised Outlier Detection Methoden, um die Labels für die Outlier automatisch zu generieren. Anschließend trainieren die Autoren auf Basis dieser erzeugten Labels ein supervised Klassifikationsmodell, das in der Lage ist, die Outlier des gesamten Datensatzes zu finden. Um die Labels für einen Datensatz zu generieren, stellen die Autoren zwei Varianten vor. In der ersten Variante wird ein kleiner Teil zuverlässiger Labels erzeugt, indem alle unsupervised Methoden diese Punkte als Outlier klassifizieren. Diese Labels werden dann verwendet, um schlechte Detektoren zu finden und diese zu entfernen, bis nur noch ein kleiner Teil guter Detektoren übrig bleibt. Die Detektoren sind die verschiedenen Outlier Detection Methoden mit unterschiedlichen Parametern. Bei der zweiten Variante wird zunächst eine große Anzahl von Labels erzeugt, indem die Ergebnisse aller Detektoren berücksichtigt werden. Anschließend wird ein neuronales Netz auf den erzeugten Labels trainiert, um schlechte Detektoren zu entfernen, die anfangs einen hohen Loss auf den Trainingsdaten haben. Diese Arbeit verfolgt ein ähnliches Ziel wie AutoOD. In dieser Arbeit wird jedoch eine Datenvalidierungsmethode anhand eines Kontextmodells vorgeschlagen, was ohne einen konkreten Datensatz möglich ist. Zudem können bei dieser Arbeit verschiedene Datenvalidierungsmethoden verwendet werden und nicht nur Outlier Detection Methoden. Ein weiterer Unterschied besteht darin, dass in AutoOD mehrere Outlier Detection Methoden kombiniert werden, während in dieser Arbeit nur einzelne Methoden vorgeschlagen werden.

In „HoloClean: Holistic Data Repairs with Probabilistic Inference“ stellen Rekatsinas et al. [RCIR17] eine Methode zur automatischen Datenbereinigung vor. Dabei kombiniert HoloClean verschiedene qualitative Datenbereinigungsverfahren, die auf Integritätsbeschränkungen oder externen Datenquellen basieren, mit quantitativen Datenbereinigungsverfahren, die statistische Informationen nutzen. Dazu erzeugt HoloClean zunächst ein probalistisches Modell, das den Datensatz repräsentiert. Um anschließend die Fehler im Datensatz zu korrigieren, verwendet HoloClean statistisches Lernen und probabilistische Inferenz. Als Eingabe benötigt HoloClean den Datensatz sowie Einschränkungen für die Daten und externe Informationen.

Yakout et al. [YBE13] stellen eine Methode zur automatischen Datenreparatur vor. Die Erkennung von fehlerhaften Daten wird dabei nicht behandelt, sondern ausschließlich die Reparatur. Dazu werden die Daten zunächst in einen Teil, der als korrekt angesehen wird, und einen Teil, der Fehler enthalten kann, aufgeteilt. Die Reparatur erfolgt durch Maximierung der Likelihoodfunktion der fehlerhaften Daten, welche durch eine Datenverteilung repräsentiert werden. Dabei werden kleine Wertänderungen an den Daten vorgenommen um die Likelihoodfunktion zu maximieren. Für diese Methode sind keine Einschränkungen für die Daten durch den Benutzer erforderlich.

Im Gegensatz zu den Arbeiten zur automatischen Datenbereinigung schlägt das in dieser Arbeit entwickelte Konzept nur eine Datenbereinigungsmethode vor, führt aber keine Datenbereinigung durch. Der Vorschlag geeigneter Datenvalidierungsmethoden kann auch ohne die generierte Daten erfolgen. Außerdem muss der Benutzer keine zusätzlichen Eingaben, wie z.B. Beschränkungen für die Daten, machen.

4. Konzept für das Vorschlagen von Datenvalidierungsmethoden mithilfe von Kontextmodellen

Ziel dieser Arbeit ist es, anhand von Kontextmodellen verschiedener IoT-Umgebungen geeignete Datenvalidierungsmethoden vorzuschlagen, die in der Lage sind, die generierten Daten zu bereinigen. Dadurch soll es auch technisch unerfahrenen Anwendern ermöglicht werden, geeignete Datenvalidierungsmethoden auszuwählen, die zu der erstellten IoT-Anwendung passen. Um dieses Ziel zu erreichen, wird in diesem Abschnitt ein Konzept zum Vorschlagen von geeigneten Datenvalidierungsmethoden vorgestellt.

Um eine geeignete Empfehlung von Datenvalidierungsmethen zu erhalten, werden Recommender Systeme verwendet. Recommender Systeme werden in dieser Arbeit verwendet, da diese verwendet werden können, um basierend auf einem Nutzer passende Produkte zu empfehlen. Generell können Recommender Systeme immer dann eingesetzt werden, wenn es notwendig ist, einem Nutzer Produkte zu empfehlen, die zu ihm und seinen Anforderungen passen. In diesem Fall sind die Nutzer die Kontextmodelle, auf deren Basis geeignete Datenvalidierungsmethoden, d.h. Produkte, vorgeschlagen werden. Ein genauerer Überblick über die Funktion von Recommender Systemen ist in Abschnitt 2.5 gegeben. Da es eine Vielzahl von Recommender Systemen für den gleichen Anwendungsfall gibt, werden in dieser Arbeit verschiedene Recommender Systeme untersucht, um herauszufinden, welches für die Empfehlung von Datenvalidierungsmethoden auf Basis des Kontextmodells am besten geeignet ist.

Dieses Kapitel ist dafür in vier Abschnitte unterteilt. Zunächst wird in Abschnitt 4.1 ein Überblick über das gesamte System zur Empfehlung von Datenvalidierungsmethoden gegeben. Anschließend werden die einzelnen Komponenten im Detail beschrieben. Dabei wird zunächst in Abschnitt 4.2 erläutert, welche Arten von Recommender Systemen überhaupt in Frage kommen, da nicht jede Art von Recommender System verwendet werden kann. Anschließend wird in Abschnitt 4.3 beschrieben, wie ein Kontextmodell als Input für ein Recommender System verwendet werden kann, da Recommender Systeme nicht direkt auf dem Graphen des Kontextmodells arbeiten können. Abschließend werden in Abschnitt 4.4 die verwendeten Recommender Systeme im Detail beschrieben und begründet, warum diese ausgewählt wurden.

4.1. Übersicht über das System

Dieser Abschnitt gibt einen Überblick über das Gesamtsystem und erläutert seine Struktur. Die einzelnen Komponenten werden dann in den folgenden Abschnitten im Detail erläutert. Die Struktur des Systems ist in Abbildung 4.1 dargestellt.

Rang	Datenvalidierungsmethode	Empfehlungswert
1	Isolation Forest	0.9
2	Local Outlier Factor	0.85
3	Clustering Based Local Outlier Factor	0.75
4	k-Nearest-Neighbors	0.6
5	Angle-Based Outlier Detection	0.5
6	Principal Component Analysis	0.45
7	Deep Support Vector Data Description	0.25

Tabelle 4.1.: Beispielausgabe des Recommender Systems

Zuerst wird die RDF-Beschreibung einer IoT-Anwendung mit Hilfe eines Kontextmodells aus [DABS22] erstellt. Diese RDF-Beschreibung wird dann durch das in Abschnitt 4.3 beschriebene KG-to-Text Modell in eine Textbeschreibung umgewandelt. Die Textbeschreibungen der Datenvalidierungsmethoden werden nicht automatisch generiert, sondern müssen manuell erstellt werden. Dies muss jedoch nur einmal geschehen, da die verschiedenen Datenvalidierungsmethoden bereits im Voraus festgelegt werden. Wenn die Auswahl der Datenvalidierungsmethoden, die empfohlen werden können, erweitert werden soll, kann die Datenvalidierungsmethode einfach mit einer Textbeschreibung zum Datensatz hinzugefügt werden.

Anschließend werden die Textbeschreibungen einer Preprocessing Komponente übergeben, die die Textbeschreibungen bereinigt. Dies ist notwendig, da die Beschreibungen in natürlicher Sprache formuliert sind und daher viele Wörter enthalten, die keine zusätzlichen Informationen über das Kontextmodell oder die Datenvalidierungsmethode liefern. Außerdem können Symbole enthalten sein, die von den Feature Selection Methoden der Recommender Systeme nicht verwendet werden können. Auch diese müssen entfernt werden. Die genaue Funktionsweise der Preprocessing-Methode ist in Abschnitt 5.3 beschrieben.

Die bereinigten Textbeschreibungen werden dann von der Hauptkomponente, dem Recommender System, verwendet, um eine Empfehlung zu erstellen. Die Empfehlung gibt an, welche Datenvalidierungsmethoden für das Kontextmodell am besten geeignet sind. Das Recommender System besteht dabei aus einer in Abschnitt 4.4.1 vorgestellten Feature-Selection Methode, die aus den Textbeschreibungen Vektoren erzeugt, und einem in Abschnitt 4.4.2 vorgestellten Computational Approach, der dann einen Wert berechnet, der für die Empfehlung verwendet wird. Es ist zu beachten, dass für ein Kontextmodell immer die Empfehlungswerte für alle Datenvalidierungsmethoden berechnet werden müssen. Diese Werte können dann miteinander verglichen werden, um die beste Empfehlung zu erhalten. Wenn beispielsweise die Kosinus-Ähnlichkeit als Computational Appraoch verwendet wird, wird nach Berechnung aller Werte für die verschiedenen Datenvalidierungsmethoden die Datenvalidierungsmethode mit dem höchste Wert empfohlen. Ein Beispiel, wie eine solche Ausgabe dann aussehen könnte, ist in Tabelle 4.1 gegeben. Dem Benutzer würde dann beispielsweise Isolation Forest als am besten geeignet für sein Kontextmodell vorgeschlagen werden, da es im Vergleich zu den anderen Datenvalidierungsmethoden den höchsten Wert hat. Für die TextCF-Komponente wird ebenfalls der höchste Wert und für die euklidische Distanz der kleinste Wert gewählt. Welche Feature Selection Methode mit welchem Computational Approach am besten zusammenpasst und welcher Computational Approach die besten Ergebnisse liefert, wird in Kapitel 6 behandelt.

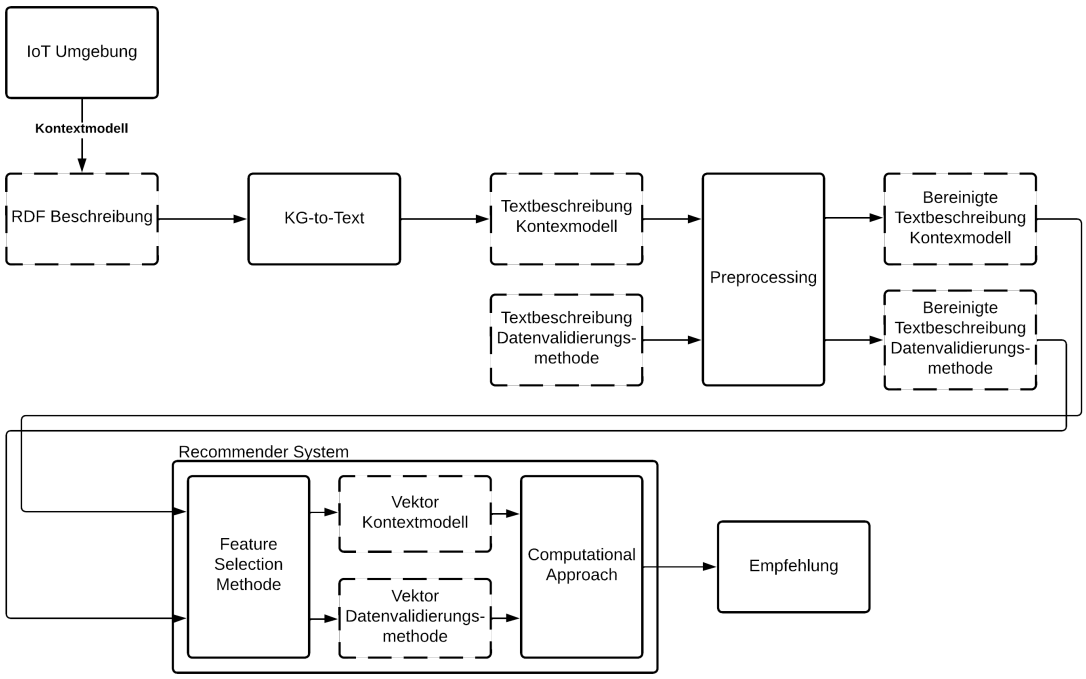


Abbildung 4.1.: Aufbau des entwickelten Systems zum Vorschlagen von Datenvalidierungsmethoden mithilfe von Kontextmodellen. Zuerst wird das Kontextmodell in Text umgewandelt. Anschließend werden die Textbeschreibungen von einem Recommender System verwendet, um Empfehlungen zu berechnen.

4.2. Auswahl eines geeigneten Recommender Systems

In diesem Abschnitt wird erläutert, welche Art von Recommender System für diese Arbeit verwendet wird, da es verschiedene Arten von Recommender Systemen gibt, die nach unterschiedlichen Prinzipien arbeiten. Dabei ist nicht jede Art von Recommender System für die Empfehlung von Datenvalidierungsmethoden geeignet.

Da DEKR [CSY+21] ein ähnliches Ziel wie diese Arbeit verfolgt, bietet es sich an, sich an dieser Arbeit zu orientieren. DEKR verwendet zwei verschiedene Recommender Systeme für die Vorhersage von Machine Learning Methoden, die am Ende zusammengeführt werden. Cao et al. [CSY+21] verwenden ein auf Knowledge Graphen basierendes Recommender System und ein auf Textbeschreibungen basierendes Recommender System. Knowledge-Based Recommender Systeme haben den Vorteil, dass bereits mit wenigen Daten über einen Nutzer eine Empfehlung erstellt werden kann, was für diesen Anwendungsfall von Vorteil ist. Außerdem können Knwoledge-Based Recommender Systeme die Probleme anderer Arten von Recommender Systemen überwinden [Agg+16]. So können sie auch Empfehlungen vorschlagen ohne das der Nutzer zuerst mit anderen Produkten interagiert haben muss. Auch diese Eigenschaft ist von Vorteil für den Anwendungsfall dieser Arbeit. Der Einsatz von Knowledge-Based Recommender Systemen ist für diese Arbeit jedoch nicht möglich, da für die Erstellung eines Knowledge Graphen zu wenige Kontextmodelle zur Verfügung standen. Dadurch war es nicht möglich, verschiedene Attribute aus den Kontextmodellen

zu extrahieren und daraus einen Knowledge Graph zu erstellen. Für eine größere Anzahl von Kontextmodellen hätten beispielsweise Attribute wie verwendete Sensoren, Maßeinheiten oder andere verwendete Geräte extrahiert werden können. Das Recommender System von DEKR, das auf textuellen Beschreibungen basiert, kann jedoch verwendet werden, wenn die Kontextmodelle in Text umgewandelt werden. Die Umwandlung in Text kann unabhängig von der Anzahl der verfügbaren Kontextmodelle erfolgen.

Die textbasierten Recommender Systeme gehören zur Kategorie der Content-Based Recommender Systeme, da die Textbeschreibungen der Produkte auch als deren Eigenschaften aufgefasst werden können. Bei Content-Based Recommender Systemen können allerdings auch noch andere Eigenschaften für die Empfehlungen genutzt werden. So könnten ähnlich wie bei Knowledge-Based Recommender Systemen bestimmte Eigenschaften des Kontextmodells extrahiert werden. Dies könnten z.B. die Anzahl der IoT-Geräte, die Anzahl der Evironments des Kontextmodells oder auch die verwendeten IoT-Geräte sein. Da aber auch hier, wie bei den Knowledge-Based Methoden, das Problem besteht, dass zu wenige Kontextmodelle zur Verfügung stehen, um ein Recommender System auf diesen Eigenschaften zu trainieren, kann auch dieser Ansatz für diese Arbeit nicht verwendet werden.

Ein häufig verwendeter Typ von Recommender-Systemen ist das Collaborative Filtering. Hier werden die Empfehlungen auf Basis von Nutzern erstellt, die in der Vergangenheit ähnliche Produkte wie der aktuelle Nutzer verwendet haben. Diese Art von Recommender Systemen ist jedoch für den Anwendungsfall dieser Arbeit nicht geeignet, da bereits einige passende Datenvalidierungsmethoden zu einem bestimmten Kontextmodell benötigt werden würden, um weitere passende Datenvalidierungsmethoden vorzuschlagen. Ein technisch unerfahrener Benutzer ist jedoch nicht in der Lage, einige geeignete Datenvalidierungsmethoden auszuwählen.

Eine weitere Möglichkeit, Vorhersagen auf Basis des Kontextmodells zu treffen, wäre die Erstellung eines Graph Embeddings des Kontextmodells und das Training eines neuronalen Netzes auf Basis dieser Embeddings. Meines Wissens nach gibt es jedoch nur Graph Embedding Methoden, die darauf abzielen, auf einem großen Graphen zu trainieren und dann nur Teilgraphen zu embedden. Dies ist jedoch bei Kontextmodellen nicht der Fall, da jedes Kontextmodell durch einen eigenen Graphen beschrieben wird. Aus diesem Grund ist es nicht möglich, diese Methode anzuwenden.

Aus den genannten Gründen fiel die Wahl auf textbasierte Recommender-Systeme. Zudem können auch Kontextmodelle verwendet werden, die auf unterschiedlichen Ontologien basieren, da eine allgemeine Textbeschreibung erstellt wird, die unabhängig von der verwendeten Ontologien ist. Werden bestimmte Eigenschaften für Knowledge-based oder Content-based Recommender Systeme extrahiert, so kann es sein, dass diese Eigenschaften in einem Kontextmodell, das eine andere Ontologie verwendet, nicht mehr verfügbar sind. Um an die Textbeschreibung des Kontextmodells zu gelangen, gibt es zwei Möglichkeiten. Zum einen könnte ein Benutzer nach der Erstellung eines Kontextmodells aufgefordert werden, eine Textbeschreibung seines erstellten Systems zu erstellen. Dabei könnte er beispielsweise nach der Domäne des IoT-Systems, den Maßeinheiten und der groben Struktur des Systems gefragt werden. Um die Erstellung des Prozesses zu automatisieren, kann auch ein KG-to-Text Modell verwendet werden. In dieser Arbeit wurde ein solches KG-to-Text Modell verwendet, um das System so zu automatisieren, dass ein Benutzer nur noch ein Kontextmodell als Eingabe benötigt. Im folgenden Abschnitt wird beschrieben, wie dieses KG-to-Text Modell funktioniert. Ein weiterer Vorteil von textbasierten Recommender Systemen für diese Arbeit besteht

darin, dass Textbeschreibungen mit ähnlichem Inhalt wie die der Kontextmodelle verwendet werden können, um das Recommender System auf einer größeren Anzahl von Daten zu trainieren und zu evaluieren.

4.3. Umwandlung des Kontextmodells in Text

Das als Graph repräsentierte Kontextmodell muss nun in eine textuelle Beschreibung umgewandelt werden, um als Input für ein textbasiertes Recommender System verwendet werden zu können. Dieser Abschnitt beschreibt, wie diese Umwandlung durchgeführt wird. Die Umwandlung erfolgt mit Hilfe eines KG-to-Text Modells.

Ein KG-to-Text Modell wandelt einen Knowledge Graph in Text um, indem es eine Beschreibung des Knowledge Graphen basierend auf den Kanten und Knoten erstellt [KJR+21]. Um ein geeignetes Modell für die Graphen der Kontextmodelle zu finden, wurden die folgenden KG-to-Text Modelle untersucht: JointGT [KJR+21], GAP [CAW22], [LTZ+21] und [RSSG20]. Es war jedoch nicht möglich, alle genannten Modelle zu testen, da verschiedene Probleme bei der Ausführung der Modelle auftraten. Entweder fehlten Dateien oder der Python-Interpreter gab Fehler aus, die nicht ohne weiteres behoben werden konnten, obwohl die angegebenen Python- und Paketversionen verwendet wurden. Aufgrund der verschiedenen Probleme konnte nur JointGT getestet werden.

4.3.1. Umwandlung in Text mit JointGT

Um die Einsatzfähigkeit des JointGT Modells für den Anwendungsfall dieser Arbeit zu testen, wurde das Modell auf dem Graphen des Kontextmodells aus [DABS22] getestet. Das Kontextmodell besteht aus einem Temperatursensor und einer Monitoring-Komponente und ist in Abbildung 4.2 dargestellt. Für die direkte Verwendung von JointGT stellen die Autoren ein bereits trainiertes Modell zur Verfügung. Da zum Zeitpunkt dieser Arbeit nur zwei Kontextmodelle, welche auf dem Framework von [DABS22] basieren, zur Verfügung standen, war es nicht möglich, das Modell auf eigenen Daten zu trainieren oder ein Finetuning des vortrainierten Modells durchzuführen. Beim Finetuning wird das bereits trainierte Modell auf eigenen Daten leicht angepasst, so dass es für diese Daten, die sich nicht zu sehr von den Originaldaten unterscheiden sollten, bessere Ergebnisse liefert[1]. Aus diesem Grund wurde das vortrainierte Modell der Autoren für den Test von JointGT verwendet.

Um den Knowledge Graph mit dem JointGT Modell verwenden zu können, muss dieser zunächst in Textform umgewandelt werden. Dazu muss jedes Knoten-Relation-Knoten Tripel in ein JSON-Format umgewandelt werden. Da jeder Knoten im Kontextmodell in Abbildung 4.2 mehrere Attribute hat, wurde, wenn möglich, das „rdfs:label“ verwendet. Wenn der Knoten kein Label hat, wurde die Bezeichnung nach „iot-ins“ verwendet. Für die Relationen wurden die Bezeichnungen der Kanten nach dem Doppelpunkt verwendet. Die Beschreibung des Graphen als JSON ist in Listing 4.1 zu finden.

Die generierte Textbeschreibung des vortrainierten JointGT Modells für das Kontextmodell sieht dann wie folgt aus:

[1] Finetuning: https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html

> The SensingDevice for Sensor1 for temperature is exposedBy and The SensingDevice for Sensor1 for temperature is exposedBy .

Wie oben zu sehen ist, gibt das Modell nur die Bezeichner der einzelnen Knoten und deren Relationen wieder und verbindet das Ganze mit verschiedenen Füllwörtern. Außerdem gibt JointGT hier nicht alle Knoten und Relationen in der Beschreibung wieder, sondern lässt einige aus. Dadurch ergeben sich zwei Probleme: Es wird nicht der gesamte Graph verwendet und es werden nur die Bezeichner der Knoten wiedergegeben.

Um das Problem zu überwinden, dass nicht der gesamte Graph verwendet wird, was zu abgeschnittenen Sätzen führt, wurde versucht, die Modellparameter so zu ändern, so dass längere Sätze erzeugt werden. Die Änderung der Parameter führte jedoch zu keinem Unterschied in der Ausgabe. Ein Grund dafür könnte sein, dass die Autoren von JointGT das Modell nur auf kleineren Graphen trainiert haben. Im Paper zu JointGT [KJR+21] geben die Autoren die Performance ihres Modells nur bis maximal sieben Triple an. Aus diesem Grund wurde versucht, einen kleineren Graphen des Kontextmodells zu verwenden. Um die Größe des Graphen zu reduzieren, wurden Knoten entfernt, so dass nur noch sechs Knoten übrig blieben. Das reduzierte Kontextmodell ist in Abbildung 4.3 dargestellt. Mit dem reduzierten Graphen erzeugt JointGT die folgende Beschreibung:

> The SensingDevice for Sensor1 has a Subsystem and a Sensor1 for temperature .

Obwohl der Satz am Ende nicht mehr abgeschnitten ist, fehlen auch hier noch Teile des Kontextmodells und das JointGT Modell gibt nur die Namen der Knoten und Relationen in Verbindung mit Füllwörtern wieder. Aufgrund der geringen Menge an Kontextmodellen ist es nicht möglich, das Modell speziell auf die Graphen der Kontextmodelle zu trainieren, was zu besseren Beschreibungen durch JointGT führen könnte.

