UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

PALUNO
The Ruhr Institute for Software Technology

Institut für Informatik und Wirtschaftsinformatik (ICB)
Lehrstuhl für Software Engineering, insb. mobile Anwendungen
Prof. Dr. Volker Gruhn

Techniken der Computerlinguistik zur Verbesserung von Suchfunktionen in der Software-Entwicklung

Bachelorarbeit

vorgelegt der Fakultät für Wirtschaftswissenschaften
der Universität Duisburg-Essen (Campus Essen) von

Leon Zimmermann
Laddringsweg 8
45219 Essen
Matrikelnummer: 3080384

Essen, den 5. September 2023

Betreuung:	Wilhelm Koop, Sascha Feldmann
Erstgutachter:	Prof. Dr. Volker Gruhn
Zweitgutachter:	Prof. Dr. Klaus Pohl
Studiengang:	Angewandte Informatik - Systems Engineering (B. Sc.)
Semester:	10

Abstract

TODO: Zusammenfassung auf Englisch

Zusammenfassung

Softwareentwickler müssen sich bei ihrer Arbeit Informationen zusammensuchen, welche sie für die weitere Arbeit benötigen. So muss ein Softwareentwickler bei der Implementierung eines Features die Intention des Features kennen. Solche Informationen können in Wissensdatenbanken hinterlegt sein. Um dort die gewünschten Informationen zu finden können Suchfunktionen verwendet werden. Insbesondere, wenn dem Softwareentwickler im Vorfeld nicht klar ist, wo er die gewünschten Informationen in der Wissensdatenbank finden kann. Aber nicht immer liefert diese Suchfunktion die gewünschten Informationen.

In dieser Arbeit wird eine semantische Suchfunktion entwickelt. Diese soll eine Verbesserung zu bestehenden Suchfunktionen bieten, wie beispielsweise die Suchfunktion von Confluence. Um festzustellen, ob die neue Suchfunktion *besser* ist als die bestehende, werden Methoden und Kriterien zur Evaluierung von Suchfunktionen erörtert. Anhand dieser Kriterien werden die neu implementierte Suchfunktion und die bestehende Suchfunktion von Confluence in einer Studie verglichen. In der Studie werden beispielhafte Sucheingaben definiert, sowie die erwarteten Ergebnisse. Die Definition der Sucheingaben erfolgt auf Basis von Anwendungsfällen. Die Anwendungsfälle beschreiben Situationen, in denen es realistisch ist, dass ein Softwareentwickler die Suchfunktion einer Wissensdatenbank verwendet. Anhand von Argumentationen werden entsprechend realistische Sucheingaben erstellt.

TODO: Ergebnis der Studie darstellen

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

NLP Natural Language Processing

1. Einleitung

Für die tägliche Arbeit benötigt ein Softwareentwickler Informationen, welche über den Code, an dem er arbeitet, hinausgehen. Um an diese Informationen zu kommen kann der Softwareentwickler eine Person suchen, welche ihm die gewünschte Information geben kann. Diese Person kann er im Projekt finden, oder über Websites, wie StackOverflow. Darüber hinaus wird in vielen Projekten eine Wissensdatenbank angelegt, welche Informationen enthält, welche spezifisch für das Projekt sind. Eine solche Wissensdatenbank ist Confluence. Sie bietet eine Suchfunktion, welche es dem Softwareentwickler erleichtern soll, die gewünschten Informationen zu finden. Beispiele für Informationen sind die Spezifikationen oder Dokumentationen eines Teiles der Software, mit welcher der Softwareentwickler gerade arbeitet. Oder aber auch Best-Practices, Guides, oder Informationen darüber, wie die Software gestartet oder ausgeliefert wird. Auch Informationen über den Projektplan sind für einen Softwareentwickler von Bedeutung. Aber nicht immer finden Softwareentwickler die gewünschten Informationen mithilfe der Suchfunktion. Ziel dieser Arbeit soll es sein, mithilfe von Wissen über Suchalgorithmen und Algorithmen aus dem Natural Language Processing zu zeigen, wie sich bestehende Suchfunktionen verbessern lassen.

Software, wie chatGPT[1] zeigt, dass Large Language Models eine valide Möglichkeit zur Information Extraction sind. Es ist denkbar, dass Suchfunktionen für Softwareentwickler durch Large Language Models verbessert werden können. Auch die Verwendung von vorgefertigten semantischen Netzen, welche in Knowledge Bases, wie DBPedia[2] zu finden sind, sind als Lösung für dieses Problem denkbar.[12] Sowohl die Verwendung von semantischen Netzen als auch die Verwendung von Vektordatenbanken kann als semantische Suche verstanden werden. Später soll noch einmal auf den genauen Unterschied zwischen den beiden Methoden eingegangen werden. In dieser Arbeit soll es lediglich um die Verwendung von Vektordatenbanken gehen.

1.1. Vorgehensweise

Die Gründe, warum eine gewünschte Information schwierig zu finden ist, sind vielzählig. Manchmal kennt der Softwareentwickler nicht das genaue Wording, um die gewünschten Informationen zu finden. Manchmal ist das Abstraktionslevel der gefundenen Informationen nicht das, welches sich der Softwareentwickler gewünscht hat. Beispielsweise, wenn eine allgemeine Definition von Domänenobjekten ge-

[1] https://openai.com/blog/chatgpt
[2] https://www.dbpedia.org/

sucht wird, aber eine Spezifikation eines Anwendungsfalls gefunden wird, in welchem das Domänenobjekt lediglich erwähnt wird. Es werden folgende Teilschritte durchlaufen, um die Suchfunktion von Wissensdatenbanken, wie Confluence, zu verbessern:

- **Definition von Anwendungsfällen:** Es werden zuerst Anwendungsfälle definiert. Damit wird das Problem der Qualität einer Suchfunktion heruntergebrochen in Teilprobleme. Die Anwendungsfälle beschreiben die konkreten Situationen, in welchen ein Softwareentwickler eine Suche nutzen könnte. Das hilft später dabei mehrere Suchfunktionen miteinander vergleichen zu können. Denn anhand der Anwendungsfälle können realistische Suchanfragen definiert werden. Diese Suchanfragen können an zwei verschiedene Suchfunktionen übergeben werden. Anschließend können die gefundenen Ergebnisse verglichen werden. Die genaue Vorgehensweise für diesen Vergleich wird in Kapitel 5 und Kapitel 6 erklärt. Außerdem bietet die Aufteilung in Anwendungsfälle bereits Aufschluss über die möglichen Verbesserungen, welche gemacht werden können. Hierauf wird in den Teilschritten "Erklärung des theoretischen Hintergrunds" und "Implementierung" weiter eingegangen.

- **Erklärung des theoretischen Hintergrunds:** Es wird die Theorie für die Implementierung einer Suchfunktion erläutert. Hier wird erklärt, welche Suchfunktionen es gibt. Außerdem werden Verfahren zur Indizierung von Dokumenten erklärt. Darüber hinaus werden NLP-Techniken erläutert, mit welchen Informationen aus gefundenen Dokumenten extrahiert werden können.

- **Implementierung:** Es wird eine neue Suchfunktion anhand der Informationen des theoretischen Hintergrunds implementiert. Es wird die Weaviate[3] Vektordatenbank verwendet, um Dokumente zu indizieren.

- **Herausarbeitung von Evaluationsmethoden und -Kriterien:** Es werden Methoden für die Bewertung herausgearbeitet. Damit wird die Frage beantwortet, wann eine Suchfunktion *gut* ist. Das hier erläuterte Wissen wird für die Durchführung der Studie benötigt.

- **Durchführung einer Studie:** Um festzustellen, ob die Implementierung eine Verbesserung darstellt, muss eine Studie durchgeführt werden. Aufgrund des Scopes der Arbeit wird nur eine rudimentäre Studie durchgeführt. Die Ergebnisse werden dargestellt und diskutiert. Dabei wird auch darauf eingegangen, an welchen Stellen die Studie weiter ausgearbeitet werden muss, um präzise Ergebnisse liefern zu können. Außerdem wird der Versuchsaufbau beschrieben und die gemessenen Daten. Die Ergebnisse werden interpretiert und es wird ein Schluss gezogen.

[3] https://weaviate.io/

1.2. Verwandte Arbeiten

Es gibt einige Arbeiten, welche die gleichen oder sehr ähnliche Probleme adressieren. Zum einen sind dies Arbeiten, welche verschiedene Arten von Suchfunktionen untersuchen. Andere Arbeiten untersuchen, wie sich die Qualität einer Suchfunktion messen lässt. Nochmals andere Arbeiten untersuchen Downstream Tasks von Suchfunktionen, also Algorithmen, welche das Ergebnis einer Suche verarbeiten, um die gleiche oder eine andere Anforderung an ein System umzusetzen.

So wird in *Automatic Query Reformulations for Text Retrieval in Software Engineering* von Haiduc et. al. ein System zur Verbesserung von Suchanfragen vorgeschlagen. Ausgangspunkt für das Paper ist das Problem der Traceability zwischen Code und anderen Softwareentwicklungs-Artefakten. Traceability bedeutet, dass sich von einer Stelle im Code, auf die entsprechenden Stellen in anderen Artefakten zurückschließen lässt. Ein Anwendungsfall für eine solche Traceability-Funktionalität ist "Feature Location", also das finden der Spezifikation eines Features, wenn nur der Code vorhanden ist. Das System, welches von Haiduc et. al. vorgeschlagen wird, verwendet Query Reformulations, um die Traceability herzustellen. Query Reformulation bedeutet, dass das System den Softwareentwickler bei der Eingabe einer Suchanfrage zur Suche nach den passenden Artefakten unterstützt. Dazu gibt der Softwareentwickler zunächst eine Suchanfrage ein, und markiert diejenigen Ergebnisse, welche am relevantesten für ihn sind. Auf Grundlage der gewählten Ergebnisse und mithilfe eines Machine Learning Algorithmus werden nun Vorschläge für eine verbesserte Suchanfrage gemacht. Dabei gibt es verschiedene Strategien. Wenn der Softwareentwickler zu Beginn eine sehr lange Suchanfrage eingegeben hat, dann kann das System eine Reduktion der Suchanfrage vorschlagen. Hat der Softwareentwickler dagegen lediglich einen Suchbegriff angegeben, so kann das System eine Erweiterung der Suchbegriffe vorschlagen. Dazu greift das System auf Synonyme des eingegebenen Suchbegriffes zurück.[8]

