Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

FAKULTÄT
FÜR MATHEMATIK, INFORMATIK
UND NATURWISSENSCHAFTEN

BACHELORARBEIT

ZUR ERLANGUNG DES AKADEMISCHEN GRADES BACHELOR OF SCIENCE (B.SC.)

Analyse von Ticketsystemen mithilfe textverarbeitender Algorithmen

---

vorgelegt von
Moritz Kreinsen

Matrikel-Nr.:	**Studiengang:**	**Fach:**
6763956	Erziehungswissenschaft / Lehramt an Gymnasien	Informatik

Erstgutachter: Dr. Lothar Hotz
Zweitgutachter: Prof. Dr. Chris Biemann

vorgelegt am
Hamburg, den 02. August 2018

Vorbemerkung

Aus Gründen der leichteren Lesbarkeit wird in der vorliegenden Bachelorarbeit die männliche Sprachform bei personenbezogenen Substantiven und Pronomen verwendet. Dies impliziert jedoch keine Benachteiligung anderen Geschlechts, sondern soll im Sinne der sprachlichen Vereinfachung als geschlechtsneutral zu verstehen sein.

Da als Grundlage dieser Arbeit unternehmensinterne Datenbanken des Ticketsystems HADES des HITeC e.V. Projektes *3S Schul-Support-Services* verwendet werden, sind im Folgenden bei textlichen oder grafischen Darstellungen personenbezogene Daten, aus datenschutzrechtlichen Gründen unkenntlich gemacht worden.

Kurzfassung

Help Desks sind heutzutage aus dem IT-Support nicht mehr wegzudenken. Wann immer ein Problem mit Hard- oder Software eines Kunden auftritt, muss dieses so schnell und effizient, wie möglich behoben werden. Um diese Kundenanfragen zu koordinieren, verwenden IT-Help-Desk in der Regel ein Ticketsystem zum Verwalten der Kunden und dessen Anliegen. Auch Schulen – seien es Förder-, Grund-, oder weiterführende Schulen – profitieren von solchen Services, da die eigenständige Wartung und Fehlerbehebung viel zu zeitintensiv wäre und häufig Fachkenntnisse erfordert, die ein Lehrer nicht zwangsläufig besitzt.

Diese Arbeit behandelt die Entwicklung eines Empfehlungsdienstes mithilfe textverarbeitender Algorithmen und Technologien für das Ticketsystem eines Schul-Support-Services mit Dienstleitungen im IT-Sektor. Dabei werden Verfahren des Natural Language Processings (NLP) und des Information Retrievals (IR), insbesondere das Text Mining zur Analyse von natürlichsprachlichen Textdokumenten beschrieben und angewendet.

Es wird gezeigt, dass sich unter Verwendung von «sklearn», einem Python-Modul zum maschinellem Lernen und dessen Algorithmus zur Berechnung von Kosinus-Ähnlichkeiten zwischen den Elementen einer Eingabematrix, paarweise ähnliche Tickets identifizieren lassen und dessen tatsächliche Ähnlichkeit beziffern und in ein Ranking einordnen lässt. Ebenfalls wird demonstriert, wie die Modellierung und Modifizierung der Ticketmetadaten in Form von natürlichsprachlichen Ticketdokumenten die Qualität dieser Ähnlichkeiten zwischen den Tickets beeinflusst und die Optimierung des Rankings quantitativ untersucht. Dazu wird ein Algorithmus zur Berechnung des «Normalized Discounted Cumulative Gain» (nDCG), einem Gütemaßstab für Rankingqualität, verwendet.

Inhaltsverzeichnis

1. Einleitung

1.1 Aufgabenstellung und Zielsetzung

Unternehmen, Vereine und Hochschulen setzen häufig auf Ticketsysteme zum Bearbeiten von Anfragen, beispielsweise bezüglich technischer Probleme, Verwalten von Informationen und Kundenkontakten bzw. Ansprechpartnern und zum Festhalten der benötigten Zeit pro Auftrag, Projekt oder Anfrage.

Ziel der Bachelorarbeit soll es sein, die Effizienz im Support von Kunden durch Mitarbeiter eines Unternehmens, also den Nutzern des Ticketsystems, zu steigern, indem ein Empfehlungsdienst entwickelt wird, der mithilfe algorithmenbasierter, textanalytischer Ansätze Tickets analysiert und den Mitarbeitern Vorschläge zu inhaltlich ähnlichen, älteren und bereits vorhandenen Tickets anbietet. So sollen Lösungen zu Problemen oder Fehlern, welche beim Kunden auftreten schneller gefunden und Zeit eingespart werden.

Die Fragestellung, der dabei nachgegangen wird ist, welche Technologien, bezogen auf die Anwendung in einem Ticketsystem, man hierzu einsetzen kann und welche algorithmenbasierten Verfahren im Bereich der Verarbeitung natürlicher Sprache oder des maschinellen Lernens angewendet werden könnten. Dazu werden die notwendigen Grundlagen erarbeitet. Weiterhin sollen Anforderungen an den Empfehlungsdienst, dessen Backend und mögliche Probleme für praktischen Einsatz untersucht werden.

1.2 Verwandte Arbeiten

Das Gebiet der Empfehlungsdienste ist sehr weit gefächert. Ob in sozialen Medien, im Online-Handel oder einer Suchmaschine, wie Google. Der Nutzer wird in all diesen Bereichen mit Empfehlungen mit ähnlichem Content zu dem aktuell oder kürzlich Beobachteten konfrontiert.

Die Anwendung eines solchen Dienstes innerhalb eines Ticket- oder Help-Desk-Systems wurde bislang noch wenig erforscht, was diese Arbeit motiviert.

Oliveira et al. (2014) entwickelten ein «Mobile Service Desk», eine Art Ticket- bzw. Incident-Management-System, welches dem Nutzer Lösungsvorschläge, aufgrund vergangener Fälle, für bestehende Probleme empfiehlt, um so Zeit und Kosten im IT-Support einzusparen und die Notwendigkeit der mehrfachen Lösungssuche für das gleiche Problem abzuschaffen. Dazu wurden die Vorverarbeitung von Textdaten und unterschiedliche Herangehensweisen zur Ähnlichkeitsberechnung aufgezeigt, beispielsweise der Jaro-Winkler Algorithmus, Levenshtein Distanz und PHP's «similar_text()»-Funktion.

Hierzu erarbeiteten auch Zhou et al. (2016) eine Lösung, welche sich allerdings auf automatisch eintreffende Ereignismeldungen von Monitoring Systemen bezieht und einer automatischen Verarbeitung und Lösungsfindung zu diesen Ereignissen dient. So auch Tang et al. (2013), welcher in seinem Ansatz einen K-Nearest-Neighbor (KNN)-Algorithmus zur Ähnlichkeitsbestimmung verwendete.

Eine sehr ähnliche Vorgehensweise zu dieser Arbeit, stellten vor kurzem Elsafty et al. (2018), mit der Entwicklung einer content-basierten Methode zur Empfehlung von Stellenangeboten in einer Online-Jobbörse für deutschsprachige Anzeigen, vor.

In einer ebenfalls erst kürzlich veröffentlichten Arbeit von Silva et al. (2018), wird mithilfe von maschinellem Lernen und der Variation verschiedener Techniken des Natural Language Processings (NLP)[1] gezeigt, wie sich unter Verwendung von tf-idf-vektorisierten natürlichsprachlichen Ticketdokumenten am effizientesten Tickets kategorisieren lassen.

Angelehnt an die oben genannten Arbeiten, wird diese Arbeit die Effizienzoptimierung durch Ähnlichkeitsempfehlungen unter dem Einsatz verschiedener NLP-Techniken, anhand des Ticketsystems HADES des Hamburger 3S Schul-Support-Services, untersuchen.

1.3 Aufbau der Bachelorarbeit

Zunächst werden im Kapitel 2 «Grundlagen» die theoretischen Grundlagen zu Ticketsystemen am Beispiel von HADES, des Information Retrieval und des Natural Language Processing, sowie Empfehlungsdiensten erklärt und entsprechende Technologien vorgestellt. Weiter werden in Kapitel 3 die damit verbundenen Anforderungen und Problemstellungen benannt. In Kapitel 4 «Anwendungskonzept und Entwicklung» wird das Konzept und der Entwicklungsverlauf beschrieben. In Kapitel 5 «Auswertung und Qualitätsentwicklung der Empfehlungen» wird die Qualität des Empfehlungsdienstes durch Variation verschiedener Rankingindikatoren ausgewertet, untersucht und diskutiert. Anschließend werden in Kapitel 6 «Ausblick» weitere Möglichkeiten zur Umsetzung aufgezeigt. Abschließend folgt eine Zusammenfassung der Arbeit.

[1] Siehe hierzu Abschnitt «*2.2 Information Retrieval und Natural Language Processing*» im Kapitel 2 «*Grundlagen*»

2. Grundlagen

Dieses Kapitel beschäftigt sich mit den Grundlagen zur Analyse von Ticketsystemen mithilfe textverarbeitender Algorithmen. Es beschreibt den Aufbau und den Anwendungsbereich eines Ticketsystems am Beispiel des Ticketsystems *HADES* des Hamburger HITeC e.V.-Projektes *3S-Schul-Support-Service* und dessen Help Desk und behandelt die theoretischen Ausgangspunkte des Information Retrieval (IR). Außerdem wird auf die Funktionsweise und Verwendung von Empfehlungsdiensten (engl.: «Recommender Systems») eingegangen.

2.1 Ticketsysteme und Help Desks

Ticketsysteme (engl.: «Issue-Tracking-System») sind sozio-technische Systeme, das heißt sie implizieren sowohl menschliche als auch maschinelle Bestandteile. Sie zählen zu der Gruppe der Informationssysteme. Ziel dieser Systeme ist es, „Informationen optimal bereitzustellen und optimal nach wirtschaftlichen Kriterien (Kosten-Nutzen-Aspekten) zu kommunizieren. Vor diesem Hintergrund sind Systeme generell als eine Menge von Elementen aufzufassen, die im Kontext zueinanderstehen. Die Kommunikation in Informationssystemen ist dabei zu verstehen, als der erforderliche Informationstransfer zwischen den Elementen eines Systems und zwischen dem System und seiner Umwelt. Hierbei werden Maschinen als Anwendungen beschrieben, die auf einer Hardware betrieben werden. Anwendungen hingegen nutzen Daten für interne Prozesse“ (Petzold & Westerkamp 2018, S. 69 zitiert nach Krcmar 2015, S. 22). Wie die Anwendungen und Prozesse im Ticketsystem zusammenwirken, wird im folgenden Abschnitt «2.1.1 Aufbau und Struktur von Ticketsystemen am Beispiel von HADES» näher erläutert.

Ein *Help Desk* ist ein System im Kundensupport eines Unternehmens, welches in Form von E-Mail-Verkehr, einer Telefonhotline bzw. einem Call-Center oder auch hybrid vorkommen kann. Kunden des Unternehmens melden sich bei diesem Help Desk unter anderem mit Problemen bei den vom Unternehmen bereitgestellten Dienstleistungen, Produktproblemen oder allgemeinen Fragen zu Produkten, Leistungen oder Kosten. Die Mitarbeiter des Unternehmens, welche den Help Desk betreuen, dokumentieren diese Anliegen in Form von Tickets im unternehmenseigenen Ticketsystem, beraten die Kunden, vereinbaren Termine und erteilen Aufträge. Dabei wird jeder Schritt in dem für das Anliegen erstellte Ticket dokumentiert. Jeder Mitarbeiter hat Zugriff auf diese Information und kann über das System Aufträge und Termine erhalten und selber Informationen zum Ticket hinzufügen, etwa zum Protokollieren von durchgeführten Arbeiten.

Es werden verschiedene Support-Level differenziert (vgl. Wei et al. 2007, S. 852):

- Level 1: Dieses Supportlevel wird umgangssprachlich auch als der oben genannte *Help Desk* bezeichnet. Anfragen auf diesem Level sind häufig einfach und schnell in circa 10-15 Minuten durch einen Call-Center-Mitarbeiter am Telefon oder per E-Mail lösbar.
- Level 2: Dieses Supportlevel ist ähnlich zu Level 1 mit dem Unterschied, dass die Bearbeitungszeit der Anfrage bzw. des Problems bis zu sechs Stunden dauern kann, sodass der Kunde währenddessen nicht anwesend sein muss, sondern sich das Call-Center nach Bearbeiten der Anfrage zurückmeldet.

- Level 3 und 4: Diese Supportlevel gehen über die Arbeit im Call-Center hinaus. Für solche Anfragen wird weiteres, technisches Personal und ein oder mehrere Vor-Ort-Termine beim Kunden benötigt.

2.1.1 Aufbau und Struktur von Ticketsystemen am Beispiel von HADES

Das Ticketsystem ist aus mehreren einzelnen Komponenten aufgebaut, die über eine Webanwendung bedient werden. Zu diesen Komponenten zählen:

1. Ein Datenbankserver, auf dem sämtliche Ticket-, Kunden-, Mitarbeiter-, und Systeminformationen in einer Datenbank gespeichert sind.
2. Ein Webserver, auf dem die Webanwendung mit einer grafischen Oberfläche läuft.
3. Eine PHP-Engine, über den auf dem Webserver Skripte laufen können.

Die verwendete Skriptsprache des in dieser Arbeit benutzten Beispiels ist PHP[2]. Es bietet die Möglichkeit, ähnlich wie JavaScript, in HTML-Code Anweisungen auszuführen. Hierzu wird standardmäßig eine Webseite in HTML geschrieben und im Code durch den Operator *<?php* die eigentlichen Skriptanweisungen eingefügt. Der Unterschied zu JavaScript besteht allerdings darin, dass das Skript auf dem Server ausgeführt, das Ergebnis in HTML konvertiert und in diesem ausgegeben wird. So ist es clientseitig nicht möglich in der HTML-Datei einer Webseite die Hintergrundprozesse nachzuvollziehen.[3]

Mittels Datenbankabfragen werden die benötigten Informationen ad-hoc, das bedeutet in dem Moment, in dem das Skript aufgerufen wird, geladen und können über Eingabefelder angefordert bzw. geschrieben werden, solange die Rechte hierfür vorhanden sind. Um auf das System zuzugreifen, muss man sich zunächst authentifizieren. Die Nutzerdaten sind ebenfalls in der Datenbank gespeichert und unterschiedliche Benutzergruppen haben unterschiedliche Rechte, um auf Informationen zuzugreifen und neue Informationen hinzuzufügen. Die Benutzergruppen sind im Rahmen dieser Arbeit irrelevant, da die Anwendung hiervon nicht tangiert wird.

Die Startseite des Ticketsystems (siehe Abbildung 1) bietet einen Überblick über alle Tickets und ist unter anderem gruppiert in offene, geschlossene, zeitlich eskalierte, eigene Tickets und solche, welche die eigene Organisation betreffen. Denn auch die eigene Organisation (hier: 3S-Schul-Support-Service) ist als Auftraggeber im Ticketsystem hinterlegt, da so interne Aufgaben an Mitarbeiter verteilt und protokolliert werden können. Über eine Schaltfläche am oberen linken Rand ist es möglich, ein neues Ticket zu eröffnen. Hierzu sind als Pflichteingaben die Zusammenfassung und die Auswahl eines Auftraggebers und Ansprechpartners einzugeben. Anschließend kann man dem Ticket die Tags zuordnen und das Ticket editieren.