4.3.2. Umwandlung in Text mit ChatGPT

Da kein Modell neu trainiert werden kann, wurde ChatGPT getestet, das bereits über eine große Menge an Hintergrundwissen verfügt. Für die folgenden Ausgaben von ChatGPT wurde GPT-3.5 verwendet. Durch das Hintergrundwissen weiß ChatGPT beispielsweise anhand der Namen einzelner Sensoren aus Abbildung 4.4, was diese messen und wofür die Sensoren typischerweise eingesetzt werden[2]. Darüber hinaus kennt ChatGPT die Bedeutung der Bezeichner der in den Kontextmodellen verwendeten Ontologien. Dabei kann ChatGPT auch für proprietäre Ontologien wie z.B. „iot-context:hasMeasurement“ sinnvolle Beschreibungen liefern[3]. Um die Fähigkeit von ChatGPT zu testen, Knowledge Graphen in eine Textbeschreibung umzuwandeln, wurden die zwei verfügbaren IoT-Kontextmodelle zum Testen verwendet. Die beiden Kontextmodelle sind in Abbildung 4.2 und Abbildung 4.4 abgebildet. Dabei wurde die RDF-Beschreibungen der beiden Knowledge Graphen verwendet. Die RDF-Beschreibungen finden sich in Anhang A in Listing A.1 und Listing A.2. Bei den RDF-Beschreibungen wurden jedoch die Attribute „iot-context:hasTimeStamp“ und „iot-context:hasValue“ entfernt, da es sich hierbei um Messwerte handelt, die keinen Mehrwert für die allgemeine Beschreibung des Kontextmodells liefern. Für die Generierung der Textbeschreibungen wurde der folgende Prompt verwendet:

[2] ChatGPT-Konversation 1: https://chat.openai.com/share/c92b4a52-bd23-4fb1-b0f2-127aadbc3712
[3] ChatGPT-Konversation 2: https://chat.openai.com/share/fdb2a4a3-9571-4d87-8aa9-045e8b25d1ff

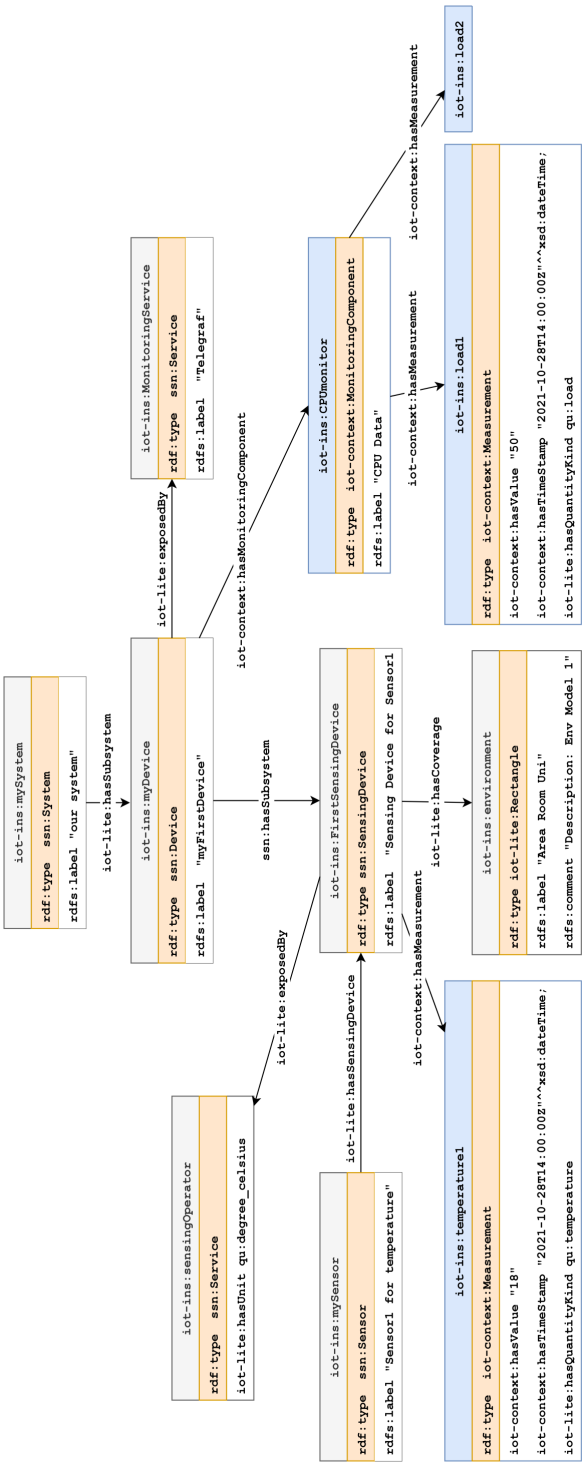


Abbildung 4.2.: Graph eines Kontextmodells mit einem Temperatursensor und einer Monitoring-Komponente. Das Kontextmodell stammt aus [DABS22].

```
{
    "id":1,
    "kbs": {
        "W100": [
            "myFirstDevice",
            "myFirstDevice",
            [[
                "hasSubsystem",
                "our system"
            ]]],
        "W101": [
            "Telegraf",
            "Telegraf",
            [[
                "exposedBy",
                "myFirstDevice"
            ]]],
        "W102": [
            "CPU Data",
            "CPU Data",
            [[
                "hasMonitoringComponent",
                "myFirstDevice"
            ]]],
        "W103": [
            "load2",
            "load2",
            [[
                "hasMeasurement",
                "CPU Data"
            ]]],
        "W104": [
            "load1",
            "load1",
            [[
                "hasMeasurement",
                "CPU Data"
            ]]],
        ...
        "W109": [
            "sensingOperator",
            "sensingOperator",
            [[
                "exposedBy",
                "SensingDevice for Sensor1"
            ]]]
    }
}
```

Listing 4.1: JSON-Beschreibung des Kontextmodells aus Abbildung 4.2. Die JSON-Beschreibung wurde aus Platzgründen gekürzt.

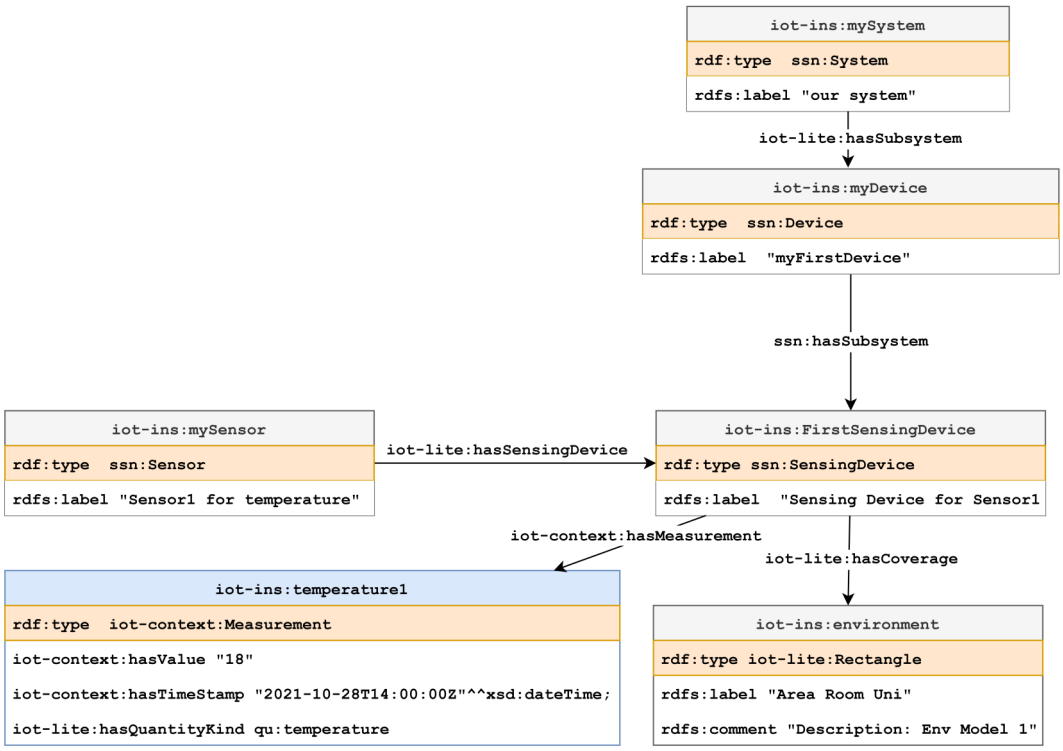


Abbildung 4.3.: Reduzierter Graph des Kontextmodells aus Abbildung 4.2. Die Monitoring-Komponente und der Monitoring-Service sind hier entfernt. Das ursprüngliche Kontextmodell stammt aus [DABS22].

> I want you to act as a KG-to-Text model and convert the knowledge graph you receive into a text description.
>
> The knowledge graph is provided in the RDF format.
>
> I want you to only reply with the description and nothing else. I want you to use the following rules when writing the descriptions:
> only use natural language,
> include the type of measurement the sensors take,
> do not use the rdf:label attribute except to describe the environment the sensors are in.
> Keep the description as short as possible.

In diesem Promt wird ChatGPT zunächst mitgeteilt, dass es einen Graphen in Text umwandeln soll und dass dieser Graph im RDF-Format beschrieben ist. Außerdem wird die Regel aufgestellt, dass nur mit der Textbeschreibung des Graphen geantwortet werden soll. Ohne diese Regel kann es passieren, dass ChatGPT eine ausführliche Antwort ausgibt, die für diese Anwendung unnötigen Text enthält. Die Regel, dass in natürlicher Sprache geantwortet werden soll, wird verwendet, um im Gegensatz zu JointGT ganze Sätze zu erzeugen. Die Regel, dass die Maßeinheiten der Sensoren in der Beschreibung enthalten sein sollten, wird verwendet, da beispielsweise bestimmte Datenvalidierungsmethoden für bestimmte Anwendungsgebiete besser geeignet sein können als andere. Außerdem liefern die Maßeinheiten einige Informationen darüber, um welche Art von

IoT-Anwendung es sich handelt. Damit die Struktur der IoT-Anwendung beschrieben wird und nicht nur die vom Benutzer gewählten Bezeichner für die einzelnen Komponenten verwendet werden, wird die Regel aufgestellt, dass ChatGPT das Attribut „rdf:label“ nicht verwenden soll. Eine Ausnahme soll jedoch für die Environments gemacht werden, da die Ontologien hier meist nur grobe Beschreibungen wie „iot-lite:Rectangle“ oder „iot-lite:Circle“ vorgeben. Abschließend wird darauf hingewiesen, dass die Beschreibung so kurz wie möglich gehalten werden soll, damit die für die Beschreibung verwendeten Wörter so aussagekräftig wie möglich sind. Auf eine genaue Vorgabe, wie z.B. drei Sätze, wurde hier verzichtet, da ChatGPT in der Regel nicht in der Lage ist, eine solche konkrete Beschränkung einzuhalten.

Mit Hilfe dieses Prompts und der RDF Beschreibung aus Listing A.1 generiert ChatGPT die folgende Beschreibung für das Kontextmodell in Abbildung 4.2[4]:

> In our system, there is a sensing device known as „Sensing Device for Sensor1“ that measures temperature in the environment labeled „Area Room Uni“. This device is part of a subsystem and is exposed by a monitoring service named „Telegraf“. Additionally, there is a monitoring component labeled „CPU Data“ that measures load1 and load2, and these measurements quantify the load of the system. The unit of measurement for temperature is degrees Celsius, and for load, it is unspecified. The sensing device is associated with Sensor1 for temperature.

Für das Kontextmodell in Abbildung 4.4 kann mit Hilfe des oben genannten Prompts und der RDF-Beschreibung aus Listing A.2 die folgende Beschreibung für das Kontextmodell generiert werden[5]:

> MA TestSystem consists of multiple devices, including device_main and actuatingDevice. device_in_1 and device_in_2 have sensing devices (esp8266_1 and esp8266_2) that monitor temperature using ds18b20 sensors. device_out has sensing devices (aqara_multisensor_1 and aqara_multisensor_2) monitoring temperature with aqara_temp sensors. These sensors measure temperature in degrees Celsius. The sensors are deployed in room1 and room2, as well as outside. The measurements are monitored by the Telegraf service, and the NET component is being monitored, measuring load.

Im Vergleich zu den Beschreibungen von JointGT enthalten die Beschreibungen von ChatGPT deutlich mehr Komponenten der Kontextmodelle. Außerdem werden die einzelnen Komponenten des Systems nicht nur mit Füllwörtern nacheinander aufgelistet, sondern ChatGPT gibt Sätze aus, die der natürlichen Sprache ähneln. Ein weiterer Vorteil ist, dass kein Modell trainiert werden muss. Dadurch können auch Kontextmodelle in Text umgewandelt werden, die sich stark von den bisher bekannten Kontextmodellen unterscheiden. Aus diesem Grund wird ChatGPT in dieser Arbeit als KG-to-Text Modell verwendet.

Zusätzlich kann ChatGPT das System auch in Stichworten beschreiben. Um Stichwörter zu erhalten, kann ChatGPT nach dem obigen Prompt folgende Anweisung gegeben werden:

[4] ChatGPT-Konversation 3: https://chat.openai.com/share/e72222fd-a852-4a5b-b325-f16194315254
[5] ChatGPT-Konversation 4: https://chat.openai.com/share/38554f47-f344-407b-9539-f0fc0e12b820

> Now describe the knowledge graph with only five keywords. I want you to only reply with the keywords and nothing else. Separate the keywords with a comma. Do not use the following keywords: System, Device, Sensing, Monitoring Service, Environment, Measurement, Sensor

Dabei werden auch Wörter aufgelistet, die ChatGPT nicht verwenden soll, da es sich hierbei nur um Bezeichner der verschiedenen Ontologien handelt, wie z.B. „ssn:System“. Diese Wörter können je nach verwendeter Ontologie geändert werden. Für das Kontextmodell aus Abbildung 4.2 gibt ChatGPT die folgenden Stichwörter aus[6]: „Temperature, Load, CPU, Area Room Uni, Telegraf“. Für das Kontextmodell aus Abbildung 4.4 werden die folgenden Stichwörter generiert[7]: „Temperature, Actuating Device, Quantity Kind, Unit, Resolution“.

Der Ablauf der KG-to-Text Komponente ist in Abbildung 4.5 dargestellt. Das Kontextmodell wird zunächst in eine RDF Beschreibung umgewandelt, die dann von der KG-to-Text Komponente in eine Textbeschreibung umgewandelt wird.

4.4. Aufbau des textbasierten Recommender Systems

In diesem Abschnitt wird der Aufbau des für diese Arbeit entwickelten Recommender Systems vorgestellt. Da es sich um ein textbasiertes Recommender System handelt, besteht es aus zwei Komponenten: Feature Selection und Computational Approach [KNMN21]. Die Feature Selection ist für die Generierung von Vektoren verantwortlich, die anschließend vom Computational Approach verwendet werden, um passende Empfehlungen zu produzieren. Die Feature Selection verwendet verschiedene Methoden aus dem Bereich des Natural Language Processing (NLP) und ist notwendig, da es nicht möglich ist, die Empfehlungen direkt auf dem Text zu berechnen. Für die Computational Approaches werden zwei Ähnlichkeitsmaße verwendet, die die Ähnlichkeit der Vektoren aus der Feature Selection messen, sowie ein Verfahren, das auf einem neuronalen Netz basiert. Es werden sowohl mehrere Feature Selection Methoden als auch Computational Approaches verwendet, um beurteilen zu können, welche Kombination aus Feature Selection Methoden und Computational Approaches für die Zielsetzung dieser Arbeit am besten geeignet ist. Die Ähnlichkeitsmaße basieren auf [KNMN21] und die Methode auf Basis eines Neuronalen Netzes auf [CSY+21]. In Abbildung 4.6 ist die grobe Struktur des Recommender Systems dargestellt. Dabei werden die Vektoren aus der Feature Selection an die Computational Approaches übergeben, die dann die Empfehlungen der Datenvalidierungsmethoden berechnen.

4.4.1. Umwandlung von Textbeschreibungen in Vektoren

Für die Umwandlung von Textbeschreibungen in Vektoren werden sogenannte Feature Selection Methoden verwendet. Die Umwandlung in Vektoren ist notwendig, da die Computational Approaches nur mit Zahlen und nicht mit Wörtern arbeiten können. Für die Feature Selection wird die Term

[6] ChatGPT-Konversation 3: https://chat.openai.com/share/e72222fd-a852-4a5b-b325-f16194315254
[7] ChatGPT-Konversation 4: https://chat.openai.com/share/38554f47-f344-407b-9539-f0fc0e12b820

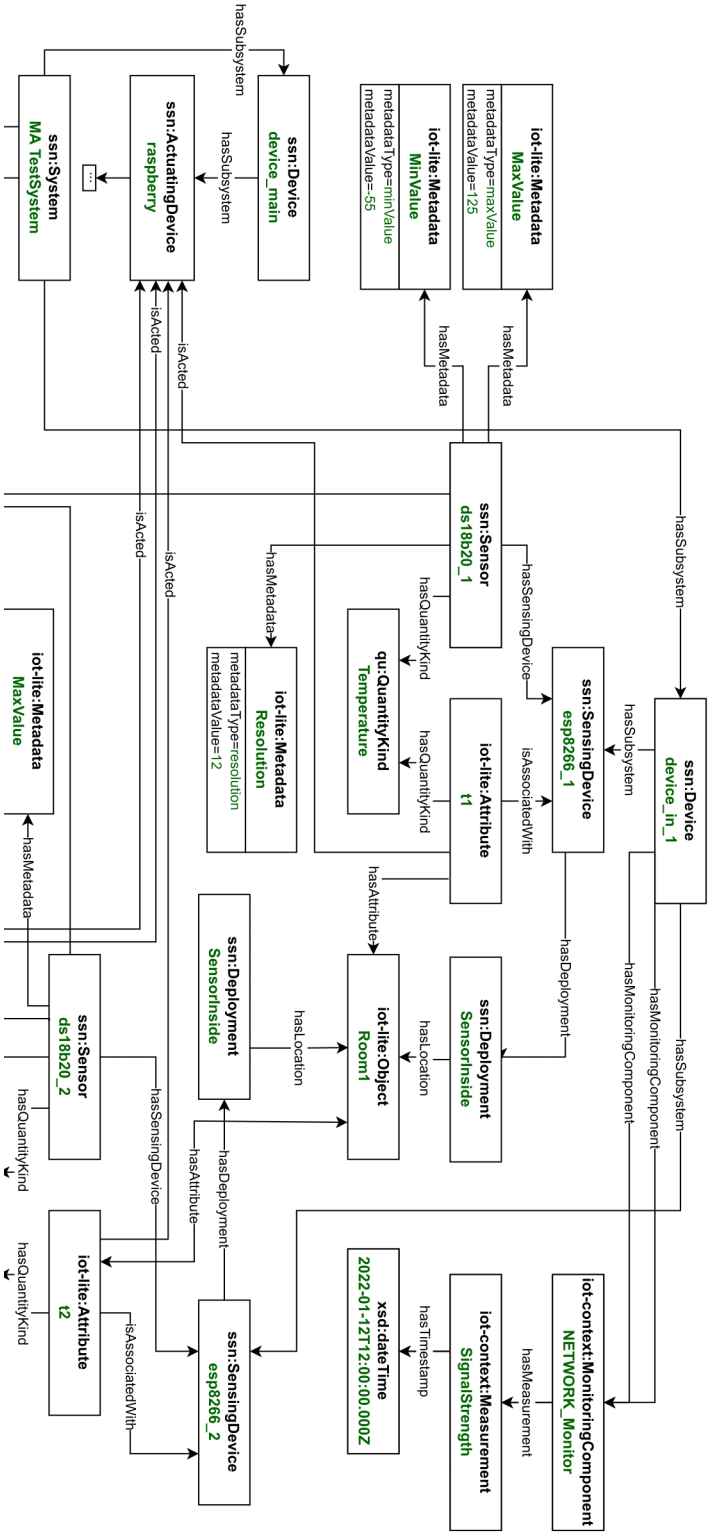


Abbildung 4.4.: Kontextmodell eines IoT-Szenarios mit mehreren Sensoren und Komponenten. Aus Platzgründen wird nur ein Ausschnitt des Kontextmodells gezeigt. Das Kontextmodell stammt aus [Sch22]

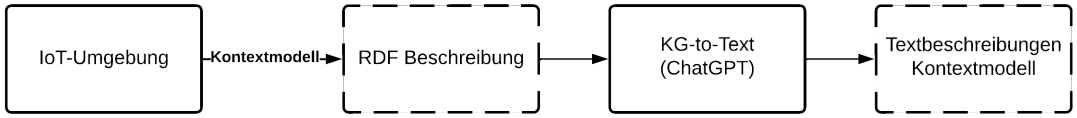


Abbildung 4.5.: Umwandlung von Kontextmodellen in Textbeschreibungen

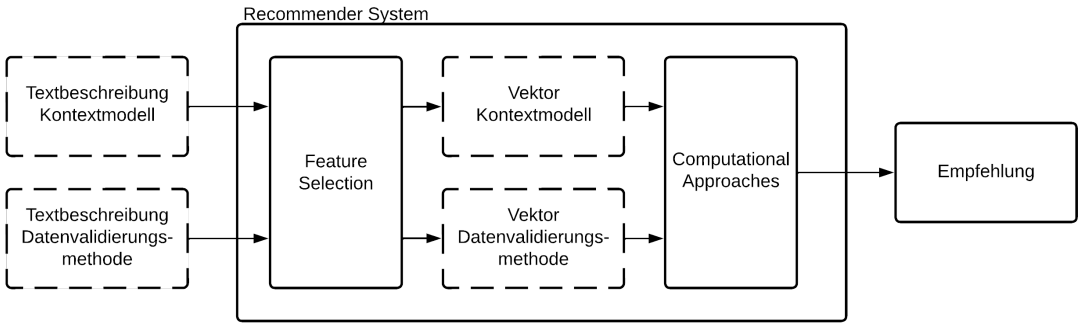


Abbildung 4.6.: Aufbau des textbasierten Recommender Systems, bestehend aus einer Feature Selection, die aus den Textbeschreibungen Vektoren erzeugt, und einem Computational Approach, der auf Basis der Vektoren eine Empfehlung produziert.

Frequency - Inverse Document Frequency und verschiedene Word Embeddings verwendet. In den folgenden Abschnitten wird begründet, warum diese Methoden verwendet werden und welche Vor- und Nachteile sie für den Anwendungsfall dieser Arbeit haben.

Term Frequency - Inverse Document Frequency

Das Term Frequency - Inverse Document Frequency (TF-IDF) Maß ist eine der einfacheren Methoden und gehört zur Klasse der Keyword basierten Techniken, bei denen wichtige Begriffe aus einem Dokument oder Text extrahiert werden. Obwohl TF-IDF eine einfache Methode ist, wird sie auch in neueren textbasierten Recommender Systemen verwendet [KNMN21]. Die Methode wird für die Zielsetzung dieser Arbeit untersucht, da sie eine häufig verwendete Methode in textbasierten Recommender Systemen ist und zudem einfach auf Textbeschreibungen angewendet werden kann. Dabei werden Textbeschreibungen, die die gleichen Wörter enthalten, in ähnliche Vektoren umgewandelt. Wenn beispielsweise die Beschreibung des Kontextmodells die Domäne/Anwendungsgebiet des IoT-Systems enthält und die Beschreibung der Datenvalidierungsmethode angibt, dass diese gut für das Anwendungsgebiet des Kontextmodells funktioniert, werden diese Beschreibungen in ähnliche Vektoren umgewandelt.

TF-IDF funktioniert, indem es die Häufigkeit eines Wortes in einem Text misst und diese Häufigkeit dann mit seiner Relevanz gewichtet. Die Relevanz wird mit Hilfe der inversen Dokumentfrequenz gemessen, d.h. es wird gemessen, wie spezifisch ein bestimmter Begriff in einem Text ist. Die Idee dahinter ist, dass ein spezifischer Begriff seltener vorkommt und auch eine höhere Aussagekraft hat als häufig vorkommende Wörter.

Wort	TF		IDF	TF-IDF	
	Satz 1	Satz 2		Satz 1	Satz 2
ein	$\frac{1}{7}$	$\frac{1}{7}$	$log(\frac{2}{2}) = 0$	0	0
iot	$\frac{1}{7}$	$\frac{1}{7}$	$log(\frac{2}{2}) = 0$	0	0
system	$\frac{1}{7}$	$\frac{1}{7}$	$log(\frac{2}{2}) = 0$	0	0
zur	$\frac{1}{7}$	$\frac{1}{7}$	$log(\frac{2}{2}) = 0$	0	0
überwachung	$\frac{1}{7}$	0	$log(\frac{2}{1}) = 0.69$	$\frac{1}{7} * 0.69 = 0.098$	0
der	$\frac{1}{7}$	$\frac{1}{7}$	$log(\frac{2}{2}) = 0$	0	0
temperatur	$\frac{1}{7}$	0	$log(\frac{2}{1}) = 0.69$	$\frac{1}{7} * 0.69 = 0.098$	0
feststellung	0	$\frac{1}{7}$	$log(\frac{2}{1}) = 0.69$	0	$\frac{1}{7} * 0.69 = 0.098$
luftfeuchtigkeit	0	$\frac{1}{7}$	$log(\frac{2}{1}) = 0.69$	0	$\frac{1}{7} * 0.69 = 0.098$

Tabelle 4.2.: TF-IDF Beispiel für die Sätze „Ein IoT System zur Überwachung der Temperatur“ und „Ein IoT System zur Feststellung der Luftfeuchtigkeit“. Groß- und Kleinschreibung wurde bei der Berechnung ignoriert.