In dem Paper *From Word Embeddings To Document Similarities for Improved Information Retrieval in Software Engineering* von Ye et. al. wird beschrieben, wie Word Embeddings dazu verwendet werden können, um Traceability zwischen Code und anderen Softwareentwicklungs-Artefakten herzustellen. Word Embeddings sind eine Datenstruktur, welche einem Wort einen Vektor in einem n-dimensionalen Raum zuweist. Anhand dieses Vektors kann die Ähnlichkeit zwischen Wörter beschrieben werden. Ähnliche Wörter haben eine geringe Distanz im n-dimensionalen Raum. Unähnliche Wörter haben eine hohe Distanz. Der Algorithmus, welcher die Ähnlichkeit der Wörter bestimmt, macht Gebrauch von der Distributional Hypthesis. Dieser besagt, dass Wörter, welche im gleichen Kontext verwendet werden, eine ähnliche Semantik besitzen. Hermit wird also die Ähnlichkeit der Wörter bestimmt. Dieses Verfahren wird nun sowohl auf den Code angewendet als auch auf die Softwareentwicklungs-Artefakte.[16]

In dem Paper *Information Retrieval Models for Recovering Traceability Links*

between Code and Documentation verwenden Antoniol et. al. einen ähnlichen Ansatz, wie Ye et. al. Auch hier werden Word Embeddings verwendet um Softwareentwicklung Artefakte gegen den Code zu matchen. Hier durchlaufen die Artefakte und der Code zwei verschiedene Pipelines. Die Wörter der Artefakte in natürlicher Sprache werden in lowercase umgewandelt. Anschließend werden Stoppwörter entfernt. Zuletzt werden Flexionen entfernt. Aus dem Code werden zunächst Identifier extrahiert. Identifier, welche mehrere Wörter unter Verwendung von CamelCase oder snake_case beinhalten, werden in die einzelnen Wörter aufgeteilt. Anschließend werden die Identifier auf die gleiche Art und Weise normalisiert, wie die Wörter der Softwareentwicklungs-Artefakte. Dann erfolgt sowohl für die Identifier als auch für die Wörter aus den Artefakten die Indizierung, also die Umwandlung in Word Embeddings.[1]

In dem Paper *TaskNav: Task-based Navigation of Software Documentation* von Treude et. al. geht es um die Entwicklung einer Oberfläche, welche die Suche von *Tasks* ermöglicht. Dabei ist unter Task eine Operation im Code zu verstehen. Das Paper beschreibt einen Task als Verben, welche mit einem direkten Objekt oder einer Präposition in Verbindung stehen. Die Autoren nennen die Phrasen *get iterator* und *get iterator for collection* als Beispiele. Die Software analysiert nun die gesamte Dokumentation und extrahiert Tasks. Die Tasks werden in einen Index geschrieben, sodass der Softwareentwickler nach ihnen suchen kann. So wie die vorherigen Paper soll auch dieses Paper eine Brücke zwischen Dokumentation und Code schaffen.[15]

Das Paper *Estimating the recall performance of Web search engines* von Clarke und Willet misst die Qualität von Suchfunktionen des World Wide Webs anhand dessen Recalls. Um dies zu ermöglichen wird ein Datensatz generiert, welche Sucheingaben beinhaltet, sowie alle relevanten Dokumente für eine Sucheingabe.[5] Die gleichen Metriken werden auch in *Methods for measuring search engine performance over time* von Bar-Ilan verwendet.[2] Hier wird ebenfalls auf die Problematik der Messung von Recall eingegangen. Es wird erläutert, dass zur Messung des Recalls a-priori bestimmt werden muss, welche Dokumente als relevant für eine gegebene Sucheingabe erachtet werden sollten. In dem Paper wird eine Referenz genannt, welche behauptet, dass die Bestimmung der Relevanz lediglich dem Nutzer mit dem Bedürfnis nach der Information überlassen ist. Es wird eine weitere Referenz genannt, welche behauptet, dass die Bestimmung der Relevanz durch ein Experten-Panel durchgeführt werden sollte. *On Search Engine Evaluation Metrics* von Sirotkin betrachtet verschiedene Ansätze zur Messung der Performance von Suchfunktionen. Neben den bereits genannten Metriken von Precision und Recall werden andere Metriken, wie Mean Reciprocal Rank und Maximal Marginal Relevance.

Suchfunktionen sind Document Retrieval Systeme. Sie liefern Dokumente, welche zu der Sucheingabe des Nutzers passen. Document Retrieval Systeme sind eine Unterkategorie von Information Retrieval Systemen. Information Retrieval Systeme liefern auf Anfrage Informationen an den Nutzer. Im Fall einer Suchfunktion

werden Dokumente geliefert, welche diese Information beinhalten. Mithilfe der Dokumente ist der Ort, an dem sich die, vom Nutzer gewünschte, Information befindet eingegrenzt. Nichtsdestotrotz muss der Nutzer aus dieser Eingrenzung die gewünschte Information manuell extrahieren. Ganz im Gegensatz zu Question Answering Systemen. *A survey for Efficient Open Domain Question Answering* von Zhang et. al. untersucht verschiedene Herangehensweisen zur Implementierung von Open-Domain Question Answering Systemen.[17] Darüber hinaus schließt das Paper auf essenzielle Techniken für Open-Domain Question Answering. Open-Domain Question Answering Systeme beantworten allgemeine Fragen eines Nutzers, z.B. basierend auf Informationen von Wikipedia. Closed-Domain Question Answering Systeme beantworten dagegen Fragen im Kontext einer spezifischen Domäne, z.B. basierend auf unternehmensinternen Informationen.

TODO: Informationen lassen sich auch anreichern: NER etc. (NLP Techniken)

TODO: Informationen lassen sich aufbereiten: RAG

2. Definition von Anwendungsfällen

In diesem Kapitel sollen Anwendungsfälle ausgewählt werden, für welche später Lösungsansätze entwickelt werden. Die Anwendungsfälle beschreiben die Situationen, in denen Softwareentwickler die Suchfunktionen von Wissensdatenbanken verwenden könnten. Zur Identifikation von Anwendungsfällen wurde zunächst Literatur herangezogen. Die Literatur ist bereits unter den verwandten Arbeiten aufgeführt. In den Verwandten Arbeiten wurden Arbeiten genannt, welche ähnliche Probleme lösen sollen. Diese Fokussieren sich vorallem auf die *Feature Location*, *Bug Localization* und die Traceability zwischen Code und anderen Artefakten. Mit anderen Worten: Die Suchfunktion soll eine Brücke zwischen Code und Dokumentation herstellen. Neben den beiden Anwendungsfällen Feature Location und Bug Localization aus der Literatur, konnten noch weitere Anwendungsfälle ermittelt werden. So ist das Onboarding im Projekt ein Anwendungsfäll für die Verwendung der Suchfunktion einer Wissensdatenbank. Die weiteren Anwendungsfälle sind *Informationen über das Projektmanagement finden*, *Implementierung nach Spezifikation* und *Abgleich mit Spezifikation*.
Die Folgenden Kapitel beschreiben die genannte Anwendungsfälle und erläutern, warum sie als Anwendungsfälle ausgesucht wurden. Bei dem Vergleich zwischen Suchfunktionen werden die Anwendungsfälle herangezogen werden, um ein fiktionales Szenario zu erstellen. Das fiktionale Szenario bietet die Grundlage für den Vergleich zwischen Suchfunktionen. Denn durch dieses werden die Sucheingaben und die erwarteten Ergebnisse definiert, welche die Suchfunktionen liefern sollen.

2.1. Onboarding im Projekt

Wenn ein neuer Softwareentwickler in einem Softwareprojekt startet, dann muss er sich zunächst einmal mit dem Projekt vertraut machen. Das bedeutet, dass er verstehen muss, was das Projekt eigentlich ist. Er muss verstehen, was das eigentliche Problem des Kunden ist. Außerdem muss er verstehen wie die Software dieses Problem löst.

Die Leitung eines Projektes wünscht sich eine möglichst schnelle Einarbeitung von neuen Mitarbeitern. Dazu können bereits etablierte Mitarbeiter herangezogen werden, welche den neuen Mitarbeiter bei der Einarbeitung unterstützen. Der Nachteil besteht darin, dass hierdurch Kapazitäten gebunden werden, welche für die aktive Entwicklung der Software benötigt werden. Daher kann das Zurückgreifen auf eine Wissensdatenbank durch den neuen Mitarbeiter sinnvoll sein. Wenn der neue Mitarbeiter sich nun beim Lesen von Dokumenten in der Wissensdatenbank Fragen stellt, dann ist es hilfreich, wenn die Suchfunktion der Wissensdatenbank die Antworten auf diese Fragen liefern kann.

Dazu muss der Softwareentwickler sehr allgemeine Informationen über das Projekt finden können. Er könnte Dinge suchen, wie einen Projektüberblick oder ein Glossar. Neben diesen allgemeinen Informationen muss sich der neue Softwareentwickler mit dem Code vertraut machen. Er muss verstehen, welche Technologien verwendet werden, welche Best-Practices, Code-Styles, Guidelines, Prozesse und Quality Gates eingehalten werden müssen. Und er muss verstehen, wie die Software lokal oder in einer Testumgebung ausgeführt werden kann.

2.2. Implementierung nach Spezifikation

Bei der Implementierung von neuen Anforderungen ist es wichtig, dass sich der Softwareentwickler an die Spezifikation hält. Nur so bekommt der Kunde die Software, die er sich gewünscht hat. Dazu sollte der Softwareentwickler es schaffen alle relevanten Dokumente zu finden, die zu der Spezifikation dazugehören. Zuerst sollte er die Feature-Spezifikation selbst finden können. Er sollte die Dokumentation der damit einhergehendenden Prozesse finden, und auch die Domänenobjekte, welche bei der Implementierung relevant sein werden. Er sollte Diagramme finden können, welche zu dem Anwendungsfall gehören, und auch die weiteren Dokumente, welche den Kontext der Anforderung erläutern. Außerdem wäre es hilfreich für den Softwareentwickler auch gleich die relevanten Stellen im Code angezeigt zu bekommen.

Feature Location ist das Vorgehen, den Einstiegspunkt im Code zu finden, welcher für ein Feature verantwortlich ist. Wenn eine Software das Feature besitzt Mails zu verschicken, dann könnte der Einstiegspunkt für dieses Feature eine Schnittstelle sein, welche den Inhalt und Empfänger der Email entgegennimmt. Der Inhalt wird anschließend in ein Mail-Template eingefügt, und die Mail wird verschickt.