[2] PHP («Hypertext Preprocessor») ist eine Skriptsprache zur Webprogrammierung.
[3] Abgerufen am 28. Juni 2018 von http://de2.php.net/manual/de/intro-whatis.php

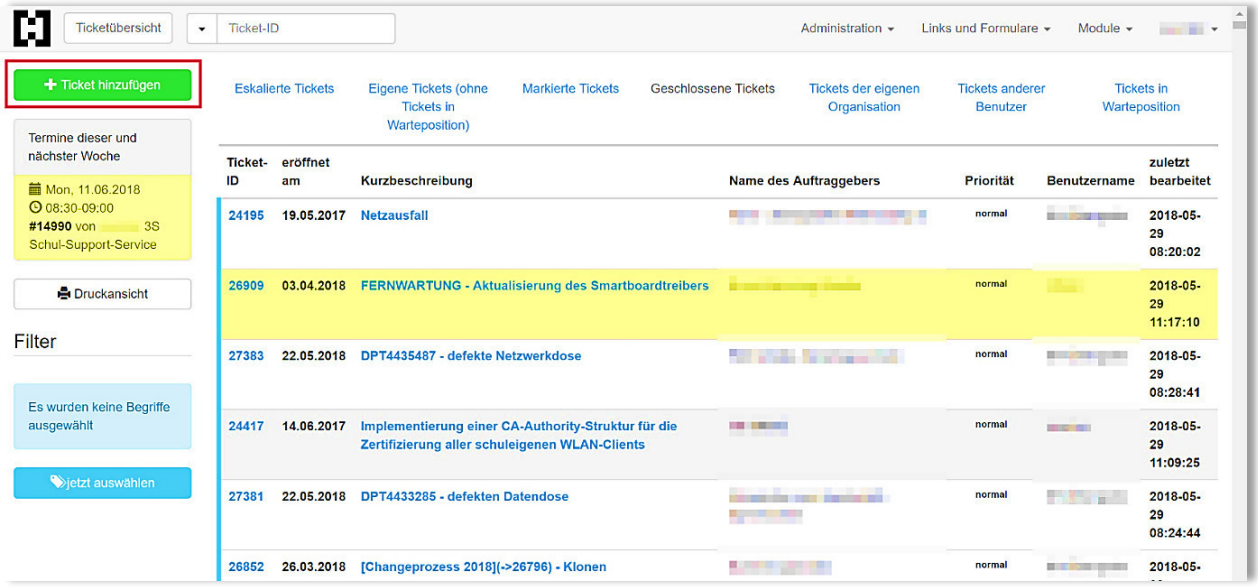

Abbildung 1: Screenshot - HADES Ticketübersicht

2.1.2 Aufbau und Struktur der HADES-Tickets

Das Wort «Ticket» ist umgangssprachlich und bezeichnet eine Form von Protokoll, das aus Informationen über den Inhalt (Metainformationen) und dem eigentlichen Inhalt mit Einträgen bzw. Ereignissen besteht.

Zu den Metainformationen zählen unter anderem der Titel bzw. die Zusammenfassung des Tickets, dessen Status, den aufgenommenen Informationen zum Anliegen bzw. die Beschreibung, Zuständigkeiten, interne Zuordnungen/Kategorien (Tags), die für dieses Ticket investierte Zeit und dessen Priorität.

So beschreibt die Zusammenfassung des Tickets mit so wenig Worten wie möglich den Inhalt des Anliegens und der Ticketstatus gibt an, ob ein Ticket offen (in Bearbeitung), zeitlich eskaliert, (lange nicht mehr bearbeitet) oder geschlossen (Anliegen behoben/gelöst) ist. Bei der Priorität kann zwischen niedrig, normal, hoch, dringend und Expertenticket gewählt werden. Als Expertenticket wird ein Ticket markiert, welches nach fünf Lösungsversuchen noch nicht gelöst wurde.

Ein Ticket kann in zwei Informationsbereiche eingeteilt werden. Der Bereich des Tickets, in dem die obenstehenden Informationen dargestellt sind, nennt man Ticketheader (siehe Abbildung 2).

Der zweite Informationsbereich enthält die eigentlichen Informationen bzw. Protokolldaten des Tickets, die sogenannten Ticketereignisse.

Ein Ereignis kann beispielsweise ein Tätigkeitsbericht, Datenergänzung, Zeiterfassung, Information über ein geführtes Telefonat oder Schriftverkehr sein. Wird ein Ereignis aufgrund einer Freitexteingabe generiert oder vom System erzeugt, wird dieser über das letzte, in dem Ticket vor-

handene Ereignis gesetzt und bekommt eine Nummer. Diese Nummern sind demnach chronologisch vom ältesten Ereignis (ganz unten) bis zum neuesten Ereignis (ganz oben) sortiert und fortlaufend nummeriert.[4]

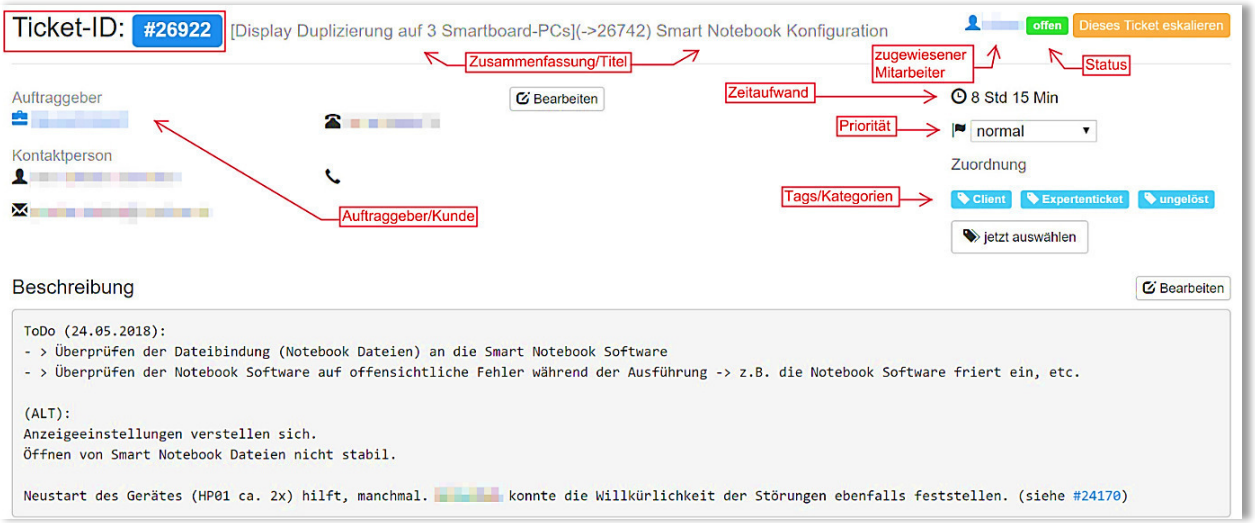

Abbildung 2: Screenshot - HADES Ticketheader (Beispiel)

Zwischen diesen beiden Informationsbereichen liegt ein Toolbereich (siehe Abbildung 3), welcher ein Eingabefeld zur Ereignisgenerierung enthält und weitere Modifikationen, wie beispielsweise eine Terminvergabe an Mitarbeiter zu diesem Ticket und Hochladen von Dateianhängen. Ereignisse hierzu werden, beispielsweise nach der Auswahl eines Datums und einer Uhrzeit des Termines vom System gesetzt und eine E-Mail-Benachrichtigung an den Teilnehmer vom System gesendet.

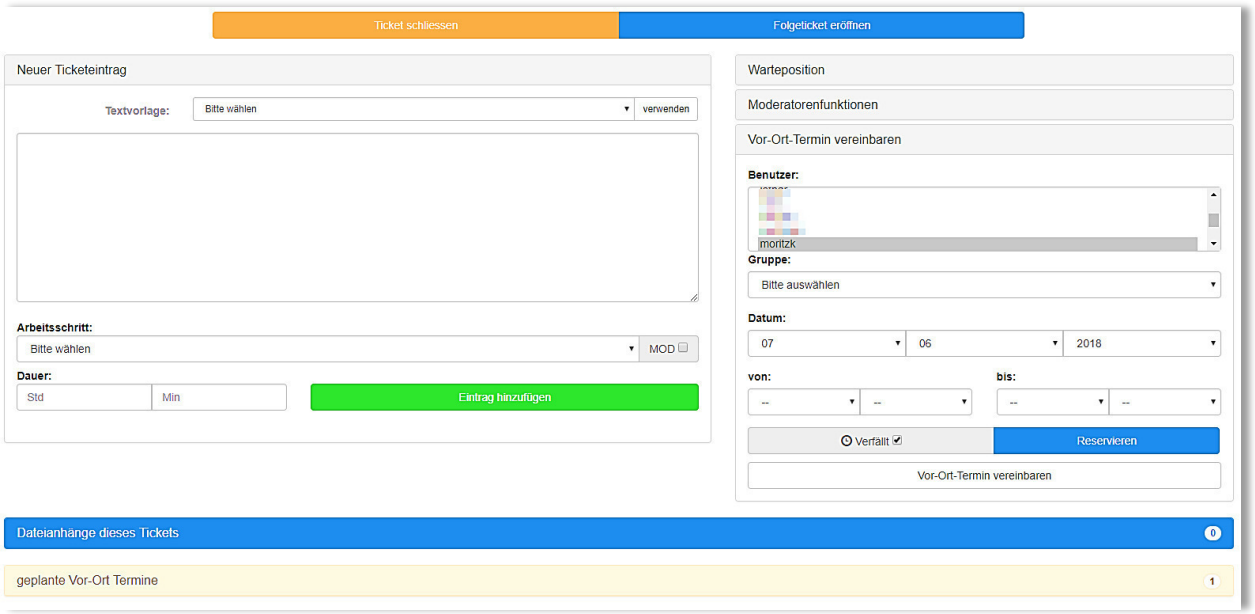

Abbildung 3: Screenshot - HADES Ticket-Toolbereich

[4] Siehe hierzu *»2.1.3 Datenbankstruktur«* unter dem Abschnitt *»2.1 Ticketsysteme und Help Desks«* im Kapitel 2 *»Grundlagen«*

Eine Besonderheit des Ticketsystems ist, dass es eine Funktion zum Erstellen eines Folgetickets gibt. Ein Folgeticket ist ein direkt mit dem Ticket verknüpftes, untergeordnetes Ticket, welches als solches in der Zusammenfassung und in den Ticketereignissen gekennzeichnet ist. Ein Folgeticket zu «*#15348 SMART Notebook Software startet nicht*» heißt beispielsweise «*#15365 [SMART Notebook Software startet nicht] (->#15348) Neuinstallation der Software per Fernwartung*», wobei die Ziffern am Anfang des Titels die Ticketnummer des aktuellen Tickets sind, die Zeichenfolge in den eckigen Klammern die Zusammenfassung des Ursprungstickets und in den runden Klammern dahinter die Ticket-ID des Ursprungstickets.

2.1.3 Datenbankstruktur

Alle Informationen, auf die HADES zurückgreift, sind in einer relationalen MySQL-Datenbank gespeichert. Diese Informationen umfassen Ticket- und Ticketsystemkomponenten-informationen, Kunden- bzw. Schulinformationen, Mitarbeiterdaten oder auch Meta-informationen, wie Zeitstempel und Kategorien von Tickets.

Im Rahmen dieser Arbeit sind nicht alle der dort abgespeicherten Informationen relevant. Nachfolgend (siehe Tabelle 1) werden die für diese Bachelorarbeit relevanten Tabellen der relationalen Datenbank mit ihren Inhalten aufgelistet:

Tabelle	Inhalte
ticket	Ticket-ID, Ticketname/-zusammenfassung, Ticketbeschreibung, Client-ID, zugeordneter Mitarbeiter, Information über Ansprechpartner sowie Kontaktdaten, Endgerät, Standort, Priorität des Tickets, Status
client	Client-ID, Schulform, Name und Adressdaten, Vertragsform, Kontaktdaten, IP-Range des Schulnetzes
tag	Tag-ID, Tagname
event	Event-ID, Eintragsnummer, Ticket-ID des betroffenen Tickets, Beschreibung, Status, Benutzer des verfassten Eintrags, Dauer, Datum
tickethastag	Ticket-ID, Tag-ID

Tabelle 1: Ausgewählte SQL-Datentabellen und Inhalte

Die Primärschlüssel der einzelnen Entitäten sind jeweils durch eine Unterstreichung gekennzeichnet. Die Tabelle «tickethastag» ist eine Relation von «ticket» und «tag» und wird verwendet, um einem Ticket einen Tagnamen zuzuordnen.

Die Tabelle «event» hat als Fremdschlüssel die Ticket-ID von «ticket» und in ihr sind alle in den Tickets generierten Einträge bzw. Ereignisse inklusive der systemgenerierten Ereignisse gespeichert. Im Attribut «Eintragsnummer» ist hierbei die im Ticket dem Ereignis zugeordnete Nummer beginnend bei «1» gespeichert, nach der die Reihenfolge der Ereignisse in der Weboberfläche festgelegt wird.

2.2 Information Retrieval und Natural Language Processing

Natural Language Processing (kurz: NLP; deut.: «Sprachtechnologie/Automatische Sprachverarbeitung») bezeichnet in der Computerlinguistik den Bereich, der sich mit der maschinellen Verarbeitung von menschlicher/natürlicher Sprache beschäftigt und sich als Teildisziplin des Information Retrieval entwickelt hat (vgl. Nadkarni et al. 2011, S. 544).

Information Retrieval (kurz: IR; deut.: «Informationsgewinnung») ist heutzutage für fast jede Person Normalität. Das Konzept, welches dahinter steckt ist denkbar einfach: Man benötigt eine Information, erstellt hierzu eine Sucheingabe und führt diese Suche aus. Dank dieser Suchmaschine erhält man so die gewünschte Information innerhalb weniger Sekunden (vgl. Kowalski 2011, S. 1).

Die Grundlagen dieses Prozesses und wie diese Informationen gefunden werden, soll im Folgenden näher untersucht werden.

2.2.1 Text- und Data-Mining

Der Begriff des Data Mining (DM) ist sehr vielfältig und weit gefächert. Gorunescu (2011, S. 4) bezeichnet es als die Wissenschaft der Extraktion nützlicher Daten aus großen Datenbeständen oder Datenbanken. Weiterführend setzt sich Data Mining ebenfalls damit auseinander diese Daten zu analysieren und zu strukturieren, also sinnvoll aufzubereiten. Hierzu zählen unter anderem das Zusammenfassen und Kategorisieren bestehender Daten, das Finden von ähnlichen Daten und diese zu gruppieren.

Data Mining setzt hierzu auf Anwendungen von maschinellem Lernen (engl.: «machine learning»), Mustererkennung, Statistik, Datenbanken und Visualisierung, um das Problem der Informationsextraktion aus großen Datenbanken zu lösen (vgl. Sharma 2014, S. 1 zitiert nach Piatetsky-Shapiro 2000).

Die Daten können in unterschiedlichen Formen vorliegen, beispielsweise Zahlen, Uhrzeit- bzw. Datumsinformation oder in Textform. Unternehmen und Organisation jeglicher Art erzeugen täglich riesige Mengen solcher Daten in verschiedensten Formaten und speichern diese in großen Datenbanken ab (vgl. Sharma 2014, S. 1).

Auch können Daten gebündelt vorkommen, als sogenanntes Datenset, ein Zusammenschluss mehrerer Einzeldaten der oben genannten Formen. Es gibt verschiedene Typen und Gruppen von Datensets. Beispielsweise zählen zur Gruppe der gespeicherten Datensets (engl.: «records») Datenmatrizen, Daten in Dokumenten und Transaktionsdaten, um nur einige zu nennen (vgl. Gorunescu 2011, S. 46). Im weiteren Verlauf dieser Arbeit wird genauer auf Dokumentendaten (engl.: «document data») eingegangen.

Eine Variante und wichtige Disziplin des Data Mining heißt Text Mining (TM) und beschreibt die Analyse großer Mengen von sprachlich-aufbereiteten[5] bzw. vorverarbeiteten Daten, welche in

[5] Siehe hierzu *»2.2.2 Sprachliche Aufbereitung/Vorverarbeitung von Textdaten (*pre-processing)« unter dem Abschnitt «*2.2 Information Retrieval und Natural Language Processing*» im Kapitel 2 «*Grundlagen*»

Textform vorliegen (vgl. Biemann & Mehler 2014, S. V). Dabei werden die verschiedenen Techniken des Data Mining auf Textdaten angewendet.

Der Unterschied zwischen Text- und Data Mining ist, dass Data Mining mit strukturierten Daten arbeitet, während Text Mining auch halb- bzw. unstrukturierte Daten in Textform behandelt. Unter strukturierten Daten versteht man Daten, die in festen Feldern verharren, wie in einer Tabelle oder eine Datenbank. Unstrukturiert sind Daten, die keinen festen Feldern zugewiesen sind und ungeordnet, beispielsweise als String vorliegen. Sie sind das Gegenteil von strukturierten Daten. Halb-strukturiert sind Daten dann, wenn sie weder in freier, ungebundener, noch in einer tabellenartiger Form wiederzufinden sind. Sie kommen demnach nicht in Datenbanken vor, haben aber trotzdem ein Format, beispielsweise XML[6]. Die Herausforderung von Text Mining ist es, neue, zuvor unbekannte Daten aus verschiedenen vorhandenen Datensätzen zu gewinnen (vgl. Vijayarani et al. 2015, S. 7).

2.2.2 Sprachliche Aufbereitung/Vorverarbeitung von Textdaten (pre-processing)

Um aus großen Mengen an Text die gewünschten Informationen herauszufiltern oder Textdaten miteinander zu vergleichen, ist es nötig, diese zunächst einmal zu bereinigen und auf das Wesentliche zu reduzieren. Dies dient unter anderem dazu, ungewollte Formate wie beispielsweise XML oder HTML auf den bloßen Textinhalt zu kürzen. Vorher ist der Einsatz von Textmining nicht möglich. Hierfür gibt es eine Reihe von Methoden und Algorithmen (vgl. Vijayarani et al. 2015, S. 7ff), aus denen im Folgenden die für diese Arbeit relevanten Schritte erläutert werden.

1. Extraktion und Tokenisierung: Liegen die Daten als unstrukturierter Text vor, wird dieser Schritt benötigt, um den Inhalt des Dokumentes in separate Wörter zu extrahieren. Dieser Prozess heißt Tokenisierung (engl.: «tokenization») und zerlegt die Zeichenkette dabei in Nomen, Verben, Adjektive und Sonderzeichen. Die Sonderzeichen werden dabei aus den Daten entfernt.

2. Entfernen von Stoppwörtern: Als Stoppwort (engl.: «stopword») bezeichnet man Wörter, die keine Relevanz für den Inhalt eines Dokumentes haben. Diese machen das Dokument nur voller und sind weniger wichtig für die Analyse. Zu diesen Wörtern zählen unter anderem Artikel, Präpositionen und Pronomen. Es gibt unterschiedliche Methoden, um diese Wörter zu definieren und zu identifizieren. Im Rahmen dieser Arbeit beschränken wir uns auf die Stoppliste (engl.: «stop list»), in der diese Wörter für die deutsche Sprache enthalten sind (vgl. Vijayarani et al. 2015, S. 7ff). Eine gute Stoppliste kann ein Dokument um bis zu 50% von irrelevanten Wörtern befreien (vgl. Oliveira 2014, S. 73).

[6] «Extensible Markup Language», ein Textformat, dass hierarchisch strukturierte Daten aufweist

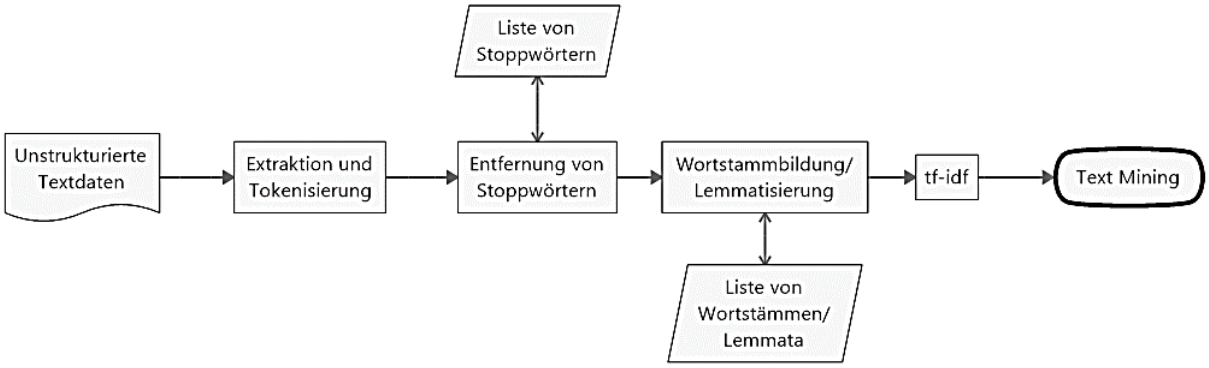

Abbildung 4: Flowchart Pre-Processing von Textdaten (vgl. Vijayarani et al. 2015, S. 9)

3. Wortstammbildung/Lemmatisierung: Abschließend werden die Wörter des Dokumentes noch auf ihren Wortstamm reduziert, das heißt alle Affixe[7] entfernt (engl.: «stemming»). Dies spart bei der späteren Analyse Zeit und Speicher und macht diese vergleichbarer. Auch hierzu gibt es eine Reihe von Methoden und Algorithmen, um das gewünschte Ziel zu erreichen. In dieser Arbeit werden unter anderem statistische Verfahren, wie N-GRAM[8] Anwendung finden (vgl. Vijayarani et al. 2015, S. 10f). Neben der Wortstammbildung gibt es noch die Lemmatisierung[9]. Diese bezeichnet die Abbildung von einer Wortform auf das entsprechende Lemma, dem Vertreter seines Lexems. Ein Lexem ist ein Set von Wortformen, die eine gemeinsamen Wortstamm bzw. eine Wortabstraktion im semantischen Sinne haben (vgl. Olive et al. 2011, S. 74). Im Unterschied zum Stemming, ist die Lemmatisierung ein automatische lexikonbasiertes Verfahren, welches beispielsweise auch unregelmäßige Verben erkennt und auf ihre Grundform zurückführt, als nur auf den gemeinsamen Wortstamm zu reduzieren.

2.2.3 Lexikalische Expansion mit JoBimText

Als lexikalische Expansion bzw. Textexpansion bezeichnet man das Generieren von zusätzlichen lexikalischen Elementen für einen bestimmten Textabschnitt innerhalb eines natürlich-sprachlichen Textkorpus, welche die textliche Darstellung erweitern. Hierbei wird diese Erweiterung für alle vorliegenden lexikalischen Elemente, unter Berücksichtigung des Kontextes, durchgeführt (vgl. Biemann & Riedl 2013, S. 56).

Das Projekt JoBimText[10] der Language-Technology Gruppe der Universität Hamburg eine Softwarelösung für eine automatische Textexpansion unter Verwendung kontextualisierter Verteilungsähnlichkeit. Dessen Ansatz stellt ein generatives, unbewachtes Modell für semantische

[7] Oberbegriff für Präfix, Suffix, Zirkumfix und Interfix, also die Zeichenketten vor, hinter, um oder zwischen einem Wortstamm.
[8] Siehe hierzu Abschnitt «*4.2 Implemen*» im Kapitel 4 «*Anwendungskonzept und Entwicklung*«
[9] Siehe hierzu Abschnitt «*4.2 Implemen*» im Kapitel 4 «*Anwendungskonzept und Entwicklung*«
[10] Siehe hierzu https://sourceforge.net/p/jobimtext/wiki/Home/

Ähnlichkeit im Kontext dar, mit dem lexikalische Expansionen für unbekanntes Textmaterial erzeugt werden können. Diese Erweiterungen tragen dazu bei, die lexikalische Lücke in der Semantik zu überbrücken und dienen als wertvoller Vorverarbeitungsschritt für viele Ansätze in der Computersemantik, wie Wortsinn-Disambiguierung[11], semantische Textähnlichkeit, Passagenbewertung und Textsegmentierung (ebd.).

2.2.4 Identifizierung von Wortarten in natürlich-sprachlichen Texten

Identifizierung bzw. Klassifizierung von Wortarten, wie beispielsweise Nomen, Verben, Adjektive oder Adverbien, bezeichnet man im Natural Language Processing als «Part-of-Speech (POS) Tagging» (deut.: «Wortart Markierung»). Hierzu werden den Wörtern innerhalb eines Textes Markierungen angehängt, anhand dessen das Wort vom Computer als solches einer bestimmten Wortart identifiziert werden kann. Diese Markierungen (engl.: «tags») sind, zum Beispiel #NN (Nomen), #VV (Verben) oder #ADJD (Adjektive). Entsprechend würde beispielsweise das Wort «Schule» vom POS-Tagger als «Schule#NN» markiert werden.

Um dieses Verfahren erfolgreich umsetzen zu können, wird der POS-Tagger mit einem Training-Set an Text trainiert. Aus den markierten Wörtern wird eine Liste konstruiert (engl.: «in-vocabulary list»). Ebenfalls wird automatisch ein Wortschatz (engl.: «distributional thesaurus (DT)») anhand eines großen Textkorpus generiert, zum Beispiel einem Set von Nachrichtenartikel mehrerer Jahre.

Um nun bei einem neuen Eingabetext die Wörter den Wortarten zuordnen zu können, werden die in diesem Text enthaltenen Wörter zunächst mit dem Vokabular aus dem Training-Set abgeglichen, um etwaige Wörter außerhalb des Vokabulars (engl.: «out of vocabulary (OOV)») zu identifizieren. Diese Wörter werden anschließend mithilfe des DT automatisch durch ihr nächstähnlichstes Wort (Synonym), welches im Vokabular enthalten ist, ersetzt. Dann werden die Wörter des geänderten Eingabetextes mit dem POS-Tagger markiert und wieder auf den Originaltext abgebildet (vgl. Biemann & Riedl 2013, S. 83).

2.2.5 Term-Frequency und Inverse-Document-Frequency

Term-Frequency - Inverse-Document-Frequency (tf-idf) ist ein statistisch-numerischer Wert, der angibt, wie wichtig ein Wort eines Dokumentes in der Gesamtheit aller Dokumente ist. Die Gesamtheit aller betrachteten Dokumente nennt man Korpus oder Kollektion. Der tf-idf Wert wird im Text Mining zur Gewichtung von Dokumenten eingesetzt (vgl. Vijayarani et al. 2015, S. 14).

Der Wert berechnet sich für ein Wort w in Dokument D im Korpus K mit $w \in D$ und $K = \Sigma D_i$ wie folgt:

$$tf(w, D) = \frac{Häufigkeit\ des\ Wortes\ w\ in\ Dokument\ D}{Max.Häufigkeit\ von\ w'\ in\ Dokument\ D} \quad (1)$$

[11] Identifizierung von Wortbedeutungen im semantischen Kontext.

$$idf(w) = \log\left(\frac{Anzahl\ der\ Dokumente\ im\ Korpus\ \mathrm{K}}{Anzahl\ der\ Dokumente, die\ w\ beinhalten}\right) \quad (2)$$

$$tf\text{-}idf(w, D) = tf \cdot idf \quad (3)$$

Entsprechend ist dieser Wert besonders groß, wenn ein Wort w häufig in einem kleinen Korpus vorkommt und besonders klein, wenn es seltener in Dokumenten auftaucht oder in nur sehr wenigen. Dabei spielen Position und Reihenfolge der Wörter im Dokument keine Rolle. Man nennt dieses Modell auch «Bag of Words», da man die Dokumente nur quantitativ betrachtet (vgl. Manning et al. 2008, S. 117f).

2.2.6 Kosinus-Ähnlichkeit und Vector-Space-Modell

Betrachtet man jedes Dokument als einen Vektor mit einer Komponente für jedes Wort im Wörterbuch[12] (engl.: «dictionary») und der Gewichtung des Dokumentes, die durch den tf-idf-Wert gegeben ist, lässt sich auf eine sehr einfache trigonometrische Art die Ähnlichkeit zwischen den einzelnen Dokumenten berechnen. Diese Vektorform ist sehr hilfreich für das Scoring und Ranking[13]. Die Darstellung eines Korpus von Dokumenten als Vektor in einem Vektorraum bezeichnet man als «Vector Space Model» (siehe Abbildung 5) (vgl. Manning et al. 2008, S. 119ff).

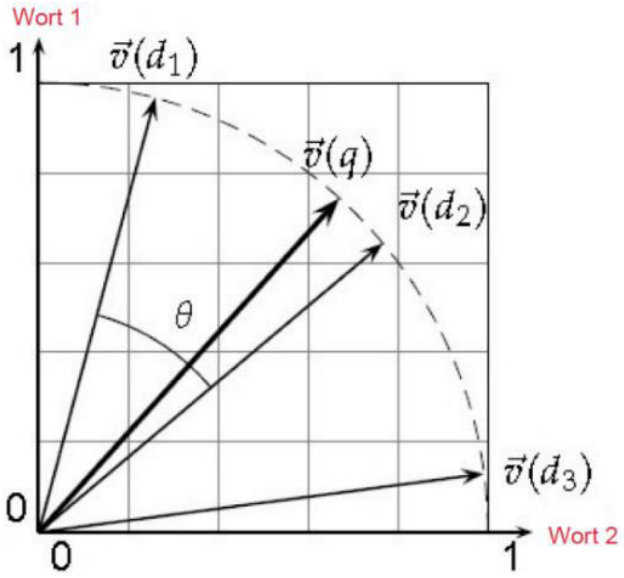

Abbildung 5: Vector Space Model mit Dokumentvektoren (aus Manning et al. 2015, S. 121)

Die Vektoren längs der Achsen in diesem Raum, repräsentieren je ein Wort in einem Dokument, zu welchen der tf-idf-Wert bestimmt wurde. So ergibt sich für m-Wörter ein m-dimensionaler Vektorraum. Die Vektoren $\vec{v}$ zwischen den Wortvektoren zeigen je nachdem, wie groß der tf-Wert für dieses Wort in dem Dokument ist, mehr oder weniger in die gleiche Richtung. Für Abbildung 5 lässt sich versinnbildlichen:

- $\vec{v}(d_1)$ repräsentiert ein Dokument im Korpus, welches eine hohe relative Häufigkeit gleich «Wort 1» hat, aber kaum Worthäufigkeiten von «Wort 2» aufweist

[12] Bezeichnet die Vereinigung der Elemente jedes Dokumentes in einem Dokumentenkorpus
[13] Siehe hierzu Abschnitt «*4.2 Implemen*» im Kapitel 4 «*Anwendungskonzept und Entwicklung*»

- $\vec{v}(d_2)$ repräsentiert ein Dokument im Korpus, welches «Wort 1» und «Wort 2» gleichermaßen häufig enthält
- $\vec{v}(d_3)$ repräsentiert ein Dokument im Korpus, welches eine hohe relative Häufigkeit gleich «Wort 2» hat, aber kaum Worthäufigkeiten von «Wort 1» aufweist
- $\vec{v}(q)$ repräsentiert ein Anfragedokument im Korpus, dessen ähnliche Dokumente man bestimmen will.

Um mit diesem Modell nun die Ähnlichkeit zwischen zwei Dokumenten d_1 und d_2 zu bestimmen, berechnet man die Kosinus-Ähnlichkeit zwischen den beiden Vektoren, welche die Dokumente repräsentieren (ebd.):

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| \cdot |\vec{V}(d_2)|} \tag{4}$$

Im Zähler steht das Skalarprodukt der beiden Dokumentvektoren. Dieses berechnet sich nach $\vec{x} \cdot \vec{y} = \sum_{i=1}^{M} x_i y_i$. Der Nenner stellt dessen euklidischen Abstand dar. Dieser ist gegeben durch $\sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)}$ wobei M der Anzahl der Komponenten $(\vec{V}_1, \dots, \vec{V}_M)$ des Vektors entspricht. Durch Längennormalisierung der Vektoren $\vec{V}(d_1)$ und $\vec{V}(d_2)$ zu Einheitsvektoren $\vec{v}(d_i)$ mit $\vec{v}(d_i) = \frac{\vec{V}(d_i)}{|\vec{V}(d_i)|}$ lässt sich Gleichung (4) umschreiben zu (ebd.):

$$sim(d_1, d_2) = \vec{v}(d_1) \cdot \vec{v}(d_2) \tag{5}$$

Gleichung (5) kann jetzt verwendet werden, um zu einer Anfrage passende Dokumente zu finden. Diesen Wert nennt man Score (ebd.).

$$score(q, d) = sim(q, d) = \vec{v}(q) \cdot \vec{v}(d) \tag{6}$$

mit q als Anfragedokument und $d \in K$.

2.2.6.1 Vom Vektor zur Matrix

Die Darstellung von Dokumenten einer Sammlung als Vektoren lässt sich vereinfacht in Form einer $M \times N$ Dokumentenmatrix darstellen. Dabei entsprechen die M-Zeilen den Dimensionen der einzelnen Vektoren, also den unterschiedlichen Wörtern und ihren Häufigkeiten, und die N-Spalten ihren Dokumenten. Dies wird in Abschnitt «4.2 Implemen» im Kapitel 4 «Anwendungskonzept und Entwicklung» näher ausgeführt (vgl. Manning et al. 2008, S. 123).

Diese Form der Darstellung ist sehr nützlich, um ein Ranking zur Abfrage von ähnlichen Dokumenten aufzustellen. Hierzu berechnet man jeweils den Score aus Gleichung (6) für jedes Element der Matrix und sortiert diese Werte entsprechend. Der hierbei kleinste resultierende Wert ist dann das Skalarprodukt der Einheitsvektoren der Abfrage und des hierzu ähnlichsten Dokumentes (ebd.).

Typischerweise interessiert einen allerdings nicht nur das beste Ergebnis, sondern eine Liste der k-besten Scores, denn möglicherweise hat das Dokument mit dem besten Score nicht gleich die höchste inhaltliche Qualität, sondern erst jenes mit dem zweit- oder sogar das mit dem drittbesten Score.

2.2.7 (Normalized) Discounted Cumulative Gain

In einem Ranking sollte stets das qualitativ beste Ergebnis an oberster Stelle stehen und die n-nächstbesten Ergebnisse in absteigender Reihenfolge darunter. Wenn man beispielsweise eine Abfrage in einer Suchmaschine abschickt, interessieren einen meistens nur die obersten Ergebnisse, da diese bereits die bestmöglichsten Resultate seiner Suche sind.

Um die Qualität eines Rankings zu beziffern, verwendet man das DCG-Maß. Das Maß bewertet die Ergebnisse nach ihrer Position im Ranking. Ein Ergebnis, das weit oben in der Liste platziert ist, hat einen besseren Wert als Ergebnisse weiter unten (vgl. Kowalski 2011, S. 268). Das DCG berechnet sich

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(1+i)} \tag{7}$$

mit k als Anzahl der Rangposition im Ranking und rel_i die Relevanz des Dokumentes an Position i (ebd.). Die Relevanz wird bestimmt, in dem man das Ergebnis bewertet. Diese Bewertungen richten sich nach einer 3-Punkte-Skala: Nicht relevant, relevant und sehr relevant. Bei solchen Gütemaßstäben werden typischerweise numerische Werte hierfür verwendet, wobei das Maß von dem Wert «0» (nicht relevant) bis zum höchsten Wert «2» (sehr relevant) reicht (vgl. Lupu et al. 2011, S. 81).

Da dieser Wert durch unterschiedliche Trefferlisten bei unterschiedlichen Abfragen beeinflusst wird, normalisiert man diesen über Abfragen hinweg. Hierzu sortiert man die Ergebnisse des Rankings nach ihrer Relevanz, wodurch ein ideales Ranking entsteht, bei dem für ein gewisses k das bestmögliche DCG ($DCG_{k,max}$) berechnet wird. Dieses normalisierte DCG bezeichnet man als «normalized discounted cumulative gain (nDCG)» und hat aufgrund der Normalisierung einen Wert zwischen 0 und 1, wobei 1 bedeutet, dass bereits das optimalste Ranking vorliegt (vgl. Kowalski 2011, S. 268). Der nDCG-Wert berechnet sich

$$nDCG_k = \frac{DCG_k}{DCG_{k,max}} \tag{8}$$

Es kann auch vorkommen, dass nicht alle Dokumente bewertet werden (vgl. Blanco et al. 2013, S. 42). Sollte ein nicht bewertetes Dokument im Ranking auftauchen, wird diesem ein dezidierter Wert zugewiesen, beispielsweise «-1».

2.3 Empfehlungsdienste

Ein Empfehlungsdienst (engl.: «Recommender/Recommendation System») kombiniert Berechnungstechniken, um benutzerdefinierte Elemente basierend auf den Interessen der Benutzer und dem Kontext auszuwählen, in dem sie sich befinden. Er hilft Benutzern dabei, mit Informationsüberflutung umzugehen und bietet ihnen Empfehlungen, angepasste Inhalte oder Dienste an (vgl. Oliveira et al. 2014, S. 73).

Man unterscheidet aufgrund der Art von Empfehlungen in verschiedene Empfehlungsdienste (ebd.):

1) content-basiert (engl.: «content-based»): empfiehlt Inhalte, die denen ähneln, die für den Benutzer von Interesse sind.

2) kollaboratives Filtern (engl.: «collaborative filtering»): identifiziert Nutzer mit ähnlichen Präferenzen zu den eigenen, um Inhalte zu empfehlen, die für den ähnlichen Nutzer von Interesse sind.

3) hybrider Ansatz (engl.: «hybrid approach»): Kombination vom inhaltsbasierten und kollaborativen Ansatz.

In dieser Arbeit wird lediglich der content-basierte Empfehlungsdienst betrachtet. Die content-basierte Methode zielt darauf ab, ähnliche Inhalte zu dem aktuell betrachteten Inhalt des Nutzers anzuzeigen, basierend auf der Berechnung von hierzu ähnlichen Inhalten, wie sie im vorigen Abschnitt «2.2.6 Kosinus-Ähnlichkeit» beschrieben ist.

3. Problemstellung und Anforderungen

Im Weiteren werden die Motivation, die Problemstellung und die Anforderungen zur Realisierung eines Empfehlungsdienstes für Ticketsysteme entwickelt.

Ziel eines Call-Centers ist es 60-75% aller Anfragen über den First-Level-Support, 15% im Second-Level-Support und nur sehr komplexe Probleme über Vor-Ort-Termine zu beheben (vgl. Wei 2007, S. 852). Aus diesem Anlass ist es von großer Bedeutung Strategien zu entwickeln, um die Bearbeitungszeit für eingehende Anfragen zu optimieren.

Motivation dieser Arbeit ist, dass die Suche und Analyse nach Problemlösungen innerhalb eines Ticketsystems anhand von bloßen Datenbankabfragen nicht effektiv genug ist, sehr viel Zeit in Anspruch nimmt und einer besseren Darstellung bedarf, um eine höhere Effektivität am Beispiel des IT-Support für Schulen zu erreichen.

Beispielsweise (siehe Abbildung 6) kommt im technischen Call-Center des Schul-Supports eine Anfrage über Störungen eines bestimmten Rechnertypen am Standort einer Schule A im Gebäude H. Ein Mitarbeiter eröffnet ein neues Ticket zu der Anfrage, mit einer passenden, prägnanten Zusammenfassung als Titel und dem Kunden, welcher die Anfrage gestellt hat als Auftraggeber. Der Call-Center-Mitarbeiter trägt die Problembeschreibung in ein dafür vorgesehenes Feld im Header des Tickets ein und ordnet dem Ticket passende Kategorien (Tags) zu. Es befindet sich bereits ein ähnliches Ticket, welches sich ebenfalls auf Störungen dieses Rechnermodells bezieht, im System und besitzt eine eindeutige Ticketnummer.

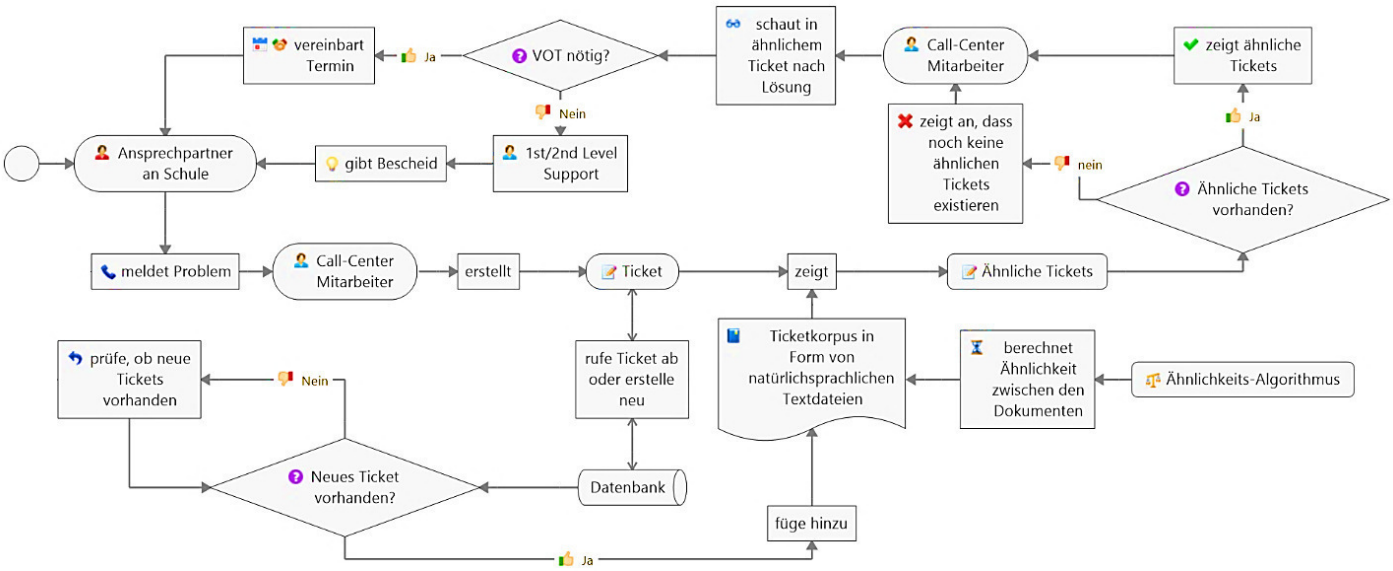

Abbildung 6: Help-Desk Concept-Flowchart

Wiederholte Ereignisse erzeugen ähnliche Tickets, die wiederum eine große Anzahl von wiederholten Problemlösungen innerhalb des Ticketsystems anreichern (vgl. Zhou et al. 2016, S. 1). Basierend auf der Arbeit von Oliveira et al. (2013) können in solchen Systemen Text-Mining Techniken und textverarbeitende Algorithmen auf die Beschreibung von neuen Tickets angewandt werden, um diese sprachlich so aufzubereiten, dass sich später die Ähnlichkeit zwischen ihnen analysieren und ähnliche Fälle vergangener Anfragen in der Datenbank identifizieren lassen (vgl. Oliveira et al. 2013, S. 73).

Ein Ähnlichkeitsalgorithmus findet aufgrund der Eingaben des Call-Center-Mitarbeiters in einem Anfrageticket andere, historische Tickets mit dem gleichen Problem und das System zeigt diesem Mitarbeiter mit absteigender Wahrscheinlichkeit diese als *Empfehlungen* in dem Anfrageticket an.

In dem oben genannten Beispiel gab es bereits an einer anderen Schule B einige Monate zuvor das gleiche Problem mit dem Rechnertypen und ein Mitarbeiter war damals bereits vor Ort, um den Fehler einzugrenzen. Dieser trug anschließend alle Erkenntnisse und Informationen in das Ticket der Schule B ein, nachdem er das Problem behoben hatte. Da dem Call-Center Mitarbeiter dieses Ticket empfohlen wurde kann er hier nachschauen, um den Fehler zu identifizieren oder einzugrenzen. Dann löst er den Vorfall gleich selber oder bildet eine Task-Force, um das aktuelle Problem aus dem Anfrageticket beheben zu lassen.

Mögliche Indikatoren für den Algorithmus sind dabei der Titel des Tickets, die Tags und die Beschreibung des Problems oder auch die Angabe des Endgerätes, welches betroffen ist, welche durch den Mitarbeiter gesetzt wurden. Grundsätzlich unterscheidet man hier in verschiedene Arten von Tickets: Change- bzw. Changeanfragetickets, Tickets zur internen Aufgabenverteilung und Personalia, LAN-Support-Tickets und Help-Desk-Tickets.

Im konkreten Beispiel von HADES sind beispielsweise solche, die als Auftrag, ausgehend vom eigenen Unternehmen aufgrund von Service- oder Vertragsleistungen angeboten werden und angefragt wurden die «Change- bzw. Changeanfrage»- und dessen «Folgetickets».

Ausgehend vom eigenen Unternehmen gibt es auch solche, die bei der internen Organisation helfen und Büroarbeiten an Mitarbeiter zuweisen, oder solche die der Personal- und Geschäftsverwaltung dienen.

Ebenfalls werden auch Anfragen für externe Dienstleister angenommen und als solche gekennzeichnet. In HADES heißen diese «LAN-Support-Tickets» und werden in der Zusammenfassung mit DPT-XXXXXXXX gekennzeichnet. Hierbei sind die X's Platzhalter für eine eigene Ticketnummer des externen Dienstleisters, welche bei diesem angefragt wird. Im System selbst, wird aber auch eine Ticketnummer von HADES für diese Art von Tickets vergeben.

Die am häufigsten auftreffenden Tickets sind die «Help-Desk-Tickets», welche aufgrund von Kundenanfragen bezüglich Fehlern an Hard- oder Software oder Peripherie in einer Schule von Mitarbeitern im Call-Center erstellt werden.

Der für den Empfehlungsdienst des Ticketsystems zu implementierende Ähnlichkeits-algorithmus muss in der Lage sein, diejenigen Tickets auszuschließen, welche nicht in die zuletzt genannte Kategorie «Help-Desk-Ticket» fallen. Der entsprechende Empfehlungsdienst muss hiermit umgehen können und den Dienst nur anbieten, wenn das Ticket dieser Kategorie zugeordnet ist.

Im Backend des Empfehlungsdienstes muss ein Skript ständig prüfen, ob neue Tickets vorhanden sind oder zu bereits vorhandenen Tickets neue Informationen hinzukamen, um auch diese bei den Empfehlungen zu berücksichtigen. Da beim Text Mining die Daten nicht in einer relationalen Datenbank vorliegen können, müssen diese Informationen im Vorfeld in Form von natürlich-

sprachlichen Dokumenten transferiert werden. Anschließend kann die Ähnlichkeit zwischen Paaren von diesen Dokumenten bestimmt werden.

Es gibt eine Vielzahl von Techniken, um Paare von Dokumenten zu identifizieren, die einander ähnlich sind. Diese unterscheiden sich hinsichtlich des Endziels, des betrachteten Korpus, der pro Dokument identifizierten Merkmale und Wortindikatoren und der Art diese zu komprimieren (vgl. Manku et al. 2007, S. 145). Hierauf wird im folgenden Kapitel näher eingegangen.

4. Anwendungskonzept und Entwicklung

Dieses Kapitel erläutert die Konzeption und die Implementierung eines Empfehlungsdienstes für Ticketsysteme. Hierzu wird eine Herangehensweise mit dem Python-Modul «scikit-learn[14]» und der Softwarelösung «JoBimText» der Language-Technology Gruppe der Universität Hamburg beschrieben und erprobt.

4.1 Konzept und Anwendungsarchitektur

Ausgangspunkt für das Konzept ist das Help-Desk-Fallbeispiel aus Kapitel 3 «Problemstellung und Anforderungen». Ziel ist es, einen content-basierten Empfehlungsdienst für das Ticketsystem HADES zu konstruieren und zu implementieren, der den Mitarbeitern des Call-Centers des Schul-Support-Services Empfehlungen zu ähnlichen Tickets bezogen auf das Anfrageticket bereitstellt, um so die Effizienz im Support zu steigern.

Hierzu soll vor allem das Backend realisiert werden und ein Implementierungssansatz für die Darstellung in der Ticketansicht geschaffen werden. Als «Backend» wird der Teil des Informationssystems bezeichnet, der für die Ausführung von Skripten, Abfragen und Aktionen im Hintergrund verantwortlich ist. Der Nutzer des Systems bekommt nur die Interaktion als Resultat der Hintergrundprozesse auf der grafischen Oberfläche des Systems mit, dem sogenannten «Frontend». Bezogen auf die Arbeit ist dies die Ticketansicht.

Die informationelle Grundlage für diesen Entwicklungsschritt bildet ein Datensatz an 7055 Tickets aus der Produktivumgebung des Ticketsystems HADES aus den Jahren 2015 und 2016 als Training-Set.

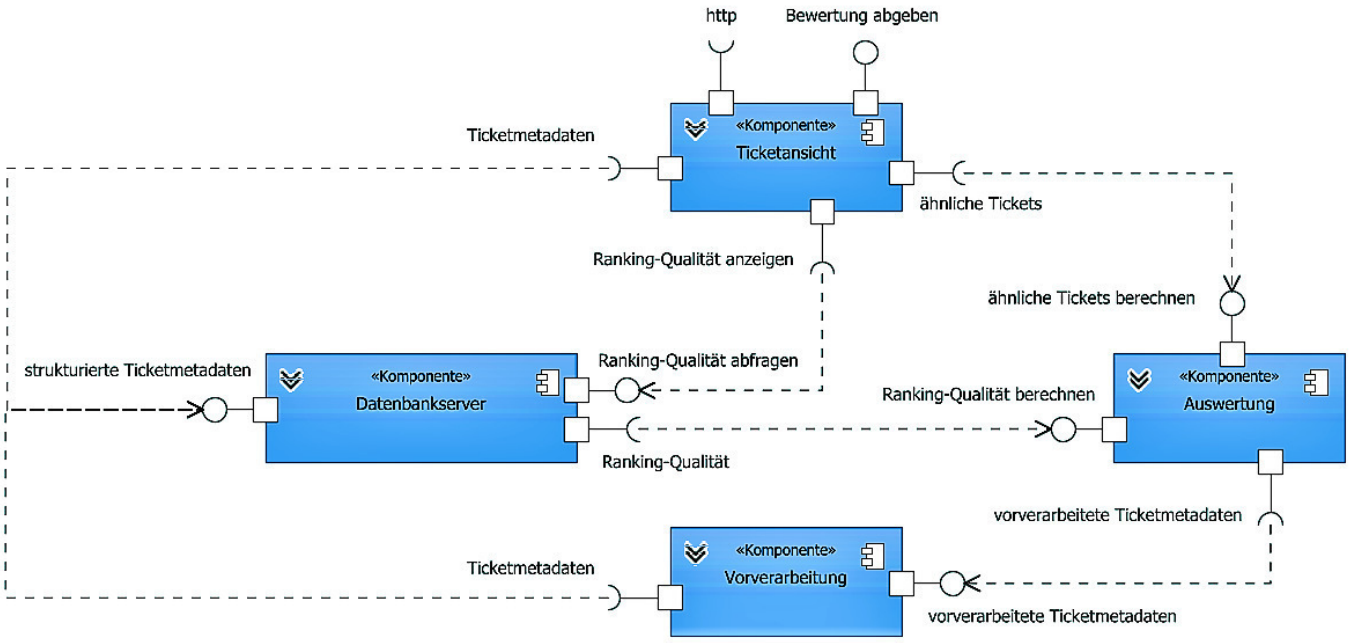

Abbildung 7: Anwendungsarchitektur

[14] Siehe hierzu http://scikit-learn.org/stable/index.html

In Abbildung 7 wird die Anwendungsarchitektur mit den einzelnen Komponenten und deren Schnittstellen und Abhängigkeiten zwischen diesen dargestellt. Hier unterscheidet man zwischen angebotenen Schnittstellen (□—○) und erforderlichen Schnittstellen (□—( ).

Die Ticketansicht soll die Ticketmetadaten in einem geeigneten Layout auf einer Weboberfläche mit http-Zugriff anzeigen, indem diese Komponente die erforderlichen Daten beim Aufrufen vom Datenbankserver abfragt. Zudem soll die Ticketansicht ein Ranking von ähnlichen Tickets anzeigen und für diese eine Bewertungsfunktion anbieten.

Hierfür sind allerdings zunächst ähnliche Tickets erforderlich, welche von einem Auswertungs-Modul mithilfe eines Skriptes berechnet werden. Dieses kann dann aufgrund der Bewertungen die Ranking-Qualität und somit die Qualität des Empfehlungsdienstes – und den empfohlenen Tickets der Ticketansicht – die Bewertungen in Form eines nDCG-Wertes über einen Datenbankserver zur Verfügung stellen.

Um die Auswertung vornehmen zu können, sind allerdings bereits verarbeitete Ticketmetadaten erforderlich, welche durch das Vorverarbeitungs-Modul zur Verfügung gestellt werden. Dieses Modul nutzt die, auf dem Datenbankserver liegenden, strukturierten Ticketmetadaten und führt darauf die in Abschnitt 2.2.2 bis einschließlich 2.2.4 beschriebenen Verfahren zur sprachlichen Aufbereitung, Identifizierung von Wortarten und der lexikalischen Expansion dieser Daten durch. Wie im vorangegangen Kapitel bereits erwähnt, wird von den Tickets nur diejenige Teilmenge betrachtet, welche in die Kategorie «Help-Desk-Ticket» fallen. Hierzu wird die Datenbank selektiv auf bestimmte Tags als Indikatoren solcher Tickets durchsucht und die vorverarbeiteten Ticketmetadaten in Form von Dokumenten auf dem Server speichert. Zur Berechnung der ähnlichen Tickets kann dann vom Auswertungsmodul auf eine Vektordarstellung dieser Ticketdokumente zurückgegriffen und dessen Kosinus-Ähnlichkeiten ermittelt werden.

4.2 Implementierung des Empfehlungsdienstes

4.2.1 Voraussetzungen und Server

Am Anfang der Arbeit stand das Konfigurieren eines Servers, auf dem alle benötigten Dienste laufen und somit eine Entwicklungsumgebung geschaffen werden kann. Dieser wurde als virtuelle Maschine im Rechenzentrum der Universität Hamburg erstellt, um jederzeit von überall darauf zugreifen zu können. Als Betriebssystem wird die Linux-Distribution Ubuntu 16.04 verwendet. Diese steht als Open-Source zur Verfügung und ist einfach in der Handhabung. Über den integrierten Paketmanager können Dienste und Anwendungen welche die Skripte benötigen, geladen werden.

Hauptaufgabe des Ubuntu-Servers ist die Bereitstellung von LAMP. Dies ist ein Akronym für **L**inux, **A**pache, **M**ySQL, **P**HP und stellt ein Paket dar, das einen Apache Web-Server mit PHP und einen MySQL-Datenbank-Server unter Linux zur Verfügung stellt. Zusätzlich ist phpMyAdmin installiert, welches über Apache den Datenbank-Server in PHP darstellen lässt. So ist eine benutzerfreundliche Interaktion mit der Datenbank möglich. Hiermit wurden anschließend die HADES Datensätze in zwei unterschiedliche Datenbanken importiert, «hades_db» und «hades_db_test».

Ebenfalls zur Grundausstattung gehört noch ein Python-Interpreter, um Python-Skripte auf dem Server laufen zu lassen. Die Skriptsprache Python wird genauer im nachfolgenden Abschnitt beschrieben.

4.2.2 Einbettung der algorithmischen Verfahren in Python

Um die Ähnlichkeiten zwischen zwei Tickets berechnen zu können, werden die im Konzept genannten Verfahren und Algorithmen aus dem Bereich des Natural Language Processing (NLP) verwendet. Die Skriptsprache, die hierzu eingesetzt wird ist Python in Version 2.7.

Python ist eine objektorientierte Programmiersprache. Sie unterstützt Module und Pakete, was die Modularität des Programms und die Wiederverwendung von Code fördert. Der Python-Interpreter und die umfangreiche Standardbibliothek sind für alle gängigen Plattformen kostenlos und mit Open-Source-Lizenz verfügbar.[15] Die Python-Programmiersprache ist eine der beliebtesten Sprachen für wissenschaftliches Rechnen. Dank seiner interaktiven Natur und seines reifenden Ökosystems aus wissenschaftlichen Bibliotheken ist es eine attraktive Wahl für die algorithmische Entwicklung und explorative Datenanalyse (vgl. Pedregosa et al. 2011, S. 2826 zitiert nach Dubois, 2007 und Milmann & Avaizis, 2011). Aus diesem Grund ist die Programmiersprache auch besonders zum Einsatz in der Computerlinguistik und somit der Verarbeitung natürlicher Sprache geeignet.

Die Python-Module, welche zur Entwicklung des Empfehlungsdienstes verwendet wurden, sind «nltk», «spaCy» und «sklearn» (scikit-learn). Diese werden nachfolgend vorgestellt.

Das Natural Language Toolkit (NLTK) definiert eine Infrastruktur, die zum Erstellen von NLP-Programmen in Python verwendet werden kann. Es bietet grundlegende Klassen für die Darstellung von Daten, die für die Verarbeitung natürlicher Sprache relevant sind. Das NLTK liefert Standardschnittstellen zum Ausführen von Aufgaben, wie zum Beispiel Teilen von Wortmarken, syntaktischem Parsen, Textklassifizieren und Standardimplementierungen für jede Aufgabe, die kombiniert werden können, um komplexe Probleme zu lösen (vgl. Bird et al. 2009).

Dieses Toolkit wurde in der Entwicklung des Empfehlungsdienstes zur sprachlichen Aufbereitung der Ticketdokumente im txt-Format im Vorverarbeitungs-Skript eingesetzt, da es speziell hierfür besonders geeignete Funktionen bereitstellt. Da das NLTK auf dem englischsprachigen «WordNet[16]» zur Lemmatisierung setzt, ist dieses hierfür leider nicht zu gebrauchen. Ebenfalls wird die Stoppwort-Reduktion und das POS-Tagging von NLTK nur auf Englisch angeboten. Die Herausforderung war, entsprechende Funktionen für die deutsche Sprache zu finden. Diese Herausforderung wurde durch «spaCy» gelöst. Dieses Modul ist zum Einsatz im maschinellem Lernen entwickelt worden und bietet ein deutschsprachiges Paket mit einem Lemmatizer für die deutsche Sprache an.

[15] Abgerufen am 22. Juni 2018 von https://www.python.org/doc/essays/blurb/

[16] Eine sehr große und umfangreiche lexikalisch-semantische Ressource für englische Sprache.

Das Natural Language Toolkit wurde anschließend nur noch zum Entfernen von Satz- und Sonderzeichen verwendet. Als Stoppliste wird eine sehr umfangreiche, öffentlich verfügbare Liste von Diaz (2016)[17] verwendet und die darin enthaltenen Stoppwörter, durch einen Abgleich mit den Ticketdokumenten, aus diesen entfernt.

In dem Vorverarbeitungs-Skript wurde zudem eine Funktion zur lexikalischen Expansion der Ticketdaten mithilfe von «JoBimText» eingebettet. Hiermit wird zunächst Part-Of-Speech Tagging der Ticketdaten vorgenommen, um anschließend genau definieren zu können, mit welchen Wortarten expandiert werden soll. Da die Softwarelösung der Language-Technology Gruppe eine REST[18]-API besitzt, können mittels http-GET-Anfrage an den Server, auf dem die Software gehostet ist, Daten zu Wortarten, Wortbedeutungen und Wortähnlichkeiten im semantischen Sinne, abgerufen und in Python weiterverarbeitet werden.

REST-Komponenten führen Aktionen an einer Ressource durch, indem sie eine Repräsentation verwenden, um den aktuellen oder beabsichtigten Status dieser Ressource zu erfassen und diese Repräsentation zwischen Komponenten zu übertragen. Eine Repräsentation ist eine Sequenz von Bytes plus Repräsentationsmetadaten zur Beschreibung dieser Bytes. Andere häufig verwendete, aber weniger genaue Namen für eine Darstellung sind «Dokument», «Datei» und «HTTP-Nachrichteninstanz» (vgl. Fielding 2000, S. 90).

Näheres zur Verwendung von «JoBimText»-Abfragen im Python-Skript wird im Kapitel 5 «Auswertung und Qualitätsentwicklung der Empfehlungen» beschrieben.

Der Inhalt eines Ticketdokumentes wird anhand eines Ticket-Beispiels in Tabelle 2 gezeigt.

Problem mit F-Prot [Schulbesuch von [...] am 15.07.14] Mir ist während des Besuchs aufgefallen, dass die Aktualisierung der Signaturen nicht richtig funktioniert oder unregelmäßig stattfindet. Die Clients sind auf Update via LAN (NAS) konfiguriert. Zuletzt haben sie die Signaturen Ende Mai erhalten. Der Ursache konnte ich heute nicht weiter nachgehen, da das zeitlich nicht möglich war (z.B. ist mir nicht bekannt, welcher PC das Internet-Update macht). Ich habe mit der AP nicht weiter über die Sache gesprochen. Die Freigabe auf dem NAS (\\nas\f-prot) existiert und ist erreichbar. Auch auf der Freigabe liegen nur Signaturdateien von Ende Mai. Client

Tabelle 2: Beispiel eines unverarbeiteten Ticketdokumentes

Nach der Ausführung des Vorverarbeitungs-Skriptes zur sprachlichen Vorverarbeitung der Ticketdokumente, sieht diese aus, wie in Tabelle 3 dargestellt.

Problem mit F Prot Schulbesuch von [...] am ich sein während der Besuch auffallen dass der Aktualisierung der Signaturen nicht richtig funktionieren oder unregelmäßig stattfinden Die Clients sein auf Update via LAN NAS konfigurieren Zuletzt haben ich der Signaturen enden Mai erhalten Der Ursache können ich heute nicht weit nachgehen da der zeitlich nicht möglich sein B sein sich nicht bekennen welch PC der Internet Update machen Ich habe mit der AP nicht weit über der Sache sprechen Die Freigabe auf der NAS nas f prot existieren und sein erreichbar Auch auf der Freigabe liegen nur Signaturdateien von enden Mai Client

Tabelle 3: Beispiel eines sprachlich-aufbereiteten Ticketdokumentes

[17] Abgerufen am 21. Juni 2018 von https://github.com/stopwords-iso/stopwords-de
[18] «REpresentational State Transfer»

Alle vorkommenden Wörter wurden mithilfe von «spaCy» lemmatisiert. Alle Sonderzeichen[19] und Stoppwörter aus der Liste von Diaz (2016) wurden entfernt.

Weiterhin wurde ein Auswertungs-Skript erstellt, welches die Berechnung und Ausgabe der ähnlichen Tickets und der Berechnung des nDCG-Wertes und dessen Eingabe in die Datenbank vornimmt. Dieses beinhaltet unter anderem den Scikit-Learn-Algorithmus zur Erstellung von tf-idf-Matrizen und der Berechnung der paarweisen Kosinus-Ähnlichkeiten.

Scikit-learn umfasst eine große Vielfalt an Machine-Learning-Algorithmen, sowohl überwacht als auch unbeaufsichtigt, unter Verwendung einer konsistenten, aufgabenorientierten Schnittstelle, wodurch ein einfacher Vergleich von Methoden für eine gegebene Anwendung ermöglicht wird. Da es sich auf das wissenschaftliche Python-Ökosystem stützt, kann es leicht in Anwendungen außerhalb des traditionellen Bereichs der statistischen Datenanalyse integriert werden (vgl. Pedregosa et al. 2011, S. 2829).

In dieser Arbeit, wird scikit-learn allerdings nicht für maschinelles Lernen verwendet, sondern seine eingebauten Funktionen ausgenutzt, wie zum Beispiel der «tf-idf-Vectorizer» und der «cosine-similarity»-Funktion unter Angabe einer Dokumentenmatrix für einen gegebenen Index.

Der mit scikit-learn implementierte Algorithmus basiert auf einem Ansatz von Mark Needham (2016)[20] zur Berechnung von Ähnlichkeiten zwischen wissenschaftlichen Berichten in der Informatik. Dieser kann der

Abbildung *8* entnommen werden.

```
################################################ SCIKIT-LEARN ALGORTIHM ################################################
#ref https://markhneedham.com/blog/2016/07/27/scitkit-learn-tfidf-and-cosine-similarity-for-computer-science-papers/

if("skl" in algorithm_to_use):
    # Iterate through the ticket directory and build an array containing the ticket
    import glob

    corpus = []
    indexincorpus=0
    for file in glob.glob("python/tickets/*.txt"):
        with open(file, "r") as ticket:
            corpus.append((file, ticket.read()))
        if(file in glob.glob("python/tickets/"+ticketid+".txt")):
            indexincorpus=(len(corpus)-1)


    # build a TF/IDF matrix for each ticket
    from sklearn.feature_extraction.text import TfidfVectorizer

    tf = TfidfVectorizer(analyzer='word', ngram_range=(1,3), min_df = 0)
    tfidf_matrix =  tf.fit_transform([content for file, content in corpus])


    # writes a function that will find the top n similar tickets based on cosine similarity
    from sklearn.metrics.pairwise import linear_kernel

    def find_similar(tfidf_matrix, index, top_n = int(top_k)):
        cosine_similarities = linear_kernel(tfidf_matrix[index:index+1], tfidf_matrix).flatten()
        related_docs_indices = [i for i in cosine_similarities.argsort()[::-1] if i != index]
        return [(index, cosine_similarities[index]) for index in related_docs_indices][0:top_n]
```

[19] Das in Tabelle 2 und Tabelle 3 enthaltene Element «[...]» symbolisiert einen Platzhalter für den Namen eines Mitarbeiters, welcher aus datenschutzrechtlichen Gründen nicht genannt werden kann.
[20] Abgerufen am 23. März 2018 von https://markhneedham.com/blog/2016/07/27/scitkit-learn-tfidf-and-cosine-similarity-for-computer-science-papers/

Abbildung 8: Screenshot - Scikit-Learn Algorithmus in Anlehnung an Mark Needham (2016)

Folgende Schritte werden dabei durchlaufen:

1. Es wird durch alle zuvor vom Vorverarbeitungs-Modul aus der Datenbank extrahierten und vorverarbeiteten Ticketdokumente iteriert und ein zweidimensionales Array bestehend aus jeweils dem Namen der Datei - in diesem Fall der Ticket-ID – und dem Inhalt des Ticketdokumentes erstellt. Das entstandene Array bildet das Dokumentenkorpus ab.
2. Anschließend wird mithilfe des tf-idf-Vectorizer aus dem Dokumentenkorpus eine tf-idf-Dokumentenmatrix erstellt.
3. Es wird eine Funktion definiert, die unter Angabe der Matrix paarweise Kosinus-Ähnlichkeiten berechnet und die Relationen dieser indiziert. Ruft man die Funktion für ein gegebenes Ticket auf – was passiert, wenn das Skript über die Weboberfläche beim Betrachten eines Tickets ausgeführt wird – dann werden über den Index die Top k ähnlichsten Tickets ausgegeben.

Das Skript wird dabei für jedes Ticket individuell mit mehreren Parametern aufgerufen. Im Aufruf wird spezifiziert,

- wie die Ticket-ID lautet, für welche die ähnlichen Tickets gelistet werden sollen,
- wie viele Top k Tickets im Ranking berücksichtigt werden sollen,
- welche Technologie verwendet werden soll, falls es mehrere gibt.

Beim letztgenannten Punkt wurden unterschiedliche Herangehensweisen getestet. Näheres hierzu ist im Kapitel 5 «Auswertung und Qualitätsentwicklung der Empfehlungen» beschrieben.

Die Ausgabe des Skriptes ist eine Abfolge von Ticket-ID des ähnlichen Tickets und dem Score der Kosinus-Ähnlichkeit, für die Top k ähnlichsten Tickets.

Ebenfalls wird im Auswertungs-Skript, wie bereits erwähnt, der nDCG-Wert berechnet. Hierzu wurde eine Python-Funktion zur Berechnung des DCG- und des nDCG-Wertes aus einem, als Open-Source verfügbaren Skript von White (2012)[21] für Ranking-Maße, verwendet und im Auswertungs-Skript eingebettet. Beim Aufrufen der Funktion «*ndcg_at_k(r, k)*» wird eine Liste der Bewertungen des betrachteten Tickets und die Anzahl der zu berücksichtigenden Top k ähnlichen Tickets übergeben.

Zuvor werden von der Datenbank die Bewertungen für das Ähnlichkeitsranking des betrachteten Tickets geladen (siehe nachfolgenden Abschnitt «4.2.3 Implementierung auf Datenbankebene») und in eine Liste geschrieben, welche von der Funktion verwendet werden kann.

Der resultierende Wert zwischen 0 und 1 wird anschließend in der Datenbank in Relation zum dazugehörigen Ticket gespeichert.

[21] Abgerufen am 08. Mai 2018 von https://gist.github.com/bwhite/3726239

Im Laufe der Arbeit wurde eine weitere Technologie namens «elasticsearch» ausgetestet. «elasticsearch» (ES) ist eine verteilte, skalierbare Echtzeit-Such- und Analyse-Engine. Diese ermöglicht es, Daten zu suchen, zu analysieren und zu erforschen (vgl. Gormley & Tong, 2015).


```
{
  "_index" : "tickets",
  "_type" : "ticket",
  "_id" : "15126",
  "_version" : 3,
  "found" : true,
  "_source" : {
     "Zusammenfassung" : "Problem mit F-Prot",
     "Tags" : "Client",
     "Ticket-ID" : "15126",
     "Endgerät" : "",
     "Beschreibung" : "[Schulbesuch von [...] am 15.07.14] Mir ist während des Besuchs aufgefallen, dass die Aktualisierung der Signaturen nicht richtig funktioniert oder unregelmäßig stattfindet. Die Clients sind auf Update via LAN (NAS) konfiguriert. Zuletzt haben sie die Signaturen Ende Mai erhalten. Der Ursache konnte ich heute nicht weiter nachgehen, da das zeitlich nicht möglich war (z.B. ist mir nicht bekannt, welcher PC das Internet-Update macht). Ich habe mit der AP nicht weiter über die Sache gesprochen. Die Freigabe auf dem NAS (\\nas\f-prot) existiert und ist erreichbar. Auch auf der Freigabe liegen nur Signaturdateien von Ende Mai."
    }
}
```


Tabelle 4: Beispiel eines elasticsearch-indizierten Tickets

Die Daten, die von ES verarbeitet werden, liegen im JSON-Format indiziert vor (siehe Tabelle 4). Die Herausforderung für die Arbeit mit den HADES-Ticketdaten hierbei ist, diese in das besagte Format zu konvertieren und die Ergebnisse von Abfragen in einem HADES-ähnlichen Format darstellbar zu machen.

ES verwendet ebenfalls eine REST-API, um Daten zu indizieren, zu löschen oder abzufragen. Das Ausgabeformat hierzu ist ebenfalls JSON.

Für Python gibt es ebenfalls ein ES-Modul, welches in das Skript importiert werden kann, um die API in vollem Umfang auch aus Python heraus anzusprechen. Beim Testen dieser Technologie wurde ein Index «tickets» angelegt, in denen die Ticket-ID's als ID's der Indexelemente vergeben und die Ticketdaten in das Feld «_source» hineingeschrieben wurden. Ebenso soll es mit ES genauso möglich sein, Synonyme in die Anfrage mit einzubeziehen und Wörter verschieden zu gewichten. Die http-GET-Abfragesyntax hierfür ist in der Tabelle 5 dargestellt. Mit dem «Boost»-Faktor, kann angegeben werden, wie hoch das jeweilige Wort im darüberstehenden Feld gewichtet werden soll.

Trotzdessen, dass ES ähnliche Funktionen bietet, wie die die in dieser Arbeit entwickelte Kombination aus «sklearn» und «JoBimText», wurde die Technologie in der Entwicklung nicht weiterverfolgt und die Konzentration aus Zeitgründen auf die Funktionalität von «sklearn» gelenkt. Nichts desto trotz hat die Verwendung von «elasticsearch» viele Vorteile und wäre eine weitere Möglichkeit ähnliche Tickets für den Empfehlungsdienst zu berechnen.

```
GET _search
{
 "query": {
  "bool": {
   "must": [
    {
     "match": {
      "Zusammenfassung": {
       "query": "Problem mit F-Prot",
       "auto_generate_synonyms_phrase_query": false
      }
     }
    },
    {
     "match": {
      "Beschreibung": {
       "query": "Update Location",
       "auto_generate_synonyms_phrase_query": false
      }
     }
    }
   ],
   "should": [
    {
     "match": {
      "Zusammenfassung": {
       "query": "F-Prot",
       "boost": 3
      }
     }
    }
   ]
  }
 }
}
```

Tabelle 5: Beispiel eines Elasticsearch Query's mit Synonym-Expansion und Wortgewichtung

4.2.3 Implementierung auf Datenbankebene

Die Struktur und die relevanten Datentabellen der HADES Datenbank wurden bereits in Abschnitt «2.1.3 Datenbankstruktur» im Kapitel 2 «Grundlagen» beschrieben.

Um das Ranking der ähnlichen Tickets bewerten zu können wurden der Datenbank zwei weitere Tabellen hinzugefügt (siehe Tabelle 6).

Tabelle	Inhalte
ticketmatchticket	Ticket-ID des aktuellen Tickets, Ticket-ID des referenzierten Tickets, Bewertung, Ähnlichkeitsscore
ndcg	Ticket-ID, NDCG-Wert des Ticketrankings

Tabelle 6: Ergänzte Datenbanktabellen für die Auswertung des Empfehlungsdienst

Die Tabelle «ticketmatchticket» ist eine Relation zwischen dem aktuellen Ticket und einem, im Ranking referenzierten Ticket mit den jeweiligen Ticket-ID's als Fremd-, und der Kombination aus aktuellem und referenzierten Ticket als Primärschlüssel. In ihr werden zudem die Bewertungen, welche beim Bewerten der einzelnen Rankingpositionen anfallen und einen Wert von -1 (nicht bewertet) oder 0 bis 2 haben können, und der Ähnlichkeitsscore, welcher als *float* vom Auswertungs-Modul zurückgegeben wird, gespeichert.

Die Tabelle «ndcg» dient dazu, den – aufgrund der Bewertung – vom Algorithmus berechneten nDCG-Wert zu speichern, um ihn in der Ticketansicht dynamisch laden und später auswerten zu können. Da jedem Ticket nur ein Wert zugeordnet ist – welcher aktualisiert wird, sollte das Ranking nachbewertet werden – ist die Ticket-ID dessen Primärschlüssel.

4.2.4 Visualisierung der Inhalte mit PHP

Um die Ticketinhalte während der Entwicklung darzustellen und eine Möglichkeit zu schaffen, das Ranking von ähnlichen Tickets später auf der Weboberfläche anzuzeigen, wurde mithilfe der Skriptsprache PHP - in Anlehnung an die Original-Weboberfläche von HADES und unter Verwendung von dessen Quellcode - diese weitestgehend nachgebaut und um ein paar Features ergänzt. Dies soll unter anderem auch eine spätere Integration in das Produktivsystem erleichtern.

Analog zu HADES wurden drei PHP-Skripte für die Entwicklung erstellt:

- index.php ist die Startseite des Testsystems und zeigt eine Übersicht aller Tickets gelistet mit allen Metainformationen, die für den Empfehlungsdienst relevant sind. Über diese Seite lässt sich ein entsprechendes Ticket auswählen oder ein neues Ticket erstellen.
- addticket.php ist ein kurzes Skript, welches ein HTML-Formular zum Anlegen eines neuen Tickets zeigt, das die relevanten Ticketmetainformation hierfür abfragt und über das sich dem neuen Ticket Tags zuordnen lassen. Über einen Submit-Button werden diese Daten dann der Datenbank hinzugefügt.
- editticket.php ist das für diese Arbeit relevanteste Skript. Dieses zeigt das eigentliche Ticket (Ticketansicht) für eine gegebene Ticket-ID an und lädt dynamisch die dazugehörigen Informationen zur Ticket-ID aus der Datenbank und schreibt sie in die entsprechenden Felder. Bestandteile der resultierenden Seite sind – in Anlehnung an HADES – der Ticketheader mit Ticket-ID und Zusammenfassung, die Ticketbeschreibung und die Tags. Zusätzlich werden auch noch betroffene Endgeräte angezeigt, sollten diese in den Ticketinformationen vorhanden sein. Im Folgenden betrachten wir ausschließlich dieses PHP-Skript.

Diese PHP-Skripte wurden zusammen mit den Python-Skripten für den Zugriff von außerhalb der virtuellen Maschine im Document-Root des Web-Servers abgelegt und aufgrund der sensiblen Daten der Tickets mit einem Passwort versehen.

Der Aufruf des Python-Skriptes zur Berechnung der ähnlichen Tickets wird in editticket.php mit dem «shell_exec()»-Befehl auf dem Server ausgeführt, die Ausgabe des Skriptes verarbeitet und in eine Tabelle geschrieben. Diese wird unterhalb der Ticketbeschreibung eingefügt und ist somit direkt beim Aufruf des Tickets für die Mitarbeiter sichtbar. Das PHP-Skript selbst führt keine Berechnungen mehr aus, sondern wird lediglich zur Datenabfrage und Ausgabe verwendet. So wird beispielsweise bei der Tabelle durch die Ausgabe des Python Skriptes iteriert und jeweils in die Felder der Tabelle geschrieben.

Diese Felder sind die Ticket-ID, die Zusammenfassung, die Ticketbeschreibung und der Ähnlichkeits-Score. Ebenso ein Feld, in den der Rückgabewert der Bewertung geschrieben wird und ein

Feld, um die aktuelle Qualität des referenzierten Tickets zum aktuellen Tickets mittels drei unterschiedlicher Radio-Buttons zu bewerten, gefolgt von einem Submit-Button, um die Bewertung an die Datenbank zu schicken und einem Reset-Button, um diese wieder aus der Datenbank zu löschen, sollte man versehentlich falsch bewertet haben. Auch existiert ein Feld, in welches der nDCG-Wert des Tickets geladen wird, um diesen beim Ticketaufruf zu visualisieren.

Dabei werden Zusammenfassung und Beschreibung des Tickets anhand der Ticket-ID aus der Datenbank geladen. Den Bewertungsfeldern kommt ausschließlich bei der Auswertung eine Rolle zu. In der späteren Ticketansicht der Produktivumgebung werden diese nicht mehr sichtbar sein.

Unter der Tabelle mit den ähnlichen Tickets folgen anschließend noch die Ticketereignisse des betrachteten Tickets zur Erschließung des Kontextes und der Historie, welche vom Skript von der Datenbank abgerufen und in eine Tabelle geschrieben werden.

5. Auswertung und Qualitätsentwicklung der Empfehlungen

Im Weiteren wird die Qualität der Anwendung des in Kapitel 4 «Anwendungskonzept und Entwicklung» konzipierten Empfehlungsdienstes ausgewertet und diskutiert.

Die Auswertung geschah in Hinblick darauf, wie sich die Qualität des Rankings verändert, wenn man

(1) unterschiedliche Metadaten alleine oder im Zusammenhang mit anderen berücksichtigt,

(2) gewisse Wörter unterschiedlich gewichtet,

(3) Synonym-Expansion vornimmt oder

(4) die Anzahl der k berücksichtigten ähnlichen Tickets variiert.

Ebenfalls wurden im Verlauf der Entwicklung unterschiedliche Technologien ausprobiert. Darunter fällt neben «scikit-learn» und «JoBimText» noch «elasticsearch». Hierauf wird später noch eingegangen.

Im Abschnitt «4.2.4 Visualisierung der Inhalte mit PHP» im vorangegangenem Kapitel wurde bereits erklärt, wie die Rankingtabelle für ähnliche Tickets erstellt wird. Hierbei sind für die Auswertung relevante Spalten mitimplementiert worden, die eine Möglichkeit bieten die aktuelle Qualität des Rankings zu erfassen und zu beurteilen (siehe Abbildung 9 und Abbildung 10). Mithilfe dieser Funktion kann eine Qualitätssteigerung oder -abnahme in Hinblick auf die oben genannten Punkte (1) bis (4) untersucht werden, um so das bestmöglichste Ranking zu erhalten.

Die Tickets werden von einem Benutzer dahingehend bewertet, ob sie hilfreich im Sinne der Ähnlichkeit zum aktuellen Ticket sind. Die Optionen sind in Form von Radio-Buttons dargestellt, wovon je referenziertem Ticket nur einer auswählbar ist.

Man kann alle oder nur einige Ticketreferenzen bewerten. Entsprechend besteht die Möglichkeit noch nicht bewertete Referenzen nachzubewerten (siehe Abbildung 10).

Die drei Bewertungsoptionen sind mit einem Wert von 0 (Nein) bis 2 (Ja) beziffert, welche in der Datenbank gespeichert werden. Die nDCG-Funktion im Auswertungs-Modul kann sich dieser Werte dann bedienen und den nDCG-Wert für das Ticket im aktuellen Bewertungszustand berechnen, welcher in der Ticketansicht angezeigt wird.

So kann man beispielsweise beobachten, wie sich der nDCG-Wert des betrachteten Tickets ändert, wenn man nur teilweise Bewertungen abschickt und entsprechend die Konsistenz dieser Funktion überprüfen.

Als Test-Set für die Auswertung wurde ein Datensatz an 4702 Tickets aus der Produktivumgebung des Ticketsystems HADES aus dem Zeitraum 2017 bis Mai 2018 verwendet.

#15126: Problem mit F-Prot

Tags: Client

Beschreibung:

[Schulbesuch von [redacted] am [redacted]] Mir ist während des Besuchs aufgefallen, dass die Aktualisierung der Signaturen nicht richtig funktioniert oder unregelmäßig stattfindet. Die Clients sind auf Update via LAN (NAS) konfiguriert. Zuletzt haben sie die Signaturen Ende Mai erhalten. Der Ursache konnte ich heute nicht weiter nachgehen, da das zeitlich nicht möglich war (z.B. ist mir nicht bekannt, welcher PC das Internet-Update macht). Ich habe mit der AP nicht weiter über die Sache gesprochen. Die Freigabe auf dem NAS (\\\\nas\f-prot) existiert und ist erreichbar. Auch auf der Freigabe liegen nur Signaturdateien von Ende Mai.

Diese Tickets sind vielleicht hilfreich:

Ticket-Nr.	Zusammenfassung	Beschreibung	hnlichkeit	Bewertung	Hilfreich?
#22016 (edit\ticket.php?ticketid=22016)	F-Prot LAN-Update Problem (edit\ticket.php?ticketid=22016)	- kopieren der Antivir.def vom \\\\nas\f\prot auf d:\F-Prot Antivirus for Windows F-Prot LAN-Update zieht vom NAS keine Updates. Die Antivir.def auf dem NAS ist aktuell. Update PC - 5565HP03A0001 (PC-R). Beim vor Ort Termin bearbeiten: - Ticket-ID: #22015 [Changeprozess 2016](->212263) - Nacharbeiten - Kommunikationsticket - Ticket-ID: #21879 SmartBoards - Bildausgabe auf Klonen einstellen - Ticket-ID: #21683 NAS Migration - Ticket-ID: #22016 F-Prot LAN-Update Problem - Ticket-ID: #22017 Software Knobbelland - Nachinstallation - Ticket-ID: #21689 Changeabnahme - Unterschrift (edit\ticket.php?ticketid=22016)	98.52398983029289	Nicht bewertet!	◉Ja ○Teilweise ○Nein
#18727 (edit\ticket.php?ticketid=18727)	F-Prot LAN-Update korrekt einrichten (edit\ticket.php?ticketid=18727)	F-Prot LAN-Update ist nicht richtig eingerichtet worden. Die Signatur ist aktuell, auch auf dem Client, aber F-Prot gibt an, die Datei sei *missing*. Dieses ist anscheinend beim Change nicht aufgefallen. (edit\ticket.php?ticketid=18727)	98.42792835414937	Nicht bewertet!	○Ja ○Teilweise ○Nein
#16072 (edit\ticket.php?ticketid=16072)	Klonen eines neuen Rechner als Smartboard-PC (edit\ticket.php?ticketid=16072)	Die Schule schafft einen neuen Smartboard und dazu einen neuen Rehner. Smartboard soll ende Mai geliefert werden.Der neue Rechner soll mit dem Image in der Schule geklont werden. (edit\ticket.php?ticketid=16072)	98.11159031486628	Nicht bewertet!	○Ja ○Teilweise ◉Nein
#17929 (edit\ticket.php?ticketid=17929)	Ein PC stürzt unregelmäßig ab (edit\ticket.php?ticketid=17929)	Wir haben einen PC, der immer wieder völlig unregelmäßig abstürzt. Ist ärgerlich, da es ein Smartboardrechner ist und dann immer die Arbeitsergebnisse verschunden sind. Ich dachte zuerst es wäre vielleicht ein Hardware-Problem, hatte den Techniker von Cancom da. Der hat einen Ramstick getauscht, aber das hat keine Besserung gebracht. Vielleicht ist es ja doch ein Softwareproblem? In den Fehlermeldungen gibt es immer einen F-Prot-Fehler - kann das damit zutun haben? Eine Lösung des Problems wäre wirklich klasse, da die Kollegin das Smartboard so nicht richtig nutzen kann. Die Icons unten rechts in der Menüzeile werden auch nicht richtig angezeigt - vielleicht ist das auch ein Symptom des Problems... Bitte ebenfalls bearbeiten: #17949 - Defekter W-Lan Drucker (edit\ticket.php?ticketid=17929)	97.96182388187283	Nicht bewertet!	○Ja ◉Teilweise ○Nein
#18744 (edit\ticket.php?ticketid=18744)	Ein HP-Drucker druckt nur unregelmäßig. (edit\ticket.php?ticketid=18744)	Außerdem druckt der HP Farbdrucker nur unregelmäßig. Ich weiß nicht woran es liegt. Wenn man ihn kurz ausgestellt hat, druckt er einen Auftrag und dann druckt er nicht mehr. (edit\ticket.php?ticketid=18744)	97.73604120279214	Nicht bewertet!	○Ja ○Teilweise ○Nein

Normalized Discounted Cumulative Gain (nDCG): 1

Bewertungen abschicken

Abbildung 9: Beispiel Ticketranking ohne Bewertung

#15126: Problem mit F-Prot

Tags: Client

Beschreibung:

[Schulbesuch von [redacted] am [redacted]] Mir ist während des Besuchs aufgefallen, dass die Aktualisierung der Signaturen nicht richtig funktioniert oder unregelmäßig stattfindet. Die Clients sind auf Update via LAN (NAS) konfiguriert. Zuletzt haben sie die Signaturen Ende Mai erhalten. Der Ursache konnte ich heute nicht weiter nachgehen, da das zeitlich nicht möglich war (z.B. ist mir nicht bekannt, welcher PC das Internet-Update macht). Ich habe mit der AP nicht weiter über die Sache gesprochen. Die Freigabe auf dem NAS (\\\\nas\\f-prot) existiert und ist erreichbar. Auch auf der Freigabe liegen nur Signaturdateien von Ende Mai.

Diese Tickets sind vielleicht hilfreich:

Ticket-Nr.	Zusammenfassung	Beschreibung	hnlichkeit	Bewertung	Hilfreich?
#22016 (editticket.php?ticketid=22016)	F-Prot LAN-Update Problem	- kopieren der Antivir.def vom \\\\nas\\fprot auf d:\\F-Prot Antivirus for Windows F-Prot LAN-Update zieht vom NAS keine Updates. Die Antivir.def auf dem NAS ist aktuell. Update PC - 5565HP03A0001 (PC-R). Beim vor Ort Termin bearbeiten: - Ticket-ID: #22015 [Changeprozess 2016](->212623) - Nacharbeiten - Kommunikationsticket - Ticket-ID: #21879 SmartBoards - Bildausgabe auf Klonen einstellen - Ticket-ID: #21683 NAS Migration - Ticket-ID: #22016 F-Prot LAN-Update Problem - Ticket-ID: #22017 Software Knobbelland - Nachinstallation - Ticket-ID: #21689 Changeabnahme - Unterschrift (editticket.php?ticketid=22016)	98.52398983029289	Bewertet (2)	
#18727 (editticket.php?ticketid=18727)	F-Prot LAN-Update korrekt einrichten (editticket.php?ticketid=18727)	F-Prot LAN-Update ist nicht richtig eingerichtet worden. Die Signatur ist aktuell, auch auf dem Client, aber F-Prot gibt an, die Datei sei *missing*. Dieses ist anscheinend beim Change nicht aufgefallen. (editticket.php?ticketid=18727)	98.42792835414937	Nicht bewertet!	○Ja ○Teilweise ○Nein
#16072 (editticket.php?ticketid=16072)	Klonen eines neuen Rechner als Smartboard-PC (editticket.php?ticketid=16072)	Die Schule schafft einen neuen Smartboard und dazu einen neuen Rehner. Smartboard soll ende Mai geliefert werden.Der neue Rechner soll mit dem Image in der Schule geklont werden. (editticket.php?ticketid=16072)	98.11159031486628	Bewertet (0)	
#17929 (editticket.php?ticketid=17929)	Ein PC stürzt unregelmäßig ab (editticket.php?ticketid=17929)	Wir haben einen PC, der immer wieder völlig unregelmäßig abstürzt. Ist ärgerlich, da es ein Smartboardrechner ist und dann immer die Arbeitsergebnisse verschunden sind. Ich dachte zuerst es wäre vielleicht ein Hardware-Problem, hatte den Techniker von Cancom da. Der hat einen Ramstick getauscht, aber das hat keine Besserung gebracht. Vielleicht ist es ja doch ein Softwareproblem? In den Fehlermeldungen gibt es immer einen F-Prot-Fehler - kann das damit zutun haben? Eine Lösung des Problems wäre wiktich klasse, da die Kollegin das Smartboard so nicht richtig nutzen kann. Die Icons unten rechts in der Menuzeile werden auch nicht richtig angezeigt - vielleicht ist das auch ein Symptom des Problems... Bitte ebenfalls bearbeiten: #17949 - Defekter W-Lan Drucker (editticket.php?ticketid=17929)	97.96182388187283	Bewertet (1)	
#18744 (editticket.php?ticketid=18744)	Ein HP-Drucker druckt nur unregelmäßig. (editticket.php?ticketid=18744)	Außerdem druckt der HP Farbdrucker nur unregelmäßig. Ich weiß nicht woran es liegt. Wenn man ihn kurz ausgestellt hat, druckt er einen Auftrag und dann druckt er nicht mehr. (editticket.php?ticketid=18744)	97.73604120279214	Nicht bewertet!	○Ja ○Teilweise ○Nein
		Normalized Discounted Cumulative Gain (nDCG): 0.51675			Bewertungen abschicken

Abbildung 10: Beispiel Ticketranking mit Bewertung

Im Folgenden werden die Anpassungen des Algorithmus in den Punkten (1) bis (4) in Hinblick auf die Ranking-Qualität – welche mit dem nDCG-Wert beziffert werden kann – ausgewertet. Hierfür wurde der durchschnittliche nDCG-Wert von je 30 willkürlich aus diesem Test-Set ausgewählten Tickets über alle sinnvollen Kombinationen der Anpassungen (1) bis (4) berechnet, indem die Empfehlungen für jedes dieser 30 Tickets von einem Nutzer händisch und über die Weboberfläche bewertet wurden. Je mehr Tickets in der richtigen Reihenfolge dem betrachteten Ticket ähnlich waren, desto größer ist der nDCG-Wert und desto besser ist die Qualität des Empfehlungsdienstes.

Zunächst (1) wurde eine Variation der Ticketmetadaten für den Gebrauch im Ticketdokument getestet. Hierzu ist das Vorverarbeitungs-Skript so angepasst worden, dass sich über den Aufruf die gewünschten Metainformationen mitgeben lässt, welche in das Ticketdokument des jeweiligen Tickets geschrieben werden soll. So wurde für jede Kombination dieses Skript einmal aufgerufen, anschließend die ähnlichen Tickets für ein gegebenes Ticket berechnet und bewertet, um den nDCG-Wert zu bestimmen.

Dabei sind einige Kombinationen ausgelassen worden. Auf Grundlage der Ergebnisse von Silva et al. (2018), dass sich durch die Hinzunahme der Kurzbeschreibung bzw. der Zusammenfassung des Tickets die besten Modelle bilden lassen, wurden nur Kombinationen betrachtet, welche die Zusammenfassung enthielten. Des Weiteren wurde das Endgerät, sollte eines im Ticket angegeben sein, immer dem Ticketdokument hinzugefügt. Dies scheint sinnvoll zu sein, da weniger als 10% der Tickets in der Datenbank überhaupt eine Angabe des Endgeräts besitzen und diese bei der Mehrheit aller Tickets nicht berücksichtigt wird, da keine vorhanden ist.

Als nächstes (2) wurde getestet, wie sich die Qualität des Tickets ändert, wenn man verschiedene Wörter unterschiedlich gewichtet. Hierzu wurden für die Kombinationen aus (1) exemplarisch jeweils die Nomina doppelt gewichtet. Dies wurde realisiert, indem mithilfe von «JoBimText» für die Wörter im Inhalt der Ticketdatei «Part-of-Speech (POS) Tagging» vorgenommen wurde und die erkannten Nomina nochmals dem Ticketdokument hinzugefügt. Hierfür wurde eine Anfrage an die REST-API des JoBimText-Servers gesendet, welche für Nomen (#NN), Verben (#VV) und Adjektive (#ADJD) ausgeführt wurde und die Antwort des Servers in Python analysiert wurde. Beinhaltet die Antwort für ein gegebenes Anfragewort mit dem Tag #NN kein Wort mit einer Übereinstimmung von 100%, wird das Wort nicht als Nomen gewertet und folglich auf Verben geprüft und so wurde weiter verfahren. Anschließend wurde hierauf ebenfalls der Ähnlichkeitsalgorithmus angewendet und die Ergebnisse bewertet.

Des Weiteren (3) wurde jeweils für jede Variation des Ticketdokumentes Synonym-Expansion vorgenommen, das heißt für alle enthaltenen Nomina jeweils eine http-GET-Anfrage an «JoBimText» geschickt und ähnliche Wörter abgefragt. Die Antwort wurde in Python dann wieder so verarbeitet, dass das Synonym als String resultiert und an das Ticketdokument angehängt wurde.

Für die Untersuchungen in Punkt (4) wurden jeweils die zu berücksichtigenden Rankingergebnisse für $k = 5$ und $k = 10$ variiert und für alle Kombinationen getestet.

#	(1)				(2)	(3)	Ø nDCG
	Zusammenfassung	Beschreibung	Tags	Endgerät	Wortge-wichtung	Synonym-Expansion	
1	X	X		X	X	X	0,619638500
2	X	X		X	X		0,607336167
3	X	X	X	X	X	X	0,598962500
4	X	X	X	X	X		0,583451233
5	X	X	X	X			0,545560167
6	X	X		X		X	0,543186667
7	X	X		X			0,523471167
8	X	X	X	X		X	0,513476767
9	X			X		X	0,368857267
10	X			X			0,358259200
11	X			X	X		0,349574567
12	X			X	X	X	0,345850667
13	X		X	X			0,315886567
14	X		X	X	X	X	0,315873733
15	X		X	X	X		0,309722567
16	X		X	X		X	0,282553233

Tabelle 7: Resultate zur Ranking-Qualitätsoptimierung für Top k=5

#	(1)				(2)	(3)	Ø nDCG
	Zusammenfassung	Beschreibung	Tags	Endgerät	Wortge-wichtung	Synonym-Expansion	
1	X	X		X	X	X	0,596817867
2	X	X		X	X		0,594791933
3	X	X	X	X	X	X	0,588184400
4	X	X		X			0,586386400
5	X	X	X	X	X		0,546378567
6	X	X	X	X			0,528534133
7	X	X		X		X	0,506085167
8	X	X	X	X		X	0,484273000
9	X		X	X	X	X	0,374651867
10	X			X	X		0,373822667
11	X		X	X			0,372486067
12	X			X	X	X	0,356939367
13	X			X		X	0,349759100
14	X			X			0,349440200
15	X		X	X	X		0,339551667
16	X		X	X		X	0,319069133

Tabelle 8: Resultate zur Ranking-Qualitätsoptimierung für Top k=10

Die Ergebnisse dieser Untersuchungen sind für die Anpassungen in Punkt (4) für $k = 5$ in Tabelle *7* und für $k = 10$ in Tabelle *8* dargestellt. Diese sind mit den Wahrheitswerten der Kombinationen von Anpassung (1) «Variation der Ticketmetadaten», Anpassung (2) «Wortgewichtung» und (3) «Synonym-Expansion» für das jeweils beste Ergebnis absteigend sortiert.

Die ertragsreichste Anpassung des Ticketdokumentes zur Verwendung im Algorithmus ist demnach die Kombination aus Ticketzusammenfassung und -beschreibung, der Angabe des Endgeräts, soweit diese vorhanden ist, der doppelten Gewichtung von Nomen in diesen vorverarbeiteten Metadaten und der Expansion des Ticketdokumentes mit Synonymen, der in den Metadaten enthaltenen Wörter und der Betrachtung der Top-5 Resultate mit dem größten Ähnlichkeitsscore.

Betrachtet man die drei jeweils ertragreichsten Resultate in Tabelle *7* und Tabelle *8*, ist auffällig, dass die Wahrheitswerte für $k = 5$ und $k = 10$ identisch sind und die Diskrepanz zwischen den Werten sehr gering, wobei die Werte für $k = 5$ minimal besser ausfallen. Daraus lässt sich ableiten, dass bei den hier vorliegenden Ergebnissen auf eine kleinere Auswahl an Rankingergebnissen tendenziell qualitativ-bessere Empfehlungen, gemessen am nDCG-Wert – also implizit der Rankingreihenfolge – angezeigt werden, als bei einer breiteren Spanne an Ergebnissen.

Ebenfalls ist aus den Ergebnissen ersichtlich, dass die Hinzunahme der Beschreibung zum Ticketdokument, im Gegensatz zu den Tags, ein wesentliches Einflussmerkmal auf die Güte der Empfehlungen darstellt. So fällt der Wert beim Weglassen der Beschreibung, im direkten Vergleich zum letzten Wert unter Hinzunahme der Beschreibung, bei $k = 10$ um 0,11 Punkte und bei $k = 5$ sogar um fast 0,14 Punkte. Hieraus lässt sich schlussfolgern, dass der Kontext eines Tickets, welcher in der Beschreibung vorhanden ist, die Empfehlungsqualität um ein Vielfaches steigert.

Beim Bewerten des Rankings ist bezüglich der inhaltlichen Qualität des Rankings aufgefallen, dass bei der Betrachtung von Top-10 Ticketempfehlungen tendenziell inhaltlich bessere Resultate empfohlen wurden, die zur Lösung eines bestehenden Problems im Anfrageticket beitragen können, welche allerdings verstreut über das ganze Ranking positioniert waren und deshalb zu einem schlechteren nDCG-Wert führten.

Weiterhin ist eine Quelle für mangelhafte Ranking-Ergebnisse, dass Mitarbeiter im Call-Center häufig weitere Aufgaben, die beim Kunden vor Ort zu erledigen sind und keinen Bezug zum aktuell betrachteten Anliegen haben, mit in die Ticketbeschreibung eingeben, anstatt ein separates Ticket hierfür anzulegen. So werden unter Betrachtung der Ticketbeschreibung weniger relevante Ergebnisse für das hauptsächliche Problem empfohlen.

Eine Herausforderung, die sich bei der Bewertung ergeben hat und bei manchen Tickets für schlechte Empfehlungen sorgte ist, dass einige Wörter, wie zum Beispiel der Name des von 3S angepassten Microsoft Windows 10 Musterimages «Butterscotch» für Schulen oder der Begriff des «Auditmode», der erst in Windows 10 eingeführt wurde, in früheren Tickets nie genannt wurden, da diese Begriffe zu der Zeit noch nicht existierten und somit auch nicht im Vokabular des Ticketdokumentenkorpus im Training-Set vorhanden sind. Entsprechend konnten für die betrachteten Tickets, welche solche Begriffe enthalten, keine passenden Empfehlungen gemacht werden.

Weiterhin ist auffällig, betrachtet man die Ähnlichkeiten zwischen dem aktuellen und dem referenzierten Ticket genauer, dass bei Tickets mit Peripherie-Problemen, wie beispielsweise defekte Drucker oder Smartboards das Ranking ein gutes Ergebnis erzielte, jedoch bei Soft- oder Hardware-Probleme, weniger eindeutig-ähnliche Tickets angezeigt wurden. Eine Möglichkeit diese Rankings zu optimieren, wäre, die Tags, welche die Probleme klassifizieren, stärker zu gewichten und so die Empfehlungen kategorieabhängig zu machen.

Im Allgemeinen lässt sich hier beurteilen, dass die Güte der vorhandenen Tickets ebenfalls eine wichtige Rolle zur Ähnlichkeitsberechnung beiträgt, da viele Tickets teilweise sehr ungenaue Zusammenfassungen haben. So lässt sich auch begründen, dass bei einem Ticketdokument, welches nur aus der Zusammenfassung eines Tickets und Endgerät – sofern eines angegeben ist – besteht, wenig Nutzen für den Ähnlichkeitsalgorithmus hat.

Eine weitere Herausforderung, die sich bei der Bewertung der Tickets ergeben hat, ist, dass bei $k = 5$ nicht alle ähnlichen Tickets angezeigt wurden, die vielleicht zur Lösung des Problems notwendig gewesen wären. Dabei gäbe es wissentlich noch mindestens ein weiteres zutreffendes Ergebnis, welches allerdings nicht im betrachteten Ranking aufgeführt war, weil die Anzahl der betrachteten Resultate zu gering war. So wurde dieses Ergebnis erst bei $k = 10$ sichtbar. Ein gleicher Effekt kann aber auch bei $k = 10$ auftreten, sodass eventuell ein Wert von $k = 20$ ausgetestet werden müsste, um eine weitere passende Empfehlung zu erhalten. Die Frage, die sich hier auftut, ist demnach, inwiefern der Algorithmus verändert werden müsste, um genau die richtige Anzahl an Ergebnissen anzuzeigen, sodass ein möglicherweise wichtiges, ähnliches Ticket, welches eventuell sogar eine Lösung für ein bestehendes Problem beinhaltetet hätte, nicht übersehen wird.

6. Ausblick

Das Kapitel liefert einen Ausblick auf weiterführende Entwicklungsmöglichkeiten und Maßnahmen für den Empfehlungsdienst im Ticketsystem.

Im Kapitel 4 «Anwendungskonzept und Entwicklung» wurde ein Ansatz mithilfe von Scikit-Learn und «JoBimText» erläutert und dargestellt und dieser im Kapitel 5 «Auswertung und Qualitätsentwicklung der Empfehlungen» ausgewertet und diskutiert.

Weiterführend gibt es noch eine Menge anderer Technologien, mit denen sich Ticketähnlichkeiten auf effiziente Art und Weise hin berechnen lassen könnten, welche im Rahmen der Arbeit allerdings nicht untersucht werden konnten.

Eine dieser Technologien ist das bereits genannte «elasticsearch», dessen Möglichkeiten im Vergleich zum Ansatz des Autors, im vorigen Kapitel bereits erwähnt worden und dessen Einsatz und ein Implementierungssansatz bereits während der Entwicklung getestet wurde. Festzuhalten ist hier, dass «elasticsearch» die gleichen Funktionen, wie der, in dieser Arbeit präsentierte Ansatz, bietet, wie zum Beispiel der lexikalischen Expansion und dem Scoring und deshalb das gleiche Potenzial besitzt, im Rahmen eines Empfehlungsdienstes für das Ticketsystem HADES eingesetzt zu werden.

Weiterführende Maßnahmen zur Optimierung des in dieser Arbeit entwickelten Empfehlungsdienstes, können zum Beispiel eine Verbesserung der Stoppliste, durch ergänzen bestimmter, semantisch nicht notwendiger Wörter oder die Verwendung andere Python-Module zur natürlichsprachlichen Textverarbeitung, sein. Diese Maßnahmen gilt es noch zu testen und die resultierende Ranking-Qualität zu evaluieren.

Es ist denkbar, mithilfe der in dieser Arbeit verwendeten Methoden der natürlich-sprachlichen Textverarbeitung unter Verwendung von Feature-Extraktionen aus den verarbeiteten Ticketdateien und maschinellem Lernen mit Scikit-Learn den Empfehlungsdienst dahingehend zu erweitern, dass nicht mehr nur ähnliche Tickets, sondern direkt auch die darin enthaltene Lösungen für ein aktuelles Problem im Anfrageticket, angezeigt werden, um so die maximale Effizienz im Kundensupport zu schaffen.

7. Zusammenfassung

In dieser Arbeit wurde auf der Grundlage von realen Help-Desk-Tickets aus dem Ticketsystem HADES des Hamburger *3S Schul-Support-Services* ein Empfehlungsdienst für ähnliche Help-Desk-Tickets implementiert und gezeigt, dass Textverarbeitung bzw. textverarbeitende Algorithmen bei der Analyse von Ticketsystemen in Hinblick auf Ähnlichkeiten zwischen einzelnen Anfragetickets eine bedeutende Rolle spielen und großen Einfluss auf die Ranking-Qualität des Empfehlungsdienstes haben.

Die Ergebnisse legen dar, dass die Modellierung der lexikalischen Grundlage textverarbeitender Technologien ausschlaggebend für dessen Verwendung in einem Empfehlungsdienst für Ticketsysteme und die Qualität dessen Empfehlungen ist. Demnach führen Synonym-Expansion und Wortgewichtung einzelner Wörter oder Terme von Ticketmetadaten, wie Ticketzusammenfassung/-kurzbeschreibung, Ticketbeschreibung und Endgeräten, auf die in den Tickets Bezug genommen wird, zu einem sehr genauen Ergebnis mit qualitativ hochwertigen Empfehlungen. Sie zeigen aber auch, dass die Qualität der Empfehlungen von der Genauigkeit und dem Umfang der Ticketmetadaten stark abhängig ist, welche wiederum von den Mitarbeitern, welche das Ticket anlegen und bearbeiten beeinflusst wird.

Bezogen auf die Fragestellung lässt sich festhalten, dass sich das Python-Modul «sklearn» und dessen algorithmischen Verfahren zur Erstellung von Term-Dokument-Matrizen und Berechnung von Kosinus-Ähnlichkeiten sehr gut für die Verwendung in einem Empfehlungsdienst eignen. Die lexikalische Grundlage hierfür kann mithilfe diverser Vorverarbeitungsschritte wie dem Entfernen von Stoppwörtern, Satz- und Sonderzeichen, der Lemmatisierung und der Identifizierung von Wortarten (engl.: «Part-of-Speech (POS) Tagging») gebildet und unter Verwendung quantitativer Bewertungsmaßstäbe, wie dem «Normalized Discounted Cumulative Gain» (nDCG), modelliert und modifiziert werden.

Der Ausblick zeigt, dass es noch viele weitere Möglichkeiten gibt den Empfehlungsdienst weiterzuentwickeln und zu optimieren. Dabei wird wohl kein Weg am maschinellem Lernen vorbeiführen, um beispielsweise die Ticketmetadaten selbst aufzuwerten und auf Grund von Bewertungen die Relevanz von Rankingergebnissen einzuschätzen. Grundlage für hierfür und für weitere Prozesse bildet jedoch immer die Analyse der Ticketdaten mithilfe textverarbeitender Algorithmen.

Abbildungs- und Tabellenverzeichnis

Literaturverzeichnis

Biemann, C., & Mehler, A. (2014). *Text Mining: From Ontology Learning to Automated Text Processing Applications.* Cham, s.l.: Springer International Publishing.

Biemann, C., & Riedl, M. (2013). Text: now in 2D! A framework for lexical expansion with contextual similarity. In *Journal of Language Modelling* (S. 55-95). Warsaw, PL: Institute of Computer Science, Polish Academy of Sciences.

Bird, S., Klein, E., & Loper, E. (2009). Preface. In *Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit.* Sebastopol, Calif. [u.a.]: O'Reilly Media.

Blanco, R., Cambazoglu, B. B., Mika, P., & Torzec, N. (2013). Entity Recommendations in Web Search. In H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. X. Parreira, . . . K. Janowicz, *The semantic web - ISWC 2013 : 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21 - 25, 2013 ; proceedings, part II* (S. 33-48). Berlin [u.a.]: Springer.

Dubois, P. F. (2007). Python: Batteries Included. *Computing in Science & Engineering. IEEE/AIP.*

Elsafty, A., Riedl, M., & Biemann, C. (2018). Document-based Recommender System for Job Postings using Dense Representations. *Proceedings of NAACL industry track.* New Orleans, USA.

Fielding, R. T. (2000). *Dissertation: Architectural Styles and the Design of Network-based Software Architectures.* Irvine: University of California.

Gormley, C., & Tong, Z. (2015). *Elasticsearch: the definitive guide.* Beijing [u.a.]: O'Reilly Media.

Gorunescu, F. (2011). *Data Mining : Concepts, Models and Techniques.* Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Kowalski, G. (2011). *Information Retrieval Architecture and Algorithms.* Boston, MA: Springer Science+Business Media, LLC.

Krcmar, H. (2015). *Informationsmanagement.* Wiesbaden: Springer Gabler Verlag.

Lupu, M., Mayer, K., Tait, J., & Trippe, A. J. (2011). *Current Challenges in Patent Information Retrieval.* Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Manku, G. S., Jain, A., & Das Sarma, A. (2007). Detecting NearDuplicates. In *WWW 2007* (S. 141-149). Alberta, CA: International World Wide Web Conference Committee.

Manning, C., Raghavan, P., & Schütze, H. (2008). Scoring, term weighting, and the vector space model. In *Introduction to Information Retrieval* (S. 109-133). Cambridge: Cambridge University Press.

Milmann, K. J., & Avaizis, M. (2011). Scientific Python. *Computing in Science & Engineering. IEEE/AIP.*

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. In *Journal of the American Medical Informatics Association* (S. 544-551). London, UK: BMJ Group.

Olive, J., Christianson, C., & McCary, J. (2011). *Handbook of Natural Language Processing and Machine Translation: DARPA Global Autonomous Language Exploitation.* New York, NY: Springer Science+Business Media, LLC.

Oliveira, T., Becker Nunes, F., Bierhalz Voss, G., Medina, R., & Valdeni de Lima, J. (2014). Recommendation System for Assisting the Management of Information Technology. In J. L. Mauri, *Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2014)* (S. 72-79). Rome, IT: IARIA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*.

Petzold, J., & Westerkamp, M. (2018). Begriff, Aufbau und Komponenten von Controlling-Informationssystemen. In J. Petzold, & M. Westerkamp, *Informationssysteme im wertorientierten Controlling* (S. 67-86). Wiesbaden: Springer Gabler.

Piatetsky-Shapiro, G. (2000). The Data-Mining Industry Coming of Age. *IEEE Intelligent Systems*.

Sharma, M. (2014). Data Mining: A Literature Survey. *International Journal of Emerging Research in Management &Technology*.

Silva, S., Ribeiro, R., & Pereira, R. (2018). Less is more in incident categorization. *7th Symposium on Languages, Applications and Technologies (SLATE 2018).* Guimarães, Portugal: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany.

Tang, L., Li, T., Shwartz, L., & Grabarnik, G. (2013). Recommending Resolutions for Problems Identified by Monitoring. *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM2013)* (S. 134-142). Ghent, Belgium: IEEE.

Vijayarani, S., Ilamathi, J., & Nithya. (2015). Preprocessing Techniques for Text Mining - An Overview. *International Journal of Computer Science & Communication Networks*, S. 7-16.

Wei, X., Sailer, A., Mahindru, R., & Kar, G. (2007). Automatic structuring of it problem ticket data for enhanced problem resolution. *IFIP/IEEE International Symposium on Integrated Network Management, 2007. IM'07.10th* (S. 852-855). IEEE.

Zhou, W., Tang, L., Li, T., Shwarzt, L., & Grabarnik, G. (2016). Resolution Recommendation for Event Tickets in Service Management. In *IEEE Transactions on Network and Service Management.* IEEE.

Eigenständigkeitserklärung

Hiermit versichere ich an Eides statt, dass ich die Arbeit eigenständig verfasst habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht.

Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium.

Hamburg, den 02. August 2018 

Ort, Datum Unterschrift