Um das TF-IDF-Maß für jedes Wort zu erfassen, wird ein Vektor erstellt, der für jedes Wort einen Eintrag hat. Es wird also ein m-dimensionaler Vektor erstellt, wobei m die Anzahl der verschiedenen Wörter in allen Beschreibungen ist. Die j-te Position des Vektors gibt dann das TF-IDF-Maß für das j-te Wort an. Die Berechnung des TF-IDF-Maßes erfolgt nach [KNMN21] wie folgt:

$$w_i = \mathrm{tf}_i * log\left[\frac{n}{\mathrm{if}_i}\right]$$

Dabei steht tf_i für die relative Häufigkeit eines Wortes in der Beschreibung, n für die Anzahl der verschiedenen Beschreibungen und if_i für die Anzahl der Beschreibungen, die das Wort i enthalten.

Ein Beispiel für die Berechnung des TF-IDF-Maßes findet sich in Tabelle 4.2. Dort wird das TF-IDF-Maß für die Sätze „Ein IoT System zur Überwachung der Temperatur“ und „Ein IoT System zur Feststellung der Luftfeuchtigkeit“ berechnet. Bei der Berechnung wurde nicht zwischen Groß- und Kleinschreibung unterschieden. Die Spalte TF gibt die relative Häufigkeit der Wörter im ersten und zweiten Satz an. Die IDF gibt die Relevanz der Wörter an. Für Wörter, die in beiden Sätzen vorkommen, ist der IDF-Wert 0, da sie keine zusätzliche Information enthalten. Im Gegensatz dazu haben Wörter, die nur in einem der beiden Sätze vorkommen, einen höheren IDF-Wert, da sie den Inhalt des Satzes besser wiedergeben. Der TF-IDF-Wert wird dann durch Multiplikation von TF und IDF berechnet. In diesem Beispiel sind die Wörter „Überwachung“, „Temperatur“, „Feststellung“ und „Luftfeuchtigkeit“ im Vergleich zu den anderen Wörtern aussagekräftig. Dadurch wird deutlich, dass es sich bei der ersten Beschreibung um die Überwachung der Temperatur und bei der zweiten Beschreibung um die Feststellung der Luftfeuchtigkeit handelt. Somit wäre der TF-IDF Vektor für den ersten Satz $[0, 0, 0, 0, 0.098, 0, 0.098, 0, 0]$ und der Vektor für den zweiten Satz $[0, 0, 0, 0, 0, 0, 0, 0.098, 0.098]$.

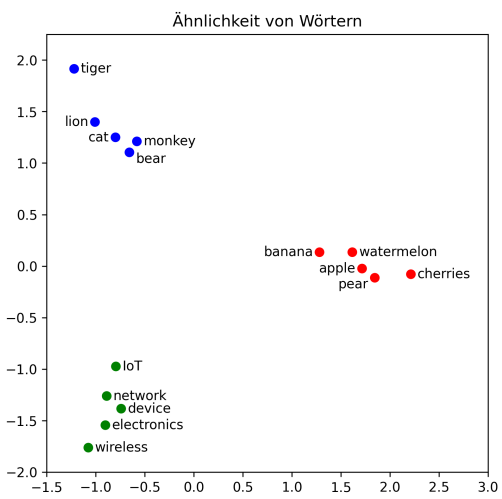
Dies führt zu ähnlichen Vektoren für Sätze, die die gleichen Wörter enthalten. Die Idee dahinter ist, dass Produkte, d.h. Datenvalidierungsmethoden, die für den Nutzer, d.h. Kontextmodelle, in Frage kommen, ähnliche Beschreibungen haben und daher auch ähnliche Wörter enthalten.

Word Embedding Methoden zur Umwandlung von Text in Vektoren

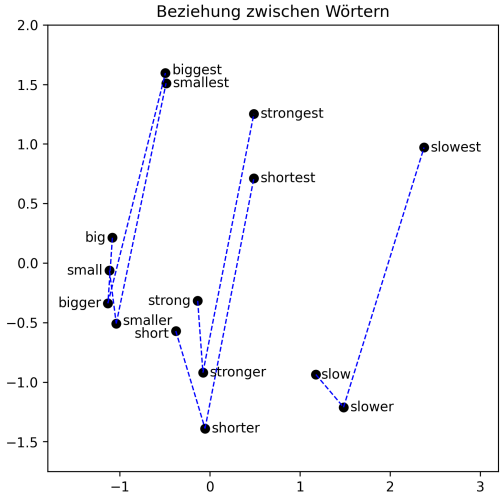
Neben der TF-IDF Methode werden drei Word Embedding Methoden verwendet, um Vektorrepräsentationen von Textbeschreibungen zu erhalten. Ziel der Embedding-Methoden ist es, die Wörter so in Vektoren umzuwandeln, dass ähnliche Wörter in ähnliche Vektoren umgewandelt werden. Außerdem sollen die Beziehungen zwischen den Wörtern auch im Vektorraum erhalten bleiben. Beispielsweise sind Begriffe, die mit „IoT“ verwandt sind, wie „network“ nahe bei „IoT“, während Tiere oder Früchte weiter entfernt sind. Außerdem haben die Steigerungsformen verschiedener Adjektive ähnliche Abstände zu den ursprünglichen Adjektiven. In Abbildung 4.7 sind diese beiden Eigenschaften dargestellt. Die auf Embeddings basierenden Methoden werden für diese Arbeit verwendet, da das Lernen der Repräsentation der Wörter zu aussagekräftigen Vektoren führen kann und die Beziehungen zwischen ähnlichen Wörtern besser erfasst werden können als mit TF-IDF. Zum Beispiel würden die Sätze „Olaf Scholz ist der Bundeskanzler von Deutschland“ und „Joe Biden ist der Präsident der Vereinigten Staaten“ mit TF-IDF zu sehr unterschiedlichen Vektoren führen, da sie fast keine Wörter gemeinsam haben, während Word Embedding Methoden die Beziehungen zwischen Olaf Scholz und Deutschland und Joe Biden und den Vereinigten Staaten lernen können. In IoT-Systemen können so beispielsweise Beschreibungen, die Wörter wie „Heizung“ oder „Temperatur“ enthalten und in die Kategorie Smart Homes fallen, in Vektoren umgewandelt werden, die näher beieinander liegen, wie Vektoren von Beschreibungen, die „EKG“ oder „MRT“ enthalten und in die Kategorie Smart Hospitals fallen. Auf diese Weise erhält man durch ähnliche Beschreibungen auch ähnliche Vektoren, was bei der Auswertung durch die Computational Approaches helfen kann. Für die Embedding Methoden wurden die folgenden drei Methoden ausgewählt: Word2Vec [MCCD13], GloVe [PSM14] und fastText [BGJM17]. Diese drei Modelle wurden ausgewählt, da es sich um häufig verwendete Modelle für textbasierte Recommender Systeme handelt [KNMN21]. Zudem funktionieren die Methoden nach unterschiedlichen Prinzipien, was zu Unterschiedlichen Resultaten führt. Um zu testen, welche Embeddings für Kontextmodelle und Datenvalidierungsmethoden am besten funktionieren, wurden daher diese drei getestet. Wie die Methoden im Detail funktionieren, ist in Abschnitt 2.6 beschrieben.

4.4.2. Vorschlagen geeigneter Datenvalidierungsmethoden anhand der erzeugten Vektoren

Damit das Recommender System passende Empfehlungen vorschlagen kann, muss anhand der erhaltenen Vektoren berechnet werden, welche Nutzer und Produkte - in diesem Fall Kontextmodelle und Datenvalidierungsmethoden - am besten zueinander passen. Bei textbasierten Recommender Systemen wird dieser Teil als Computational Approaches bezeichnet [KNMN21]. Im Folgenden werden verschiedene Verfahren vorgestellt, die versuchen, die besten Empfehlungen zu berechnen. Dabei werden zwei Ähnlichkeitsmaße sowie ein Verfahren auf Basis eines neuronalen Netzes verwendet. Die Ähnlichkeitsmaße eignen sich besonders gut für diese Arbeit, da sie nicht erlernt werden müssen und sofort Empfehlungen berechnen können. Aufgrund des kleinen Datensatzes ist es schwierig, neuronale Netze gut zu trainieren und richtig zu validieren. Dennoch wird in dieser Arbeit eine Methode untersucht, die auf neuronalen Netzen basiert, da diese oft eine bessere Leistung bieten und speziell auf die Daten einer bestimmten Aufgabe trainiert werden. In der Literatur gewinnen neuronale Netze für Recommender Systeme zunehmend an Popularität [KNMN21]. Die in dieser Arbeit verwendeten Ähnlichkeitsmaße sind die Kosinus-Ähnlichkeit und der euklidische



(a) Gruppierung von Wörtern zu IoT, Tieren und Früchten



(b) Beziehungen zwischen den Steigerungen verschiedener Adjektive

Abbildung 4.7.: Darstellung der Vektoren des vortrainierten Word2Vec-Modells [MSC+13] der einzelnen Wörter. Die Reduktion auf zwei Dimensionen erfolgte mittels Principal Component Analysis.

Abstand. Die Auswahl der Ähnlichkeitsmaße erfolgte in Anlehnung an [KNMN21]. Bei dem auf einem neuronalen Netz basierenden Verfahren handelt es sich um die TextCF-Komponente der DEKR Methode [CSY+21].

Euklidischer Abstand

Der euklidische Abstand wird verwendet, um die Distanz zwischen zwei Punkten zu messen. Dabei wird die Länge der Verbindungslinie zwischen den beiden Punkten gemessen [Ras23]. Um den euklidischen Abstand zur Messung der Ähnlichkeit zwischen zwei Textbeschreibungen zu verwenden, werden die Vektoren, die durch eine Feature Selection Methode erzeugt werden, als zwei Punkte betrachtet. Dann wird der euklidische Abstand zwischen diesen beiden Punkten berechnet. Je kleiner der euklidische Abstand ist, desto ähnlicher werden die Textbeschreibungen interpretiert. Daher wird für jedes Paar aus Kontextmodell und Datenvalidierungsmethode der euklidische Abstand berechnet und die Empfehlungen nach diesem Abstand sortiert. Das bedeutet, dass die Datenvalidierungsmethode mit der geringsten Distanz zu einem Kontextmodell an erster Stelle empfohlen wird.

Der euklidische Abstand wird in dieser Arbeit als Ähnlichkeitsmaß verwendet, da er einfach zu berechnen ist und in der Mathematik die am häufigsten verwendete Methode ist, um den Abstand zwischen zwei Punkten zu berechnen [KNMN21]. Außerdem muss kein Modell trainiert werden, was aufgrund des kleinen Datensatzes von Vorteil ist. Darüber hinaus werden bei Embedding basierten Methoden zur Feature Selection ähnliche Wörter in Vektoren umgewandelt, die nahe beieinander liegen. Dies sollte dazu führen, dass ähnliche Beschreibungen nahe beieinander liegen. In Abbildung 4.8 ist ein Beispiel für den euklidischen Abstand zwischen verschiedenen Wörtern dargestellt, die mit dem Word2Vec Modell in Vektoren umgewandelt wurden. Dabei haben Wörter,

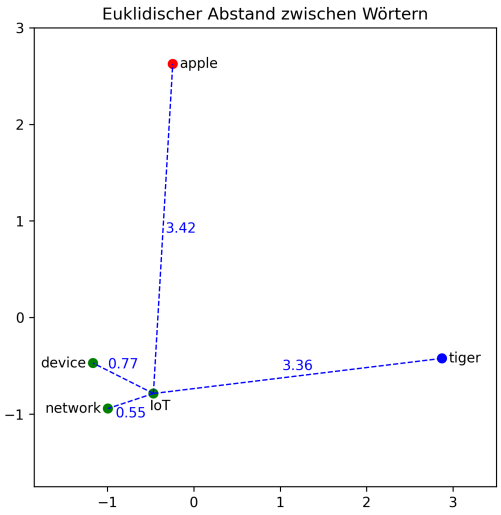


Abbildung 4.8.: Euklidischer Abstand zwischen den Vektoren des Word2Vec Modells von verschiedenen Wörtern. Der euklidische Abstand bezieht sich auf die zwei dimensionale Darstelltung, welche mit Principal Component Analysis erstellt wurde.

die häufig im Bereich des Internet of Things vorkommen, einen kleineren euklidischen Abstand zueinander als Wörter, die nichts mit dem IoT zu tun haben. In der Abbildung haben beispielsweise die Wörter „device“ und „network“ einen kleinen euklidischen Abstand zum Wort „IoT“, während „Greece“ und „tiger“ einen großen Abstand haben.

Der euklidische Abstand wird nach [KNMN21] wie folgt berechnet:

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

wobei a_i bzw. b_i der i-te Eintrag im Vektor a bzw. b ist.

Kosinus-Ähnlichkeit

Da der euklidische Abstand für hohe Dimensionen ungenauer wird [Shr20], wird zusätzlich die Kosinus-Ähnlichkeit untersucht. Der Unterschied zwischen dem euklidischen Abstand und der Kosinus-Ähnlichkeit besteht darin, dass die Kosinus-Ähnlichkeit die Ähnlichkeit anhand des Winkels der Vektoren misst. Insbesondere bei vielen Dimensionen, wie es bei Word Embeddings der Fall ist, kann der euklidische Abstand an Aussagekraft verlieren [Shr20]. Der Winkel zwischen verwandten Wörtern bleibt jedoch auch bei hohen Dimensionen klein, so dass die Kosinus-Ähnlichkeit bei hohen Dimensionen ein besseres Ergebnis liefern kann. Eine stark vereinfachte Darstellung findet sich in Abbildung 4.9. Hier ist der euklidische Abstand zwischen den beiden Vektoren relativ groß, da die Vektoren unterschiedlich lang sind, während die Kosinus-Ähnlichkeit gegen eins geht, da der Winkel zwischen den Vektoren klein ist.

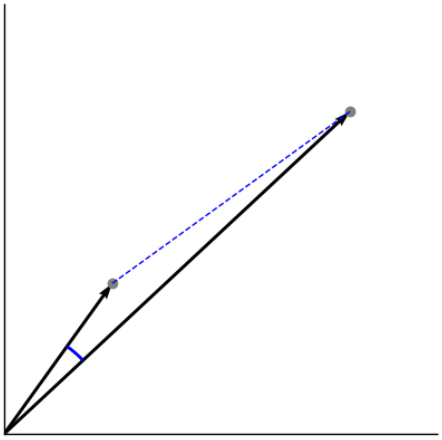


Abbildung 4.9.: Unterschied zwischen der euklidischen Distanz und der Kosinus-Ähnlichkeit. Der euklidische Abstand ist durch die blau gestrichelte Linie dargestellt. Der Winkel für die Kosinus-Ähnlichkeit ist blau dargestellt.

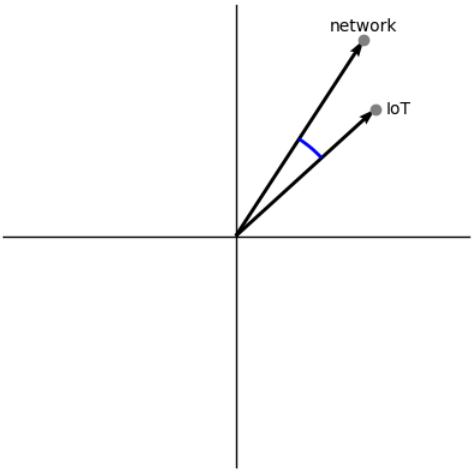
Die Kosinus-Ähnlichkeit ist ein Maß für die Ähnlichkeit zweier Vektoren. Dazu wird der Kosinus des Winkels zwischen den beiden Vektoren berechnet, indem das Skalarprodukt der beiden Vektoren durch das Produkt ihrer Längen dividiert wird. Die Ähnlichkeit der beiden Vektoren wird im Bereich $[-1, 1]$ angegeben. Je ähnlicher sich die Vektoren sind, d.h. je kleiner der Winkel zwischen ihnen ist, desto mehr geht die Kosinus-Ähnlichkeit gegen 1. Wenn die Vektoren nicht ähnlich sind und der Winkel zwischen den Vektoren groß ist, geht die Kosinus-Ähnlichkeit gegen -1 [Tec23]. In Abbildung 4.10 ist dargestellt, wie sich die Kosinus-Ähnlichkeit in Abhängigkeit vom Winkel zwischen den beiden Vektoren ändert.

Für den Anwendungsfall der Textbeschreibungen bedeutet dies, dass zwei Textbeschreibungen ähnlich sind und somit die Datenvalidierungsmethode mit höherer Wahrscheinlichkeit zum Kontextmodell passt, wenn die Kosinus-Ähnlichkeit gegen eins geht. Wenn die Datenvalidierungsmethode und das Kontextmodell nicht zusammenpassen, geht die Kosinus-Ähnlichkeit gegen 0 oder -1. Wie bei der euklidischen Distanz wird die Kosinus-Ähnlichkeit für jedes Kontextmodell-Datenvalidierungsmethode-Paar berechnet und die Empfehlungen werden nach der Kosinus-Ähnlichkeit sortiert. Datenvalidierungsmethoden mit einer Kosinus-Ähnlichkeit nahe 1 werden zuerst empfohlen.

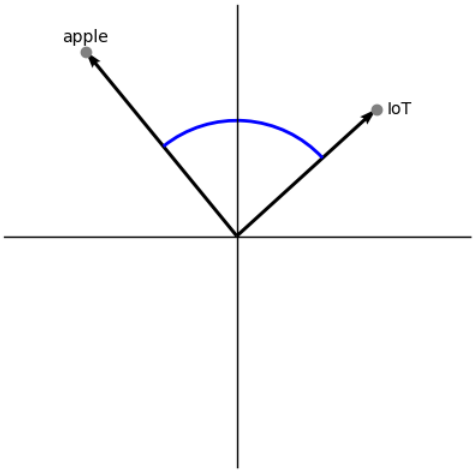
Die Kosinus-Ähnlichkeit wird in dieser Arbeit verwendet, da sie ebenfalls einfach zu berechnen ist und kein Modell dafür trainiert werden muss. Außerdem ist die Kosinus Ähnlichkeit eine der am häufigsten verwendeten Methoden für textbasierte Recommender Systeme, um die Ähnlichkeit zwischen einem Nutzer und einem Produkt anhand eines Vektors zu bestimmen [KNMN21]. Laut den Autoren wird die Kosinus-Ähnlichkeit wie folgt berechnet:

$$cos(a, b) = \frac{ab}{\|a\|\|b\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} (a_i)^2}\sqrt{\sum_{i=1}^{n} (b_i)^2}}$$

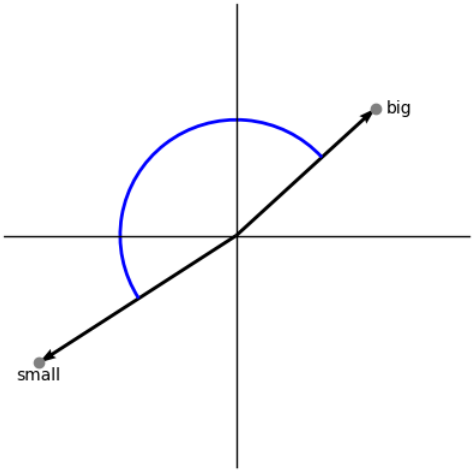
wobei a_i bzw. b_i der i-te Eintrag im Vektor a bzw. b ist.



(a) Bei ähnlichen Vektoren ist der Winkel zwischen ihnen relativ klein und die Kosinus-Ähnlichkeit geht gegen 1



(b) Für Vektoren, bei denen der Winkel etwa 90 Grad beträgt, die also nicht ähnlich sind, geht die Kosinus-Ähnlichkeit gegen 0



(c) Bei entgegengesetzten Vektoren geht die Kosinus-Ähnlichkeit gegen -1

Abbildung 4.10.: Kosinus-Ähnlichkeit von ähnlichen, unähnlichen und entgegengesetzten Vektoren. Der Winkel zwischen den Vektoren ist in blau dargestellt.

Text-based Collaborative Filtering

Im Gegensatz zum euklidischem Abstand und der Kosinus-Ähnlichkeit basiert dieses Verfahren auf neuronalen Netzen und lernt so, passende Empfehlungen vorzuschlagen, anstatt sie mit einem Ähnlichkeitsmaß zu berechnen. Dieses Verfahren ist die Komponente Text-based Collaborative Filtering (TextCF) der DEKR Methode [CSY+21]. Dabei wird Collaborative Filtering auf den Textbeschreibungen der Datenvalidierungsmethoden und Kontextmodelle durchgeführt. In diesem Fall ist es Möglich Collaborative Filtering anzuwenden, da das neuronale Netz auch auf Textbeschreibungen ähnlicher Datensätze mit den zugehörigen Datenvalidierungsmethoden trainert werden kann. Mit den klassischen Collaborative Filtering Methoden wäre das nicht möglich. Das

bedeutet, dass die Textbeschreibung eines neuen Textmodells mit den Beschreibungen anderer bereits bekannter Kontextmodelle verglichen wird und die Kontextmodelle ausgewählt werden, die eine ähnliche Beschreibung aufweisen. Anschließend werden Datenvalidierungsmethoden für das neue Kontextmodell vorgeschlagen, die sich für ähnliche, bereits bekannte Kontextmodelle bewährt haben. Im Gegensatz zu den beiden Ähnlichkeitsmaßen kann das neuronale Netz auch nichtlineare Beziehungen zwischen Beschreibungen erfassen. Die Autoren der DEKR Methode verwenden für die Erstellung der Vektoren die Embeddings des GloVe Modells. Neben den Embeddings des GloVe Modells werden in dieser Arbeit auch die Embeddings des Word2Vec und des fastText Modells verwendet. Als Grundbaustein für die TextCF-Komponente verwenden die Autoren das Neural Collaborative Filtering Framework von He et al. [HLZ+17].

Die TextCF Komponente wird für diese Arbeit verwendet, da DEKR ein ähnliches Ziel wie diese Arbeit verfolgt. Anstatt geeignete Datenvalidierungsmethoden für Kontextmodelle vorzuschlagen, wird versucht, geeignete Machine Learning Methoden für Datensätze vorzuschlagen. Dabei verwenden die Autoren textuelle Beschreibungen der Datensätze und Methoden. Da die Textbeschreibungen für beide Problemstellungen ähnlich sein sollten, wird auch in dieser Arbeit TextCF verwendet. Außerdem kann ein neuronales Netz Beziehungen zwischen Kontextmodellen und Datenvalidierungsmethoden besser lernen als ein Ähnlichkeitsmaß. Ein mögliches Problem bei der Verwendung von neuronalen Netzen ist jedoch, dass der Datensatz, mit dem das Modell trainiert wird, eine gewisse Größe haben sollte. Der Datensatz der Autoren von DEKR ist deutlich größer als der Datensatz dieser Arbeit.

Im Folgenden wird die Funktionsweise des Neural Collaborative Filtering Frameworks [HLZ+17] vorgestellt. Das Ziel von Neural Collaborative Filtering ist es, die Matrixfaktorisierung, die eine beliebte Methode für Collaborative Filtering ist, durch ein neuronales Netzwerk zu ersetzen, das in der Lage ist, beliebige Funktionen zu erlernen. Dadurch ist es möglich, neben den linearen Funktionen, die bei der Matrixfaktorisierung verwendet werden, auch nichtlineare Funktionen zu verwenden. Bei der Matrixfaktorisierung werden die Empfehlungen durch die Berechnung des Skalarprodukts der latenten Feature-Vektoren erzeugt. Diese latenten Feature-Vektoren werden durch Optimierung der Matrixfaktorisierung gelernt. Nach Ansicht der Autoren ist die lineare Funktion jedoch nicht immer in der Lage, die komplexe Struktur der Interaktionen zu erfassen. Durch die zusätzliche Verwendung von nichtlinearen Funktionen soll das Modell in der Lage sein, die komplexen Interaktionen abzubilden und somit eine bessere Performance zu bieten. Um dies zu erreichen, versucht Neural Collaborative Filtering die Funktion zur Berechnung der Empfehlungen mit Hilfe der latenten Feature-Vektoren zu erlernen, anstatt diese fest vorzugeben.