Der Abgleich mit einer Spezifikation ist notwendig, um Testfälle zu schreiben, und zu prüfen, ob das Verhalten der Anwendung korrekt ist. Natürlich gibt es Arten von Fehlern, welche erkennbar sind, ohne dafür die Spezifikation heranzuziehen. Wenn in einem Online-Shop der Preis für ein Produkt in Amerika bei 5$ liegt, aber in Deutschland der Preis bei 1.000€ liegt, dann braucht es nicht die Spezifikation, um festzustellen, dass es bei der Umrechnung von Dollar zu Euro einen Fehler gegeben hat. Aber nicht alle Fehler sind so offensichtlich. Die Spezifikation zum Abgleich mit dem Verhalten der Software heranzuziehen ist besonders in Randbedingungen hilfreich. Angenommen ein Online-Shop bietet einen kostenlosen Versand ab einem Mindestbestellwert an. Sobald der Preis den Wert von 25$ übersteigt ist der Versand gratis. Der Versand kostet im Normalfall 5$. Nun muss definiert werden, ob der Versand in dem Mindestbestelltwert miteinbezogen wird oder nicht. Wird er miteinbezogen, dann sorgt das dafür, dass ein Produkt,

welches 20$ kostet einen kostenlosen Versand hat, weil der Mindestbestellwert zuzüglich dem Versandpreis 25$ beträgt. Ob dieses Verhalten gewünscht ist oder nicht, muss in der Spezifikation festgehalten werden.

Wenn nun ein Bugticket dieser Art bei einem Softwareentwickler landet, dann muss er, wie auch bei der Implementierung, alle relevanten Dokumente heranziehen.

2.3. Bug Localization

Neben der Implementierung neuer Features ist es Aufgabe eines Softwareentwicklers bestehende Fehler in der Software zu korrigieren. Um einen Fehler zu korrigieren, ist es sinnvoll die Spezifikation heranzuziehen, um herauszufinden, wie sich die Software ordnungsgemäß verhalten sollte. TODO: Neu schreiben

Wenn der Softwareentwickler nun ein Fehlerticket erhält, dann muss er die entsprechende Spezifikation zu diesem Fehlerticket finden können. Und idealerweise wird ihm durch die Suche sogar gleich die betroffene Stelle im Code angezeigt. Das ist Bug Localization.

2.4. Informationen über das Projektmanagement finden

Softwareentwickler sollten darüber Bescheid wissen, was in dem Projekt, in dem sie arbeiten, vor sich geht. Sie sollten über organisatorische Dinge Bescheid wissen, wie die Teamaufteilung und die Zuständigkeiten in den einzelnen Teams und in dem gesamten Projekt. Das ist wichtig für eine gute Kommunikation und entsprechenden Wissensaustausch zwischen Kollegen. Außerdem sollten Softwareentwickler darüber Bescheid wissen, wann das nächste Release der Software ansteht. Diesen Termin müssen sie bei der Planung ihrer Arbeit berücksichtigen, damit sie auch rechtzeitig alle relevanten Features implementiert haben und alle kritischen Fehler behoben haben. Neben der Einhaltung von Terminen ist die Planung wichtig für die Motivation der Softwareentwickler. Denn Motivation entsteht durch die Wahrnehmung, sich einem Ziel zu nähern (Quelle: Flow - mihaly csikszentmihaly). Aus diesem Grund müssen die Softwareentwickler auch die Vision der Software verstehen und auch die übergreifende Vision des Unternehmens. Sie müssen verstehen, warum die Arbeit, welche sie erledigen so wichtig ist. Und sie müssen verstehen, für wen sie diese Arbeit tun. Auch diese Faktoren sind wichtig für eine hohe Motivation bei der Arbeit. Daher ist es so wichtig diese Informationen einfach zugänglich zu machen, also einfach auffindbar zu machen.

Im Kapitel "Evaluationsmethoden und -Kriterien" wird aufgezeigt, wie die Anforderungen der Anwendungsfälle quantifizierbar gemacht werden können. Das wird dabei helfen nachzuvollziehen, inwieweit die Anforderungen der Anwendungsfälle erreicht wurden, welche genannt wurden.

3. Konzeption der Suchfunktion

Eine einfache Implementierung einer Suchfunktion kann aus drei Komponenten bestehen. Aus dem Crawling, dem Index und dem Suchalgorithmus. Das Crawling ist zuständig für das Finden von Dokumenten[3]. Der Index speichert Informationen der Dokumente, und die Suche ist zuständig für das Verstehen der Nutzeranfrage und die Abfrage der relevantesten Informationen aus dem Index, sowie dessen Verarbeitung und Darstellung. Der Begriff Dokument wird hier verwendet, um die Dateien zu beschreiben, welche durch einen Crawler gesucht und durch den Index verarbeitet werden. Unter Dokumenten können auch eine Website verstanden, welche durch einen Webcrawler durchsucht werden. Es wird im Folgenden zunächst die Funktionsweise des Crawlings erläutert. Danach werden zwei Möglichkeiten der Indizierung beschrieben. Zuerst die Volltext-Indizierung, dann die Vektor-Indizierung. Dann werden die verschiedenen Arten von Suchalgorithmen erläutert. Und zuletzt wird auf Grundlage der dargestellten Informationen eine Konzeption zur Implementierung einer neuen Suchfunktion hergeleitet.

3.1. Crawling

Für die Implementierung einer Suchfunktion wird zunächst ein Datensatz von Dokumenten benötigt, welche über die Suchfunktion gefunden werden können. Dieser wird mithilfe eines Crawlers aufgebaut. Ein Crawler ist ein Algorithmus, welcher Techniken aus dem Natural Language Processing nutzt, um Informationen aus einem Dokument zu extrahieren.[10] Implementierungen können reguläre Ausdrücke verwenden, um die Informationen zu extrahieren, oder auch fortgeschrittenere Verfahren, wie Abstract Syntrax Trees. Im Falle von Websites kann der Crawler Hyperlinks zu weiteren Websites extrahieren. Damit kann der Algorithmus sukzessive den Datensatz von Dokumenten befüllen. Die neuen Dokumente werden durch den Index verarbeitet und wiederum auf neue Links analysiert. Dieses Verfahren kann beliebig lange und beliebig rekursiv durchlaufen werden, um den Index zu erweitern. Neben dem Crawling können Indizes befüllt werden, indem eine Liste von Dokumenten übergeben werden, welche dem Index hinzugefügt werden sollen.

3.2. Indizierung

Die Indizierung von Dokumenten kann sowohl mit einer Volltext-Indizierung oder auch einer Vektor-Indizierung umgesetzt werden. Ein Volltextindex bestimmt den Score anhand von Ausschnitten aus dem Volltext. Wenn diese bestimmte Kriterien erfüllen, dann wird das Dokument bei der Suche gefunden, andernfalls nicht.

Ein Vektorindex berechnet Vektoren anhand des Ursprungstextes. Ein Vektorindex wird auch als Vector Space Model (VSM) bezeichnet. Die Berechnung der Vektoren kann mithilfe von tf-idf, BM25, word2vec, Latent Semantic Embeddings oder auch mit Transformers erfolgen.

Das Ziel der Indizierung ist für jeden Algorithmus der gleiche: Für eine gegebene Sucheingabe sollen die relevantesten Dokumente gefunden werden. Die Algorithmen unterscheiden sich darin, wie sie diese Relevanz berechnen. Sie werden im Folgenden näher erläutert.

3.2.1. Volltext-Indizierung

Eine Möglichkeit zur Umsetzung einer Volltext-Indizierung ist der invertierte Index. Die Dokumente werden in einer Datenbank abgelegt. Anschließend wird der Index aufgebaut, indem alle Wörter extrahiert werden, welche in den Dokmenten vorkommen. Nun werden diese Wörter in eine Liste geschrieben, und jedem Wort wird zugeordnet, in welchem Dokument sich dieses Wort wiederfinden lässt. Diese Liste wird invertierter Index genannt, weil nicht die Wörter den Dokumenten zugeordnet sind, sondern die Dokumente den Wörtern. Es wird ebenfalls gespeichert, an welchen Stelle des Dokuments das Wort vorkommt, und auch in wie vielen Dokumenten ein Wort vorkommt.

Bei der Indizierung der Wörter besteht nun die Problematik, dass gleiche Wörter in unterschiedlichen Formen existieren können. So stammen *Heizung* und *heizen* beide von dem gleichen Wortstamm *heiz* ab. Um bei der Indizierung Speicherplatz zu sparen, können Wörter auf diesen Wortstamm reduziert werden, damit sie als ein einziges Wort betrachtet werden können. Die Bildung des Wortstamms wird als Stemming bezeichnet. Beim Stemming kann es jedoch zu Overstemming und Understemming kommen. Overstemming bedeutet, dass zwei Wörter, die eigentlich nichts miteinander zu tun haben, also nicht semantisch gleich sind, den gleichen Wortstamm besitzen und als ein Wort betrachtet werden. Ein Beispiel hierfür sind die Wörter *Wand* und *wandere*, wie in *ich wandere*. Beide besitzen den Wortstamm *wand* und werden entsprechend als ein Wort betrachtet. Understemming bedeutet, dass zwei Wörter, die eigentlich etwas miteinander zu tun haben, also semantisch gleich sind, nicht den gleichen Wortstamm besitzen und dadurch als zwei verschiedene Wörter betrachtet werden. Ein Beispiel hierfür sind die Wörter *absorbieren* und *Absorption*, welche die Wortstämme *absorb* und *absorp* besitzen. Es gibt Techniken zur Vermeidung solcher Probleme, wie der Einsatz vollständiger morphologischer Analysekomponenten. Hierauf soll aber nicht weiter eingegangen werden.

3.2.2. Vektor-Indizierung

Neben einer Volltext-Indizierung können Dokumente in Form von Vektoren indiziert werden. Dazu werden Dokumente zunächst, wie auch bei der Volltext-

Indizierung gecrawlt. Anschließend durchlaufen die Inhalte der Dokumente ein Preprocessing. Dieses kann je nach Implementierung variieren. Das Kapitel *Einspielen der Daten in Weaviate* beschreibt, wie in der hier aufgeführten Implementierung das Preprocessing durchgeführt wird. Das Preprocessing hat den Zweck die Daten an das Schema der Datenbank anzupassen und die Qualität der Daten zu erhöhen. Außerdem sorgt es für eine kürzere Indizierungszeit.

Nach dem Preprocessing werden durch einen Transformer für die Inhalte der Dokumente Vektoren berechnet. Ein Transformer wird mithilfe von Trainingsdaten darauf trainiert, Vektoren für Wörter zu generieren. Das trainierte Modell wird nach dem Preprocessing durchlaufen. Zuletzt werden die Daten in einer Vektordatenbank gespeichert. Eine Vektordatenbank speichert Daten, wie eine dokumentenbasierte Datenbank. Dort werden nun sowohl die rohen Daten als auch die Vektoren gespeichert, welche von dem Transformer berechnet wurden. Die Vektoren haben den Vorteil, dass die Daten in der Datenbank nicht linear gespeichert sind. Sie sind in einem n-dimensionalen Raum gespeichert, mit dessen Hilfe die semantische Nähe zwischen Dokumenten ausgedrückt werden kann. Das funktioniert auf die gleiche Weise, wie bereits in dem Kapitel *Semantic Search* beschrieben.