Der grobe Aufbau des Neural Collaborative Filter Frameworks ähnelt dem der Matrixfaktorisierung. Zuerst wird ein latenter Vektor erstellt, der auf dem Benutzer und dem Produkt basiert, in diesem Fall auf dem Kontextmodell und der Datenvalidierungsmethode. Diese beiden Vektoren dienen dann als Input für ein nuronales Netzwerk, das einen Wert für die Wahrscheinlichkeit vorhersagt, dass Nutzer und Produkt zusammenpassen. Bei der Matrixfaktorisierung wird anstelle des neuronalen Netzes das skalare Produkt verwendet.

Für die Erstellung des neuronalen Netzes stellen die Autoren drei Komponenten vor: Generalized Matrix Factorization (GMF), Multi-Layer Perceptron (MLP) und NeuMF. Die GMF-Komponente soll dazu dienen, die Matrixfaktorisierung zu imitieren und somit die Matrixfaktorisierung mit Hilfe des Neural Collaborative Filtering Frameworks zu ermöglichen. Dazu verwendet die GMF-Komponente wie die Matrixfaktorisierung das Skalarprodukt. Im Detail sieht die Ebene wie folgt aus:

$$\hat{y}_{ui} = a_{out}(\mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i))$$

Dabei steht a_{out} für die Aktivierungsfunktion und $\mathbf{h}$ für die Gewichte der Output Ebene. $\mathbf{p}_u$ bzw. $\mathbf{q}_i$ stehen für den latenten Vektor des Nutzers bzw. des Produkts. Die Autoren merken an, dass, wenn für a_{out} die Identitätsfunktion und für $\mathbf{h}$ ein Vektor mit 1 an jeder Position gewählt würde, die GMF-Ebene genau die Matrixfaktorisierung darstellen würde. Für ihre Implementierung wählen die Autoren jedoch die Sigmoidfunktion $\sigma(x) = \frac{1}{1+e^{-x}}$ für a_{out} und lernen $\mathbf{h}$.

Die MLP-Komponente soll das Lernen von Interaktionen zwischen latenten Vektoren, der Nutzer und Produkten ermöglichen. Als Input dient die Konkatination der latenten Vektoren des Nutzers und des Produktes. Insbesondere soll diese Ebene in der Lage sein, auch nichtlineare Interaktionen zwischen den beiden Vektoren zu lernen, die von der GMF-Komponente bzw. der Matrixfaktorisierung nicht gelernt werden können. Um dies zu erreichen, verwenden die Autoren ein Standard-Multilayer-Perceptron, das aus mehreren hidden Layern besteht. Im Detail sieht die Komponente wie folgt aus:

$$\begin{aligned}
\mathbf{z}_1 &= \phi_1(\mathbf{p}_u, \mathbf{q}_i) = \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix} \\
\phi_2(\mathbf{z}_1) &= a_2(\mathbf{W}_2^T \mathbf{z}_1 + \mathbf{b}_2) \\
&\dots \\
\phi_L(\mathbf{z}_{L-1}) &= a_L(\mathbf{W}_L^T \mathbf{z}_{L-1} + \mathbf{b}_L) \\
\hat{y}_{ui} &= \sigma(\mathbf{h}^T \phi_L(\mathbf{z}_{L-1}))
\end{aligned}$$

wobei W_x die Matrix mit den Gewichten, $\mathbf{b}_x$ der Bias-Vektor und a_x die Aktivierungsfunktion der x-ten Ebene ist. Die Autoren weisen darauf hin, dass die Aktivierungsfunktion frei gewählt werden kann, empfehlen jedoch die ReLU-Funktion: $f(x) = max(0, x)$. Außerdem empfehlen die Autoren, die unterste Ebene so breit wie möglich zu wählen und die folgenden Ebenen immer schmaler zu gestalten.

Die letzte Komponente, die NeuMF-Komponente, hat die Funktion, GMF und MLP zusammenzuführen. Die Komponente sieht wie folgt aus:

$$\begin{aligned}
\phi^{GMF} &= \mathbf{p}_u \odot \mathbf{q}_i \\
\phi^{MLP} &= a_L(\mathbf{W}_L^T(a_{L-1}(...a_2(\mathbf{W}_2^T \left[\mathbf{p}_u\ \mathbf{q}_i\right] + \mathbf{b}_2)...)) + \mathbf{b}_L) \\
\hat{y}_{ui} &= \sigma(\mathbf{h}^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix})
\end{aligned}$$

Dabei ist $\hat{y}_{ui}$ die Wahrscheinlichkeit, dass Nutzer und Produkt zusammenpassen.

In Abbildung 4.11 ist der Aufbau des Neural Collaborative Filtering Framework dargestellt. Für die Eingabe in die GMF Layer und die MLP Layer wird jeweils der Vektor des Kontextmodells und der Datenvalidierungsmethode verwendet. Dieser Vektor wird erstellt, indem der Durchschnitt

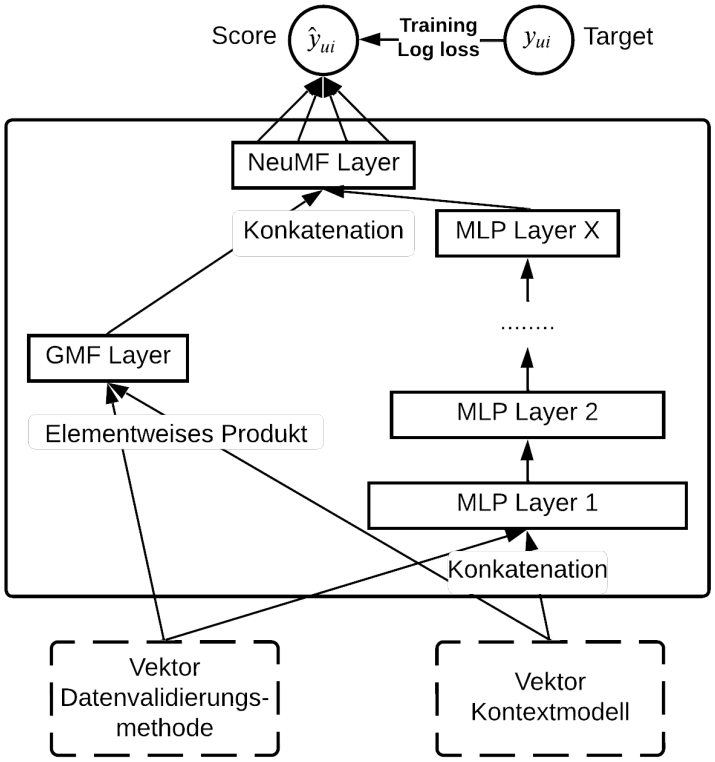


Abbildung 4.11.: Aufbau des Neural Collaborative Filtering Frameworks nach [HLZ+17]

der Word Embeddings der einzelnen Wörter der Beschreibung gebildet wird. $\hat{y}_{ui}$ beschreibt dann die Wahrscheinlichkeit, dass das Kontextmodell und die Datenvalidierungsmethode zusammen passen.

4.4.3. Kombination der Methoden zur Umwandlung von Textbeschreibungen in Vektoren und der Methoden zum Vorschlagen von Datenvalidierungsmethoden

Um die beste Kombination zwischen Feature Selection Methoden und Computational Approaches zu finden, werden alle Feature Selection Methoden als Input für die Computational Approaches verwendet. Eine Ausnahme bildet die Kombination aus TF-IDF und TextCF. Diese Kombination wird nicht verwendet, da die TF-IDF Vektoren eine zu hohe Dimension haben, so dass die TextCF Komponente nicht mehr in der Lage ist, sinnvolle Beziehungen zwischen den Vektoren zu erlernen. Stattdessen werden für die TextCF Komponente nur die Word Embedding Methoden verwendet, da alle vorgestellten Word Embedding Methoden eine maximale Dimension von 300 haben. Eine Veranschaulichung, wie die Kombination der Feature Selection Methoden und der Computational Approaches aussieht, ist in Abbildung 4.12 gegeben. Welche der Kombinationen am besten für das Vorschlagen von Datenvalidierungsmethoden geeignet ist, wird in Kapitel 6 evaluiert.

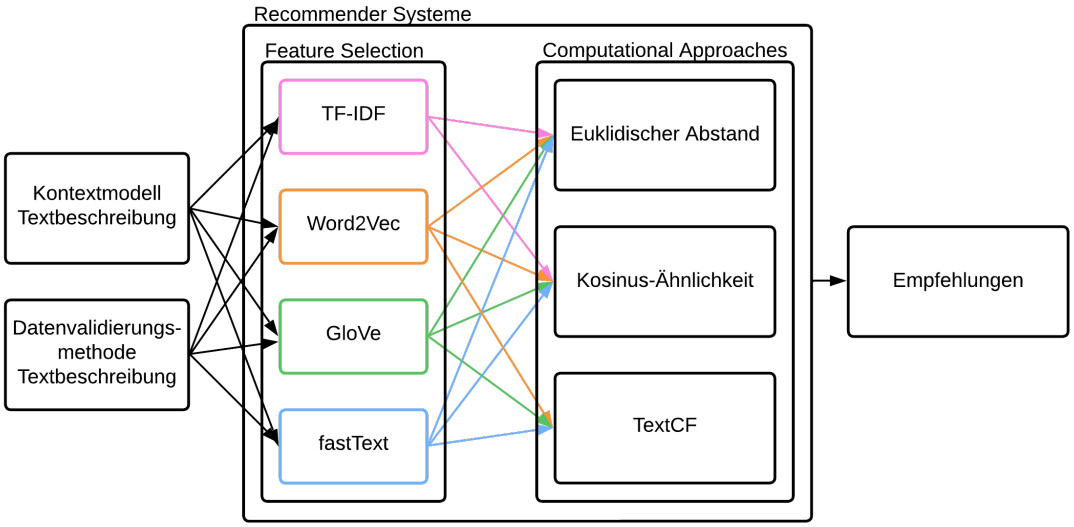


Abbildung 4.12.: Mögliche Kombinationen der Feature Selection Methoden und der Computational Approaches

5. Implementierung

In diesem Kapitel wird die prototypische Implementierung der vorgestellten Konzepte beschrieben. Die gesamte prototypische Implementierung ist in Python[1] realisiert, da durch die vielen verfügbaren Pakete für Python bereits viele Funktionalitäten vorhanden sind. Für die Implementierung wurden hauptsächlich Jupyter Notebooks verwendet. Die TextCF Komponente sowie die Methoden zur Evaluierung wurden nicht in einem Jupyter Notebook implementiert. Um die prototypische Implementierung vorzustellen, ist dieses Kapitel in fünf Abschnitte unterteilt. Zunächst wird in Abschnitt 5.1 erklärt, wie ein Datensatz für das Training der Recommender Systeme erstellt wurde. In Abschnitt 5.2 wird gezeigt, wie die Kontextmodelle automatisch in eine Textbeschreibung umgewandelt werden können. Anschließend wird in Abschnitt 5.3 beschrieben, wie die Textbeschreibungen modifiziert werden, bevor sie von den Feature Selection Methoden verwendet werden, um sie in Vektoren umzuwandeln. Anschließend wird in Abschnitt 5.4 die Implementierung der verschiedenen Feature Selection Methoden beschrieben, die die Textbeschreibungen in Vektoren umwandeln. Zuletzt wird in Abschnitt 5.5 die Implementierung der verschiedenen textbasierten Recommender Systeme vorgestellt.

5.1. Erstellung des Datensatz

In diesem Abschnitt wird beschrieben, wie der Datensatz für das Training und die Evaluierung der verschiedenen Recommender Systeme erstellt wurde. Da nur wenige verschiedene Kontextmodelle für IoT-Umgebungen verfügbar waren und diese keine geeigneten Datenvalidierungsmethoden enthalten, musste zunächst ein Datensatz erstellt werden, der das Training und die Evaluation der zu untersuchenden Recommender Systeme ermöglicht. Da ein solcher Datensatz meines Wissens weder in der Literatur noch online verfügbar ist, wird im Folgenden beschrieben, wie der Datensatz für diese Arbeit erstellt wurde. Der erstellte Datensatz enthält die textuelle Beschreibung von Datensätzen und Datenvalidierungsmethoden sowie die Performance der Methoden auf den entsprechenden Datensätzen. Dabei wurden also die Kontextmodelle durch unterschiedliche Datensätze ersetzt. Die Wahl fiel auf Datensätze, da für diese in der Literatur bereits verschiedene Datenvalidierungsmethoden evaluiert wurden und sich die Beschreibungen nicht wesentlich von denen der Kontextmodelle unterscheiden. Für die Auswahl geeigneter Datenvalidierungsmethoden wurde eine Auswahl der in [DEK23] vorgestellten Methoden verwendet, da in dieser Arbeit eine Auswahl von Outlier Detection Methoden mit unterschiedlichen Funktionsweisen vorgestellt wird. Die sieben ausgewählten Datenvalidierungsmethoden sind die folgenden: Angle-Based Outlier Detection, Local Outlier Factor, Deep Support Vector Data Description, Clustering Based Local Outlier Factor, k-Nearest-Neighbors, Isolation Forest, Principal Component Analysis. Basierend auf

[1] Python: https://www.python.org/

diesen sieben Datenvalidierungsmethoden wurden anschließend geeignete Paper gesucht, die die Methoden auf verschiedene Datensätze anwenden und deren Performance messen. Tabelle 5.1 gibt einen Überblick über die verwendeten Paper/Arbeiten.

Der Datensatz wurde erstellt, indem die Interaktionen zwischen den Datensätzen und der Datenvalidierungsmethode sowie deren Leistung notiert wurden. Eine Interaktion bedeutet dabei, dass eine Datenvalidierungsmethode auf einen Datensatz in der Literatur angewendet wurde. Die Performance wird in diesem Datensatz anhand der AUC-ROC Metrik gemessen. Die Wahl der Metrik fiel dabei auf AUC, da die meisten Paper, die die Performance von Datenvalidierungsmethoden angeben, diese Metrik verwenden. Da nicht alle Paper eine Performance angeben oder die Performance nicht in der AUC-Metrik angeben, enthält nicht jede Interaktion von Datansatz und Datenvalidierungsmethode einen AUC-Wert im erstellten Datensatz. Ein weiteres Problem ist, dass verschiedene Paper für die gleiche Datensatz-Methoden Interaktion leicht unterschiedliche AUC-Werte angeben, da die Paper die gleiche Methode mit unterschiedlicher Parameterwahl ausgeführt haben. In einem solchen Fall wurde der Durchschnitt aller AUC-Werte für eine Datensatz-Methoden Interaktion gebildet.

Für die Erstellung der Textbeschreibung der Datensätze und Methoden wurden zwei verschiedene Methoden verwendet. Dabei liefert eine Methode lange Beschreibungen, wohingegen die zweite Methode nur Stichworte liefert. Es wurden sowohl lange als auch kurze Textbeschreibungen erstellt, um vergleichen zu können, welche Beschreibungen bessere Ergebnisse liefern. So kann es z.B. sein, dass die auf Basis der Textbeschreibungen erstellten Embeddings aussagekräftiger sind, wenn die Beschreibung aus weniger Wörtern besteht.

Für die langen Textbeschreibungen wurde manuell nach den Datensätzen mit Hilfe einer Suchmaschine gesucht. Die Beschreibung wurde dann von den Webseiten, welche den Datensatz zur Verfügung stellen, kopiert. Die kurzen Beschreibungen, welche nur aus fünf Stichworten bestehen, wurden mit ChatGPT erstellt. Da in der Literatur meist die gleichen Datensätze zum Vergleich von Datenvalidierungsmethoden verwendet werden und diese Datensätze auch frei im Internet verfügbar sind, sollte ChatGPT in der Lage sein, aussagekräftige Stichwörter für diese Datensätze zu generieren. Die Prompts, für die Erstellung der kurzen Beschreibungen der Datenvalidierungsmethoden sind wie folgt:

> Describe the following data validation methods in five keywords each based on their application: Angle-Based Outlier Detection, Local Outlier Factor, Deep Support Vector Data Description, Clustering Based Local Outlier Factor, k-Nearest-Neighbors, Isolation Forest, Principal Component Analysis
>
> Format the list into a csv file with ; as operator

Für die Erstellung der kurzen Beschreibungen der Datensätze wurden die folgenden Prompts verwendet:

> Describe the following Machine Learning datasets in five keywords each and dont use the word dataset: Annthyroid, Arrhythmia, BreastW, Cardio, ForestCover, Glass, Http (KDDCUP99), Ionosphere, Mammography, Mnist, Musk, Optdigits, Pendigits, Pima, Shuttle, Smtp (KDDCUP99), Speech, Thyroid, Wine, Caltech 101, Zoo, Cardiotocography, SpamBase, Internet Advertisements, ISOLET, Multiple Features, Iris, FEI Face Database, lymphography, satimage-2, Vertebral, Vowels, WBC, Satellite, ALOI, Waveform, WDBC, WPBC, HeartDisease, Hepatitis, PageBlocks, Parkinson,

> Stamps, Wilt, NHL96 dataset, Adult, Skin Segmentation, AirBnB New User Bookings competition, Hepc, TAO, Covertype, Gisette, Diabetes, Letter Recognition, Madelon, Cora, Citeseer, Pubmed, CIFAR-10, GTSRB, splice, cod-rna, KDD intrusion detection, AUSLAN, ECG
>
> Format the list into csv file with ; as separator

In Tabelle 5.2 ist ein Auszug aus dem erstellten Datensatz gegeben, der die lange und kurze Textbeschreibung für zwei Datensätze zeigt.

5.2. Implementierung der KG-to-Text Komponente

Damit nicht für jedes Kontextmodell manuell eine Textbeschreibung mit ChatGPT erstellt werden muss, wird im Folgenden beschrieben, wie das Kontextmodell automatisch in eine Textbeschreibung umgewandelt werden kann. Dazu wird die API von OpenAI[2] verwendet. Diese ermöglicht es, die Modelle von OpenAI in Python oder JavaScript zu verwenden und somit auch automatisch Nachrichten an ein ChatGPT-Modell zu senden. Als Modell kann zwischen GPT-3.5 und GPT-4 gewählt werden, wobei für die Implementierung des KG-to-Text Modells in dieser Arbeit GPT-3.5-turbo verwendet wurde. Der Grund dafür ist, dass kein signifikanter Unterschied zwischen den Ausgaben von GPT-4 und GPT-3.5 festgestellt werden konnte. Außerdem sind die Kosten für das Modell GPT-3.5-turbo um ein Vielfaches niedriger als für GPT-4. Darüber hinaus wurde GPT-3.5 auch in Abschnitt 4.3 verwendet, um die Fähigkeit von ChatGPT als KG-to-Text-Modell zu testen.

Um eine API-Anfrage an ChatGPT zu stellen, muss das zu verwendende Modell und eine Konversation übergeben werden. Da die Ausgaben des GPT-Modells standardmäßig nicht deterministisch sind, wird in der prototypischen Implementierung zusätzlich ein Seed in der API-Anfrage übergeben. Der Seed soll helfen, die Ausgaben des Modells so deterministisch wie möglich zu machen. Der Seed kann aber leider nicht garantieren, dass das Modell immer die gleichen Antworten ausgibt, da sich durch Änderungen am Modell durch OpenAI, die Ausgabe bei gleichem Seed ändern kann. Die übergebene Konversation besteht aus verschiedenen Nachrichten des Benutzers und Ausgaben des Modells. Für jede Nachricht muss sowohl eine Rolle als auch der Inhalt der Nachricht angegeben werden. Als Rolle können die Rollen „system“, „user“ und „assistant“ angegeben werden. Die Rolle „system“ wird verwendet, um das Verhalten des Modells zu verändern. Beispielsweise kann dem Modell eine Persona zugewiesen werden. Bei der Implementierung des KG-to-Text-Modells wird diese Rolle jedoch nicht verwendet. Die „user“ Rolle kennzeichnet die Nachrichten des Benutzers, während die „assistant“ Rolle die Antworten des GPT-Modells kennzeichnet. Bei Anfragen sollten die Antworten des Modells ebenfalls mit der „assistant“ Rolle angegeben werden, da das Modell sonst nicht weiß, was die vorherige Konversation war. Ein Beispiel für die Verwendung der API ist in Listing 5.1 gegeben.

Um die Textbeschreibungen für das Kontextmodell zu erstellen, werden zwei API-Anfragen verwendet. In der ersten Anfrage wird die lange Beschreibung des Kontextmodells erstellt. Dazu wird zunächst eine Konversation erstellt, die als erste Nachricht den Prompt aus Abschnitt 4.3 zur Erstellung der Beschreibungen enthält. Die zweite Nachricht ist einfach eine Antwort, die das

[2] OpenAI API: https://platform.openai.com/docs/api-reference/

Paper	Datenvalidierungsmethoden
An Efficient Density-Based Local Outlier Detection Approach for Scattered Data [SXR+18]	Local Outlier Factor
An Improved Data Anomaly Detection Method Based on Isolation Forest [XWMZ17]	Local Outlier Factor, Isolation Forest
An Outlier Detection Approach Based on Improved Self-Organizing Feature Map Clustering Algorithm [YWW+19]	Local Outlier Factor
Angle-Based Outlier Detection Algorithm with More Stable Relationships [LLC15]	Angle-Based Outlier Detection
Angle-based outlier detection in high-dimensional data [KSZ08]	Angle-Based Outlier Detection, Local Outlier Factor
Anomaly Detection via Online Oversampling Principal Component Analysis [LYW12]	Principal Component Analysis
Cluster-Based Improved Isolation Forest [SDYC22]	Local Outlier Factor, k-Nearest-Neighbors, Isolation Forest
Clustered Hierarchical Anomaly and Outlier Detection Algorithms [IHD21]	Angle-Based Outlier Detection, Local Outlier Factor, k-Nearest-Neighbors, Clustering Based Local Outlier Factor, Isolation Forest
Deep Dual Support Vector Data description for anomaly detection on attributed networks [ZFW+22]	Deep Support Vector Data Description
Deep One-Class Classification [RVG+18]	Deep Support Vector Data Description
Isolation Forest [LTZ08]	Local Outlier Factor, Isolation Forest
L1-Depth Revisited: A Robust Angle-Based Outlier Factor in High-Dimensional Space [Pha18]	Angle-Based Outlier Detection, Local Outlier Factor, k-Nearest-Neighbors
LOF: identifying density-based local outliers [BKNS00]	Local Outlier Factor
Nearest-Neighbor and Clustering based Anomaly Detection Algorithms for RapidMiner [AG12]	Clustering Based Local Outlier Factor
On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study [CZS+16]	Angle-Based Outlier Detection, Local Outlier Factor, k-Nearest-Neighbors
Robust factored principal component analysis for matrix-valued outlier accommodation and detection [MZW+23]	Principal Component Analysis
SDCOR: Scalable density-based clustering for local outlier detection in massive-scale datasets [NHF21]	Local Outlier Factor
Statistical Analysis of Nearest Neighbor Methods for Anomaly Detection [GAR19]	Local Outlier Factor, k-Nearest-Neighbors, Isolation Forest

Tabelle 5.1.: Liste der Paper/Arbeiten, die zur Erstellung des Datensatzes verwendet wurden. In der rechten Spalte sind die Datenvalidierungsmethoden angegeben, welche im jeweiligen Paper verwendet wurden.

Datensatz	Beschreibung
Parkinson	The data set consists of medical data distinguishing healthy people from those suffering from Parkinson's disease. The latter were labeled as outliers.
NHL96 dataset	Our base dataset is an 855-record dataset consisting of 1995-96 NHL player performance statistics. These statistics include numbers of goals assists points penalty minutes shots on goal games played power play goals etc.
Parkinson	Disease, Classification, Speech, Attributes, Diagnosis
NHL96 dataset	Hockey, Player, Statistics, Attributes, Multiclass

Tabelle 5.2.: Lange und kurze Textbeschreibung von zwei Datensätzen

```python
from openai import OpenAI
client = OpenAI(api_key='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')

completion = client.chat.completions.create(
    model='gpt-3.5-turbo',
    messages=[
        {"role": "user", "content": "In welchem Land liegt Berlin?"},
        {"role": "assistant", "content": "Berlin liegt in Deutschland."},
        {"role": "user", "content": "In welchem Land liegt Paris?"},
    ],
    seed=2023
)

print(completion.choices[0].message.content)
```

Listing 5.1: Beispiel einer API Anfrage an das GPT-3.5-turbo Modell. Der Inhalt der „assistant“ Nachricht ist frei gewählt. Die Ausgabe des Beispielprogrammes ist: „Paris liegt in Frankreich.“

GPT-Modell ausgeben würde, wenn man ihm nur die erste Nachricht senden würde. Die letzte Nachricht ist dann die RDF-Beschreibung des Kontextmodells. Diese wird jedoch zunächst leicht modifiziert. Dazu werden, wie bereits in Abschnitt 4.3 beschrieben, zunächst die Tripel entfernt, die die Relationen „iot-context:hasValue“ oder „iot-context:hasTimeStamp“ enthalten.

Die zweite Abfrage wird dann verwendet, um die Stichwörter zu erzeugen, die das Kontextmodell beschreiben. Dazu wird die Konversation aus der ersten Anfrage erweitert. Dabei wird zunächst die Antwort des GPT-Modells, die aus der textuellen Beschreibung des Kontextmodells besteht, an die Konversation angehängt. Anschließend wird der Prompt zur Erstellung der Stichwörter aus Abschnitt 4.3 an die Konversation angehängt. Die beiden Antworten des GPT-Modells sind dann die lange Textbeschreibung und die Stichwörter, die das Kontextmodell beschreiben. Der detaillierte Aufbau der KG-to-Text ist in Abbildung 5.1 dargestellt.

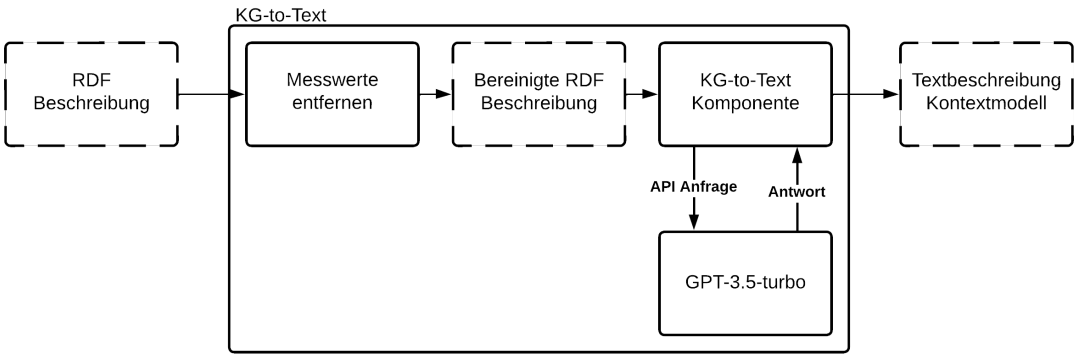


Abbildung 5.1.: Detaillierter Aufbau der KG-to-Text Komponente

5.3. Preprocessing der Textbeschreibungen

Da die Textbeschreibungen der Methoden und Datensätze in natürlicher Sprache formuliert sind, sollten diese zunächst bereinigt werden. Im Folgenden wird beschrieben, wie die Beschreibungen bereinigt werden, um die Performance der Recommender Systeme zu erhöhen.

Für das Preprocessing wurde sich an [Dee23] und [Mon18] orientiert. Dabei werden zunächst alle Satzzeichen, Ziffern und unnötige Leerzeichen entfernt, da diese entweder von den Feature Selection Methoden nicht verwendet werden können oder keine zusätzlichen Informationen liefern. Nach dem Entfernen dieser Zeichen werden alle Wörter mittels Lemmatisierung umgewandelt. Lemmatisierung beschreibt den Prozess der Umwandlung von Wörtern in ihre ursprüngliche Form. Zum Beispiel würde das Wort „umgewandelt“ in seine ursprüngliche Form „umwandeln“ geändert. Bei der Lemmatisierung wird die Stammform eines Wortes mit Hilfe einer morphologischen Analyse und eines Wörterbuchs erstellt [D2122]. Für die Lemmatisierung wird das Paket nltk[3] verwendet, das Werkzeuge für die Arbeit mit natürlicher Sprache bereitstellt. Insbesondere wird die Klasse „WordNetLemmatizer“ aus „nltk.stem“ verwendet. Zudem werden mit Hilfe des nltk-Pakets die Stoppwörter in den Beschreibungen entfernt. Dazu wird „stopwords“ aus „nltk.corpus“ verwendet. Stoppwörter sind häufig vorkommende Wörter wie „a“, „the“ oder „is“. Diese Wörter werden aus den Textbeschreibungen entfernt, da sie wenig zusätzliche Information zur Beschreibung beitragen und z.B. das Embedding eines Satzes negativ beeinflussen könnten. Außerdem werden alle Wörter entfernt, die kürzer als drei Zeichen sind, da auch diese kurzen Wörter in der Regel nicht viele Informationen enthalten.

Ein Beispiel, wie die Beschreibungen vor und nach der Vorverarbeitung aussehen, ist in Tabelle 5.3 dargestellt. Zuerst wird die unbearbeitete Textbeschreibung gezeigt, gefolgt von der bearbeiteten Textbeschreibung.

[3] NLTK: https://www.nltk.org/

Datensatz	Beschreibung
zoo	a simple database containing 17 boolean-valued attributes. the “type” attribute appears to be the class attribute. here is a breakdown of which animals are in which type: (i find it unusual that there are 2 instances of “frog” and one of “girl”!)
zoo	simple database containing boolean-valued attribute type attribute appears class attribute breakdown animal type find unusual instance frog one girl
cardiotocography	data set related to heart diseases. it describes 3 classes: normal suspect or pathological. normal patients are treated as inliers and the remaining as outliers.
cardiotocography	data set related heart disease describes class normal suspect pathological normal patient treated inliers remaining outlier
spambase	the “spam” concept is diverse: advertisements for products/web sites make money fast schemes chain letters pornography... our collection of spam e-mails came from our postmaster and individuals who had filed spam. our collection of non-spam e-mails came from filed work and personal e-mails and hence the word ’george’ and the area code ’650’ are indicators of non-spam. these are useful when constructing a personalized spam filter. one would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter.
spambase	spam concept diverse advertisement products/web site make money fast scheme chain letter pornography ... collection spam e-mail came postmaster individual filed spam collection non-spam e-mail came filed work personal e-mail hence word ’george area code indicator non-spam useful constructing personalized spam filter one would either blind non-spam indicator get wide collection non-spam generate general purpose spam filter

Tabelle 5.3.: Auszug aus den Textbeschreibungen der Datensätze vor und nach dem Preprocessing. Dabei ist immer zuerst die Beschreibung vor dem Preprocessing, gefolgt von der Beschreibung nach dem Preprocessing gegeben.

5.4. Umwandlung der Textbeschreibungen in Vektoren

Im Folgenden wird beschrieben, wie die für diese Arbeit ausgewählten und in Abschnitt 4.4.1 beschriebenen Methoden zur Feature Selection implementiert wurden.

5.4.1. TF-IDF

Um die TF-IDF Vektoren der Beschreibungen zu erzeugen, wird das Paket sklearn[4] verwendet, das für verschiedene Machine Learning Anwendungen verwendet werden kann. Dazu wird der „TfidfVectorizer“ von „sklearn.feature_extraction.text “ verwendet. Um den TfidfVectorizer anzu-

[4]scikit-learn: https://scikit-learn.org/

passen, wird zuerst ein Korpus mit allen Textbeschreibungen erstellt und dann der TfidfVectorizer auf dieses Korpus angepasst. Anschließend werden alle Textbeschreibungen in einen TF-IDF-Vektor umgewandelt. Dies geschieht sowohl mit den normalen Textbeschreibungen als auch mit den Stichwörtern.

5.4.2. Word2Vec

Für die Erstellung der Features mit Hilfe des Word2Vec-Modells wird ein vortrainiertes Modell verwendet, das von den Autoren zur Verfügung gestellt wird. Das vortrainierte Modell[5] stammt aus [MSC+13]. Das Modell wurde auf einem Datensatz bestehend aus Google News trainiert und umfasst die Embeddings für 3 Millionen Wörter. Die Dimension der Embeddings ist 300.

Zum Laden des Modells wird das Paket gensim[6] verwendet, welches das Laden und Erstellen verschiedener NLP-Modelle ermöglicht. Da das vortrainierte Modell immer nur für einzelne Wörter ein Embedding zur Verfügung stellt, können die kompletten Beschreibungen nicht direkt in ein Embedding umgewandelt werden. Daher wird zunächst für jedes Wort, das in der Beschreibung vorkommt, ein Embedding erzeugt und anschließend über alle Embeddings der Durchschnitt gebildet. Existiert für ein Wort kein Embedding im vortrainierten Modell, wird dieses Wort einfach übersprungen.

5.4.3. GloVe

Für die Erstellung der Vektoren mithilfe des GloVe Modells [PSM14] wird ein vortrainiertes Modell verwendet, das von den Autoren zur Verfügung gestellt wird. Die Autoren stellen verschiedene vortrainierte Modelle zur Verfügung. Dabei wurden die Modelle auf unterschiedlichen Datensätzen trainiert und umfassen daher jeweils eine unterschiedliche Anzahl an Vokabular. Des Weiteren stehen die Modelle mit unterschiedlichen Dimensionen der Embeddings zur Verfügung. Für die Implementierung in dieser Arbeit wurde das Modell mit 2,2 Millionen Vokabular und 300 Dimensionen ausgewählt[7]. Das vortrainierte Modell liegt als Textdatei vor, wobei in jeder Zeile ein Wort mit seinem zugehörigen Embedding steht. Dies ermöglicht die Erstellung eines Dictionary, das für jedes Wort das entsprechende Embedding enthält.

Da auch dieses Modell nur Embeddings für einzelne Wörter liefert, wird auch hier, wie beim Word2Vec Modell, das Embedding für einen Satz aus dem Durchschnitt der Embeddings der einzelnen Wörter gebildet. Auch bei diesem Modell werden Wörter, für die im vortrainierten Modell kein Embedding vorhanden ist, einfach übersprungen.

[5] Word2Vec Embeddings: https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing
[6] Gensim: https://radimrehurek.com/gensim/
[7] GloVe Embeddings: https://nlp.stanford.edu/data/glove.840B.300d.zip

5.4.4. fastText

Auch beim fastText Modell wird, wie bei den beiden anderen Embedding basierten Methoden, ein vortrainiertes Modell verwendet. Dazu wird ein vortrainiertes Modell aus [MGB+18] verwendet. Genauer gesagt wird dafür das Modell mit einem Vokabular von 2 Millionen Wörtern und Subwort Informationen verwendet[8]. Es wurde explizit die Variante mit subword Informationen verwendet, da so auch Embeddings für Wörter erstellt werden können, die nicht im Vokabular vorkommen. Das Modell liegt hier nicht wie bei den beiden anderen Methoden in einem Textformat vor, weshalb das Modell nicht einfach in einer Art Wörterbuch gespeichert werden kann. Stattdessen wird das fasttext-Paket[9] für Python verwendet, um das Modell zu laden. Auch hier wird einfach der Mittelwert über alle Wörter der Beschreibung gebildet. Es müssen jedoch keine Wörter ausgelassen werden, da auch für Wörter, die nicht im Vokabular enthalten sind, ein Embedding erstellt werden kann.

5.5. Vorschlagen geeigneter Datenvalidierungsmethoden

Um die Vektoren der Feature Selection Methoden in eine Empfehlung umzuwandeln, werden Computational Approaches verwendet. Im Folgenden wird beschrieben, wie diese implementiert wurden.

5.5.1. Euklidischer Abstand

Um den euklidischen Abstand zwischen allen Beschreibungen zu berechnen, wird das Paket sklearn[10] verwendet. Genauer gesagt wird die Methode „euclidean_distances“ aus „sklearn.metrics.pairwise“ verwendet. Die Methode verwendet als Eingabe zwei Matrizen, die die Vektoren der Beschreibungen darstellen. Die erste Matrix repräsentiert die Nutzer, d.h. die Kontextmodelle/Datensätze und die zweite die Produkte, d.h. die Datenvalidierungsmethoden. Die Matrizen enthalten in der Zeile i den Vektor der i-ten Beschreibung. Die Methode berechnet dann die Distanzen zwischen jedem Nutzer und allen Datenvalidierungsmethoden. Anschließend werden die berechneten Distanzen aufsteigend sortiert, so dass die besten Ergebnisse an erster Stelle stehen. Auf dieser Grundlage können dann die besten Datenvalidierungsmethoden für einen Nutzer empfohlen werden. Ein Beispiel, wie die Abstände für einen Nutzer aussehen, ist in Tabelle 5.4 zu finden.

5.5.2. Kosinus-Ähnlichkeit

Die Kosinus-Ähnlichkeit wird ebenfalls mit Hilfe des Pakets sklearn implementiert. Dabei wird die Methode „cosine_similarity“ aus „sklearn.metrics.pairwise“ verwendet. Wie bei der euklidischen Distanz werden die Vektoren der Beschreibungen zunächst in eine Matrix umgewandelt und dann an die Funktion zur Berechnung der Kosinus-Ähnlichkeit übergeben. Das Verfahren berechnet dann die Kosinus-Ähnlichkeit zwischen jedem Nutzer und allen Produkten. Die Ergebnisse werden

[8] fastText Model: https://dl.fbaipublicfiles.com/fasttext/vectors-english/crawl-300d-2M-subword.zip
[9] fastText Paket: https://fasttext.cc/docs/en/python-module.html
[10] scikit-learn: https://scikit-learn.org/stable/

Ranking	Datenvalidierungsmethode	Euklidischer Abstand
1	isolation forest	1.379048
2	local outlier factor	1.387016
3	angle-based outlier detection	1.390550
4	clustering based local outlier factor	1.392396
5	deep support vector data description	1.395077
6	k-nearest-neighbors	1.402434
7	principal component analysis	1.405611

Tabelle 5.4.: Euklidischer Abstand zwischen dem Vektor des Datensatzes „Annthyroid“ und den Vektoren der Datenvalidierungsmethoden. Die Vektoren wurden mit TF-IDF erzeugt.

Ranking	Datenvalidierungsmethode	Kosinus-Ähnlichkeit
1	local outlier factor	0.688228
2	deep support vector data description	0.657307
3	isolation forest	0.642664
4	clustering based local outlier factor	0.634152
5	k-nearest-neighbors	0.622239
6	principal component analysis	0.619954
7	angle-based outlier detection	0.592452

Tabelle 5.5.: Kosinus-Ähnlichkeit zwischen dem Vektor des Datensatzes „Annthyroid“ und den Vektoren der Datenvalidierungsmethoden. Die Vektoren wurden mit Hilfe der fastText Embeddings erstellt.

wiederum für jeden Nutzer sortiert, diesmal jedoch in absteigender Reihenfolge, so dass der größte Wert an erster Stelle steht. Die beste Datenvalidierungsmethode wird dann basierend auf dem kleinsten Winkel zwischen dem Vektor des Benutzers, d.h. Datensatz/Kontextmodell, und dem Vektor des Produkts, d.h. Datenvalidierungsmethode, empfohlen. In Tabelle 5.5 wird ein Beispiel gegeben, wie Kosinus-Ähnlichkeiten für einen Nutzer aussehen können.

5.5.3. Text-CF

Für die Implementierung der TextCF Komponente wurde die Implementierung der DEKR Methode[11] als Grundlage verwendet. Da die Implementierung der DEKR-Methode auch eine Komponente für Empfehlungen auf Basis von Knowledge Graphen enthält, die für diese Arbeit nicht verwendet werden kann (siehe Abschnitt 4.2), wurde diese Komponente zuerst entfernt. Dafür wurde der Teil zum Laden des Knowledge Graphen, die Aggregation über den Graphen und die anschließenden Vorhersage basierend auf dem Knowledge Graphen entfernt. Zudem wurde das Laden der notwendigen Daten angepasst. Für das Laden der Textbeschreibungen werden diese zuerst, wie in Abschnitt 5.3 beschrieben, bereiningt. Nach dem bereinigen werden dann mithilfe der Feature Selection Methoden die Vektoren der Beschreibungen erstellt. Diese Vektoren werden dann in einem Dictonary mithilfe

[11] DEKR: https://github.com/cxsss/DEKR

der save Funktion von numpy gespeichert. Dies wird gemacht, damit nicht für jede Ausführung des Modells zuerst ein Word Embedding Model geladen werden muss. Wenn das Modell dann ausgeführt wird, werden die Beschreibungen aus dem Dictonary für das entsprechende Word Embedding Modell geladen. Neben dem Entfernen der nicht mehr benötigten Funktionen und Anpassung des Data Loaders wurde zusätzlich die Evaluierungsfunktionen angepasst. Die war nötig, da die verschiedenen Evaluierungsmetriken auf den neuen Daten nicht richtig berechnet wurden.

Im Folgenden wird der Aufbau der TextCF Komponente beschrieben. Da die Methode auf einem neuronalen Netz basiert, das trainiert werden muss, werden zunächst alle Beschreibungen sowie eine Rating-Datei geladen. Die Rating-Datei enthält die Daten, anhand derer das neuronale Netz optimiert wird. Dazu enthält die Datei alle Datensatz-Datenvalidierungsmethode Interaktionen, bei denen die AUC Metrik über einem bestimmten Schwellenwert liegt. Der Schwellenwert wurde für diese Arbeit mit $0,65$ gewählt, da bei einer höheren Wahl zu wenige Daten übrig bleiben. Zusätzlich wird jedem Datensatz und jeder Datenvalidierungsmethode eine ID zugewiesen, damit diese durch eine Nummer identifiziert werden können und somit auch in einem Tensor verwendet werden können. Ein Tensor ist dabei eine Mehrdimensionale Matrix, welche nur Elemente des gleichen Typs beinhaltet[12].

Das neuronale Netz besteht aus 3 verschiedenen Layern, die jeweils mit dem Paket torch[13] erstellt werden. Der erste Layer ist ein Layer zur Reduktion der Wort Embeddings der Beschreibungen, da die Wort Embeddings 300 Dimensionen haben. Die GMF-Layer des Neural Collaborative Filtering Framework, die zum Lernen der linearen Beziehungen der Beschreibungen verwendet wird, wird durch torch.mul() ersetzt. Die reduzierten Embeddings werden dann sowohl an torch.mul() als auch an die MLP-Layer übergeben. Der MLP-Layer soll die nichtlinearen Zusammenhänge lernen. Vor der Übergabe an den MLP-Layer werden die reduzierten Embeddings zunächst konkatiniert. Der MLP-Layer besteht wiederum aus drei Layern, wobei der Output des nächsten Layers immer halb so groß ist wie der Output des vorherigen Layers. Die Ergebnisse von torch.mul() und der letzten MLP-Layer werden konkatiniert und an den NeuMF-Layer weitergegeben, der dann einen einzelnen Wert ausgibt. Der Ausgabewert kann dann als die Wahrscheinlichkeit interpretiert werden, mit der der Benutzer und das Produkt, das als Eingabe diente, zusammenpassen. Das neuronale Netz wird mit Hilfe des Adam Optimizers optimiert und als Loss Funktion wird die Binary Cross Entropy[14] verwendet. In Abbildung 5.2 ist der Aufbau des neuronalen Netzes dargestellt. Die genaue Initialisierung der einzelnen Ebenen ist in Listing 5.2 gegeben.

[12]Tensors: https://pytorch.org/docs/stable/tensors.html
[13]PyTorch: https://pytorch.org/
[14]Binary Cross Entropy: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

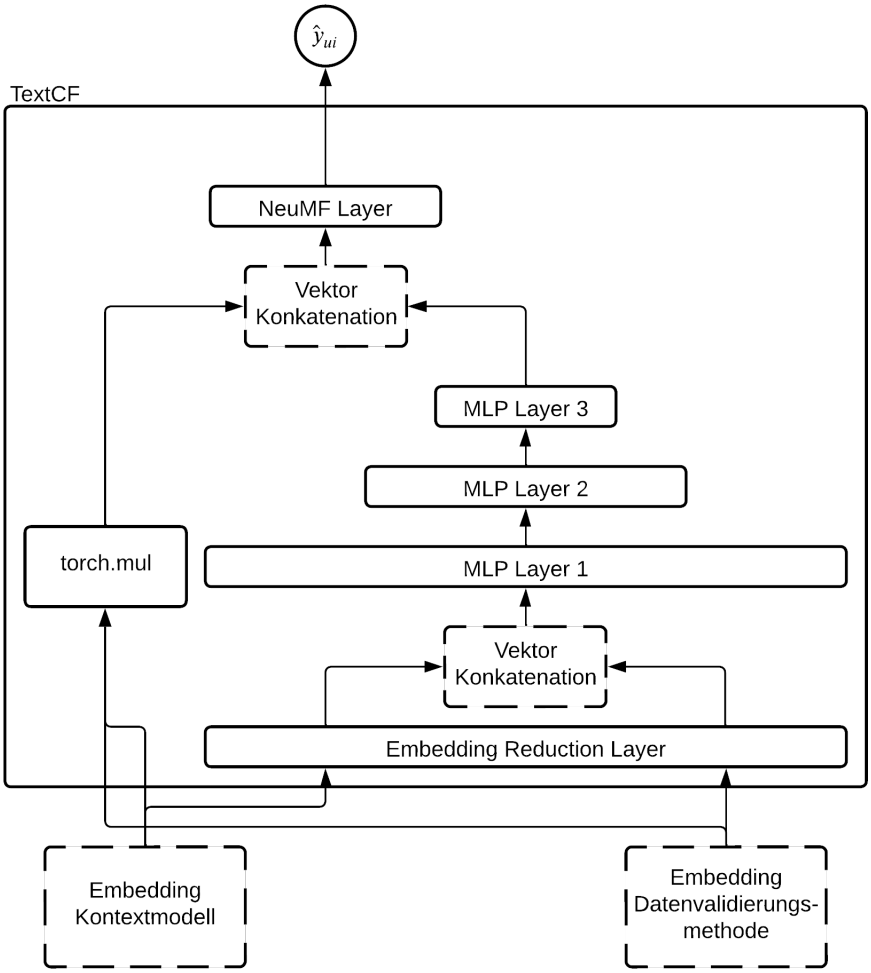


Abbildung 5.2.: Aufbau des neuronalen Netzes der TextCF Komponente.

```
import torch.nn as nn

desc_reduce = nn.Sequential(
    nn.Linear(desc_embed_size, embed_size_add),
    nn.ReLU()
)

MLP_layer123 = nn.Sequential(
    nn.Linear(embed_size_add * 2, embed_size_add * 2),
    nn.ReLU(),
    nn.Dropout(p=dropout),

    nn.Linear(embed_size_add * 2, embed_size_add),
    nn.ReLU(),
    nn.Dropout(p=dropout),

    nn.Linear(embed_size_add, embed_size_add // 2),
    nn.ReLU(),
    nn.Dropout(p=dropout)
)

NeuMF = nn.Sequential(
    nn.Linear((embed_size_add + embed_size_add // 2), 1),
    nn.Sigmoid()
)
```

Listing 5.2: Initialisierung der verschiedenen Ebenen in der TextCF Komponente. desc_embed_size gibt die Dimensionen der Embedding Vektoren an. embed_size_add ist ein Parameter, der beim Aufruf der Komponente angepasst werden kann.

6. Evaluierung

In diesem Kapitel werden die erstellten Recommender Systeme evaluiert. Dazu werden zunächst in Abschnitt 6.1 die verwendeten Metriken vorgestellt. Anschließend werden in Abschnitt 6.2 die Baselines vorgestellt, gegen die die Recommender Systeme getestet werden. In Abschnitt 6.3 werden die Ergebnisse der Recommender Systeme auf dem Datensatz aus Abschnitt 5.1 vorgestellt. Zuletzt werden in Abschnitt 6.4 die Recommender Systeme an den Kontextmodellen getestet.

6.1. Metriken

Die für die Evaluation gewählten Metriken orientieren sich an [CSY+21] und [KNMN21]. Für die Evaluation können Recommender Systeme in drei Typen eingeteilt werden: Als Regressionsproblem, als Klassifikationsproblem oder als Rankingproblem. Bei einem Regressionsproblem wird versucht, die Bewertungen der Produkte vorherzusagen, während bei einem Klassifikationsproblem nur versucht wird vorherzusagen, ob ein Produkt relevant ist. Bei einem Ranking-Problem wird versucht, die Produkte nach ihrer Relevanz zu sortieren [KNMN21]. Da für ein Kontextmodell verschiedene Datenvalidierungsmethoden funktionieren können, diese aber unterschiedlich gut Fehler erkennen, handelt es sich um ein Rankingproblem. Für die Bewertung wurden daher die folgenden Metriken verwendet: Precision@k, Recall@k und NDCG@k. Diese werden im Folgenden vorgestellt.