3.2.3. Scoring Algorithmen

TODO: Bag of words

TODO: Überarbeiten Zur Implementierung eines Volltext-Index werden Dokumente und Wörter in einer m x n Matrix angeordnet, wobei m die Anzahl der Dokumente ist und n die Anzahl der Wörter. Die Werte in dieser Matrix werden anhand einer ausgewählten Metrik bestimmt. Eine oft verwendete Metrik ist tf-idf.

$tf - idf(t, D) = tf(t, D) * idf(t)$
$tf(t, D) = \frac{\#(t,D)}{\max_{t' \in D} \#(t',D)}$
$idf(t) = \log \frac{N}{\sum_{D:t \in D} 1}$
TODO: Quelle

Der tf-Teil steht für *term frequency* und wird berechnet, indem für das jeweilige Dokument bestimmt wird, wie häufig das Wort in dem Dokument vorkommt. Damit die Metrik nicht abhängig von der Länge des Dokumentes ist, wird dieser Wert durch die insgesamte Anzahl der Vorkommnisse des Wortes dividiert. Damit ist diese Metrik relativ zur insgesamten Anzahl der Vorkommnisse des Wortes, und nicht absolut. Der idf-Teil steht für *inverse document frequency*. Er wird berechnet indem die insgesamte Anzahl der Dokumente durch die Anzahl der Dokumente dividiert wird, welche das Wort enthalten. Das Ergebnis des dividierens wird an die Logarithmus-Funktion übergeben, sodass am Ende der idf-Wert berechnet wurde. Die beiden Werte werden miteinander multipliziert. Das Ergebnis ist die tf-idf-Metrik.

Wenn das Wort charakteristisch für ein Dokument ist, dann kommt es in diesem Dokument häufig vor, aber in anderen Dokumenten nur selten. Wenn das Wort

nicht charakteristisch für das Dokument ist, dann kommt es in anderen Dokumenten genauso häufig oder häufiger vor als in diesem Dokument.

Der BM25 Algorithmus ist eine Erweiterung der tf-idf Metrik. TODO: BM25

3.3. word2vec

TODO: word2vec erläutern

3.4. LSE

TODO: Was sind Latent Semantic Embeddings

Wenn ein Softwareentwickler ein bestehendes Feature anpassen muss, dann muss er zunächst den Einstiegspunkt im Code kennen. Sind keine weiteren Informationen vorhanden, dann muss der Softwareentwickler dazu den Code manuell durchsuchen. Mithilfe von Techniken aus dem Natural Language Processsing, wie Latent Semantic Indexing (LSI), Latent Dirichlet Allocation (LDA) und Vector Space Models kann dieser Prozess automatisiert werden.[7] Mithilfe eines gemeinsamen Index von den Inhalten der Wissensdatenbank und dem Quellcode lässt sich ein Zusammenhang zwischen Feature-Spezifikation und Klassen im Quellcode herstellen. Die Herangehensweise erfordert eine Möglichkeit beide Datenquellen auf die gleiche Art und Weise zu indizieren. Dazu werden LSI und Vector Space Models verwendet.[1]

3.5. Transformers

TODO: transformers erläutern

3.6. Suchalgorithmen

Im Folgenden werden zuerst gängige Suchalgorithmen erklärt, welche in nahezu jeder Suchfunktion zu finden sind. Danach wird im speziellen auf die strukturierte Suche und die semantische Suche eingegangen. Die semantische Suche wird später bei der Implementierung einer Suchfunktion verwendet werden.

3.6.1. Die gängigen Suchalgorithmen

In dem Kapitel *Evaluationsmethoden und -Kriterien* wurde bereits beschrieben, dass eine Suche das Qualitätskriterium der Konformität erfüllen sollte. Das bedeutet, dass eine Suchfunktion die gängigen Arten von Suchanfragen unterstützen muss. Dieses Kapitel soll die Arten von Suchanfragen vorstellen. Es ist wichtig diese zu kennen, um das Qualitätskriterium später bei der Implementierung erfüllen zu können. Es gibt die gängigen Suchalgorithmen Keyword Search, Phrase Search,

Boolean Search, Field Search.

Eine Keyword Search durchsucht Dokumente nach der Sucheingabe des Nutzers. Die Eingabe wird dabei nicht als ganzes betrachtet, sondern jedes Wort einzeln. Für jedes Keyword werden Dokumente als Ergebnis angezeigt, wenn dieses in dem Dokument vorhanden ist. Wenn ein Dokument mehrere der Keywords beinhaltet, wird dessen Relevanz höher eingeschätzt als für Dokumente, welche weniger Keywords enthalten. Dokumente mit höherer Relevanz werden weiter oben in der Ergebnisliste angezeigt.

Eine Phrase Search ist die Suche nach Textausschnitten in Dokumenten. Hier werden nicht mehrere Keywords einzeln betrachtet, sondern die gesamte Eingabe in das Suchfeld als eine Einheit. Es reicht also nicht mehr aus, dass ein Dokument eines der Wörter enthält. Es muss die gesamte Sucheingabe als ein String enthalten sein.

Die Boolean Search bietet die Möglichkeit einen boolschen Ausdruck als Sucheingabe zu machen. Ein Beispiel dafür ist die Sucheingabe *Dokumentation AND Angular*. Die Sucheingabe bedeutet, dass die Suchfunktion nur Dokumente als Ergebnis darstellen soll, welche beide Keywords Dokumentation und Angular enthalten. Die Boolean Search kann auch eine Phrase, wie bei der Phrase Search, beinhalten: *"Dokumentation von Software" AND Angular*. In diesem Beispiel werden nur Dokumente als Ergebnis nur angezeigt, wenn sie den gesamten String *Dokumentation von Software* enthalten, sowie das Keyword *Angular*. Bei einer Boolean Search können die boolschen Operatoren *AND*, *OR*, *NOT* beliebig kombiniert werden.

Eine Field Search sucht Dokumente anhand von Attributen. Der Nutzer kann diese Attribute auswählen. Wenn der Nutzer beispielsweise ein Dokument sucht, welches am 01.01.2005 erstellt wurde, dann kann die Eingabe der Suche so aussehen: *erstelldatum: 01.01.2005*. Es können beliebig viele Attribute verwendet werden, um die Suche einzugrenzen. Neben der Verwendung der Attribute für die Suche selbst, können die Attribute komplementär zu einer anderen Art von Suche verwendet werden. So kann eine Suchfunktion Buttons bereitstellen, über welche Filter festgelegt werden. Nun kann eine Keyword Search durchgeführt werden, aber die gefundenen Dokumente werden mithilfe der Filter weiter eingeschränkt. Neben diesen gängigen Suchalgorithmen gibt es weitere Suchalgorithmen, wie die strukturierte Suche und die semantische Suche.

3.6.2. Die strukturierte Suche

Eine klassische Suchfunktion verwendet Keywords, um relevante Dokumente für eine Sucheingabe zu ermitteln. Der Vorteil dieser Art von Suche ist, dass die technische Struktur, in der die Daten vorliegen und gespeichert sind, nicht bekannt sein müssen. Der Nachteil ist auf der anderen Seite, dass die Suchergebnisse unpräzise

sein können. Wenn beispielsweise der Suchbegriff Kamera eingegeben wird, dann werden Kameras als Ergebnisse zurückgegeben. Soll die Kamera nun bestimmte Eigenschaften besitzen, dann müssen diese Eigenschaften ebenfalls als Keywords angegeben werden. Nun wird aber nicht die Suche auf Ergebnisse eingegrenzt, bei denen eine Kamera diese bestimmten Eigenschaften besitzt. Stattdessen werden Suchergebnisse angezeigt, bei denen einige dieser Keywords vorkommen. Demgegenüber stehen Datenbankabfragen, beispielsweise mithilfe von SQL. Bei einer Datenbankabfrage können Objekte abgefragt werden, dessen Eigenschaften ganz bestimmte Werte haben. Die Ergebnisse, die eine solche Abfrage zurückgibt, sind dabei vollkommen genau. Es werden keine Objekte zurückgegeben, welche diese Kriterien nicht erfüllen. Voraussetzung für eine solche Suchabfrage ist allerdings, dass die Struktur der Datenbank a priori bekannt ist. Dem Nutzer muss der Name der Datenbank, der relevanten Tabellen, sowie der relevanten Properties bekannt sein, damit er das passende SQL für die Datenbankabfrage schreiben kann. Strukturierte Suchen sollen die Vorteile beider Vorgehensweisen kombinieren. Die Struktur der Daten soll a priori nicht bekannt sein müssen, aber trotzdem sollen die Suchergebnisse vollkommen präzise sein. TODO: Wie wird das ermöglicht?

3.6.3. Die semantische Suche

Unter einer semantischen Suche wird im Allgemeinen verstanden, dass die Suche nicht nur eine syntaktische Suche einer Zeichenkette ist, sondern auch Techniken, wie technische Analysen verwendet, um die Bedeutung der Sucheingabe nachzuvollziehen.[6] Eine semantische Suche hat den Zweck, die Ähnlichkeit und Beziehungen zwischen Wörtern zu verstehen. Sie kennt Homonyme, Synoyme und Antonyme von Wörtern. So wird durch sie beispielsweise die Ähnlichkeit von den Wörtern *rollout deployment* abgebildet, und dass diese Wörter oft im gleichen Kontext verwendet werden.

Die technische Umsetzung einer semantischen Suche kann auf verschiedene Arten erfolgen. Zum einen besteht die Möglichkeit ein explizites semantisches Netz heranzuziehen, und so die Zusammenhänge von Wörtern abzubilden. Ein semantisches Netz ist eine Menge aus Aussagen in der Form *Subjekt Prädikat Objekt*. Anhand dieser Aussagen wird ein Graph aus Beziehungen zwischen Wörtern hergestellt.[11] TODO: Erläuterende Grafik + Beispiel.
Das hat den Vorteil, dass ein solches explizites semantisches Netz von Menschen erstellt wurde, und damit eine hohe Qualität der Daten einhergeht. Denn an der Erstellung von semantischen Netzen sind mehrere Menschen beteiligt, welche zuerst einen Konsens über die Eigenschaften und Zusammenhänge von Wörtern schaffen müssen. Der Nachteil dieser Methode ist der große Arbeitsaufwand, welcher mit der Erstellung eines solchen semantischen Netzes einhergeht. Ein weiterer Nachteil ist die mögliche Unvollständigkeit, welche ein solches semantisches Netz besitzen könnte. Eine Wissensdatenbank, welche für ein Projekt erstellt wurde, kann Definitionen von Begriffen beinhalten, welche in allgemeinen semantischen Netzen nicht vorhanden sind. Bei der Umsetzung einer semantischen Suche mithil-

fe eines semantischen Netzes müsste also zuerst ein allgemeines semantisches Netz herangezogen werden, beispielsweise von DBPedia. Anschließend müsste dieses semantische Netz um die neuen Definitionen, welche nur im Kontext des Projektes gelten, erweitert werden.