6.1.1. Precision@k

Die Metrik Precision@k misst die Precision der ersten k Produkte, die vom Recommender System vorgeschlagen werden. Dafür misst die Precision@k den Anteil relevanter Produkte in den ersten k Empfehlungen. Die Precision@k wird berechnet, indem die Anzahl der relevanten Produkte in den ersten k Vorschlägen durch k dividiert wird [KNMN21]:

$$P@k = \frac{|\{\text{Relevante empfohlene Produkte @k}\}|}{k}$$

Ein Beispiel für die Berechnung der Precsiion@k für vorgeschlagene Produkte ist in Abbildung 6.1 gegeben.

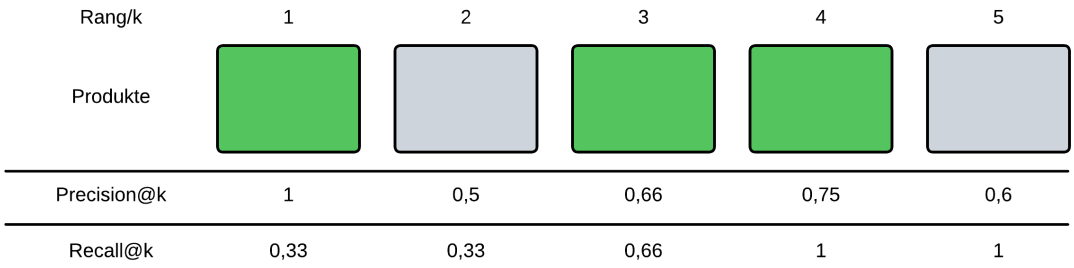


Abbildung 6.1.: Beispiel für die Berechnung von Precision@k und Recall@k. Die grünen Felder stehen für relevante Produkte, die grauen für nicht relevante Produkte.

6.1.2. Recall@k

Die Metrik Recall@k gibt den Recall in den ersten k empfohlenen Produkten an. Der Recall beschreibt dabei den Anteil der relevanten Produkte an allen relevanten Produkten, die das System in den obersten k Empfehlungen finden konnte. Dazu werden die relevanten Produkte in den obersten k Empfehlungen durch alle relevanten Produkte dividiert [KNMN21]:

$$R@k = \frac{|\{\text{Relevante empfohlene Produkte @k}\}|}{|\{\text{Relevante Produkte}\}|}$$

In Abbildung 6.1 ist ein Beispiel für die Berechnung der Recall@k Metrik gegeben.

6.1.3. NDCG@k

Da sowohl Precision als auch Recall die Platzierung der Produkte nicht berücksichtigen, also z.B. ob die relevanten Produkt auch auf dem ersten Platz empfohlen werden, wird zusätzlich die Metrik NDCG@k (Normalized discounted cumulative gain) verwendet. Diese Metrik berücksichtigt ebenfalls, auf welchem Platz die relevanten Produkte empfohlen werden. Dabei ist der NDCG@k hoch, wenn die relevanten Produkte auch auf den ersten Plätzen empfohlen werden. Werden dagegen zuerst irrelevante Produkte und erst danach relevante Produkte empfohlen, ist der NDCG@k klein. Dagegen kann z.B. der Recall@k bei einem hohen k sehr hoch sein, obwohl die relevanten Produkte erst spät empfohlen werden. Zur Berechnung des NDCG@k wird der DCG@k durch den IDCG@k dividiert. Der DCG@k gibt den Discounted Cumulative Gain für die ersten k empfohlenen Produkte an, während der IDCG@k den Discounted Cumulative Gain für die ersten k Produkte in einer idealen Reihenfolge angibt. Der NDCG@k wird wie folgt berechnet [Dhi23]:

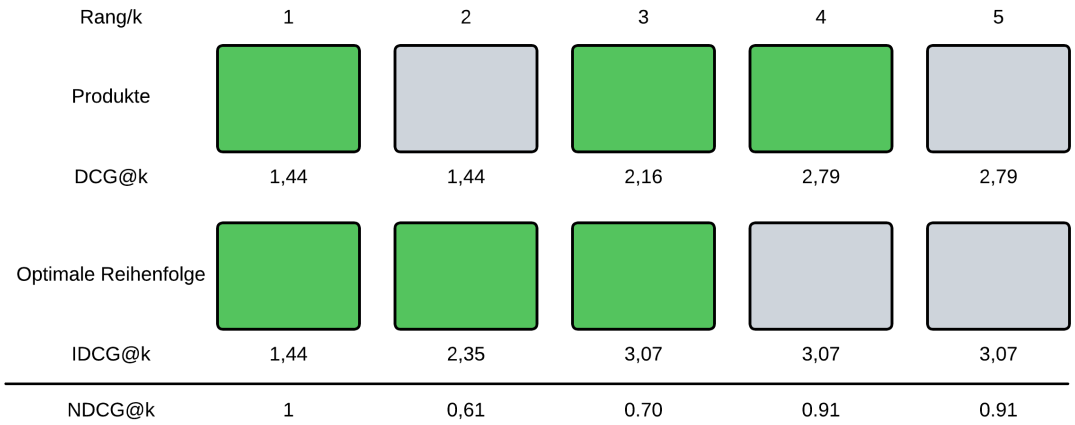


Abbildung 6.2.: Beispiel für die Berechnung der NDCG@k Metrik. Die grünen Felder stehen für relevante Produkte, die grauen für nicht relevante Produkte. Zuerst werden DCG@k und IDCG@k berechnet.

$$\text{NDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}}$$

$$\text{DCG@k} = \sum_{i=1}^{k \text{ actual order}} \frac{rel_i}{\log_2(i+1)}$$

$$\text{IDCG@k} = \sum_{i=1}^{k \text{ ideal order}} \frac{rel_i}{\log_2(i+1)}$$

Ein Beispiel für die Berechnung von DCG@k, IDCG@k und NDCG@k findet sich in Abbildung 6.2.

6.2. Baseline

Um die Performance der in dieser Arbeit verwendeten Recommender Systeme messen zu können, wird eine Baseline benötigt, die einen Vergleich ermöglicht. Da es meines Wissens keinen öffentlichen Datensatz oder andere Recommender Systeme gibt, die für die Zielsetzung der Arbeit als Vergleich dienen könnten, wurden zwei Baselines erstellt, die das Verhalten eines Benutzers des Systems modellieren sollen. Für die erste Baseline wird davon ausgegangen, dass der Benutzer unerfahren im Umgang mit Datenvalidierungsmethoden ist und daher einfach eine zufällige Reihenfolge für das Ranking der zur Auswahl stehenden Datenvalidierungsmethoden wählt. Bei der zweiten Baseline wird davon ausgegangen, dass der Nutzer die Rangfolge nach der Popularität der verschiedenen Datenvalidierungsmethoden erstellt. Das bedeutet, dass die Datenvalidierungsmethode, die im gesamten Datensatz am häufigsten verwendet wird, an erster

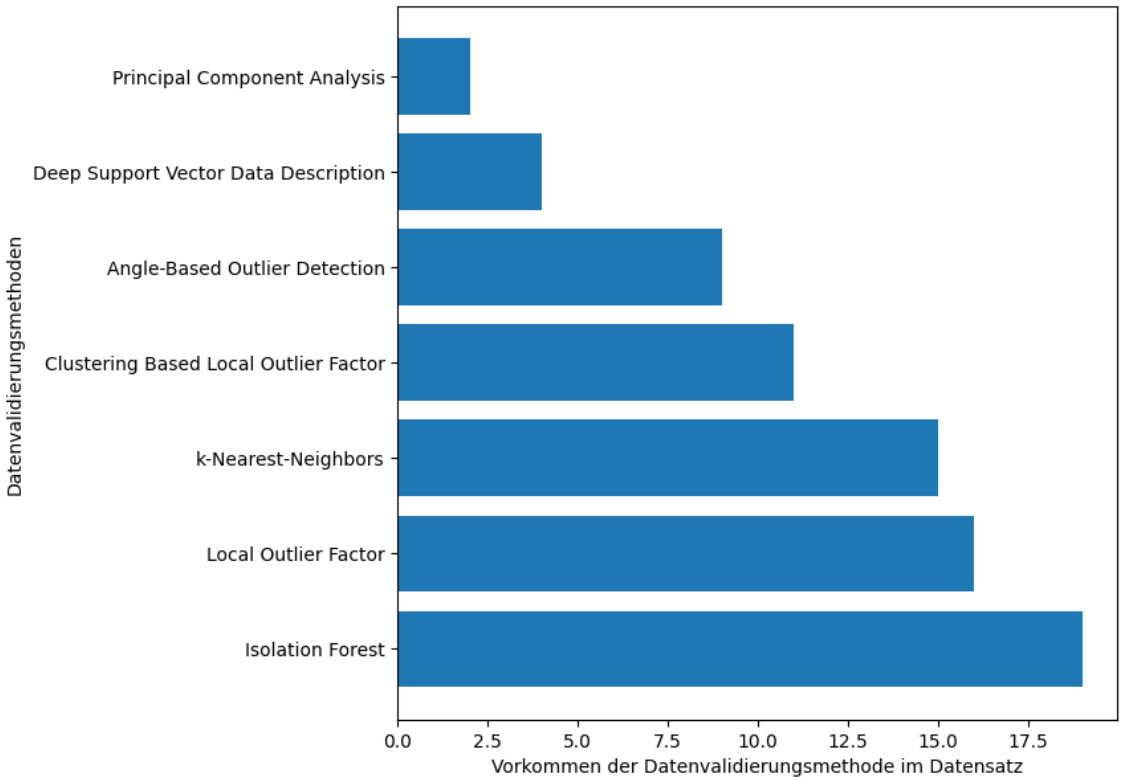


Abbildung 6.3.: Anzahl der Vorkommen jeder Datenvalidierungsmethode in dem in Abschnitt 5.1 erstellten Datensatz.

Stelle steht und die Methode, die im gesamten Datensatz am seltensten verwendet wird, an letzter Stelle steht. Die Baseline nach Popularität wurde zusätzlich gewählt, da einige Methoden im Datensatz deutlich häufiger vorkommen als andere, wie in Abbildung 6.3 dargestellt ist.

6.3. Evaluierung anhand des erstellten Datensatzes

Im Folgenden wird die Performance der verschiedenen Computational Approaches mit den verschiedenen Feature Selection Methoden aus Abschnitt 4.4 anhand des erstellten Datensatzes aus Abschnitt 5.1 vorgestellt. Dabei werden die Recommender Systeme hauptsächlich an diesem Datensatz evaluiert, da zum Zeitpunkt dieser Arbeit nur sehr wenige Kontextmodelle zur Verfügung standen und für diese keine Daten mit geeigneten Datenvalidierungsmethoden vorlagen bzw. die Fehler teilweise nur künstlich hinzugefügt wurden. Im Folgenden werden für jeden Computational Approach nur die besten Feature Selection Methoden gezeigt, um die Abbildungen übersichtlicher zu gestalten. Die Plots mit allen Feature Selection Methoden sind in Anhang B zu finden. Die Metriken wurden jeweils für $k \in \{1, 2, 3, 4, 5, 6, 7\}$ berechnet. Dies ist darauf zurückzuführen, dass insgesamt sieben verschiedene Datenvalidierungsmethoden zur Auswahl standen. Eine Datenvalidierungsmethode wurde als relevant für den Datensatz angesehen, wenn ihr AUC-Score für diesen Datensatz größer als $0,65$ ist. Dieser Wert wurde gewählt, da bei einem höheren Grenzwert zu wenige relevante Datensatz-Datenvalidierungsmethode-Paare übrig blieben.

6.3.1. Euklidischer Abstand

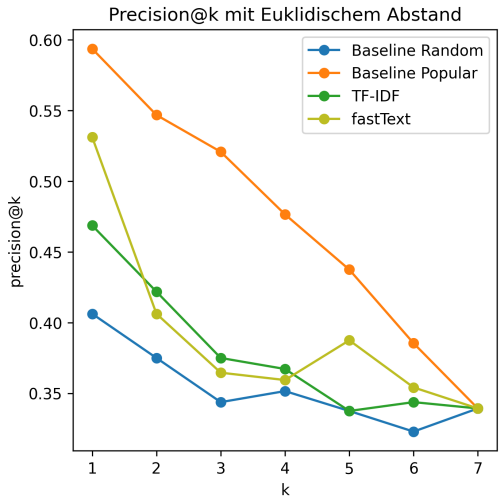
Für die euklidische Distanz liefern TF-IDF und die Embeddings des fastText-Modells die besten Ergebnisse. Für die Evaluierung wurde der gesamte Datensatz verwendet, da bei dieser Methode kein Modell trainiert werden muss. In Abbildung 6.4a ist die Precision@k mit TF-IDF und den fastText Embeddings sowie den beiden Baselines dargestellt. Der Recall@k ist in Abbildung 6.4b und der NDCG@k in Abbildung 6.4c dargestellt. Im Allgemeinen haben sich die normalen Textbeschreibungen für die euklidische Distanz als besser erwiesen als nur die Stichwörter. TF-IDF und fastText Embeddigs sind besser als die Baseline, bei der der Benutzer die Datenvalidierungsmethoden zufällig auswählt, aber schlechter als die Baseline mit den populärsten Methoden. Für $k \in \{1, 5\}$ sind die fastText Embeddings besser als TF-IDF für Precision@k und Recall@k. Für NDCG@k sind die fastText Embeddings nur für $k = 5$ besser als TF-IDF. Insgesamt sind die Ergebnisse mit euklidischer Distanz nur geringfügig besser als die Zufallsauswahl.

6.3.2. Kosinus-Ähnlichkeit

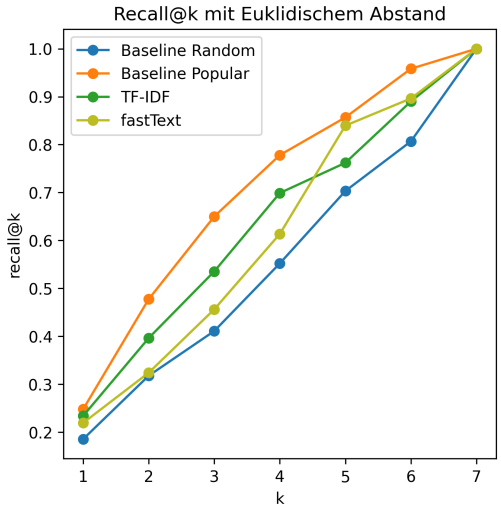
Bei der Kosinus-Ähnlichkeit wurden die besten Ergebnisse mit den GloVe- und den fastText-Embeddings erzielt. Für die Evaluierung wurde der vollständige Datensatz verwendet. In Abbildung 6.5 sind die Ergebnisse für die beiden Embedding Methoden mit der Kosinus-Ähnlichkeit dargestellt. Das fastText Embedding liefert bessere Ergebnisse als das Glove Embedding, mit Ausnahme von $k \in \{2, 5\}$ für Precision@k und $k = 5$ für den Recall@k. Die Ergebnisse sind besser als die Random Baseline, aber schlechter als die Popular Baseline. Die Ergebnisse liegen ungefähr zwischen den beiden Baselines und liefern somit ein besseres Ergebnis als die euklidische Distanz.

6.3.3. TextCF

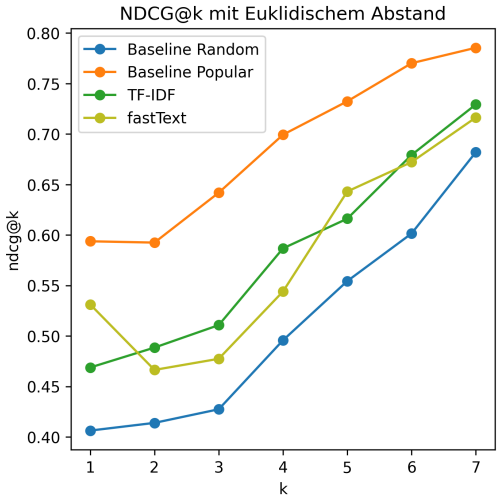
Für die Evaluierung der TextCF Komponente wurden ca. 20% der Daten aus dem Datensatz für die Evaluierung verwendet und mit den restlichen Daten wurde das Modell trainiert. Die Baselines wurden für die Evaluierung ebenfalls nur mit den Daten erstellt, die für die Evaluierung der TextCF Komponente verwendet wurden. Daher ist der Verlauf hier etwas anders als bei der euklidischen Distanz und der Kosinus-Ähnlichkeit. Die TextCF Komponente wurde mit verschiedenen Dimensionen der MLP Layer ausgeführt, um herauszufinden, welche Dimension für die einzelnen Embeddings am besten funktioniert. Die getesteten Dimensionen waren 2, 4, 6 und 8. Bei höheren Dimensionen wichen der Train- und Test-Loss zu stark voneinander ab. Die Lernrate des Adam-Optimizers wurde auf $1 \cdot 10^{-3}$ gesetzt und das Modell wurde immer für 70 Epochen trainiert. Für die Endergebnisse wurde jeweils der Durchschnitt der Ergebnisse der letzten fünf Epochen gebildet, da aufgrund der geringen Datenmenge die Ergebnisse teilweise schwankten. Am besten funktionieren die GloVe Embeddings auf Stichwörtern mit Dimension=6, die fastText Embeddings auf Stichwörtern mit Dimension=2, die Word2Vec Embeddings auf Stichwörtern mit Dimension=8 und die GloVe Embeddings auf normalen Beschreibungen mit Dimension=8. Die Ergebnisse dieser Kombinationen sind in Abbildung 6.6 dargestellt. Insgesamt funktioniert von den genannten Kombinationen die GloVe Embeddings auf Stichwörtern mit der Dimension 6 am besten.



(a) Precision@k für den euklidischen Abstand mit TF-IDF und fastText Embeddings.



(b) Recall@k für den euklidischen Abstand mit TF-IDF und fastText Embeddings.



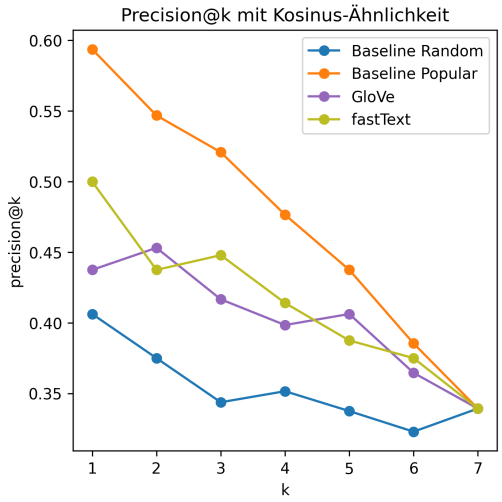
(c) NDCG@k für den euklidischen Abstand mit TF-IDF und fastText Embeddings.

Abbildung 6.4.: Beste Ergebnisse mit euklidischem Abstand

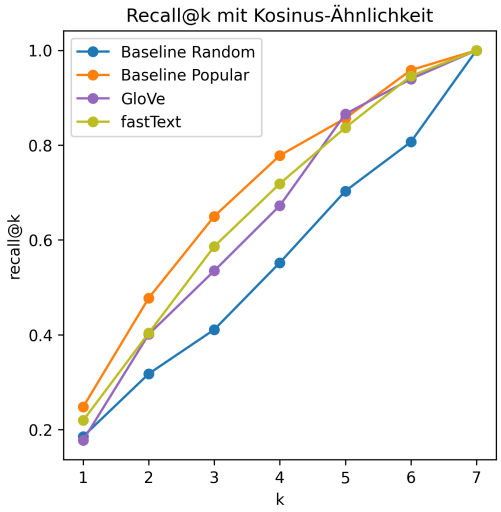
Die Ergebnisse sind deutlich besser als die Random Baseline und übertreffen für Precision@k und NDCG@k auch die Popular Baseline. Lediglich für Recall@k ist die Popular Baseline teilweise besser.

6.3.4. Diskussion der Ergebnisse

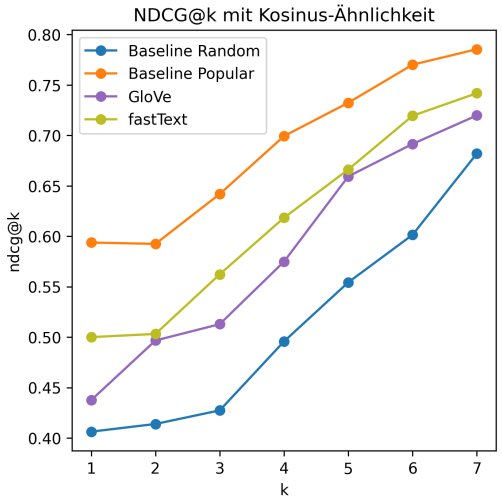
Die Ergebnisse zeigen, dass alle Methoden in der Lage sind, einem Benutzer, der mit den Datenvalidierungsmethoden nicht vertraut ist, einen besseren Vorschlag zu machen, als es der Benutzer selbst könnte. Aufgrund der ungleichen Verteilung im Datensatz ist die Popular Baseline für diesen Datensatz relativ stark. Dies bedeutet jedoch nicht, dass dies immer der Fall sein muss.



(a) Precision@k für die Kosinus-Ähnlichkeit mit GloVe und fastText Embeddings.



(b) Recall@k für die Kosinus-Ähnlichkeit mit GloVe und fastText Embeddings.

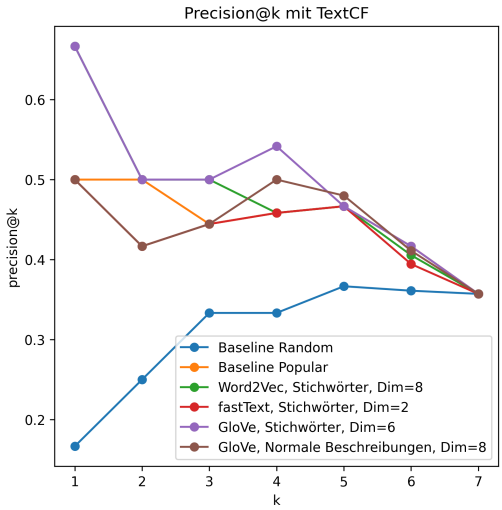


(c) NDCG@k für die Kosinus-Ähnlichkeit mit GloVe und fastText Embeddings.

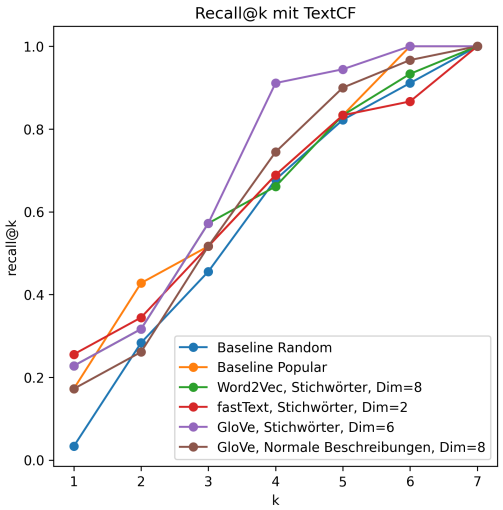
Abbildung 6.5.: Beste Ergebnisse mit Kosinus-Ähnlichkeit

Beispielsweise kann sich die Verteilung der am häufigsten verwendeten Datenvalidierungsmethoden ändern, wenn dem Datensatz mehr Datenvalidierungsmethoden hinzugefügt werden oder andere Datensätze mit geeigneten Datenvalidierungsmethoden hinzugefügt werden. Zudem weiß ein technisch unerfahrener Benutzer nicht immer, welche Datenvalidierungsmethoden am häufigsten verwendet werden. Daher wird in dieser Arbeit bereits das Übertreffen der Random Baseline als Nutzen für den Anwender des Systems betrachtet.

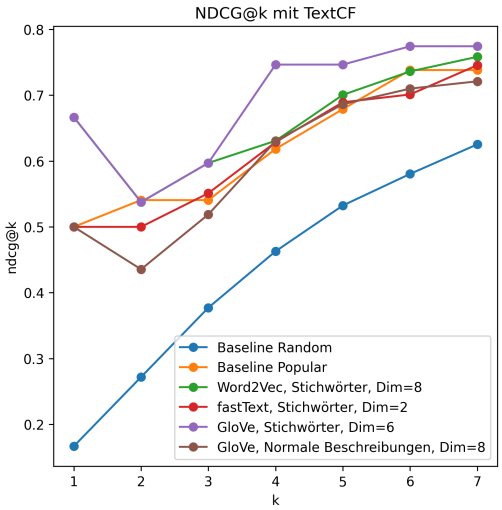
Für den euklidischen Abstand und die Kosinus-Ähnlichkeit werden die besten Ergebnisse mit den normalen Beschreibungen der Datensätze und Datenvalidierungsmethoden erzielt. Dies kann darauf zurückzuführen sein, dass die Vektoren der normalen Beschreibungen näher beieinander liegen als die Vektoren der Stichwörter. Für die TextCF Komponente liefern vor allem die Stichwörter



(a) Precision@k für TextCF mit den besten Ergebnissen.



(b) Recall@k für TextCF mit den besten Ergebnissen.



(c) NDCG@k für TextCF mit den besten Ergebnissen.

Abbildung 6.6.: Beste Ergebnisse der TextCF Komponente

ein besseres Ergebnis als die normalen Beschreibungen. Dies kann daran liegen, dass durch die Durchschnittsbildung über alle Wörter die bedeutungsvollen Wörter durch andere Wörter unbedeutend gemacht werden. Möglicherweise kann die TextCF-Komponente die Beziehungen zwischen den Beschreibungen besser lernen, wenn weniger Wörter verwendet werden oder wenn die verwendeten Wörter aussagekräftiger sind.

Im Allgemeinen lieferte die TextCF Komponente die besten Ergebnisse in Kombination mit den GloVe Embeddings der Stichwörter und einer Dimension von 6 für die MLP-Layer. Dabei wurde in 66,7% der Fälle eine relevante Datenvalidierungsmethode an erster Stelle vorgeschlagen. Obwohl die Precision@k und der NDCG@k für $k \in \{2, 3\}$ wieder abnehmen, liefert die Methode immer

noch bessere Ergebnisse als die Popular Baseline. Somit ist die Methode in der Lage, einem Benutzer sinnvolle Vorschläge für eine Datenvalidierungsmethode zu liefern, selbst wenn der Benutzer bedingt mit den Datenvalidierungsmethoden vertraut ist und immer die populärste Methode wählt.

Die Fähigkeit des Recommender Systems, passende Datenvalidierungsmethoden zu liefern, wurde mit einem Datensatz durchgeführt, der Datensatz-Datenvalidierungsmethoden-Paare enthält, da für eine Evaluation mit Kontextmodellen zu wenige Kontextmodelle zur Verfügung standen. Die Ergebnisse sollten aber auch für Kontextmodelle relevant sein, da die Recommender Systeme die Vorhersagen auf Basis von Textbeschreibungen getroffen haben, die sich für Datensätze und Kontextmodelle nicht zu sehr unterscheiden sollten, da beide Beschreibungen in der Regel ähnliche Inhalte, wie z.B. die Art der gemessenen Daten, enthalten.

Zusammenfassend können Recommender Systeme verwendet werden, um geeignete Datenvalidierungsmethoden vorzuschlagen. Dabei liefert die TextCF Komponente, die auf einem Machine Learning Ansatz basiert, bessere Ergebnisse als einfache Ähnlichkeitsmetriken. Die TextCF Komponente lieferte die besten Ergebnisse mit Hilfe der Embeddings des GloVe Modells.

6.4. Evaluationsergebnisse unter Verwendung der Kontextmodelle

Da in Abschnitt 6.3 die Kontextmodelle noch nicht enthalten sind, werden hier noch die Ergebnisse der Recommender Systeme auf den Textbeschreibungen der Kontextmodelle gezeigt. Dabei werden in Tabelle 6.1 die Ergebnisse der TextCF Komponente mit GloVe Embeddings auf den Stichwörtern und einer Dimension von 6 gezeigt. Die GloVe Embeddings wurden hier verwendet, da sie, wie in Abschnitt 6.3 gezeigt, am besten mit der TextCF Komponente funktioniert haben. Die Ergebnisse für den euklidischen Abstand und Kosinus-Ähnlichkeit sind in Tabelle 6.2 und Tabelle 6.3 zu finden. Mit „Einfaches Kontextmodell“ ist das Kontextmodell aus [DABS22] und mit „IoT-Szenario“ das Kontextmodell aus [Sch22] gemeint. Da für die Kontextmodelle keine geeigneten Datenvalidierungsmethoden bekannt sind und die Daten auch keine Outlier aufweisen, ist eine detaillierte Auswertung leider nicht möglich. Es ist jedoch anzumerken, dass das Recommender System, welches auf der TextCF Komponente basiert, in diesem Fall jeweils Deep Support Vector Data Description an letzter Stelle empfohlen hat. Dies macht durchaus Sinn, da diese Methode meist für Daten verwendet wird, die eine hohe Dimension haben [RVG+18]. Die simpleren Methoden wurden dagegen früher empfohlen.

Rang	Einfaches Kontextmodell		IoT-Szenario	
	Datenvalidierungsmethode	Score	Datenvalidierungsmethode	Score
1	Isolation Forest	0, 9877	Isolation Forest	0, 9792
2	Local Outlier Factor	0, 7366	k-Nearest-Neighbors	0, 7446
3	k-Nearest-Neighbors	0, 6390	Local Outlier Factor	0, 6763
4	Principal Component Analysis	0, 6364	Clustering Based Local Outlier Factor	0, 3937
5	Clustering Based Local Outlier Factor	0, 6213	Principal Component Analysis	0, 3063
6	Angle-Based Outlier Detection	0, 2911	Angle-Based Outlier Detection	0, 2091
7	Deep Support Vector Data Description	0, 2117	Deep Support Vector Data Description	0, 1481

Tabelle 6.1.: Vorschläge für die Kontextmodelle mit der TextCF-Komponente und Glove Embeddings auf Stichwörter mit Dim=6. Das einfache Kontextmodell stammt aus [DABS22] und das Kontextmodell IoT-Szenario stammt aus [Sch22]. „Score“ gibt die ausgegebene Wahrscheinlichkeit der TextCF-Komponente an.

Rang	TF-IDF				fastText			
	Einfaches Kontext-modell	Score	IoT-Szenario	Score	Einfaches Kontext-modell	Score	IoT-Szenario	Score
1	kNN	1, 3633	DSVDD	1, 3643	DSVDD	0, 1413	DSVDD	0, 1399
2	DSVDD	1, 3899	LOF	1, 4001	IForest	0, 1489	CBLOF	0, 1493
3	PCA	1, 3976	PCA	1, 4020	kNN	0, 1504	kNN	0, 1496
4	IForest	1, 3979	kNN	1, 4039	PCA	0, 1528	PCA	0, 1515
5	ABOD	1, 3995	ABOD	1, 4142	ABOD	0, 1570	IForest	0, 1535
6	LOF	1, 4093	CBLOF	1, 4142	CBLOF	0, 1581	ABOD	0, 1568
7	CBLOF	1, 4142	IForest	1, 4142	LOF	0, 1593	LOF	0, 1624

Tabelle 6.2.: Vorschläge für die Kontextmodelle mit euklidischen Abstand und TF-IDF und fastText Embeddings. Aus Platzgründen wurden die Methoden zur Outlier Detection abgekürzt. Das einfache Kontextmodell stammt aus [DABS22] und das Kontextmodell IoT-Szenario stammt aus [Sch22]. „Score“ gibt den euklidischen Abstand an.

Rang	GloVe				fastText			
	Einfaches Kontext-modell	Score	IoT-Szenario	Score	Einfaches Kontext-modell	Score	IoT-Szenario	Score
1	IForest	0, 8578	DSVDD	0, 8222	DSVDD	0, 8954	DSVDD	0, 8837
2	DSVDD	0, 8479	IForest	0, 8169	IForest	0, 8876	IForest	0, 8710
3	ABOD	0, 8277	ABOD	0, 7977	kNN	0, 8811	kNN	0, 8666
4	kNN	0, 8060	CBLOF	0, 7718	PCA	0, 8771	PCA	0, 8629
5	CBLOF	0, 8046	LOF	0, 7702	ABOD	0, 8706	CBLOF	0, 8611
6	PCA	0, 8012	kNN	0, 7656	LOF	0, 8701	ABOD	0, 8548
7	LOF	0, 7995	PCA	0, 7587	CBLOF	0, 8666	LOF	0, 8526

Tabelle 6.3.: Vorschläge für die Kontextmodelle mit Kosinus-Ähnlichkeit und GloVe und fastText Embeddings. Aus Platzgründen wurden die Methoden zur Outlier Detection abgekürzt. Das einfache Kontextmodell stammt aus [DABS22] und das Kontextmodell IoT-Szenario stammt aus [Sch22]. „Score“ gibt die Kosinus-Ähnlichkeit an.

7. Zusammenfassung und Ausblick

In dieser Arbeit wurde ein Konzept vorgestellt, um Datenvalidierungsmethoden basierend auf Kontextmodellen vorzuschlagen. Dies ermöglicht es auch technisch unerfahrenen Benutzern, geeignete Datenvalidierungsmethoden für ihre IoT-Anwendung auszuwählen.

Der Einsatz von IoT ist mittlerweile in vielen verschiedenen Anwendungsbereichen wie Smart Homes oder Industrie 4.0 präsent. Dabei ist die Anzahl der IoT-Geräte und damit auch die Menge der generierten Daten in letzter Zeit stark angestiegen. Die erzeugten Daten können dabei verschiedene Fehler enthalten, die unter anderem durch defekte Sensoren oder auch durch Fehler bei der Datenübertragung entstehen können. Zudem ist es durch modellgetriebene Entwicklung auch für technisch unerfahrene Nutzer einfacher geworden, solche IoT-Anwendungen zu erstellen. Diese Anwendungen können mithilfe von Kontextmodellen dargestellt werden. Dabei ist es für technisch unerfahrene Nutzer nicht immer einfach, geeignete Datenvalidierungsmethoden für die erzeugten Daten der IoT-Anwendungen auszuwählen.

Das in dieser Arbeit entwickelte Konzept ermöglicht es auch technisch unerfahrenen Benutzern, geeignete Datenvalidierungsmethoden zu verwenden, indem dieses mit Hilfe des Kontextmodells der IoT-Anwendung geeignete Datenvalidierungsmethoden vorschlägt. Zunächst wurden verschiedene Typen von Recommender Systemen untersucht, um eine für die Problemstellung geeignete Variante zu finden. Textbasierte Recommender Systeme erwiesen sich als am besten geeignet, da diese auch mit wenigen Kontextmodellen trainiert werden können. Um eine textuelle Beschreibung eines Kontextmodells zu erhalten, die dann als Input für die Recommender Systeme verwendet werden kann, wird ein KG-to-Text Modell verwendet. Hierfür wurden verschiedene Modelle untersucht, wobei sich Chat-GPT für diese Arbeit am besten eignet, da das Modell nicht erst trainiert werden muss, was bei der geringen Anzahl verfügbarer Kontextmodelle von Vorteil ist. Um die Recommender Systeme auf einem etwas größeren Datensatz trainieren zu können, als es nur mit den verfügbaren Kontextmodellen möglich gewesen wäre, wurde zusätzlich ein Datensatz aus verschiedenen Outlier Detection Methoden und Datensätzen erstellt. Der Datensatz umfasst dabei die Performanz der Methoden auf den verschiedenen Datensätzen, sowie die textuellen Beschreibungen der Methoden und Datensätze. Für die Feature Selection der Recommender Systeme werden verschiedene Word Embedding Methoden sowie Term Frequenzy - Inverse Document Frequency verwendet, da diese Methoden eine einfache und sinnvolle Umwandlung von Text in Vektoren bieten. Um eine Empfehlung zu erhalten werden verschiedene Computational Approaches vorgestellt. Diese sind der euklidische Abstand, die Kosinus-Ähnlichkeit und eine auf neuronalen Netzen basierende Methode.

Das System zum Vorschlagen von Datenvalidierungsmethoden wurde prototypisch in Python implementiert. Dabei wird die RDF-Beschreibung des Kontextmodells zunächst mit Hilfe des KG-to-Text-Modells in eine Textbeschreibung umgewandelt, welche dann von einer Preprocessing-Komponente bearbeitet wird. Die aufbereiteten Textbeschreibungen werden dann an das Recommender System

übergeben, welches diese in Vektoren und diese wiederum in ein Ranking von Empfehlungen umwandelt. Dabei ist die Kombination der GloVe Embeddings aus Stichwörtern in Kombination mit der TextCF-Komponente am besten für die Vorhersage von Datenvalidierungsmethoden geeignet.

Die Evaluierung zeigt, dass die Empfehlungen des Recommender Systems besser sind als die eines Benutzers, der zufällige Methoden zur Datenvalidierung auswählt. Das System kann also einem technisch unerfahrenen Benutzer, der aus Unwissenheit zufällige Methoden wählt, helfen, eine geeignete Datenvalidierungsmethode zu finden.

Ausblick

In dieser Arbeit wird nur die Textbeschreibung als Input für die Recommender-Systeme verwendet. Dies ist unter anderem auf die geringe Verfügbarkeit von Kontextmodellen zurückzuführen, die auf der gleichen Ontologie basieren. Wenn mehr Kontextmodelle der gleichen Ontologie zur Verfügung stehen, können weitere Eigenschaften der Kontextmodelle genutzt werden, um die Performance der Recommender Systeme zu verbessern und somit ein hybrides Recommender System zu verwenden. Dazu kann z.B. ein Knowledge Graph erstellt werden, der bestimmte Eigenschaften aus den Kontextmodellen extrahiert und Eigenschaften über die Datenvalidierungsmethoden enthält. Anstelle eines Knowledge Graphen können auch nur verschiedene Attribute aus den Kontextmodellen extrahiert und auf deren Basis eine Empfehlung gegeben werden.

In letzter Zeit wurden zahlreiche Deep Learning basierte Recommender Systeme vorgestellt, die in der Lage sind auch nichtlineare Beziehungen zwischen Nutzern und Produkten zu erfassen [Mu18]. Es bietet sich daher an, auch für die Problemstellung dieser Arbeit Recommender Systeme basierend auf Deep Learning zu untersuchen. Dies war im Rahmen dieser Arbeit nur bedingt möglich, da hierfür in der Regel ein größerer Datensatz benötigt wird.

Für die Datenvalidierung wurden zahlreiche Datenvalidierungsmethoden entwickelt [AHS23]. Diese Arbeit untersucht nur einen kleinen Teil der verfügbaren Datenvalidierungsmethoden und beschränkt sich auf Methoden zur Erkennung von Ausreißern. In Zukunft können weitere Datenvalidierungsmethoden in die Empfehlungen aufgenommen werden.

Für die Evaluierung in dieser Arbeit wurde nur eine kleine Anzahl von Kontextmodellen für die Evaluierung verwendet. Wenn im Laufe der Zeit mehr Kontextmodelle erstellt werden, kann das System auf einer breiteren Basis evaluiert werden und es kann untersucht werden, wie sich die Recommender Systeme bei vielfältigeren Kontextmodellen verhalten.

Eine weitere Optimierung wäre eine stärkere Automatisierung des Systems, da die einzelnen Komponenten bisher nur prototypisch implementiert sind. So könnte z.B. eine graphische Benutzeroberfläche erstellt werden, die den Benutzern eine einfachere Anwendung ermöglicht.

Literaturverzeichnis

[AG12] M. Amer, M. Goldstein. „Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer“. In: *Proc. of the 3rd RapidMiner Community Meeting and Conference (RCOMM 2012)*. 2012, S. 1–12 (zitiert auf S. 54).

[Agg+16] C. C. Aggarwal et al. *Recommender systems*. Bd. 1. Springer, 2016 (zitiert auf S. 18, 19, 29).

[AHS23] M. Abdelaal, C. Hammacher, H. Schoening. „Rein: A comprehensive benchmark framework for data cleaning methods in ml pipelines“. In: *arXiv preprint arXiv:2302.04702* (2023) (zitiert auf S. 17, 78).

[BEBT16] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor. „IoT-Lite: a lightweight semantic model for the Internet of Things“. In: *2016 INTL IEEE conferences on ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress (uic/atc/scalcom/cbdcom/iop/smartworld)*. IEEE. 2016, S. 90–97 (zitiert auf S. 16, 17).

[BGJM17] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov. „Enriching word vectors with subword information“. In: *Transactions of the association for computational linguistics* 5 (2017), S. 135–146 (zitiert auf S. 20, 23, 41).

[BKNS00] M. M. Breunig, H.-P. Kriegel, R. T. Ng, J. Sander. „LOF: identifying density-based local outliers“. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, S. 93–104 (zitiert auf S. 54).

[BZA20] A. Boukerche, L. Zheng, O. Alfandi. „Outlier detection: Methods, models, and classification“. In: *ACM Computing Surveys (CSUR)* 53.3 (2020), S. 1–37 (zitiert auf S. 17).

[CAW22] A. Colas, M. Alvandipour, D. Z. Wang. „GAP: A graph-aware language model framework for knowledge graph-to-text generation“. In: *arXiv preprint arXiv:2204.06674* (2022) (zitiert auf S. 31).

[CIKW16] X. Chu, I. F. Ilyas, S. Krishnan, J. Wang. „Data cleaning: Overview and emerging challenges“. In: *Proceedings of the 2016 international conference on management of data*. 2016, S. 2201–2206 (zitiert auf S. 17).

[CSY+21] X. Cao, Y. Shi, H. Yu, J. Wang, X. Wang, Z. Yan, Z. Chen. „DEKR: description enhanced knowledge graph for machine learning method recommendation“. In: *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021, S. 203–212 (zitiert auf S. 16, 25, 29, 37, 42, 45, 65).

[CYW+23] L. Cao, Y. Yan, Y. Wang, S. Madden, E. A. Rundensteiner. „AutoOD: Automatic Outlier Detection“. In: *Proceedings of the ACM on Management of Data* 1.1 (2023), S. 1–27 (zitiert auf S. 13, 25).

[CZS+16] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, M. E. Houle. „On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study“. In: *Data mining and knowledge discovery* 30 (2016), S. 891–927 (zitiert auf S. 54).

[D2122] D212digital. *What is Lemmatization and Stemming in NLP?* 2022. URL: https://212digital.medium.com/what-is-lemmatization-and-stemming-in-nlp-e25e142332c4 (zitiert auf S. 56).

[DABS22] D. Del Gaudio, B. Ariguib, A. Bartenbach, G. Solakis. „A live context model for semantic reasoning in IoT applications“. In: *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2022, S. 322–327 (zitiert auf S. 13, 16, 17, 28, 31, 33, 35, 73–75).

[Dee23] Deepanshi. *Text Preprocessing in NLP with Python Codes*. 2023. URL: https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/ (zitiert auf S. 56).

[DEK23] M. Dallinger, P. Epple, L. Kollikowsk. „Categorization and evaluation of methods and techniques for validating IoT data“. In: 2023 (zitiert auf S. 51).

[Dhi23] A. Dhinakaran. *Demystifying NDCG*. 2023. URL: https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0 (zitiert auf S. 66).

[FWM+15] M. U. Farooq, M. Waseem, S. Mazhar, A. Khairi, T. Kamal. „A review on internet of things (IoT)“. In: *International journal of computer applications* 113.1 (2015), S. 1–7 (zitiert auf S. 15).

[GAR19] X. Gu, L. Akoglu, A. Rinaldo. „Statistical analysis of nearest neighbor methods for anomaly detection“. In: *Advances in Neural Information Processing Systems* 32 (2019) (zitiert auf S. 54).

[GBB18] P. Gokhale, O. Bhat, S. Bhat. „Introduction to IOT“. In: *International Advanced Research Journal in Science, Engineering and Technology* 5.1 (2018), S. 41–44 (zitiert auf S. 15).

[GKT19] L. Georgios, S. Kerstin, A. Theofylaktos. „Internet of things in the context of industry 4.0: An overview“. In: (2019) (zitiert auf S. 13).

[GPVW17] J. M. Gomez-Perez, J. Z. Pan, G. Vetere, H. Wu. „Enterprise knowledge graph: An introduction“. In: *Exploiting linked data and knowledge graphs in large organisations*. Springer, 2017, S. 1–14 (zitiert auf S. 16).

[GS09] N. Gibbins, N. Shadbolt. „Resource Description Framework (RDF)“. In: 2009. URL: https://api.semanticscholar.org/CorpusID:3288537 (zitiert auf S. 16).

[GZQ+20] Q. Guo, F. Zhuang, C. Qin, H. Zhu, X. Xie, H. Xiong, Q. He. „A survey on knowledge graph-based recommender systems“. In: *IEEE Transactions on Knowledge and Data Engineering* 34.8 (2020), S. 3549–3568 (zitiert auf S. 16).

[HBC+21] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier et al. „Knowledge graphs". In: *ACM Computing Surveys (Csur)* 54.4 (2021), S. 1–37 (zitiert auf S. 16).

[HLZ+17] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua. „Neural collaborative filtering". In: *Proceedings of the 26th international conference on world wide web*. 2017, S. 173–182 (zitiert auf S. 13, 17, 18, 46, 48).

[IHD21] N. Ishaq, T. J. Howard, N. M. Daniels. „Clustered hierarchical anomaly and outlier detection algorithms". In: *2021 IEEE International Conference on Big Data (Big Data)*. IEEE. 2021, S. 5163–5174 (zitiert auf S. 54).

[KBV09] Y. Koren, R. Bell, C. Volinsky. „Matrix factorization techniques for recommender systems". In: *Computer* 42.8 (2009), S. 30–37 (zitiert auf S. 13).

[KJR+21] P. Ke, H. Ji, Y. Ran, X. Cui, L. Wang, L. Song, X. Zhu, M. Huang. „Jointgt: Graph-text joint representation learning for text generation from knowledge graphs". In: *arXiv preprint arXiv:2106.10502* (2021) (zitiert auf S. 31, 32).

[KK20] A. Khanna, S. Kaur. „Internet of things (IoT), applications and challenges: a comprehensive review". In: *Wireless Personal Communications* 114 (2020), S. 1687–1762 (zitiert auf S. 13).

[KNMN21] S. Kanwal, S. Nawaz, M. K. Malik, Z. Nawaz. „A review of text-based recommendation systems". In: *IEEE Access* 9 (2021), S. 31638–31661 (zitiert auf S. 37, 39–44, 65, 66).

[KSZ08] H.-P. Kriegel, M. Schubert, A. Zimek. „Angle-based outlier detection in high-dimensional data". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, S. 444–452 (zitiert auf S. 54).

[LLC15] X. Li, J. C. Lv, D. Cheng. „Angle-based outlier detection algorithm with more stable relationships". In: *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, Volume 1*. Springer. 2015, S. 433–446 (zitiert auf S. 54).

[LTZ+21] J. Li, T. Tang, W. X. Zhao, Z. Wei, N. J. Yuan, J.-R. Wen. „Few-shot knowledge graph-to-text generation with pretrained language models". In: *arXiv preprint arXiv:2106.01623* (2021) (zitiert auf S. 31).

[LTZ08] F. T. Liu, K. M. Ting, Z.-H. Zhou. „Isolation forest". In: *2008 eighth ieee international conference on data mining*. IEEE. 2008, S. 413–422 (zitiert auf S. 54).

[LYW12] Y.-J. Lee, Y.-R. Yeh, Y.-C. F. Wang. „Anomaly detection via online oversampling principal component analysis". In: *IEEE transactions on knowledge and data engineering* 25.7 (2012), S. 1460–1470 (zitiert auf S. 54).

[MCCD13] T. Mikolov, K. Chen, G. Corrado, J. Dean. „Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013) (zitiert auf S. 20–22, 41).

[MGB+18] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, A. Joulin. „Advances in Pre-Training Distributed Word Representations". In: *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*. 2018 (zitiert auf S. 59).

[MMIP14] A. Moreno, L. Marin, D. Isern, D. Perelló. „Dynamic learning of keyword-based preferences for news recommendation“. In: *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*. Bd. 1. IEEE. 2014, S. 347–354 (zitiert auf S. 13).

[MNG+17] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, I. Yaqoob. „Big IoT data analytics: architecture, opportunities, and open research challenges“. In: *ieee access* 5 (2017), S. 5247–5261 (zitiert auf S. 13).

[Mon18] D. Monsters. *Text Preprocessing in Python: Steps, Tools, and Examples*. 2018. URL: https://medium.com/product-ai/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908 (zitiert auf S. 56).

[MSC+13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean. „Distributed representations of words and phrases and their compositionality“. In: *Advances in neural information processing systems* 26 (2013) (zitiert auf S. 42, 58).

[Mu18] R. Mu. „A survey of recommender systems based on deep learning“. In: *Ieee Access* 6 (2018), S. 69009–69022 (zitiert auf S. 78).

[MZW+23] X. Ma, J. Zhao, Y. Wang, C. Shang, F. Jiang. „Robust factored principal component analysis for matrix-valued outlier accommodation and detection“. In: *Computational Statistics & Data Analysis* 179 (2023), S. 107657 (zitiert auf S. 54).

[NHF21] S. A. N. Nozad, M. A. Haeri, G. Folino. „SDCOR: Scalable density-based clustering for local outlier detection in massive-scale datasets“. In: *Knowledge-based systems* 228 (2021), S. 107256 (zitiert auf S. 54).

[Pha18] N. Pham. „L1-depth revisited: A robust angle-based outlier factor in high-dimensional space“. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2018, S. 105–121 (zitiert auf S. 54).