Eine weitere Möglichkeit zur Umsetzung einer semantischen Suche ist die Verwendung von Transformers und Vektordatenbanken. Um zu verstehen, welche Wörter kontextuell zusammengehören, werden hier die Wörter in einem n-dimensionalen Raum positioniert. Wörter, die sich sehr ähnlich sind, also im gleichen Kontext verwendet werden, haben in diesem n-dimensionalen Raum eine geringe Distanz zueinander. Wörter, die sich eher unähnlich sind, wie *"rollout" und "API"*, haben eine größere Distanz. Der Vorteil einer semantischen Suche ist, dass der genaue Begriff, welcher gesucht wird nicht bekannt ist. Wenn sich der Nutzer also über ein Thema informieren möchte, mit welchem er nicht gut vertraut ist, dann kann die semantische Suche hilfreich sein. Denn der Nutzer kann nun einen Begriff eingeben, der zu dem Thema passt, und den er kennt. Er findet anschließend Dokumente, welche vielleicht nicht genau diesen Begriff beinhalten, aber welche thematisch dennoch ähnlich sind. Genau dieser Vorteil soll bei der Implementierung später genutzt werden.

Um eine semantische Suche zu implementieren, werden die Technologien von Transformern und Vektordatenbanken verwendet. Ein Transformer erhält als Input eine große Menge an Text und mappt die einzelnen Wörter auf einen Vektor einer beliebigen Länge. Der Vektor, der am Ende herauskommt, beschreibt die Position des Wortes in dem n-dimensionalen Raum. Der Vektor beschreibt gewissermaßen, wie stark ein Wort in eine abstrakte Kategorie einzuordnen ist. Jeder Wert im Vektor entspricht einer Kategorie. Mithilfe der Vektoren können verschiedene Wörter hinsichtlich ihrer Ähnlichkeit analysiert werden. Ähnliche Wörter habe eine große räumliche Nähe, während zwei Wörter, die in vollkommen unterschiedlichen Kontexten verwende werden eine sehr große Distanz im Raum besitzen. Nehmen wir für ein Beispiel einen dreidimensionalen Raum an. Die X-Achse ist beschriftet mit dem Wort „Tier“, die Y-Achse ist beschriftet mit dem Wort „Computer“ und die Z-Achse ist beschriftet mit dem Wort „Mensch“. Nun geben wir einem Transformer das Wort „Katze“, und der Transformer berechnet einen dreidimensionalen Vektor, welcher das Wort „Katze“ im Raum positioniert. Weil eine Katze ein Tier ist, ist der X-Wert des Vektors eins. Der Wert eins bedeutet, dass das Wort vollständig zu dieser Kategorie gehört. Da eine Katze überhaupt nichts mit einem Computer zu tun hat, ist der Y-Wert des Vektors 0.

Nun ist eine Katze kein Mensch, aber eine Katze ist ein Haustier von Menschen. Es ist denkbar, dass die Wörter Katze und Mensch oft im gleichen Kontext verwendet werden, sodass der Wert bei 0,3 liegen könnte. Damit der Transformer einen Vektor berechnen kann, braucht er eine Menge Daten. Diese Daten erhält er aus vielen Texten. Werden zwei Wörter oft im gleichen Text genannt oder kommen zwei Wörter in vielen Texten sehr nahe beieinander vor, dann geht der Trans-

former davon aus, dass die beiden Wörter ähnlich sind, und berechnet ähnliche Vektoren. Zuvor müssen die Texte allerdings bereitgestellt werden. Dazu kann beispielsweise das Internet gecrawlt werden. Die Ergebnisse des Transformers werden in einer Vektordatenbank gespeichert. Eine Vektordatenbank ist eine Datenbank, welche Vector Embeddings, also ein Objekt als Key und dessen Vektor als Value speichert. Bei dem Objekt kann es sich um Wörter handeln, dann wird auch von Word Embeddings gesprochen. Es können aber auch Daten andere Daten, wie Bilder, Videos oder Audio gespeichert werden. Der Zweck von Vektordatenbanken ist es, Daten nicht einfach linear zu speichern, sondern in einem Raum. Die Distanz zwischen zwei Einträgen in diesem Raum beschreibt dessen Ähnlichkeit. Genau diese Informationen nutzen semantische Suchen.

3.7. Wahl der Suchfunktion

Die vergangenen Kapitel haben verschiedene Indizierungs- und Suchalgorithmen dargestellt. Jeder dieser Algorithmen hat Vor- und Nachteile. Diese werden in diesem Kapitel betrachtet, um die Entscheidung für die verwendeten Technologien der neuen Suchfunktion nachvollziehbar zu machen.

Die Confluence Suche verwendet laut Dokumentation Apache Lucene.[1] Laut der Dokumentation von Apache Lucene, verwendet dieses einen Scoring Algorithmus, welcher sowohl auf VSM als auch auf Boolean Models basiert.[2] Im Information Retrieval sind Boolean Models jene Scoring Algorithmen, welche auf boolscher Algebra basieren. Aus der Dokumentation von Apache Lucene und aus dem genannten Paper geht ebensfalls hervor, dass die Vektoren des VSM mit tf-idf berechnet werden.

In einem Paper von Choudhary et. al. wird ein Document Retrieval System entwickelt.[4] Das Document Retrieval System verwendet Bert zur Generierung von Embeddings. Der Scoring Algorithmus des Systems kombiniert Bert und tf-idf, um den Score zu berechnen. Laut dem Paper bietet eine Kombination aus tf-idf und Bert zur Implementierung eines Document Retrieval Systems signifikante Performance Verbesserungen gegenüber einem Document Retrieval System, welches lediglich tf-idf verwendet. Die Performance Verbesserung wird in dem Paper an dem MS Marco Datensatz gemessen.[3]

In einem Paper von Karpukhin et. al. wird Open-Domain Question Answering untersucht.[9] Dazu wird ein Dense Passage Retriever entwickelt. Ein Dense Passage Retriever bestimmt den Textabschnitt eines gegebenen Textes, welcher mit größter Wahrscheinlichkeit eine Antwort auf eine gestellte Frage beinhaltet. Der Input für einen Dense Passage Retriever sind Dokumente, welche zuvor mithilfe eines Scoring Algorithmus aus einem Index extrahiert wurden. Das Paper zeigt,

[1] https://confluence.atlassian.com/doc/ranking-of-search-results-1188406620.html
[2] https://lucene.apache.org/core/2_9_4/scoring.html
[3] https://microsoft.github.io/msmarco/

dass das entwickelte Passage Retrieval System besser darin ist relevante Textabschnitte zu finden als der BM25 Algorithmus. Das Paper nennt die semantische Verknüpfung von Synonymen und Paraphrasierungen mit unterschiedlichen Tokens als Vorteil gegenüber BM25 und auch TF-IDF.

Die semantische Suche ist bei dem Einspielen der Daten in die Datenbank langsamer, weil zunächst die Vektoren für die Daten generiert werden müssen. Die Zeit für die Abfrage von Daten aus einem Vector Space Model wächst mit wachsender Anzahl von Datenpunkten nicht so schnell an, wie bei einem Volltext-Index. Bei einem Volltext-Index müssen die übergebenen Keywords mit jedem Datensatz abgeglichen werden. Bei einem Vector Space Model sind zwar mit steigender Zahl von Datensätzen mehr Punkte im Raum vorhanden, aber es müssen nicht für jeden Punkt Strings verglichen werden. Der Algorithmus, welcher für die Ermittlung der passendsten Datenpunkte im Vector Space Model verwendet wird, lautet Approximate Nearest Neighbor (ANN). Dieser basiert auf linearer Algebra. TODO: Erklären, warum ANN schneller sein soll

TODO: Warum keine strukturierte Suche? TODO: Was gäbe es noch für Ansätze?

Bei der Implementierung sollen dabei keine semantischen Netze verwendet werden, sondern ein Vector Space Model. TODO: Warum?

Neben den direkten Vorteilen einer semantischen Suche gegenüber Suchfunktionen, wie eine Keyword Search, gibt es weitere indirekte Vorteile eine semantischen Suche auf Grundlage von Vector Space Models. Vector Space Models sind Multimodal. Es können also in einem Vector Space Model nicht nur Informationen über ein Medium, wie Fließtext, gespeichert werden, sondern neben Fließtext können gleichzeitig auch andere Medien, wie Code, Bilder, Audiodateien etc. gespeichert werden. Und anders als bei einer gewöhnlichen Datenbank, in welcher mehrere verschiedene Medien gespeichert werden können, beispielsweise eine relationale Datenbank, können Vector Space Models diese verschiedenen Medien miteinander semantisch verknüpfen.

Indem die Codebase, welche zu der Wissensdatenbank in Verbindung steht, ebenfalls indiziert wird, kann eine Traceability zwischen Code und Wissensdatenbank hergestellt werden. Damit können Feature Location und Bug Localization umgesetzt werden. So kann die Suchfunktion nicht nur zum Durchsuchen der Wissensdatenbank verwendet werden, sondern auch zum Durchsuchen der Codebase. Und wenn in der Suchfunktion nach einem Feature gesucht wird, dann kann neben der Spezifikation in der Wissensdatenbank auch gleich der Einstiegspunkt im Code gefunden werden. Damit verbessert sich wiederum die Wartbarkeit der Software, da das Ändern von Features oder korrigieren von Bugs erleichtert wird. TODO: Quellen

3.8. Retrieval Augmented Generation

Neben einer Suchfunktion können System entwickelt werden, welche eine Question-Answering Funktionalität bieten. Eine Question-Answering Funktionalität nimmt eine Frage des Nutzers entgegen, und liefert eine Antwort. Solche Systeme basieren auf Retrieval Augmented Generation. Das bedeutet, dass auf Grundlage der Frage des Nutzers zuerst eine Suche durchgeführt wird. Diese Suche liefert relevante Dokumente zum Beantworten der Frage. Nun werden die, für die Frage relevantesten, Stellen der Dokumente mithilfe eines Large Language Models extrahiert und umformuliert. Das Ergebnis ist eine Antwort auf die Frage des Nutzers.

4. Implementierung der neuen Suchfunktion

In diesem Kapitel wird die Implementierung einer alternativen Suchfunktion zu der Suchfunktion von Confluence dargestellt. Diese Implementierung wird anschließend mithilfe des erläuterten Versuchsaufbaus mit der bestehenden Suche von Confluence verglichen. Die Implementierung verwendet eine Vektordatenbank, welche eine semantische Suche ermöglicht. Das Filtern nach bestimmten Properties ermöglicht die Vektordatenbank ebenfalls. Es wird die Vektordatenbank Weaviate verwendet. Der Vergleich zeigt, ob eine semantische Suchfunktion *besser* ist als die Confluence-Suche.

4.1. Aufsetzen der Vektordatenbank Weaviate

Für das Aufsetzen von Weaviate werden zwei Komponenten benötigt. Zum einen die Vektordatenbank selbst. Zum anderen ein Transformer, welcher Text entgegennimmt, und diese in Vektoren umwandelt, sodass diese in der Datenbank gespeichert werden können. Um die beiden Komponenten aufzusetzen wird Docker verwendet. Das entsprechende docker-compose-yml File ist im Anhang zu finden. Hier werden die beiden Komponenten definiert. Unter *t2v-transformers* wird der Transformer konfiguriert. Es wird das Image eines bereits vortrainierten Transformers verwendet. Unter *weaviate* wird die Datenbank konfiguriert. Hier wird mithilfe von den Environment-Variablen $DEFAULT\_VECTORIZER\_MODULE$, $ENABLE\_MODULES$ und $TRANSFORMERS\_INFERENCE\_API$ konfiguriert, welcher Transformer verwendet werden soll. So wird konfiguriert, dass der Transformer der anderen Komponente verwendet werden soll.

Mithilfe der Dependency *io.weaviate:client:4.0.1* wird die Client API von Weaviate verwendet. So wird nun eine Verbindung zu der Vektordatenbank aufgebaut. Diese beinhaltet zu diesem Zeitpunkt noch keine Daten. Bevor die Daten in die Datenbank eingespielt werden, muss das Schema der Daten angegeben werden. Der entsprechende Code ist ebenfalls im Anhang zu finden. Es wird eine Klasse *Document* definiert. Diese Klasse beinhaltet die Properties *documentUrl*, *h1*, *h2* und *p*. Es werden in dieser Klasse also die URL des Dokuments, sowie die Inhalte aller h1-, h2 und p-Tags gespeichert. Außerdem wird als Vectorizer *text2vec-transformers* konfiguriert.

4.2. Einspielen der Daten in Weaviate

Um nun die Daten aus Confluence in Weaviate einzuspielen, werden zuerst die Daten aus Confluence exportiert. Beim Export von Confluence Seiten werden HTML-Dateien generiert. Es werden keine CSS-, JavaScript- oder Bilddateien generiert. Hierdurch entstehen bei der Durchführung der Studie Probleme, auf welche später eingegangen wird. Die HTML-Dateien werden preprocessed, um dem zuvor definierten Schema zu entsprechen. Es werden zuerst mithilfe von regulären Ausdrücken alle Inhalte von h1-, h2- und p-Tags herausgefiltert. Anschließend werden Punctuations und Stopwords entfernt und die Inhalte druchlaufen einen Tokenizer und einen Stemmer. Punctuations sind Zeichen, wie die folgenden: *.,;:*. Stopwords sind Wörter, welche für einen Leser notwendig sind, aber für die Verarbeitung durch einen Algorithmus als unwichtig erachtet werden[13]. Beispiele für Stopwords sind *aber, denn, der*. Ein Tokenizer trennt einen Text in einzelne Wörter auf. Aus einem Text, wie *das deployment erfolgt durch ein bash-skript* wird also ein Array, welches folgendermaßen aussieht:

$$[\text{"}das\text{"}, \text{"}deployment\text{"}, \text{"}erfolgt\text{"}, \text{"}durch\text{"}, \text{"}ein\text{"}, \text{"}bash\text{"}, \text{"}skript\text{"}].$$

Ein Stemmer bestimmt den Wortstamm für die einzelnen Wörter auf Basis von Grammatikregeln. Konkret verwendet die Software den Porter-Stemmer-Algorithmus. Aus dem Wort *deployment* wird dadurch beispielsweise *deploy*. Duplikate von Wörtern werden anschließend verworfen. Nachdem die Inhalte dieses Preprocessing durchlaufen haben, werden sie in die Datenbank eingespielt.

Im Kapitel *Vektorindizes* wurde bereits von Performancegründen gesprochen, aus denen das Preprocessing durchgeführt wird. Da nun die einzelnen Schritte des Preprocessings erklärt wurde, kann auch erklärt werden, warum das Preprocessing die Performance beim indizieren erhöht. Zuallererst werden viele Wörter gänzlich verworfen, weil sie Stopwords sind. Durch den Stemmer werden anschließend ähnliche Wörter zusammengruppiert. Die Wörter *Heizung* und *heizen* stammen beide vom gleichen Wortstamm *heiz*. Das bedeutet, dass aus zwei verschiedenen Wörtern, welche beide indiziert werden müssen, ein einziges Wort gemacht wird. Denn am Ende werden Duplikate, wie bereits erwähnt, verworfen. Die Anzahl der Wörter, welche indiziert werden müssen, wird dadurch reduziert, und damit auch die Last auf dem Transformer, welche die Vektoren für die Wörter berechnen muss.

4.3. Verwendung der Suchfunktion von Weaviate

Weaviate verwendet eine API, welche GraphQL queries entgegennimmt. Um eine Suche durchzuführen muss ein GET-Request durchgeführt werden. Nun muss angegeben werden, welche Klasse aus der Datenbank abgefragt werden soll. In diesem Fall *Document*.

4.3.1. Verwendung der Semantische Suche

Um die semantische Suche zu verwenden, muss die NearText Funktion verwendet werden.

4.3.2. Verwendung von Filtern

TODO: Ergänzen

4.3.3. Die Benutzeroberfläche

5. Evaluationsmethoden und -Kriterien

Um zu verifizieren, dass die Implementierung ihre gewünschte Wirkung erzielt, müssen zuerst Methoden und Kriterien zur Messung herangezogen werden. Karl Popper gilt als Begründer des kritischen Rationalismus. Es war der Anfang von wissenschaftlich durchgeführten Experimenten. Bei einem Experiment wird zunächst eine Hypothese aufgestellt. Diese wird anschließend durch das Experiment untersucht. Ein solches Experiment soll später für die genannten Anwendungsfälle erstellt werden. Dazu sollen für die Anwendungsfälle Sucheingaben erstellt werden. Für jede Sucheingabe wird definiert, welche Dokumente als Ergebnis erwartet werden. Die Hypothese: Mit der implementierten Suchfunktion werden die erwarteten Dokumente *besser* gefunden als mit der bisherigen Suchfunktion. Diese Hypothese muss nun quantifizierbar und messbar gemacht werden. Die subjektive Wahrnehmung reicht nicht für eine wissenschaftliche Arbeit aus.

Die Formulierung der Hypothese ist für sich genommen zu unspezifisch. Es muss geklärt werden, wann eine Information *besser* zu finden ist. Im Folgenden werden Precision, Recall und F-Maß, sowie Qualitätskriterien gemäß ISO/IEC 9126 herangezogen, um Suchfunktionen anhand dieser Eigenschaften vergleichbar zu machen. Anhand dessen wird hergeleitet, was in diesem Kontext *besser* bedeutet. Im Anschluss werden diese Eigenschaften in einem Versuchsaufbau verwendet.

5.1. Precision, Recall und F-Maß

Zur Evaluation der Suchfunktionen werden später die statistischen Messwerte Precision und Recall verwendet. Die Messwerte beschreiben, inwieweit eine Hypothese zutrifft. Wenn also die Hypothese ist, dass ein bestimmtes Dokument gefunden wird, dann ist der Precision-Wert das Verhältnis zwischen allen gefundenen Dokumenten und den gefundenen Dokumenten, die tatsächlich relevant sind. Der Wert lässt sich im Kontext der Suche nach Dokumenten wie folgt definieren[14]:

$$Precision = P = \frac{gefundenerelevanteDokumente}{gesamteAnzahlgefundenerDokumente}$$

Der Recall-Wert gibt wiederum an, wie viele von den tatsächlich relevanten Dokumenten auch gefunden wurden. Er lässt sich in diesem Kontext wie folgt definieren[14]:

$$Recall = R = \frac{gefundenerelevanteDokumente}{gesamteAnzahlrelevanterDokumente}$$

Es ist schwierig beide Werte zu optimieren, da der Precision-Wert versucht die Anzahl der gefundenen Dokumente einzugrenzen und der Recall-Wert versucht die

Anzahl der gefundenen Dokumente zu erweitern. Das F1-Maß fasst beide Werte zu einem neuen Wert zusammen[14]:

$$F_1 = 2\frac{PR}{P+R}$$

Neben der Verwendung des F1-Maß ist es wichtig sich Gedanken darüber zu machen, welcher der beiden Messwerte wichtiger ist. In diesem Fall ist es sinnvoll eher die Precision zu optimieren. Denn, wenn ein Dokument gesucht wird, aber überhaupt nicht gefunden werden kann, dann erfüllt die Suchfunktion nicht ihren Zweck. Wenn die Suchfunktion irrelevante Dokumente darstellt, kann sie trotzdem ihren Zweck erfüllen, solange die relevantesten Dokumente zuerst in der Liste der Ergebnisse dargestellt wird. Dieser Faktor gilt auch bei der Implementierung zu berücksichtigen.

5.2. ISO/IEC 9126

Die ISO/IEC 9126 legen Qualitätskriterien für die Entwicklung von Software fest. Es gibt sechs verschiedene Qualitätskriterien, welche sich wiederum in verschiedene Facetten einteilen lassen. Doch nicht alle der sechs Qualitätskriterien sind in dieser Arbeit von Relevanz. So werden Die Qualitätskriterien Wartbarkeit, Effizienz, Übertragbarkeit und Zuverlässigkeit nicht beachtet. Die Wartbarkeit bezieht sich auf die Fähigkeit von Software sich zu verändern. Diese Fähigkeit hat allerdings nichts mit der Qualität einer Suchfunktion zu tun, wie sie hier gemessen werden soll. Die Effizienz ist im Kontext von Suchfunktionen als gegeben anzunehmen. Der Author unterstellt hier, dass jede Suchfunktion, welche untersucht werden soll, auch effizient ist. Denn im Jahr 2023 ist anzunehmen, dass eine Suchfunktion auch schnell die Suchergebnisse darstellen kann. Die Übertragbarkeit beschreibt die Fähigkeit einer Software, auf verschiedenen Umgebungen lauffähig zu sein. Die Zuverlässigkeit beschreibt, ob die Software auch bei unterschiedlichen Rahmenbedingungen in der gewünschten Form funktioniert. Die beiden Kriterien liegen ebenfalls außerhalb des Scopes der Arbeit, weil auch sie sich nicht auf die Qualität der Suchfunktion als solches beziehen.

Die zu betrachtenden Qualitätskriterien sind Benutzbarkeit und Funktionalität. Die beiden Qualitätskriterien werden in den nächsten Kapiteln im Detail erläutert. Das schließt die verschiedenen Facetten der Qualitätskriterien mit ein.

5.2.1. Benutzbarkeit

Die Benutzbarkeit von Software teilt sich in die Facetten Attraktivität, Konformität, Erlernbarkeit, Verständlichkeit und Bedienbarkeit ein. Die Attraktivität soll hier nicht weiter betrachtet werden, weil die reine Optik einer Suchfunktion im Kontext dieser Arbeit keine Relevanz besitzt. Die Facetten Erlernbarkeit und Verständlichkeit ergeben sich durch die Konformität der Suche. Damit eine Suchfunktion konform ist, muss sie alle gängigen Arten von Suchen unterstützen. Also

Keyword Search, Phrase Search, Boolean Search, Phrase Search. Die unterschiedlichen Arten von Suchen werden in dem Kapitel "Theoretischer Hintergrund" näher erläutert.

Die übrige Facette ist die Bedienbarkeit. Eine Autovervollständigung von Wörtern kann es dem Softwareentwickler vereinfachen, passende Sucheingaben zu machen. Sie verhindert zum einen, dass er Tippfehler macht. Zum anderen hilft sie dem Softwareentwickler auch bei der Wortwahl. Sie sorgt also für eine bessere Bedienbarkeit.

5.2.2. Funktionalität

Die Funktionalität von Software teilt sich in die Facetten Angemessenheit, Sicherheit, Interoperabilität, Konformität, Ordnungsmäßigkeit und Richtigkeit auf.
TODO: Weiter ausführen

5.3. Versuchsaufbau

Für den Versuchsaufbau werden Dokumente benötigt, welche durchsucht werden können. Dazu sollen Daten aus einem echten Projekt verwendet werden. Da die Informationen zu dem Projekt nicht veröffentlicht werden dürfen, müssen die Sucheingaben und Ergebnisse so eingeschränkt werden, dass keine projektspezifischen Informationen preisgegeben werden. Die Daten sind bereits in einem Confluence Space vorhanden, dessen Suche als Benchmark für die neu implementierte Suchfunktion verwendet wird. Die Implementierung der neuen Suchfunktion und die Befüllung dessen Datenbank mit den gleichen Daten, wie die Confluence Suche, wird in dem Kapitel *Implementierung* näher erklärt.

Wie zuvor erwähnt, müssen nun Sucheingaben erstellt werden, welche in beiden Suchen eingegeben werden können. Zum Vergleich der beiden Suchfunktionen werden für jede Sucheingabe die obersten fünf Ergebnisse betrachtet. Für jedes Dokument in diesen Ergebnissen, welches laut Experimentaufbau als Ergebnis erwartet wird, erhält die Suchfunktionen einen Hit. Die Hits werden für beide Suchfunktionen zusammengezählt und verglichen. Anschließend wird eine einfaktorielle Varianzanalyse mithilfe von SPSS[1] durchgeführt. Die Analyse wird zeigen, ob die neu implementierte Suchfunktion signifikant mehr Hits generiert als die bestehende Confluence Suche.

5.4. Einfaktorielle Varianzanalyse

Die Varianzanalyse überprüft, ob ein signifikant unterschiedlicher Wert zwischen den beiden Suchfunktionen besteht. Wenn die neu implementierte Suchfunktion

[1] https://www.ibm.com/de-de/spss

signifikant mehr Dokumente findet als die Confluence-Suche, dann bestätigt dies die Hypothese. Die Hypothese ist dabei, dass die neu implementierte Suchfunktion eine Verbesserung zur bestehenden Confluence-Suche darstellt.

Zur Durchführung der einfaktoriellen Varianzanalyse muss die abhängige Variable und die unabhängige Variable definiert werden. Die unabhängige Variable ist die verwendete Suchfunktion. Diese muss numerisch repräsentiert werden. So steht die Zahl 0 für die Confluence-Suche, und die Zahl 1 für die neu implementierte Suchfunktion. Die abhängige Variable ist die Anzahl der gefundenen erwarteten Ergebnisse. Sie ist die abhängige Variable, weil die Anzahl von der verwendeten Suchfunktion abhängt.

6. Vergleich der Suchfunktionen

Zum Vergleich der beiden Suchfunktionen wird im Folgenden ein fiktionales Szenario dargestellt. Das Szenario beschreibt den Ablauf des Onboardings eines neuen Mitarbeiters. Dabei werden die expliziten Fragen beschrieben, welche sich der Mitarbeiter stellt, um sich einzuarbeiten. Es wird angenommen, dass der Mitarbeiter sich mithilfe der Wissensdatenbank einarbeitet und die Suchfunktion der Wissensdatenbank verwendet. Die Fragen, welche sich der Mitarbeiter des fiktionalen Szenarios stellt, werden anschließend bei der Durchführung einer Studie verwendet. Die Studie verwendet die Fragen, um die Performance zwischen Suchfunktionen zu vergleichen. Der genaue Aufbau der Studie ist im Folgenden näher beschrieben. Nachdem der Aufbau der Studie erklärt wurde, werden die Daten der Studie ausgewertet und dargestellt. Das Ergebnis wird diskutiert. Zuletzt wird auf die Validität der Daten eingegangen und es wird der Versuchsaufbau diskutiert.

6.1. Fiktionales Szenario: Onboarding eines Mitarbeiters

Für den Vergleich von Suchfunktionen müssen auf Grundlage der Anwendungsfälle realistische Fragestellungen entwickelt werden. Zu diesem Zweck sei angenommen, dass ein neuer Softwareentwickler in einem bestehenden Softwareprojekt eingearbeitet wird. Es wird also der Anwendungsfall des Onboardings betrachtet.

Zuerst wird dem Softwareentwickler aufgetragen, sich das *Getting Started* im Confluence durchzulesen. Der Softwareentwickler stellt sich also die Fragen, **wo sich das Getting Started befindet**.

Um sich mit der Software vertraut zu machen, wird dem neuen Softwareentwickler anschließend aufgetragen, die Anwendung bei sich lokal zu starten. Nachdem er sie gestartet hat, soll er sich mit der Funktionalität der Software vertraut machen. Um die Anwendung lokal zu starten, **sucht der Softwareentwickler nach einer Installationsanleitung**.

Nachdem die Software installiert ist, startet er die Anwendung. Dazu muss er sich anmelden. Er möchte herausfinden **mit welchen Anmeldedaten er sich anmelden kann**. Nachdem er sich angemeldet hat, möchte er sich eine Übersicht über die Funktionen der Software verschaffen. Dazu sucht er nach den **Use-Cases** der Anwendung. Nachdem er die Use-Cases gefunden hat, probiert er mehrere davon aus.

Einer der Use-Cases ist die Versendung von Korrespondenzen am Ende eines Workflows. Er möchte diesen Use-Case genauer nachvollziehen. Deshalb möchte er herausfinden **wie man eine Korrespondenz verschickt** und **wie man den Empfänger einer Korrespondenz einstellen kann**.

Ein weiterer Use-Case ist die Erstellung eines Auftrags. Der Softwareentwickler möchte herausfinden, **wie ein Auftrag erstellt werden kann**.

Der Softwareentwickler möchte sich weiterhin noch technischer mit der Software auseinandersetzen, und möchte sich daher das genaue Datenmodell eines Auftrags ansehen. Er **sucht also nach dem Datenmodell des Auftrags**.

Nachdem sich der Softwareentwickler eine Weile mit der Software auseinandergesetzt hat, wird ihm mitgeteilt, dass Mitarbeiter des Projektes in Teams eingeteilt sind. Dabei gibt es drei Entwicklungsteams: Team Blau, Team Rot, Team Gold. Ihm wird erklärt, dass er von nun an Teil von Team Blau sein wird. Daher beschließt der neue Softwareentwickler herauszufinden, welche anderen Mitarbeiter auch Teil seines Teams sind. Er **möchte herausfinden, wer alles zu Team Blau gehört**.

Weiterhin möchte er genauer verstehen, wie die Entwicklung in dem Projekt abläuft. Er beschließt herauszufinden, **für welche Teile die unterschiedlichen Teams und Mitarbeiter zuständig sind**.

6.2. Aufbau der Studie

Im Rahmen der Bachelorarbeit wird eine Studie durchgeführt, welche die neue Suchfunktion mit der bestehenden Confluence-Suche vergleicht. Da der Zeitrahmen begrenzt ist, in welchem die neue Suchfunktion entwickelt wird, wird nicht die tatsächliche Suchfunktion von Confluence entwickelt. Es wurde bereits erwähnt, dass die Suche von Confluence auf Apache Lucene basiert, und dass dieses ein VSM auf Basis von BM25 verwendet. Daher wird die neu entwickelte Suchfunktion in mehreren Konfigurationen verglichen. So wird die neue Suchfunktion so konfiguriert, dass diese eine reine BM25 Suche durchführt. Diese Konfiguration soll als Benchmark dienen und die tatsächliche Confluence-Suche repräsentieren. Eine weitere Konfiguration verwendet eine Mischung aus einer BM25 Suche, und einer semantischen Suche. Die beiden Suchalgorithmen werden zu gleichen Teilen verwendet. Zuletzt verwendet eine andere Konfiguration lediglich die semantische Suche auf Grundlage von LSE.

Die Eigenschaft, welche untersucht werden soll ist, ob eine semantische Suche für den gegebenen Datensatz, und im Kontext einer Wissensdatenbank in der Softwareentwicklung, ebenfalls bessere Suchergebnisse liefert, als eine Suche auf Basis von BM25. Durch die Verwendung einer einheitlichen Implementierung wird

sichergestellt, dass lediglich die Effektstärken der Suchalgorithmen untersucht werden. Es wird damit verhindert, dass die Confluence-Suche besser abschneidet, weil sie ausgereifter ist. Es wird ebenfalls sichergestellt, dass die Datensätze der Suchfunktionen identisch sind.

Für die Studie wurde ein Datensatz generiert, welcher Dokumente beinhaltet, welche durch die Suchfunktion gefunden werden sollen. Für jedes Dokument sind eine oder mehrere Sucheingaben definiert, mit dessen Eingabe das Dokument gefunden werden soll. Darüber hinaus ist für jedes Dokument festgehalten, zu welchem Anwendungsfall sich dieses zuordnen lässt.

Die Suchfunktion gibt für jeden Algorithmus fünf Dokumente als Antwort auf eine Sucheingabe zurück. Diese fünf Dokumente nach dessen Score sortiert. Das bedeutet, dass das erste Dokument der Liste jenes ist, welches von dem Algorithmus als das passendste erachtet wird. Die fünf Dokumente werden darauf untersucht, ob sich das gewünschte Dokument unter den Dokumente befindet. Das Ergebnis ist ein Precision-Score für den Suchalgorithmus. Um eine detailliertere Analyse zu ermöglichen wird nicht nur die Precision in Bezug auf die ersten fünf Dokumente gemessen. Es wird die Precision für das erste Dokument, die ersten drei Dokumente und alle fünf Dokumente gemessen. Anschließend werden die Precision-Scores der Algorithmen miteinander verglichen.

Die Studie wird vollkommen automatisch durchgeführt. Dadurch können Ergebnisse der Studie nicht durch Teilnehmer verfälscht werden. Es bedeutet auch, dass die Studie eine hohe Reliabilität hat und mit jeder Durchführung das gleiche Ergebnis liefert. Sie ist Reproduzierbar. Der Nachteil dieser Herangehensweise ist, dass die subjektive Wahrnehmung des Nutzers, in Bezug auf die Precision der Suchfunktion, nicht beachtet werden kann. So ist es denkbar, dass eine Sucheingabe nicht das gewünschte Dokument beinhaltet, aber andere Dokumente, welche ein Nutzer als sinnvoll erachten würde. Die Ergebnisse der Studie sind damit abhängig von der Vorauswahl der Dokumente, welche gefunden werden sollen, und der Sucheingaben, welche a priori als sinnvoll bestimmt wurden. Es ist möglich, dass eine Suchfunktion für eine Sucheingabe durchaus ein sinnvolles Ergebnis liefert, aber nicht das Ergebnis, welches durch den Aufbau der Studie erwartet wird.

6.3. Auswertung der Ergebnisse

TODO: Ergänzen

6.4. Diskussion des Studienaufbaus

Wie bereits beschrieben gibt der Recall an, wie viele der relevanten Dokumente gefunden wurden. Die Precision gibt lediglich an, wie viele der Dokumente, welche gefunden wurden, relevant sind. Eine optimale Suchfunktion würde per Definition

alle Dokumente finden, welche relevant sind, und keine unrelevanten Dokumente. Umgekehrt ist eine schlechte Suchfunktion eine Suchfunktion, welche keine relevanten Dokumente findet, sondern nur unrelevante. Dabei spielt es für den Nutzer aber keine Rolle, ob unrelevante Dokumente gefunden wurden. Die Tatsache, dass die relevanten Dokumente nicht gefunden wurden, machen die Suchfunktion für den Nutzer nicht benutzbar. Die Studie untersucht die Precision der Suchalgorithmen. Auf Grundlage der obigen Argumentation zeigt sich, dass der Recall-Score wichtiger ist als der Precision-Score. Denn ist der Recall gering, dann ist die Suchfunktion nicht benutzbar. Eine Suchfunktion mit einem hohen Recall, aber eine niedrigen Precision könnte dagegen viele relevanten Dokumente finden, aber auch viele irrelevante Dokumente. Solange die relevanten Dokumente in der Liste weiter oben dargestellt werden, ist diese Zusammensetzung aus Precision und Recall gut.

Die Studie hat die Precision für das erste, die ersten drei und alle fünf Dokumente gemessen. Die Studie hat allerdings keinen Recall gemessen, obwohl es sinnvoll ist den Recall als den wichtigeren Score einer Suchfunktion zu erachten. Grund dafür ist die Tatsache, dass um den Recall messen zu können, alle relevanten Dokumente für eine Sucheingabe bekannt sein müssen. Um für eine Studie a priori Sucheingaben zu bestimmen, und alle Dokumente, welche für diese Sucheingabe relevant sind, müsste der gesamte Datensatz bekannt sein. Da der Datensatz mehrere hundert Dokumente beinhaltet ist dies nicht möglich. Und auch wenn der gesamte Datensatz bekannt wäre, dann müsste trotzdem eine Entscheidung darüber getroffen werden, welche Dokumente relevant sind und welche nicht. Das wäre wiederum eine subjektive Entscheidung, sodass die Validität des Ergebnisses anzweifelbar wäre.

TODO: Question Answering (Frage: was sind die dateiformate für den auftragseingang?/was für dateien gebe ich in den auftragseingang?, Antwort: excel - dateien oder csv - dateien)

7. Zusammenfassung und Ausblick

TODO: Ergänzen

7.1. Zusammenfassung

TODO: Ergänzen

7.2. Ausblick

TODO: Ergänzen

Die neue Suchfunktion gibt wenige bestimmte Ergebnisse in sehr vielen Fällen zurück, egal ob das Ergebnis in diesem Fall sinnvoll ist oder nicht.

Die Performance der semantischen Suche ist abhängig von dem gewählten Transformer. Bei der Implementierung der neuen Suchfunktion wurde der Sentence Transformer *sentence-transformers-msmarco-distilroberta-base-v2* verwendet. Möglicherweise kann die Verwendung eines anderen Modells die Performance der neuen Suchfunktion verbessern. Es besteht die Möglichkeit die Codebase ebenfalls zum Index hinzuzufügen und damit Traceability zwischen Code und Wissensdatenbank herzustellen.

8. Literaturverzeichnis

[1] Antoniol, Canfora, Casazza und De Lucia. „Information retrieval models for recovering traceability links between code and Documentation“. In: *Proceedings International Conference on Software Maintenance ICSM-94* (2000).

[2] Judit Bar-Ilan. „Methods for measuring search engine performance over time“. In: *Journal of the American Society for Information Science and Technology* 53.4 (2002), S. 308–319.

[3] Carlos Castillo. „Effective web crawling“. In: *ACM SIGIR Forum* 39.1 (2005), S. 55–56.

[4] Sneha Choudhary, Haritha Guttikonda, Dibyendu Roy Chowdhury und Gerard P. Learmonth. „Document retrieval using deep learning“. In: *2020 Systems and Information Engineering Design Symposium (SIEDS)* (2020).

[5] Sarah J. Clarke und Peter Willett. „Estimating the recall performance of web search engines“. In: *Aslib Proceedings* 49.7 (1997), S. 184–189.

[6] Andreas Dengel. *Semantische Technologien Grundlagen - Konzepte - Anwendungen*. Spektrum, Akad. Verl, 2012.

[7] Bogdan Dit, Meghan Revelle, Malcom Gethers und Denys Poshyvanyk. „Feature location in source code: A taxonomy and survey“. In: *Journal of Software: Evolution and Process* 25.1 (2011), S. 53–95.

[8] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia und Tim Menzies. „Automatic query reformulations for text retrieval in software engineering“. In: *2013 35th International Conference on Software Engineering (ICSE)* (2013).

[9] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen und Wen-tau Yih. „Dense passage retrieval for open-domain question answering“. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2020).

[10] Moaiad Khder. „Web scraping or web crawling: State of Art, Techniques, approaches and application“. In: *International Journal of Advances in Soft Computing and its Applications* 13.3 (2021), S. 145–168.

[11] Fritz Lehmann. *Semantic networks in Artificial Intelligence*. Pergamon Pr.

[12] Y. Li, D. McLean, Z.A. Bandar, J.D. O'Shea und K. Crockett. „Sentence similarity based on semantic nets and corpus statistics“. In: *IEEE Transactions on Knowledge and Data Engineering* 18.8 (2006), S. 1138–1150.

[13] Serhad Sarica und Jianxi Luo. „Stopwords in technical language processing“. In: *PLOS ONE* 16.8 (2021).

[14] Pavel Sirotkin. *On Search Engine Evaluation Metrics*. 2012.

[15] Christoph Treude, Mathieu Sicard, Marc Klocke und Martin Robillard. „TaskNav: Task-based navigation of software documentation“. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (2015).

[16] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu und Chang Liu. „From word embeddings to document similarities for improved information retrieval in software engineering“. In: *Proceedings of the 38th International Conference on Software Engineering* (2016).

[17] Qin Zhang, Shangsi Chen, Dongkuan Xu, Qingqing Cao, Xiaojun Chen, Trevor Cohn und Meng Fang. „A survey for Efficient Open Domain Question Answering“. In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2023).

Anhang

A. Anhang

A.1. docker-compose.yml File für Weaviate

```
version: '3.4'
services:
  weaviate:
    image: semitechnologies/weaviate:1.18.3
    ports:
      - "8080:8080"
    environment:
      QUERY_DEFAULTS_LIMIT: 20
      AUTHENTICATION_ANONYMOUS_ACCESS_ENABLED: 'true'
      PERSISTENCE_DATA_PATH: "./data"
      DEFAULT_VECTORIZER_MODULE: text2vec-transformers
      ENABLE_MODULES: text2vec-transformers
      TRANSFORMERS_INFERENCE_API: http://t2v-transformers:8080
      CLUSTER_HOSTNAME: 'node1'
    volumes:
      - /var/weaviate:/var/lib/weaviate
  t2v-transformers:
    image: semitechnologies/transformers-inference:sentence-transformers-
        msmarco-distilroberta-base-v2
    environment:
      ENABLE_CUDA: 0
```

A.2. Initialisieren des Schemas in Weaviate

```
  client.schema()
    .classCreator()
    .withClass(
      WeaviateClass.builder()
        .className(ConfluenceDataService.DOCUMENT_CLASS)
        .properties(
          buildProperties(
            mapOf(
              ConfluenceDataService.DOCUMENT_URL to WEAVIATE_TEXT_DATATYPE
              ConfluenceDataService.H1_TAG to WEAVIATE_TEXT_DATATYPE,
              ConfluenceDataService.H2_TAG to WEAVIATE_TEXT_DATATYPE,
              ConfluenceDataService.PARAGRAPH_TAG to WEAVIATE_TEXT_DATATYP
```

```
            )
          )
        )
        .vectorizer(VECTORIZER)
        .build()
).run()
```

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe alle Stellen, die ich aus den Quellen wörtlich oder inhaltlich entnommen habe, als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Essen, den 5. September 2023