[PSM14] J. Pennington, R. Socher, C. D. Manning. „Glove: Global vectors for word representation“. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, S. 1532–1543 (zitiert auf S. 20–22, 41, 58).

[Ras23] V. Rastogi. *Euclidean distance and cosine similarity*. 2023. URL: https://medium.com/@vaibhav1403/euclidean-distance-and-cosine-similarity-69cbf8140fed (zitiert auf S. 42).

[RCIR17] T. Rekatsinas, X. Chu, I. F. Ilyas, C. Ré. „Holoclean: Holistic data repairs with probabilistic inference“. In: *arXiv preprint arXiv:1702.00820* (2017) (zitiert auf S. 26).

[RD+00] E. Rahm, H. H. Do et al. „Data cleaning: Problems and current approaches“. In: *IEEE Data Eng. Bull.* 23.4 (2000), S. 3–13 (zitiert auf S. 17).

[RSSG20] L. F. Ribeiro, M. Schmitt, H. Schütze, I. Gurevych. „Investigating pretrained language models for graph-to-text generation“. In: *arXiv preprint arXiv:2007.08426* (2020) (zitiert auf S. 31).

[RVG+18] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, M. Kloft. „Deep one-class classification“. In: *International conference on machine learning*. PMLR. 2018, S. 4393–4402 (zitiert auf S. 54, 73).

[Sch22] T. Schubert. „Context-aware data validation for machine learning pipelines“. Magisterarb. 2022 (zitiert auf S. 38, 73–75, 87).

[SDYC22] C. Shao, X. Du, J. Yu, J. Chen. „Cluster-based improved isolation forest“. In: *Entropy* 24.5 (2022), S. 611 (zitiert auf S. 54).

[SFBM23] Z. Shokrzadeh, M.-R. Feizi-Derakhshi, M.-A. Balafar, J. B. Mohasefi. „Knowledge graph-based recommendation system enhanced by neural collaborative filtering and knowledge graph embedding“. In: *Ain Shams Engineering Journal* (2023), S. 102263 (zitiert auf S. 16).

[Shr20] A. Shrivastav. *Curse Of Dimensionality: The curse that all ML Engineers need to deal with*. 2020. URL: https://medium.com/analytics-vidhya/curse-of-dimensionality-the-curse-that-all-ml-engineers-need-to-deal-with-5d459d39dc8a (zitiert auf S. 43).

[SL11] G. Shen, B. Liu. „The visions, technologies, applications and security issues of Internet of Things“. In: *2011 International conference on E-Business and E-Government (ICEE)*. IEEE. 2011, S. 1–4 (zitiert auf S. 15).

[Smi20] A. Smiti. „A critical overview of outlier detection methods“. In: *Computer Science Review* 38 (2020), S. 100306 (zitiert auf S. 13, 17).

[SXR+18] S. Su, L. Xiao, L. Ruan, F. Gu, S. Li, Z. Wang, R. Xu. „An efficient density-based local outlier detection approach for scattered data“. In: *IEEE Access* 7 (2018), S. 1006–1020 (zitiert auf S. 54).

[Tec23] TechClaw. *Cosine similarity between two arrays for word embeddings*. 2023. URL: https://medium.com/@techclaw/cosine-similarity-between-two-arrays-for-word-embeddings-c8c1c98811b (zitiert auf S. 44).

[VF13] O. Vermesan, P. Friess. *Internet of things: converging technologies for smart environments and integrated ecosystems*. River publishers, 2013 (zitiert auf S. 15).

[WZW+18] H. Wang, F. Zhang, J. Wang, M. Zhao, W. Li, X. Xie, M. Guo. „Ripplenet: Propagating user preferences on the knowledge graph for recommender systems“. In: *Proceedings of the 27th ACM international conference on information and knowledge management*. 2018, S. 417–426 (zitiert auf S. 13, 17, 20).

[WZX+19] H. Wang, M. Zhao, X. Xie, W. Li, M. Guo. „Knowledge graph convolutional networks for recommender systems“. In: *The world wide web conference*. 2019, S. 3307–3313 (zitiert auf S. 20).

[XWMZ17] D. Xu, Y. Wang, Y. Meng, Z. Zhang. „An improved data anomaly detection method based on isolation forest“. In: *2017 10th international symposium on computational intelligence and design (ISCID)*. Bd. 2. IEEE. 2017, S. 287–291 (zitiert auf S. 54).

[YBE13] M. Yakout, L. Berti-Équille, A. K. Elmagarmid. „Don’t be scared: Use scalable automatic repairing with maximal likelihood and bounded changes“. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. 2013, S. 553–564 (zitiert auf S. 26).

[YWW+19] P. Yang, D. Wang, Z. Wei, X. Du, T. Li. „An outlier detection approach based on improved self-organizing feature map clustering algorithm“. In: *IEEE Access* 7 (2019), S. 115914–115925 (zitiert auf S. 54).

[ZFW+22] F. Zhang, H. Fan, R. Wang, Z. Li, T. Liang. „Deep dual support vector data description for anomaly detection on attributed networks“. In: *International Journal of Intelligent Systems* 37.2 (2022), S. 1509–1528 (zitiert auf S. 54).

Alle URLs wurden zuletzt am 10. 12. 2023 geprüft.

A. RDF Beschreibungen

```
@prefix iot-context: <http://www.uni-stuttgart.de/2021/iot-context#> .
@prefix iot-ins:     <http://www.uni-stuttgart.de/2021/iot-instance#> .
@prefix iot-lite:    <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix qu:          <http://purl.org/NET/ssnx/qu/qu#> .
@prefix rdf:         <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:        <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ssn:         <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

iot-ins:mySystem
    rdf:type ssn:System ;
    rdfs:label "our system" ;
    iot-lite:hasSubsystem iot-ins:myDevice .

iot-ins:FirstSensingDevice
    rdf:type ssn:SensingDevice ;
    rdfs:label "Sensing Device for Sensor1" ;
    iot-lite:exposedBy iot-ins:sensingOperator ;
    iot-context:hasMeasurement iot-ins:temperature1 ;
    iot-lite:hasCoverage iot-ins:environment .

iot-ins:myDevice
    rdf:type ssn:Device ;
    rdfs:label "myFirstDevice" ;
    ssn:hasSubsystem iot-ins:FirstSensingDevice ;
    iot-context:hasMonitoringComponent iot-ins:CPUmonitor ;
    iot-lite:exposedBy iot-ins:MonitoringService .

iot-ins:MonitoringService
    rdf:type ssn:Service ;
    rdfs:label "Telegraf" .

iot-ins:CPUmonitor
    rdf:type iot-context:MonitoringComponent ;
    rdfs:label "CPU Data" ;
    iot-context:hasMeasurement iot-ins:load2 ;
    iot-context:hasMeasurement iot-ins:load1 .

<...>

iot-ins:environment
    rdf:type iot-lite:Rectangle ;
    rdfs:label "Area Room Uni" ;
    rdfs:comment "Description: Env Model 1" .

iot-ins:mySensor
    rdf:type ssn:Sensor ;
    rdfs:label "Sensor1 for temperature" ;
    iot-lite:hasSensingDevice iot-ins:FirstSensingDevice .
```

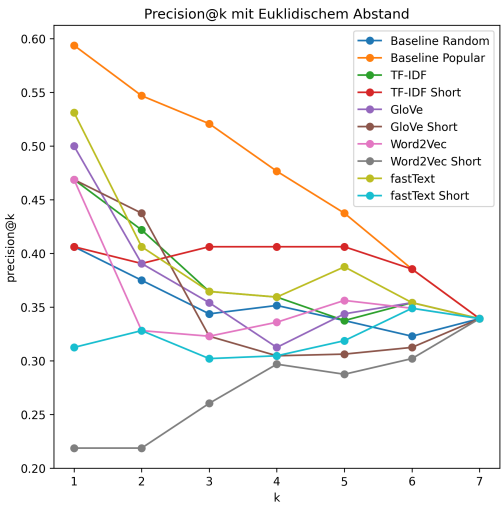
Listing A.1: Erstellte RDF-Beschreibung des Kontextmodells aus Abbildung 4.2. Die Beschreibung wurde aus Platzgründen gekürzt.

```
@prefix dc:          <http://purl.org/dc/elements/1.1/> .
@prefix geo:         <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix iot-context: <http://www.uni-stuttgart.de/2021/iot-context#> .
@prefix iot-ins:     <http://www.uni-stuttgart.de/2021/iot-instance#> .
@prefix iot-lite:    <http://purl.oclc.org/NET/UNIS/fiware/iot-lite#> .
@prefix m3-lite:     <http://purl.org/iot/vocab/m3-lite#> .
@prefix owl:         <http://www.w3.org/2002/07/owl#> .
@prefix qu:          <http://purl.org/NET/ssnx/qu/qu#> .
@prefix qu-rec20:    <http://purl.org/NET/ssnx/qu/qu-rec20#> .
@prefix rdf:         <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:        <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ssn:         <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix time:        <http://www.w3.org/2006/time#> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

iot-ins:mySystem
        rdf:type               ssn:System ;
        rdfs:label             "MA TestSystem" ;
        ssn:hasSubsystem  iot-ins:device_1 ;
        ssn:hasSubsystem  iot-ins:device_2 ;
        ssn:hasSubsystem  iot-ins:device_out ;
        ssn:hasSubsystem  iot-ins:device_main .

iot-ins:device_main
        rdf:type               ssn:Device ;
        rdfs:label             "device_main" ;
        iot-lite:id            "1" ;
        ssn:hasSubsystem       iot-ins:actuatingDevice .

iot-ins:actuatingDevice
        rdf:type                    iot-lite:ActuatingDevice ;
        rdfs:label                   "raspberry" .

<...>

iot-ins:sensor_1
        rdf:type                    ssn:Sensor ;
        rdfs:label                   "ds18b20_1" ;
        ssn:hasMetadata             iot-ins:metadata_ds18b20_min ;
        ssn:hasMetadata             iot-ins:metadata_ds18b20_max ;
        ssn:hasMetadata             iot-ins:metadata_ds18b20_res ;
        ssn:hasSensingDevice        iot-ins:sensingDevice_1 ;
        iot-lite:hasQuantityKind    qu:Temperature ;
        iot-lite:hasUnit            qu:degree_celsius .

<...>

iot-ins:measurement_3
        rdf:type                  iot-context:Measurement ;
        iot-context:hasTimeStamp  "2022-02-07T13:51:30.273Z"^^xsd:dateTime ;
        iot-context:hasValue      1 .
```

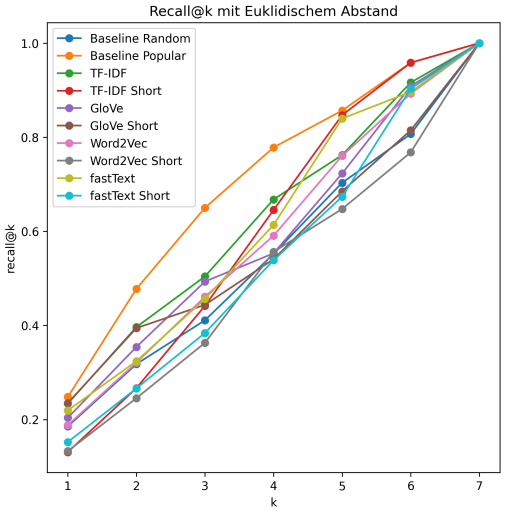
Listing A.2: RDF-Beschreibung des IoT-Kontextmodells aus [Sch22]

B. Plots

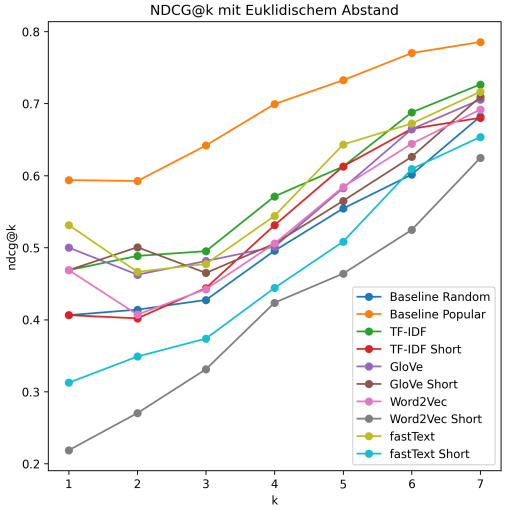
In Abbildung B.1 bis B.5 sind die Ergebnisse für die verschiedenen Metriken aus Abschnitt 6.1 aller Feature Selection Methoden im Zusammenhang mit den Computational Approaches dargestellt.



(a) Precision@k für den euklidischen Abstand, alle Feature Selection Methoden

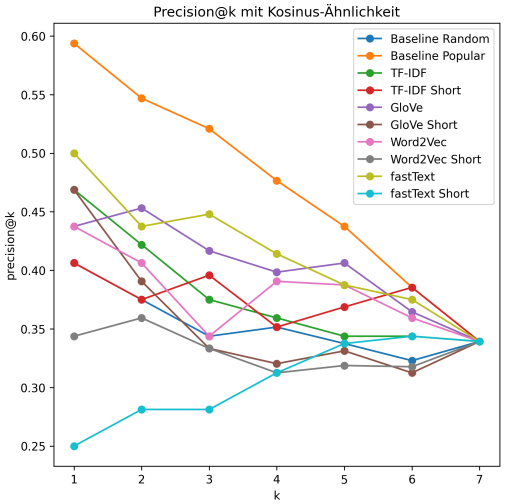


(b) Recall@k für den euklidischen Abstand, alle Feature Selection Methoden

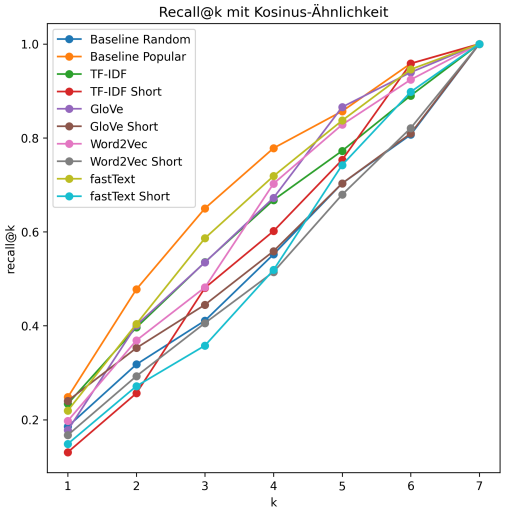


(c) NDCG@k für den euklidischen Abstand, alle Feature Selection Methoden

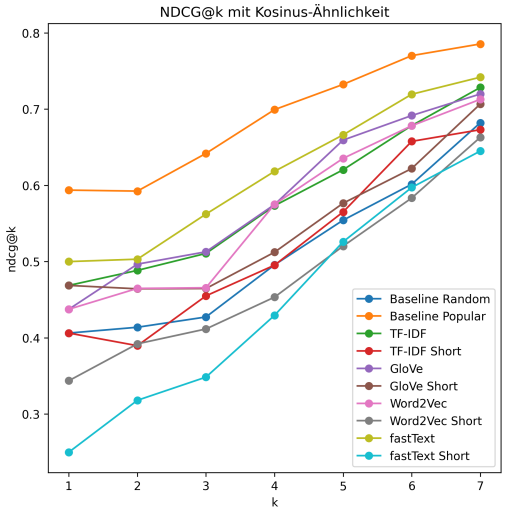
Abbildung B.1.: Vollständige Ergebnisse mit euklidischem Abstand



(a) Precision@k für für die Kosinus-Ähnlichkei, alle Feature Selection Methoden

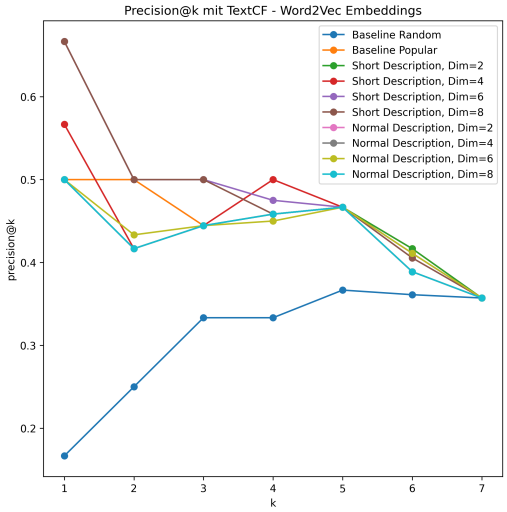


(b) Recall@k für die Kosinus-Ähnlichkei, alle Feature Selection Methoden

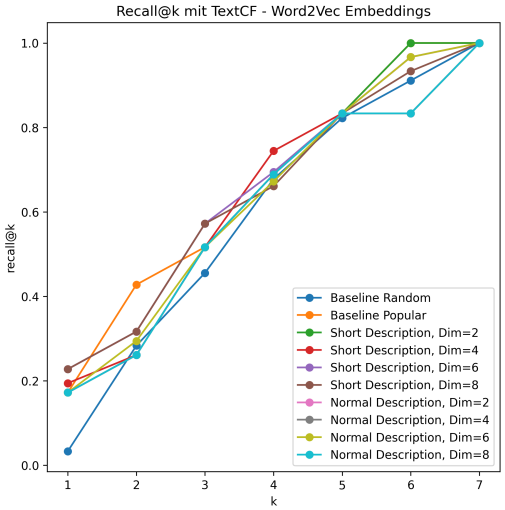


(c) NDCG@k für die Kosinus-Ähnlichkei, alle Feature Selection Methoden

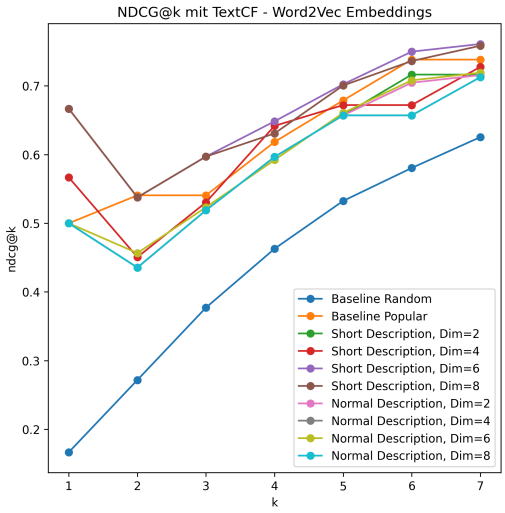
Abbildung B.2.: Vollständige Ergebnisse mit Kosinus-Ähnlichkeit



(a) Precision@k für TextCF mit Word2Vec Embeddings

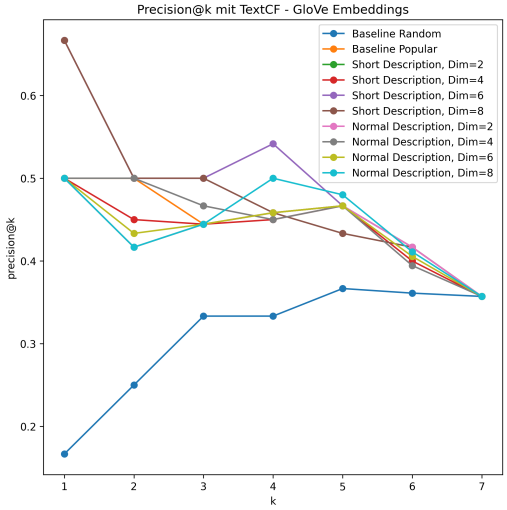


(b) Recall@k für TextCF mit Word2Vec Embeddings

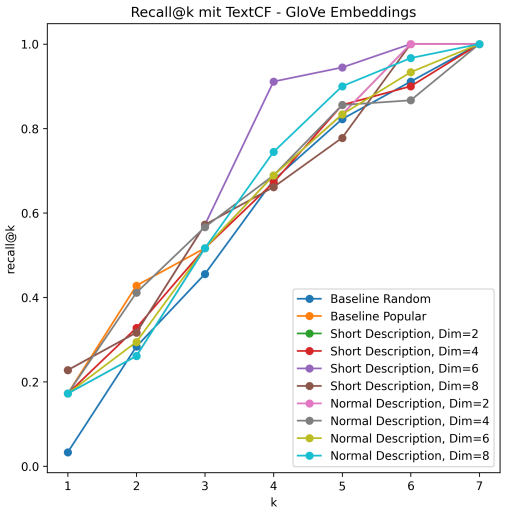


(c) NDCG@k für TextCF mit Word2Vec Embeddings

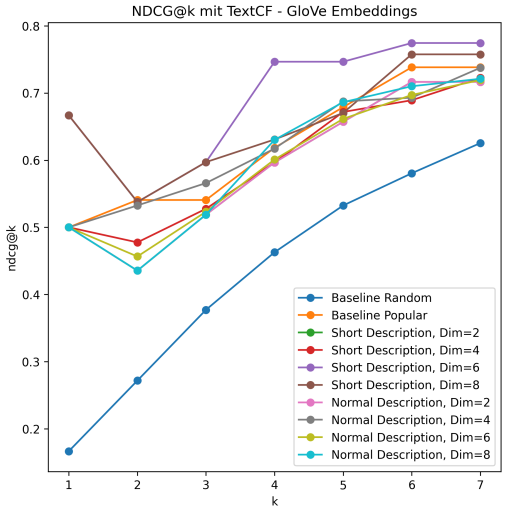
Abbildung B.3.: Vollständige Ergebnisse der TextCF Komponente mit Word2Vec Embeddings



(a) Precision@k für TextCF mit GloVe Embeddings

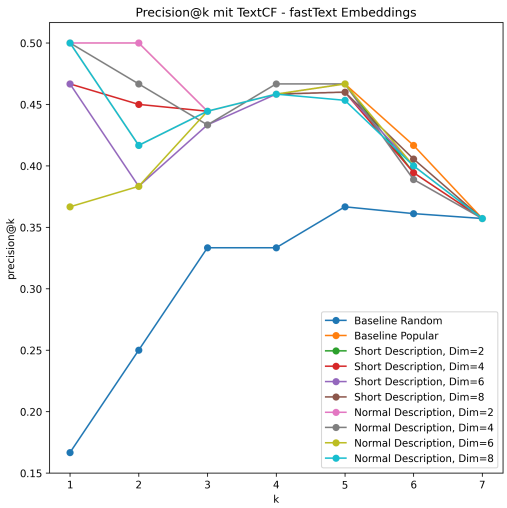


(b) Recall@k für TextCF mit GloVe Embeddings

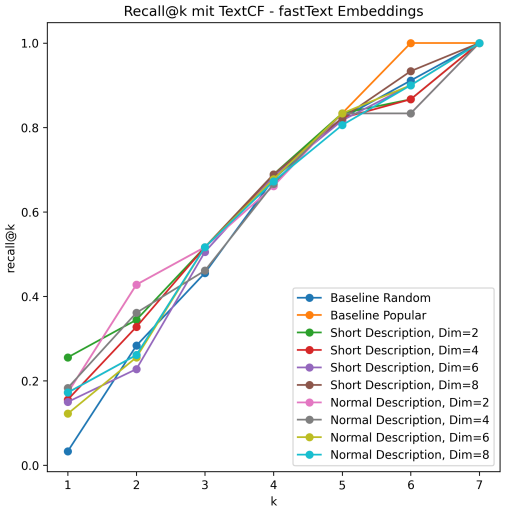


(c) NDCG@k für TextCF mit GloVe Embeddings

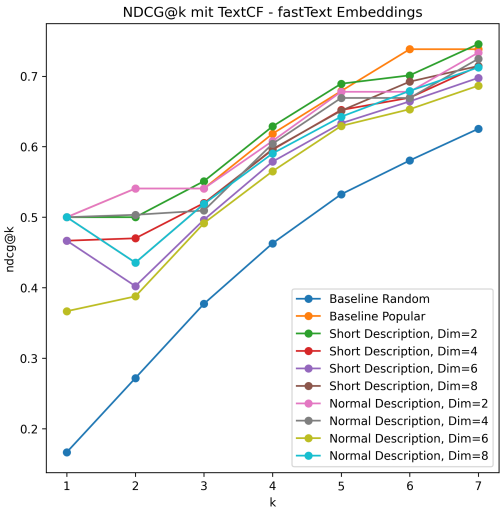
Abbildung B.4.: Vollständige Ergebnisse der TextCF Komponente mit GloVe Embeddings



(a) Precision@k für TextCF mit fastText Embeddings



(b) Recall@k für TextCF mit fastText Embeddings



(c) NDCG@k für TextCF mit fastText Embeddings

Abbildung B.5.: Vollständige Ergebnisse der TextCF Komponente mit fastText Embeddings

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift