HAW HAMBURG

BACHELORTHESIS
Fabian Peters

Automatisierte Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Fabian Peters

Automatisierte Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Olaf Zukunft
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 23. Dezember 2020

Fabian Peters

Thema der Arbeit

Automatisierte Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen

Stichworte

Twitter, Context Items, Crawler, Scraper, NLP, Spacy, GenSim, doccano, Bugreports

Kurzzusammenfassung

Twitter bietet einen direkten Kommunikationskanal zwischen Benutzer und Hersteller einer Software. Anstatt komplizierte Kontaktformulare oder Supporttickets auszufüllen, können Benutzer Probleme direkt und öffentlich wirksam in die Timeline des Hersteller schreiben. Wie können Hersteller mit diesem Feedback umgehen? Welche Informationen sind wichtig für sie und welche können sie direkt aus der Meldung des Benutzers lesen? In dieser vorliegenden Ausarbeitung wurde die Kommunikation zwischen Benutzern und Herstellern auf Twitter mit einem Twittercrawler erfasst und ausgewertet. Insgesamt wurden 5 Millionen Tweets von 19 unterschiedlichen Twitter-Accounts gesammelt und zu Truthsets mit mehr als 2.4 Millionen Tweets von Benutzern normalisiert. In Anlehnung an das Konferenzpapier 'Extracting and Analyzing Context Information in User-Support Conversations on Twitter' der Autoren Maleej und Martens [21] wurde eine Auswahl von vielversprechenden Context Items bestimmt und mit herstellerspezifischen Belegungen erfasst. Für das Context Item der Gerätebezeichnung wurden zusätzliche Belegungen mithilfe eines FastText-Modells ermittelt, welches mit den Tweets des Truthsets trainiert wurde. Für zwei ausgewählte Gruppen mit Herstellern für IDE-Software und mobiler Anwendungen wurden händisch annotierte Tweetssets erstellt. Der implementierte Extractor wurde auf seine Performance geprüft, wobei dieser für die Mehrzahl der Context Items eine Precision von 68-98 % erreichte. Die Kontrollgruppe Mobile dient zur Validierung der Herangehensweise des Konferenzpapiers, während die Kontrollgruppe IDE eine mögliche Erweiterbarkeit des Ansatzes untersucht. Diese Erweiterung zeigt grundlegende Probleme hinsichtlich Mehrdeutigkeit und Überschneidungen der individuellen Context Items, wobei die ursprüngliche Kontrollgruppe mit minimalen Abweichungen bestätigt werden konnte.

Fabian Peters

Title of Thesis

Automated elicitation of relevant context information from Twitter conversations

Keywords

Twitter, Context Items, Crawler, Scraper, NLP, Spacy, GenSim, doccano, Bugreports

Abstract

Twitter provides a direct communication channel between the user and the vendor of a software. Instead of filling out complicated contact forms or support tickets, users can post problems directly and publicly to the timeline of the vendor. What information is important to them and what can they read directly from the user's message? In this paper, the communication between users and vendors on Twitter was captured and analyzed using a Twittercralwer. A total of 5 million tweets from 19 different Twitter accounts were collected and normalized into truthsets with more than 2.4 million tweets from users. Following the conference paper 'Extracting and Analyzing Context Information in User-Support Conversations on Twitter' by authors Maleej and Martens [21], a selection of promising context items and with vendor-specific valid assignments were captured. For the context item of device name, additional assignments were determined using a FastText model trained with the tweets from the truthsets. Manually-annotated tweetsets were created for two selected groups from IDE-Software and mobile application vendors. The implemented extractor was tested for its performance, achieving a precision of 68-98 % for the majority of context items. used to validate the approach of the conference paper, while the IDE control group investigates a possible extensibility of the approach. This extension shows fundamental problems regarding ambiguity and overlap of the individual context items, while the original control group could be validated with minimal deviations.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Sobald ein Softwarehersteller ein Produkt veröffentlicht, ist dieses Produkt gewissen Benutzergemeinde ausgesetzt. Während der Verwendung des Produktes können die Benutzer auf Probleme stoßen, welche dann im besten Fall an den Softwarehersteller zurückgemeldet und von diesem behoben werden. Softwarehersteller verwenden sogenannte *Issue Tracker*, um diese aufgetretenen Probleme zu erfassen, auszuwerten und zu beheben. Idealerweise ist der Inputkanal für solche Probleme von dem Softwarehersteller so aufbereitet, dass ein Benutzer strukturierte Daten zu seinem festgestellten Problem als *Bugreport* weitergibt. Strukturierte Daten enthalten dann die Context Items, die zur Lösung des Problems benötigt werden. Diese Context Items beinhaltet den Auszug der relevanten Informationen. Dies beläuft sich auf die aufgerufene Funktion und die auftretende Fehlermeldung, sowie das Betriebssystem des verwendeten Endgerätes und der Applikation. Zusätzlich wird auch die Versionen der benutzten Applikation und die des Betriebssystems übergeben. Da Softwarehersteller auch in den Sozialen Medien vertreten sind, ist es denkbar, dass die klassischen Inputkanäle nicht verwendet werden (siehe Abbildung 1.1). Es ist davon auszugehen, dass wenig technisch-versierte Benutzer sich öffentlich (z.B. auf Twitter) zurückmelden, um den Druck auf den Hersteller zu erhöhen, sich dem Problem anzunehmen.[22]

Abbildung 1.1: Twitter-Statusmeldung File a Bugreport? [46]

Dieses Verhalten könnte auf vielen Gründe zurückführen, es liegt dennoch im Interesse der Softwarehersteller dieses Feedback zu behandeln. Gibt es bspw. seit dem letzten Update vermehrte Meldungen zu einer fehlerhaften Funktion, kann der Hersteller öffentlichkeitswirksam reagieren und zusätzliche Informationen beim Benutzer erfragen. Sobald der Fehler behoben ist, kann dieser sich direkt an den betroffenen Nutzer wenden und Lösungsvorschläge, z.B. in Form einer Updateempfehlung, geben.

1.1 Zielsetzung

Im Rahmen dieser Arbeit soll die Umsetzbarkeit eines Systems zur Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen überprüft werden. Hierzu wird ein System erstellt, das öffentlich zugängliche Twitterdaten erfasst und aus ihnen Daten gewinnt. Zur Erhebung der relevanten Kontextinformationen werden Context Items bestimmt und Pre-Defined Keyword Lists erstellt. Die Informationen sind bei den Herstellern von Anwendungen, Geräten und Betriebssystemen zu recherchieren und für die Erhebung aufzubereiten. Ziel der vorliegenden Arbeit ist im Weiteren, die Performance der Erhebung in Gruppen von artverwandten Serviceprovidern zu bestimmen und zu vergleichen. Die Performance wird gegen ausgewählte Annotations-Sets getestet. Als Vergleichsbasis dient die Ausarbeitung von Maleej und Martens [21]. Die dort präsentierte Herangehensweise wird hier mit dem Ergebnis einer ähnlichen Kontrollgruppe überprüft und durch eine zusätzliche Variante erweitert. Die zusätzliche Variante besteht aus einer weiteren Grup-

pe, die eine andere Art von Serviceprovidern beinhaltet, dessen Betriebssysteme nicht in ihrer Ausarbeitung betrachtet wurden. Hauptmerkmal liegt hierbei auf der Untersuchung der Performance. Zum Schluss werden Parallelen, sowie individuelle Stärken und Schwächen der einzelnen angewendeten Gruppen herausgestellt und diskutiert.

1.2 Aufbau der Arbeit

In Kapitel 2 werden die verwendeten Techniken und die benötigten Grundlagen für die Thematik beschrieben. Im Kapitel 3 werden die Anforderungen an ein System zur Erhebung von Kontextinformationen aus Twitterunterhaltungen zusammengefasst, sowie verwandte Arbeiten analysiert. Im darauffolgenden Kapitel 4 werden die einzelnen Teile des Systems, der Crawler und der Extractor konzipiert und ihre Implementierung erläutert. Das Kapitel 5 evaluiert die Vorbereitung der Daten und bewertet die Performance des Extrators gegenüber eines händisch annotierten Tweetsets, bevor das letzte Kapitel 6 mit einem Fazit und Ausblick abschließt.

2 Grundlagen

Dieses Kapitel fasst die Grundlagen zur Erfassung, Aufbereitung und Analyse der automatisierten Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen zusammen. Angefangen mit Erläuterung zu dem Kurznachrichtendienst Twitter wird auf die verschiedenen Möglichkeiten, Besonderheiten und Bestimmungen eines Crawlers eingegangen. In diesem Zusammenhang werden auch die verschiedenen Python-Bibliotheken näher betrachtet. Abschließend wird die Thematik der Contex Items und Issue Tracker beschrieben.

2.1 Twitter

Der Kurznachrichtendienst Twitter bietet seinen Benutzern die grundsätzlichen Funktionen einer Plattform der Sozialen Medien. Benutzer interagieren miteinander, indem diese Nachrichten, sogenannte Tweets, auf sich selbst bezogen oder an andere Benutzer gerichtet veröffentlichen. [27] Diese Tweets werden favorisiert, beantwortet oder als Retweet geteilt. Benutzer haben die Möglichkeit anderen Benutzern zu folgen, um über neue Aktivitäten informiert zu werden. Die Followerzahl eines Benutzer sagt aus, wie viele weitere Benutzer diesem Account folgen. Die daraus resultierende Kommunikation wird über die eigene Twitter-API [39] geneigten Benutzern in unterschiedlichen Stufen und Anwendungen zugänglich gemacht. In den folgenden Unterkapiteln wird auf die Datenstruktur von Twitter und den Zugang der Twitter-API eingegangen.

2.1.1 Tweet

Tweets bilden die grundsätzlichen Bausteine von Twitter. Diese enthalten Attribute wie die ID des Tweets, die ID des Autors, den Text, einen Zähler wie häufig dieser Tweet favorisiert wird oder retweetet wurde. Des weiteren sind auch Attribute zum Nachvollziehen

einer Unterhaltung enthalten. So verfügen einige Tweets über die Attribute *in_reply_to_status_id* und *in_reply_to_user_id*. Diese Attribute geben Auskunft über einen vorherigen Tweet auf den sich bezogen wird. Autoren werden im Benutzer-Objekt abgebildet. Sie enthalten Attribute wie *screen_name*, den angezeigten Namen, die Anzahl der Follower, *created_at* als Datum der Registrierung, sowie den *status_count* der die Anzahl der Tweets des Users, inklusive Retweets, auflistet. [43]

Über die Twitter-API werden Tweets als Name/Wert-Paare im JSON-Format zurückgegeben. [40]

2.1.2 Unterhaltungen

Unterhaltungen auf Twitter sind aufgrund der Attribute einzelner Tweets im Prinzip wie eine verkettete Liste organisiert. Es gibt einen Kopf-Tweet, ohne Werte in den Attributen *in_reply_to_status_id* und *in_reply_to_user_id*, auf den mit einem Tweet geantwortet wird. Auf diesen Tweet kann wiederum geantwortet werden und so entsteht eine Kette von Tweets, die eine Unterhaltung bildet.

2.1.3 Zugang

Abgesehen von dem Einbetten von Tweets auf einer Webseite, wird für die Nutzung der Twitter-API ein Twitter Entwickler-Account benötigt. Mit diesem Account kann eine Entwickler-App erstellt werden. Für diese App werden dann *API-Keys* und *User-Access-Token*, sowie *Bearer-Token* erstellt. [39] Für die Interaktion mit den einzelnen Service-Endpoints werden die HTTP-Methoden *GET* und *POST* verwendet. Sie ermöglichen die Authentifizierung und das Prüfen von Benutzer-Timelines und einzelner Statusmeldungen. [38]

2.1.4 Beschränkungen

Jeder Service-Endpoint unterliegt einem bestimmten Abfrage-Limit. Diese Limits sind unterteilt in 15-minütige Zeitfenster. [36] Für die Abfrage einer Timeline eines bestimmten Benutzers, mit der API GET statuses/user_timeline gilt ein Abfragelimit von 900 Abfragen per Benutzer bzw. 1.500 per *Access-Token* pro 15 Minuten. [37] In jeder Abfrage können bis bis zu 200 Tweets enthalten sein. Durch den Parameter max_id werden

nur Tweets mit einer ID kleiner als dieser *max_id* zurückgegeben. Dies ermöglicht es *top-down* in den Tweets eines Twitteraccounts zu crawlen. Allerdings gibt dieser Endpoint maximal 3.200 der aktuellen Tweets zurück. Das Abfrage-Maximum für einen Tag beträgt 100.000 Anfragen. [35]

2.2 Crawler

Crawler sind Programme die automatisiert das Internet durchforsten, um effektiv und effizient Informationen über Webseiten und ihren Verbindungen untereinander zu sammeln. [15]

Crawler erlauben es, große Webseiten zu verschlagworten und diese Informationen Benutzern vereinfacht zur Verfügung zu stellen. [24]

Dazu startet der Crawler von einer URL, dem sogenannten Seed, von der er weitere URLs extrahiert, um diese zu besuchen und diese nach weiterem URLs durchsucht. [20] Die extrahierten Daten können lokal in einer Datenstruktur oder in einer Datenbank gespeichert werden. [20]

Es gibt dabei zwei grundsätzliche Ansätze, den generic Crawler und den focused Crawler. [24] Diese werden in den folgenden Abschnitten definiert.

Der generic Crawler ist ein Crawler mit einem sehr einfachen Auftrag, der ohne zu filtern oder spezifischer Stichworte Informationen sammelt. [48] Durch diese ungefilterte Suche werden viele unnütze Daten heruntergeladen und nur grob kategorisiert. [7]

Ein focused Crawler hingegen sucht Informationen, die mit dem spezifischen Thema verwandt sind und filtert irrelevante Daten aus. [24] Focused Crawler können auch unterscheiden, ob es sich um relativ statische oder dynamische Inhalte handelt. Dieses fokussierte Sammeln von Informationen macht den Focused Crawler effizient in Hinsicht auf Netzwerkbandbreite und Speicherplatz. [7]

Durch die wachsende Fülle an Inhalten im Internet, bietet sich eine Parallelisierung bzw. Verteilung einzelner Crawler an. Die Parallelisierung der Crawler geht allerdings auch mit einer Verkomplizierung der Architektur einher. [23][48]

Eine spezielle Anwendung eines Crawlers ist das Webscraping. Webscraping beschreibt die automatisierte Extraktion von speziellen Informationen aus Webseiten. Es simuliert

die menschliche Herangehensweise, unstrukturierte Informationen aus dem Internet in strukturierte Daten zu transformieren. [6] Das Herauslösen der Daten beim Webscraping erfolgt entweder durch HTTP-Abfragen oder durch das Einbinden voll umfassender Webbrowser. [20] Eine besondere Herausforderung bei Scrapen ist, dass gewünschten Daten auf einer oder mehreren Webseiten unterschiedlich vorhanden sein können. Sie kommen dabei unstrukturiert und in verschiedenen Datentypen vor oder bedürfen einer Benutzereingabe, um verfügbar zu werden. [6] Diese Herausforderung stellt sich in Rahmen dieser Thesis nicht, da es sich immer um die selben Ressourcen und Webseiten handelt.

Der innerhalb dieser Arbeit implementierte Crawler arbeitet mithilfe der Python-Bibliotheken *Tweepy* und *TwitterScraper*, auf die in den folgenden Unterkapitel eingegangen wird. Die Strukturierung und das Crawling-Konzept werden in Kapitel 4 beschrieben.

2.2.1 Tweepy

Die Python-Bibliothek *Tweepy* bietet neben Schnittstellenwrappern zur Twitter-API auch Handler für die Authentifizierung mit oAuth 1a und oAuth 2. Dies ermöglicht eine einfache Authentifizierung mit dem Benutzer und auch der Applikation. [31]

Dazu werden nur die Zugangsdaten, wie im Unterkapitel 2.1.3 beschrieben, benötigt und *Tweepy* übernimmt die Authentifizierung. [30]

Tweepy bietet für die Verwendung der Twitter-API einen entsprechenden Wrapper an. Es können Timeline, Benutzer und Status mit vorgegeben Attributen abgefragt werden. So ist z.B. das in Unterkapitel 2.1.3 bereits genannte Attribut *max_id* in der *statuses_lookup()*-Methode, welche die Timeline eines Benutzers anzeigt, enthalten. [33]

Ein weiteres wichtiges Attribut ist *wait_on_rate_limit*, welches für die Einhaltung der Beschränkungen des Abfrage-Limits (vgl. Unterkapitel 2.1.4) bei der Verwendung fast aller *Tweepy*-Methoden sorgt.

Die Rückgabe der Daten der Twitter-API erfolgt in der Regel strukturiert als *Tweepy*-Objekt *User* oder *Status*, diese können dann direkt weiterverarbeitet werden. [32]

2.2.2 TwitterScraper

TwitterScraper bietet eine ähnliche Funktionalität wie *Tweepy*, verwendet aber nicht die Twitter-API. *TwitterScraper* verarbeitet mit den Python-Bibliotheken *requests* und *beautifulsoup4* HTML-Responses der Twitter-Webseite und transferiert diese in JSON-konforme Daten. Dabei sind die Abfrage-Limits (vgl. Unterkapitel 2.1.4) in der Twitter-API irrelevant, da die Ergebnisse auf der Twitter-Webseite in ihrer Anzahl keinen Beschränkungen unterliegen. [34]

Weiterhin verwendet *TwitterScraper* für die HTTP-GET-Abfrage immer unterschiedliche Proxy-Server. Diese verbergen die IP-Adresse des crawlenden Rechners und verhindern somit mögliche Sperrungen durch den Service.

Abbildung 2.1: Erweiterte Suche der Twitter-Webseite

Mit *Twitterscraper* lassen sich entweder Command-line-Befehle absetzen oder auch direkt Python-Methoden verwenden. Die Command-line-Befehle können mit verschiedenen Argumenten versehen werden, die z.B. über das Ausgabeformat oder ein Limit an Rückgabetweets bestimmen. In beiden Anwendungen wird eine Query-Abfrage als erstes Argument verwendet, die der erweiterten Suche auf der Twitter-Webseite entspricht (siehe Abbildung 2.1). [34]

2.2.3 JSON

Die Twitter-API verwendet JSON als Austauschformat. [40] Diese Notation ist leicht zu lesen und kann maschinell verarbeitet werden. JSON verwendet als Struktur neben geordneten Listen auch Name/Wert-Paare, diese werden auch von der Twitter-API, sowie von den hier bereits erläuterten Python-Bibliotheken verwendet. Außerdem ist JSON in der Python-Standard-Bibliothek enthalten und kann somit importiert werden. [9][28][40]

2.2.4 Rechtliche Bestimmungen

Twitter gestattet es seinen Benutzern gemäß der allgemeinen Geschäftsbedingung seine Services zu nutzen. Allerdings schließen diese es aus, dass Benutzer neben den zur Verfügung gestellten Schnittstellen anderweitig auf ihre Services und Daten zugreifen. *Webscraping* wird sogar ausdrücklich untersagt: *'(iii) mit anderen (automatisierten oder anderweitigen) Mitteln als mit unseren derzeit verfügbaren und veröffentlichten Schnittstellen von Twitter auf die Dienste zuzugreifen oder diese zu durchsuchen (wobei dies auch nur gemäß den anwendbaren Bedingungen erfolgen darf) bzw. dies zu versuchen, es sei denn, Sie sind aufgrund einer gesonderten Vereinbarung mit Twitter ausdrücklich hierzu befugt (HINWEIS: Das sogenannte „Crawling" der Dienste ist erlaubt, soweit dies gemäß den Vorgaben der robots.txt-Datei erfolgt. Das sogenannte „Scraping" der Dienste ist ohne vorherige Genehmigung von Twitter ausdrücklich untersagt)'* [42] Twitter behält sich das Recht vor Accounts zu sperren oder zu kündigen, wenn diese gegen die Regeln verstoßen sollten. *'Wir können Ihre Accounts sperren oder kündigen oder Ihnen die Bereitstellung der Dienste jederzeit aus beliebigem Grund ganz oder teilweise verwehren, insbesondere, wenn wir Grund zu der Annahme haben, dass: (i) Sie gegen die vorliegenden Bedingungen oder die Twitter-Regeln oder die Periscope Community Richtlinien verstoßen haben, (ii) Sie für uns eine Gefahr oder ein mögliches rechtliches Risiko darstellen oder (iii) Ihr Account aufgrund von rechtswidrigem Verhalten oder (iv) aufgrund von längerer Inaktivität entfernt werden sollte oder weil (v) es wirtschaftlich nicht mehr vertretbar ist, Ihnen die Dienste bereitzustellen.'* [42]

Über die *robots.txt* auf *www.twitter.com/robots.txt* einzusehen ist bestimmt, dass kein Webcrawler die Seite durchsuchen darf. Der Inhalt der *robots.txt* lautet:

```
User-agent: *
Disallow: /"
```

Die *robots.txt* ist ein inoffizieller Standard, der Webserver von unerwünschten Zugriffen durch Crawler schützen soll. Der Inhalt der *robots.txt* wird allerdings nur von wohlwollenden Crawlern berücksichtigt und birgt keine Garantie vor unerlaubten Zugriff. [19]

Die Nutzung von Crawlern und Scraper für wissenschaftliche Arbeiten sind zwar grundsätzlich zulässig [18], die rechtliche Lage ist jedoch unübersichtlich und sprengt den Rahmen der vorliegenden Arbeit.

2.3 spaCy

spaCy bietet als Python-Bibliothek Funktionen des Natural Language Programming an. Neben Funktionen zum POS-Tagging, Entity Linking, Named Entitty Recognition oder Lemmatization, die entweder aus vorhandenen Sprachmodellen geladen oder selbst trainiert werden können, bietet *spaCy* die Möglichkeit der Tokenization von Texten, sowie ein Rule-based Matching von Wörtern in Texten. Bei der Tokenization werden Texte in einzelnen Wörter, Satzzeichen und Zahlen, den sogenannten Token, aufgeteilt. Unter Berücksichtigung der verwendeten Sprache kann *spaCy* unterscheiden, ob es sich um eine Satzzeichen oder bspw. eine Abkürzung handelt. Nach der Tokenization kann über die einzelnen Token iteriert werden. Das Rule-based Matching ermöglicht das Auffinden von Wörtern oder Mustern in Texten. Anders als bei regulären Ausdrücken können die Mustern auch Annotationen enthalten. [16][26]

2.4 Gensim

Gensim ist eine Python-Bibliothek für semantische Textanalysen. Sie beinhaltet eine Anzahl von Algorithmen, die grundsätzlich Texte in Textcorpora zusammengefasst und in Vektoren umwandelt. Diese Vektoren bilden ein Modell, mit dessen Hilfe dann bspw. zu einem Wort ähnliche Wörter ermittelt werden können. Für die vorliegende Arbeit wird der in Gensim enthaltene Algorithmus *FastText* verwendet. [29] Dieser eignet sich insbesondere für die Erkennung von alternativen Bezeichnungen von Benutzern.

FastText erlaubt es, Textcorpora einzulesen und trainiert anhand der Wörter ein Modell mit Wortvektoren. Die Wortvektoren können als *Skipgram* oder *CBOW* (continuous-bag-of-words) erstellt werden. Durch den Vergleich dieser Vektoren lassen sich Ähnlichkeiten zwischen Wörtern aufzeigen. [8]

2.5 Context Items

Kontextinformationen, die in Texten, Unterhaltungen und Nachrichten enthalten sind, bestehen aus Context Items. *Context Items* sind im Rahmen eines Bugreports z.B. das eingesetzte Gerät, das Betriebssystem und die eingesetzte Version der Software des Benutzers.

2.6 Issue Tracker

Issue Tracker sind strukturierte Systeme, um die Entwicklung einer Software voran zu treiben. Diese Entwicklung beinhaltet die Beseitigungen von Fehlern, aber auch allgemeine Verbesserungen. Diese können von Entwicklern oder auch ggf. von den Benutzern erfasst werden. Erfassten Issues können relevante Daten zugewiesen werden. [17][45]

Im Rahmen dieser Arbeiten sind die zugewiesenen Daten die beschriebenen *Context Items*, die erfassten Issues sind Bugreports.

2.7 doccano

Mit *doccano* ist es möglich Texte mit Annotationen zu versehen. Mit einer Benutzeroberfläche und einer Import- und Exportschnittstelle können Texte geladen und mit Stichworten annotiert werden. Für diese Ausarbeitung wurde *doccano* benutzt, um Kontextinformationen aus einer gewissen Anzahl von Tweets manuell zu annotieren. Das dabei entstehende Set von Datensätzen kann dann zu Validierung der extrahierten Context Items verwendet werden. [25]

2.8 Klassifikationsmodell

Um die Ergebnisse zu kontrollieren und die händisch annotierten Werte mit den automatisch extrahierten Werten zu vergleichen, wird das Klassifikationsmodell *True vs. False and Positive vs. Negative* angewendet. Hierbei wird pro Fall zwischen vier Klassen unterschieden:

- True Positives (TP), annotierter und extrahierten Wert stimmen überein

- False Positives (FP), annotierter und extrahierter Wert stimmen nicht überein
- False Negative (FN), Wert wurde annotiert, aber nicht extrahiert
- True Negative (TN), Wert wurde weder annotiert noch extrahiert

Die Einteilung findet sich in der Tabelle 2.1.

	True Positive (TP)	**False Positives (FP)**
Annotation:	enthält Context Items	Context Items möglich
Extraktion:	erkennt Context Items	erkennt Context Items
	True Negatives (TN)	**False Negatives (FN)**
Annotation:	enthält keine Context Items	enthält Context Items
Extraktion:	erkennt keine Context Items	erkennt keine Context Items

Tabelle 2.1: Klassifikation Annotation und Extraktion

Für die erfassten summierten Klassifikationen werden mit weiteren Metriken aussagekräftige Werte wie *Accuracy*, *Precision* und *Recall* bestimmt. Die *Accuracy* bestimmt die prozentuale Genauigkeit der bestimmten Werte aller Werte, als
$\frac{TP+TN}{TP+TN+FP+FN}$.
Mit der Precision bestimmt man das Verhältnis der positiven wahren zu positiven falschen Klassifikationen: $\frac{TP}{TP+FP}$.
Der *Recall* gibt das Verhältnis zwischen richtig erkannten zu falsch erkannten Werten, als
$\frac{TP}{TP+FN}$
, an. [12][13][14]

3 Analyse

Für die automatisierte Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen müssen Vorbereitungen zur Datenermittlung, -speicherung und -verarbeitung getroffen werden. In den folgenden Unterkapiteln werden die Anforderungen an eine solche Erhebung und verwandte Arbeiten zusammengefasst.

3.1 Anforderungen

Die Anforderungen an eine automatisierte Erhebung beinhalten eine gewisse Autonomie der Datenermittlung und -verarbeitung. Ein Twittercrawler sollte ohne manuelles Zutun, Daten sammeln. Zusätzlich sollte dieser eine feste Datenstruktur besitzen, die unterschiedliche Dienste anbinden kann. Die Extraktion von relevanten Kontextinformationen ist an eine variierende Datenbasis gebunden. Context Items sind unterschiedlich relevant und ihre Erhebung ist abhängig von Stand der aktuellen Erkenntnissen und Bedürfnissen. Die herangezogene Datenbasis muss dementsprechend variabel und erweiterbar gestaltet sein. Die Extraktion der Kontextinformation darf die erhobenen Daten des Crawlers nicht verändern und erfolgt deswegen funktional.

3.2 Verwandte Arbeiten

In der Arbeit *Twitter als Basis wissenschaftlicher Studien* [27] beschreibt der Autor, dass sich Twitter als Datenbasis für wissenschaftliche Studien, aufgrund der strukturierten Bereitstellung der Daten über API, anbietet und das Filtern, Strukturieren und Analysieren beschleunigt.

Der Autor stellte fest, dass alle anderen Zugänge (APIs und Drittanbieter) unterschiedliche Einschränkungen vorwiesen und unter Umständen in diesen Fällen mehrere Ansätze

zum Sammeln von Tweets notwendig wären. Datensammlungen böten viele Möglichkeiten, die sich in Zeithorizont, Datenumfang und finanzieller Schwelle äußerten. [27]

Dagegen stehe, dass die gesammelten Daten einige Einschränkungen aufwiesen. *'Die typische Internet-Sprache mit ihren Charakteristika, wie Abkürzungen, Mehrsprachigkeit und Neologismen, verhindert eine zuverlässige, automatische Inhaltsanalyse.'* [27] Eine eingeschränkte Möglichkeit Tweets zu vernetzen, etwa bei Unterhaltungen, könne den Kontext eines Tweets ausblenden. Die Verifizierung von Aussagen und Meta-Daten sei dann nicht möglich. Grundsätzlich sei doch die Verfügbarkeit der Daten gegeben und bereits gut strukturiert vorhanden. [27]

Eine speziellere Betrachtung und Verarbeitung von erhobenen Twitterdaten erstellten und veröffentlichen Daniel Martens und Walid Maalej in ihrem 2019 erschienenen Konferenzpapier *Extracting and Analyzing Context Information in User-Support Conversations on Twitter*, über die Erfassung und Analyse von Kontextinformationen in Supportunterhaltungen auf Twitter. [21]

Benutzer interagieren vermehrt über den öffentlichen Kanal der Sozialen Medien, obwohl viele Applikationen bereits die Möglichkeit anbieten Feedback zu Fehlern oder Verbesserungen zu erfassen. Dies passierte laut den Autoren, um den öffentlichen Druck auf die Hersteller zu erhöhen. Das Feedback der Benutzer enthalte aber nicht unbedingt die benötigten Kontextinformationen (Context Items) und Struktur, die ein Entwickler benötigt um das Problem nachzustellen und zu beheben. Um diese Kontextinformationen zu erhalten müssten Supportteams diese in immer wiederkehrenden, aufwendigen Unterhaltungen erfragen. [21] Die Herausforderungen dieses Feedback zu klassifizieren, besteht also im wesentlichen aus den folgenden Punkten:

- Fehlende Informationen ermitteln
- Nicht reproduzierbare Fälle, aufgrund fehlender Kontextinformationen
- Manueller Aufwand, um fehlende Informationen in Unterhaltungen herauszulösen

Der Ansatz von Maleej und Martens ist es, Tweets zu sammeln, die grundsätzlichen Kontextinformationen aus den Tweets zu extrahieren und fehlende Kontextinformationen mithilfe eines Chatbots zu komplettieren, um anschließend einen Issue zu erfassen. Die Arbeitsschritte beschreiben die Autoren in vier Phasen:

1. Tweet Classification Phase

   Gesammelte Tweets werden klassifiziert, nur echte Bugreports werden in die nächste Phase übernommen.

2. Context Extraction Phase

   Einzelne Tweets oder Unterhaltungen werden auf die grundsätzlichen Context Items überprüft und zusätzlich im Falle einer Unterhaltung, ob alle Kontextinformationen extrahiert werden konnten.

3. Context Clarification Phase

   Wenn nicht alle Kontextinformationen extrahiert werden konnten, wird der Chatbot die fehlenden Kontextinformationen erfragen.

4. Issue Creation Phase

   Wenn alle Kontextinformationen extrahiert werden konnten, wird ein strukturierter Bugreport erstellt und mit der Twitterunterhaltung für eventuelle Rückfragen oder Erfolgsmeldungen verknüpft. [21] Die einzelnen Phasen dieses Prozesses werden in Abbildung 3.1 dargestellt und nachfolgend erläutert.

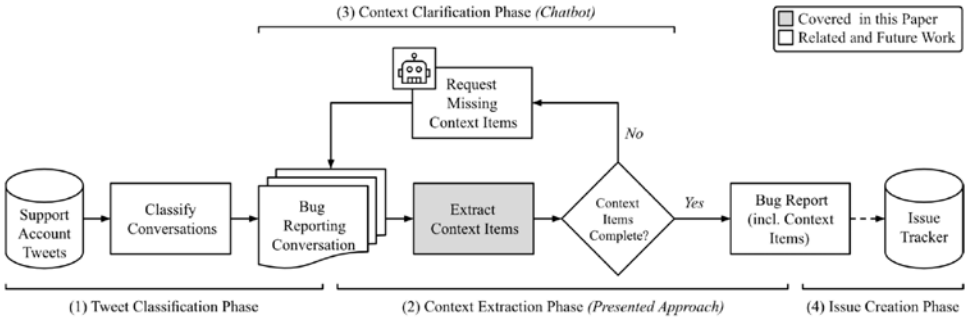


Abbildung 3.1: Übersicht der Gesamtkonfiguration der einzelnen Phasen [21]

Die wesentlichen Herausforderungen sind in dieser Arbeit die entscheidenden Phasen der *Context Extraction Phase*, die fehlende Kontextinformationen zusammenfasst und die *Context Clarification Phase*, die den manuellen Aufwand fehlende Informationen zu ermitteln, minimieren soll.

Maleej und Martens untersuchten in ihrem Papier die Unterhaltungen der offiziellen Twitteraccounts von *Spotify*, *Netflix* und *Snapchat*. Der Prozess der Datenverarbeitung wird in der Abbildung 3.2 dargestellt. Zuerst wurden Twitter-Search-API Nachrichten, die direkt an den jeweiligen Account gerichtet waren, erfasst. Das daraus resultierende Dataset umfasst über fünf millionen Tweets, die aus mehr als zwei millionen Unterhaltungen mit über einer million Benutzer besteht. [21]

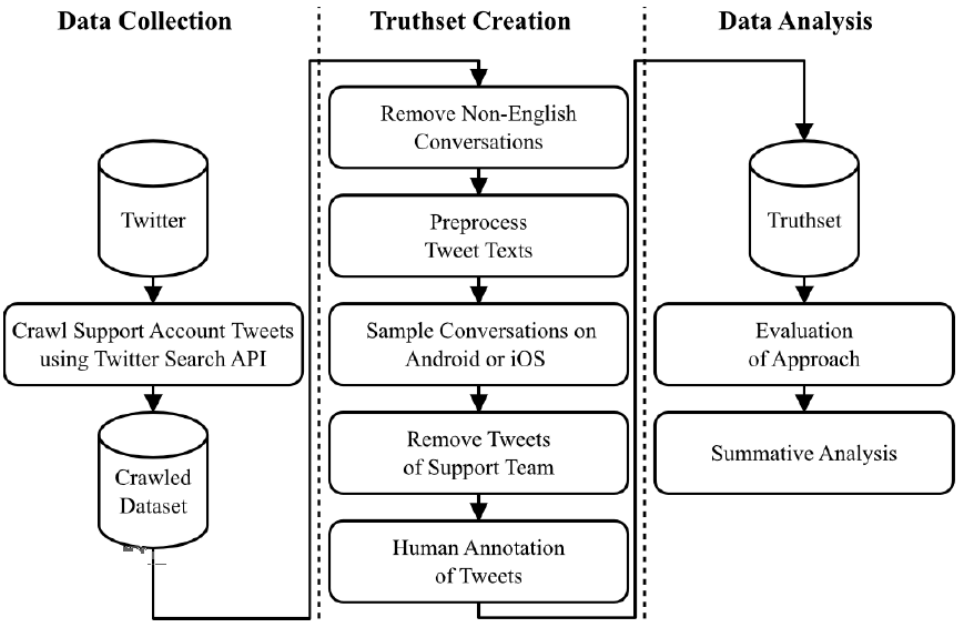


Abbildung 3.2: Übersicht der Datenverarbeitung
[21]

Die gesammelten Daten wurden anschließend zu einem Truthset weiterverarbeitet. Die Bereinigung der erfassten Tweets erfolgte so, dass nur englischsprachige Tweets enthalten blieben und überflüssige Zeilenumbrüche, Leerzeichen und Erwähnungen von Supportaccounts entfernt und Großbuchstaben in Kleinbuchstaben umgewandelt wurden.

Manuelle Stichworte, die auf Context Items hinweisen, wurden festgelegt und den Tweets mithilfe des Open-Source Tool *doccano* als Label zugewiesen. So konnten aus ungefähr 3.000 Usertweets 1.800 Context Items gewonnen werden. Die Context Items umfassten in diesem Beispiel das Betriebssystem, das Gerät, sowie die Verison des Betriebssystems und der App. Die Auswahl erfolgte durch eine ebenfalls manuelle Überprüfung von einhundert Unterhaltungen, in denen diese Context Items von den Supportaccounts erfragt wurden. Das hier entstandene Truthset wurde anschließend verwendet um zu belegen, wie gut die

automatische Extraktion der Context Items funktionierte. Die Evaluation der Ergebnisse ergaben Werte für Precision und Recall von 81 - 99 %. [21]

Die automatisierte Erhebung der ermittelten Context Items erfolgte in zwei unterschiedlichen Verfahren. Das erste Verfahren behandelte die Bezeichnung des Betriebssystems und die des Gerätes. Diese ermittelten die Autoren aus *Pre-Defined Keyword Lists* und Listen der offiziellen App Store-Betreiber und Benutzer-Communities.

Zusätzlich wurde, wie in Abbildung 3.3 zu sehen, das *Crawled Dataset* auf alternierende und nicht formelle Bezeichnungen überprüft. Das beinhaltet Abkürzungen und Fehler in der Rechtschreibung. Die Tweets wurden in einzelne *Token* aufgeteilt und in einer *Word Vector Representation*, welche es erlaubt anhand der Abstände der Vektoren Ähnlichkeiten aufzuspüren, verarbeitet. Anschließend wurden diese Wörter bis zu einem gewissen Grad den offiziellen Bezeichnungen der *Pre-Defined Keyword Lists* hinzugefügt.

Diese Liste der Bezeichnung bedurfte es im letzten Schritt noch einem manuellen *Fine-Tuning*. Wörter wie 'horizon', die gleichzeitig Bezeichnungen für Geräte oder Betriebssystemen ähneln, entfernten die Autoren aus den *Pre-Defined Keyword Lists*. Vergleichbar erfolgte auch eine manuelle Bereinigung der *Pre-Defined Keyword List with alternative Spellings*. Bezeichnungen die zu allgemein und nicht relevant im Kontext der Entwicklung erschienen, wurden entfernt. [21]

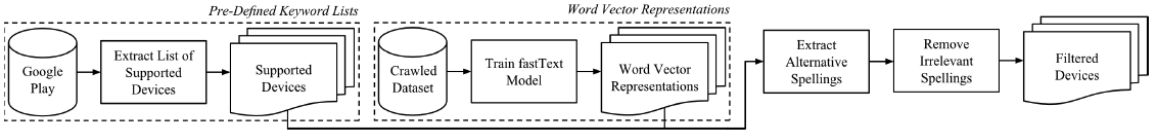


Abbildung 3.3: Erstellung gefilterter Bezeichnungen von Android-Geräten

In dem zweiten Verfahren, für die Ermittlung Betriebssystem- und Appversion, wurden ebenfalls *Pre-Defined Keyword Lists* für die offiziellen Bezeichnungen verwendet und das User-Feedback in einzelne *Token* aufgeteilt. Eine Herausforderung besteht in der Mehrdeutigkeit der App-Versionen zwischen den beiden Betriebssystemen *iOS* und *Android*. Da eine App für beide Betriebssysteme die selbe Versionsnummer verwenden kann, erlaubt dies keine Rückschlüsse von der App-Version auf das Betriebssystem. Eine zweite Herausforderung besteht in einem Feedback, welches in seinen detaillierten Bezeichnung jene offiziellen Angaben der *Pre-Defined Keyword List* übertrifft. Wird z.B. die *Subversion* zu einer offiziellen Version angegeben, dann kann diese nicht direkt verglichen

werden. Die Autoren haben hierzu einen *Version-Tree* implementiert, der es ermöglicht diese detaillierten Versionen einzuordnen. [21]

Um die Context Items der Betriebssystem- und Appversion zu extrahieren, werden die einzelnen Token mit dem *Version-Tree* verglichen und eindeutige Konfigurationen heraus gelöst. Bei Konflikten werden zusätzlich vorhandene Context Items zu Gerät und Betriebssystem zur Hilfe genommen, um diese aufzulösen. [21]

Abschießend bewerteten die Autoren die wachsende Bedeutung der Reaktion der Hersteller auf Feedback aus Sozialen Medien. Ein Drittel der erfassten *Bugreports* könnten, bei Analyse der Twitter-Daten, früher entdeckt werden. Ihr Ansatz kann dabei helfen fehlende Context Items zu ermitteln und das Feedback der Benutzer für einen Entwickler so aufzubereiten, dass dieser aktiv an einer Lösung arbeiten kann. Um die Aussagekraft des Ansatzes zu unterstützen, bedarf es aber noch weitere und fortführende Forschungen. Diese Arbeiten sollten den Ansatz auch auf Support-Accounts mit geringerer Popularität anwenden oder mögliche Auswirkungen des technischen Hintergrunds der Benutzer und der Anzahl der ausgetauschten Context Items erforschen. [21]

4 Konzept und Implementierung

Das hier entwickelte System zur Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen besteht aus zwei voneinander unabhängigen Teilen: einem Crawler und einem Extractor (vgl. Abbildung 4.1). Der Crawler ermittelt die Twitterunterhaltungen in Form einzelner Tweets bestimmter Serviceprovider und speichert diese in einer Datenstruktur. Der Extractor normalisiert die gesammelten Daten und ermittelt mit der Hilfe von *Pre-Defined Keyword Lists* Context Items in den Truthsets.

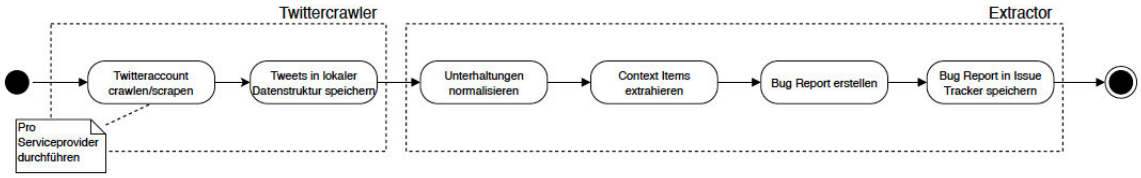


Abbildung 4.1: Übersicht automatisierte Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen

Während Webcrawler Internetseiten durchsuchen und weitere Links suchen, sortieren und priorisieren, ist der hier zu verwendende Twittercrawler ausschließlich auf der Suche nach Tweets und den dazugehörigen Antworten. Das Ziel der Suche sind Unterhaltungen zwischen Serviceprovidern und Benutzern. Der Einstiegspunkt für die Suche ist die Timeline der Serviceprovider, die auf Tweets der Benutzer antworten, also Tweets die über das Attribut *in_reply_to_status_id* verfügen. Außerdem wird mit *TwitterScraper* (vgl. Unterkapitel 2.2.2) die erweiterte Twitter-Suche zur Ermittlung der Kommunikation verwendet. Grundsätzlich werden zwei Abfragen verwendet: Tweets, die direkt an den Serviceprovider gerichtet sind und Tweets des Serviceproviders, inklusive der anfallenden Antworten.

Die Aufgabe des Extractors besteht darin, die zuvor gesammelten Tweets zu bereinigen und in ein zu verarbeitendes Format, dem *Truthset*, zu transformieren. Mit zuvor definierten *Pre-Defined Keyword Lists*, die Serviceproviderangaben mit abweichenden Bezeichnungen von Benutzern verbinden, werden Context Items aus den Unterhaltungen

extrahiert. Die Ergebnisse werden überprüft und anschließend als Bugreports in einem Issue Tracker gespeichert.

4.1 Konzeption des Crawlers

Der Twittercrawler ist in der Lage einen oder mehrere Twitter-Accounts zu verarbeiten und die Kommunikation zwischen Hersteller und Benutzer zu erfassen. Diese Kommunikation ist im Prinzip flüchtig, da Nutzer ihre Tweets nach einer gewissen Zeit löschen können oder Twitter seine API oder sonstige Onlinestrukturen ändern kann. Daher soll die Kommunikation extern gespeichert werden. Die Daten sollen für die spätere Verarbeitung strukturiert und mit wichtigen Informationen erfasst werden.

Es sollen Unterhaltungen gesichert werden, die von Benutzern mit den jeweiligen Serviceprovider initiiert werden und vielversprechend sind, was die Erhebung von Kontextinformationen betrifft. Um das volle Potential der Unterhaltungen einer Serviceprovider-Timeline zu erfassen, soll nicht nur die Twitter-API verwendet, sondern auch Techniken des Webscraping angewendet werden (siehe Abbildung 4.2). Diese haben den Vorteil, dass sie nicht den selben Beschränkungen der Twitter-API (nur die letzten 3.200 Tweets sind verfügbar) unterliegen.

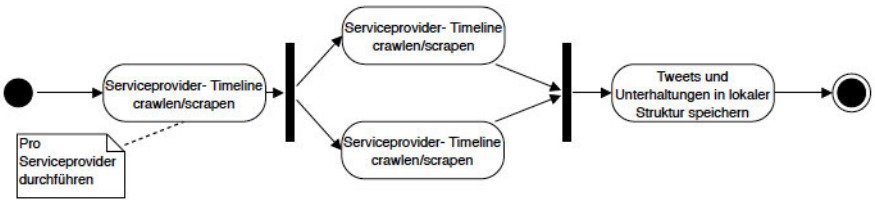


Abbildung 4.2: Übersicht Erfassung von Tweets und Unterhaltungen

Um Kontextinformationen aus Twitterunterhaltungen zu erheben, werden zuerst Datenstrukturen für Tweetobjekte benötigt, die anschließend in Unterhaltungen zusammengefasst werden. Zur späteren Auswertung und Einordnung der Serviceprovider sind Statistiken zu erfassen. Die benötigten Datenstrukturen werden in den folgenden Abschnitten beschrieben.

4.1.1 Tweet

Twitter hält eine große Menge an Attributen pro Tweetobjekt im JSON-Format vor. Für die Erhebung von relevanten Kontextinformationen werden aber nur eine Teilmenge dieser Attribute benötigt. Neben den IDs für das Tweetobjekt, Ersteller, Zuordnung zu anderen Tweets und des Textes, benötigt das Tweetobjekt Attribute für die Zuordnung zu einer Sprache und ein Erstellungsdatum. Die Tweet-ID des Tweetobjekts ist als eindeutiger Schlüssel zu verwenden. Die hier verwendeten Attribute sind:

- *ID* (Tweet-ID)
- *userid* (Tweet-Authr-ID)
- *screen_name* (Tweet-Autor Anzeigename)
- *in_reply_to_user_id* (Antwort an Tweet-autor-ID)
- *in_reply_to_user_screen_name* (Antwort an Tweet-autor Anzeigename)
- *in_reply_to_status_id* (Antwort an Tweet-ID)
- *created_at* (Timestamp JJJJ-MM-DD HH:MM:SS)
- *full_text* (Tweet-Text)
- *lang* (Sprachenkürzel)

4.1.2 Unterhaltung

Für die spätere Weiterverarbeitung sollen die Tweets in Unterhaltungen summiert werden. Die Annahme für eine Support-Unterhaltung ist, dass ein Nutzer einen Tweet an den Serviceprovider schreibt. Dieser Tweet verfügt über kein *in_reply_to_status_id*-Attribut und ist somit der Kopf einer verketteten Tweetreihe (siehe Abbildung 4.3). Die Antworten referenzieren dann über dieses *in_reply_to_status_id*-Attribut auf den Kopf der Tweetreihe oder wiederum auf die Antworten zu diesem Kopf. Für die Erhebung der relevanten Kontextinformationen soll durch die Summierung der Tweets in Unterhaltungen ein ganzheitlicher Kontext erfasst werden.

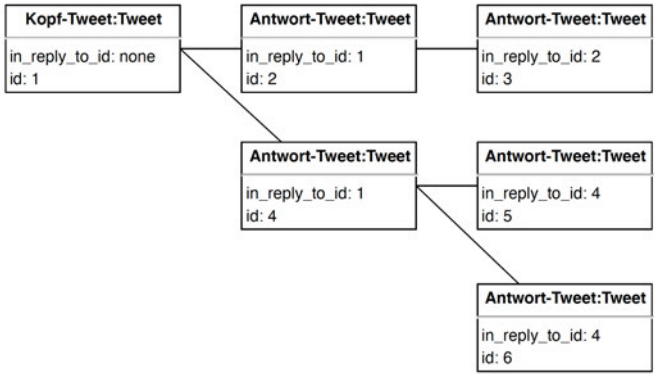


Abbildung 4.3: Schematische Verknüpfung einzelner Tweet zu einer Twitterunterhaltung

4.1.3 Statistik

Pro Serviceprovider sollen Statistiken erhoben werden. Diese fassen grundsätzliche Informationen zu dem Serviceprovider zusammen und bieten die Möglichkeit den Arbeitsfortschritt des Crawlers zu messen. Dazu sollen die folgenden Werte erhoben werden:

- *tweets* (Anzahl der gesammelten Tweets des Serviceproviders)
- *replies* (Anzahl der gesammelten Replies an den Serviceprovider)
- *conversations* (Anzahl der erstellten Unterhaltungen aus Tweets und Replies)
- *status_count* (Anzahl der Statusmeldungen (Tweets) des Serviceproviders)
- *followers_count* (Anzahl der Follower des Serviceproviders)

4.2 Konzeption des Extractors

Die Erhebung von Kontextinformationen fängt mit der Normalisierung der Basis von Tweets und Unterhaltungen an. Aus diesen werden anschließend einzelne Context Items extrahiert (siehe Abbildung 4.4).

Der Ablauf bedarf allerdings noch einer Festlegung und Aufbereitung bestimmter Context Items. Um diese aus den Unterhaltungen zu extrahieren, müssen Serviceprovideranga-ben in *Pre-Defined Keyword Lists* zusammengetragen werden. In einigen Fällen bietet es sich an, zu den Angaben der Serviceprovider alternative Bezeichnungen der Benutzer zu ermitteln und diese in *Pre-Defined Keyword Lists with alternative Spellings* (vgl. Unterkapitel 4.2.4) zusammenzufassen. Diese Listen dienen dann während der Extraktion als *Nachschlagewerk*.

Wenn im letzten Schritt zu einer Unterhaltung alle ermittelten Context Items zusammengefasst sind, müssen diese für einen Issue Tracker aufbereitet werden. Sollten die Context Items nicht komplett oder nicht eindeutig sein, müssen diese geklärt werden. Der Bugreport im Issue Tracker sollte anschließend die erfassten Context Items auflisten, die komplette Unterhaltung bereithalten, sowie die Möglichkeit anbieten direkt mit dem Benutzer in Kontakt treten zu können.

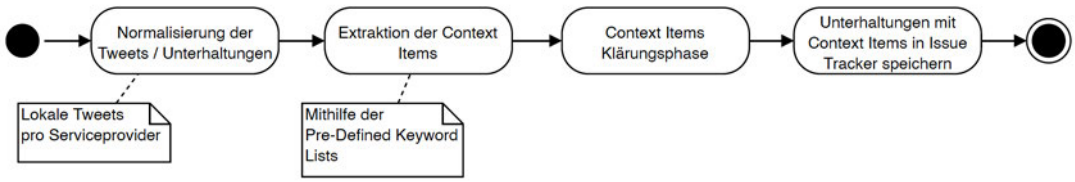


Abbildung 4.4: Übersicht Extraktion relevanter Kontextinformationen aus Twitterunterhaltungen

4.2.1 Normalisierung

Die Kommunikation zwischen Serviceprovider und Benutzer ist in den meisten Fällen eine Unterhaltung zwischen Menschen und somit in relativ freier Form (280 Textzeichen ggf. mit Bild oder Video [43][41]). Oftmals hat ein solcher Austausch nicht das ideale Format zur maschinellen Verarbeitung. Um die Kommunikation für die Verarbeitung vorzubereiten, sollen überflüssige Leerzeichen, Absätze, Erwähnungen von Accounts (@TwitterUser)

entfernt werden. Weiterhin sollen Groß- und Kleinbuchstaben in ein einheitliches, mit den *Pre-Defined Keyword Lists* übereinstimmendes Format gebracht werden.

4.2.2 Context Items

In Anlehnung an die Arbeit von Maleej und Martens [21] sollen die folgenden Context Items erhoben werden:

- Betriebssystem des Endgerätes
- Version des Betriebssystem
- Version der Applikation
- Gerätebezeichnung
- Fehlercode

4.2.3 Pre-Defined Keyword List

Die Context Items sind abhängig von der App des Herstellers. Die App lief oder läuft in verschiedenen Versionen auf einem oder mehreren Betriebssystemen, die wiederum verschiedene Versionen haben. Für Betriebssysteme gibt es wiederum kompatible Geräte. Diese müssen ermittelt und in den Pre-Defined Keyword Lists für die Erhebung verwaltet werden. Diese Erhebung kann händisch oder maschinell erfolgen. Es ist dabei abzuwiegen, ob der Aufwand einer maschinellen Erhebung den einer händischen Erhebung übersteigt.

4.2.4 Pre-Defined Keyword List with alternative Spellings

Für die Gerätebezeichnungen sollten alternative Schreibweisen ermittelt werden. Dies bietet sich an, da ein Gerät zwar einen Produktions- oder Modelnamen vom Hersteller bekommt, aber eventuell von einem Benutzer falsch oder in abgekürzter Schreibweise mitgeteilt wird. Das *iPhone 8 plus* wird dann z.B. zum *iPhone8+*. Hierzu müssen die Tweets der Benutzer analysiert werden und alternative Bezeichnungen mit den Herstellerangaben verglichen werden, um die *Pre-Defined Keyword List* zu erweitern.

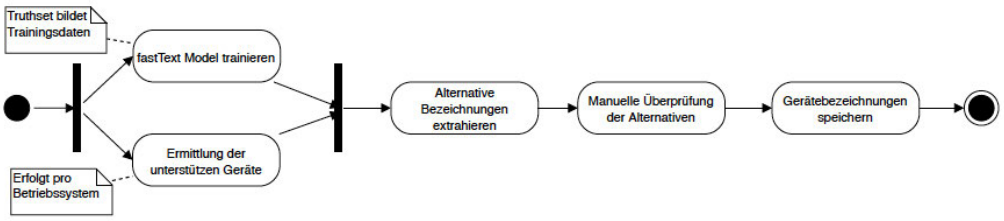


Abbildung 4.5: Erstellung Pre-definded Keyword List with alternative Spellings

In der Abbildung 4.5 ist dargestellt, dass das Training und die Ermittlung der unterstützen Geräte unabhängig voneinander erfolgen. Die alternativen Bezeichnungen werden dann anhand der erstellten Wortvektoren aus den unterstützten Geräte extrahiert und abschließend manuell überprüft.

4.2.5 Context Items Klärungsphase

Eine Klärung der ermittelten Context Items muss erfolgen, wenn die gewünschten Context Items nicht vollständig oder eindeutig ermittelt werden können. Solange die Context Items nicht eindeutig oder bis zu einem gewissen Grad ausreichend gefüllt sind, kann kein Bugreport erstellt werden (vgl. Abbildung 4.6). In jedem Fall soll zunächst ein Klärungsversuch unternommen werden.

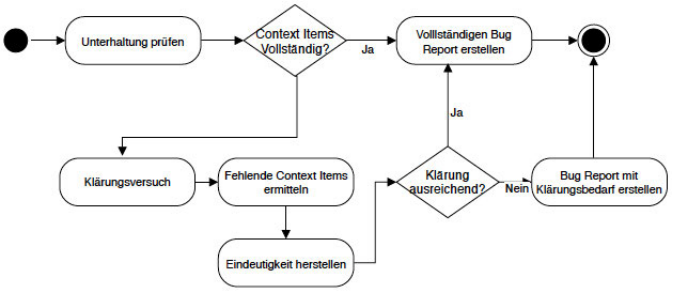


Abbildung 4.6: Context Items Klärungsphase

Fehlende Context Items können durch Abhängigkeiten einzelnen Context Items hergestellt werden. Es lässt sich z.B. von einer erhobene Gerätebezeichnung *iPhone* auf das Betriebssystem *iOS* schließen, da iPhones ausschließlich mit diesem Betriebssystem laufen.

Mehrdeutige Context Items können auftreten, wenn die Erhebung nach dem selben Muster erfolgt oder die Stichworte aus der *Pre-Defined Keyword List* ähnlich sind. Die

Erhebung der Version der App und des Betriebssystems kann mehrdeutig sein, wenn diese nach dem selben Schema erfolgen (Appversion: 14.1.2 Betriebssystemversion: 14.1.0). Diese Mehrdeutigkeit kann ebenfalls durch Abhängigkeiten zwischen bereits erfassten und eindeutigen Context Items geklärt werden. Im Zweifel ist es notwendig, mehrdeutige Werte in den Bugreport zu übergeben, um diese von einem Bearbeiter überprüfen zu lassen.

Mehrfach genannte Context Items treten in einer Unterhaltung auf, wenn mehrere Fehler, Fälle, Geräte oder verschiedenen Appversionen genannt werden. Der Extractor kann diese zwar verarbeiten, die Unterhaltung muss von einem Bearbeiter überprüft werden.

4.2.6 Bugreport

Der Bugreport soll die folgenden Merkmale enthalten:

- die komplette und nicht normalisierte Unterhaltung zwischen Benutzer und Serviceprovider
- die komplette normalisierte Unterhaltung mit Context Items pro Tweet
- die erhobenen Context Items
- Kontaktinformation des Benutzers für Erfolgsmeldung oder Rückfragen

4.3 Implementierung des Crawlers

Die Implementierung des Crawlers wurde in der Programmiersprache Python [44] vorgenommen und ist als Klasse realisiert. Sie verwendet Funktionen aus den vier Klassen: Serviceprovider, Tweet, Twitter-API und Storager (vgl. Abbildung 4.7). Der Crawler verwaltet eine Liste von Serviceprovidern, die in einer Schleife überprüft werden. Die Serviceprovider-Instanz verwaltet die Dictionaries für Tweets, Unterhaltungen und Statistiken.

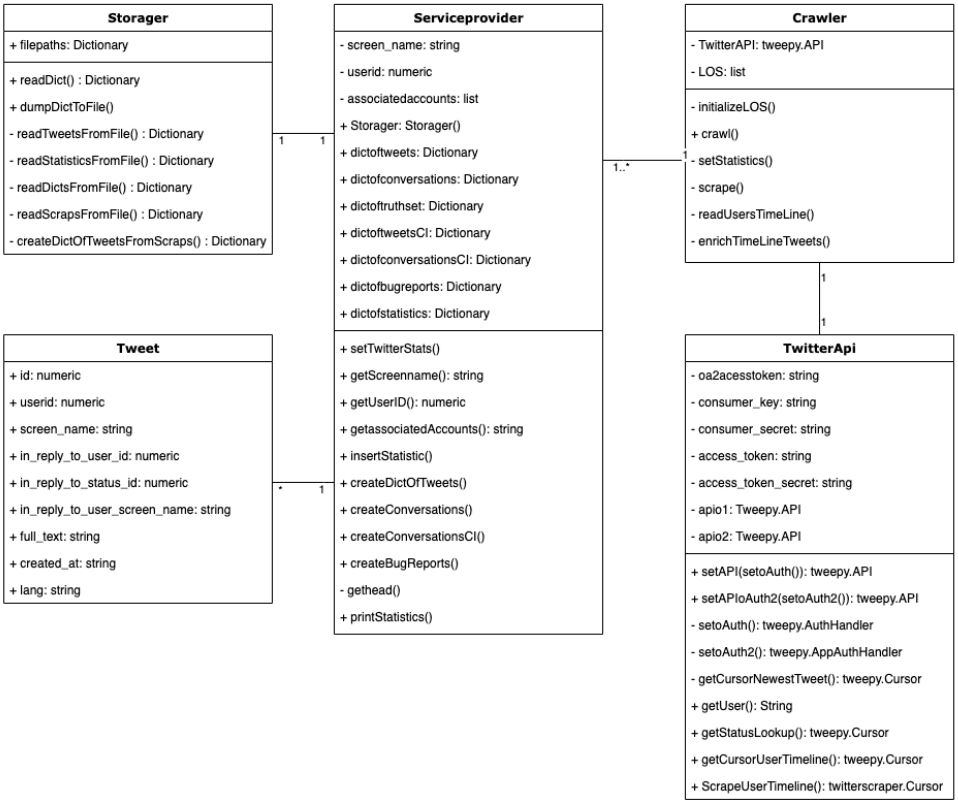


Abbildung 4.7: UML-Klassendiagramm Crawler

Die Klasse Serviceprovider bietet seine Methoden dem Crawler und dem Exractor an. Die jeweiligen Dictionaries werden mit Hilfe einer Storager-Instanz aus einem JSON-Format eingelesen und gespeichert. Die Storager-Klasse verarbeitet Aufrufe zum Laden eines Dictionaries mit dem Factory-Pattern. Die Twitter-API-Instanz wird für die Authentifizierung und Datenermittlung verwendet. Die Klasse *Tweet* ist eine Repräsentation der gewünschten Tweet-Attribute.

4.3.1 Ablauf

Der Crawler kann in einer übergeordneten Struktur, automatisiert in einer Endlosschleife, einfach oder im Stapel in Kombination mit der Extractor-Implementierung laufen und

schreibt die Ergebnisse regelmäßig in die angelegte Datenstruktur. Dabei ermittelt dieser Stammdaten des gecrawlten Twitter-Accounts und sammelt Tweets mit den Bibliotheken *Tweepy* und *TwitterScraper*. Abschließend werden die gesammelten Tweets in die Unterhaltungsstruktur abgelegt und Statistiken exportiert. Der Ablauf ist schematisch in der Abbildung 4.8 dargestellt.

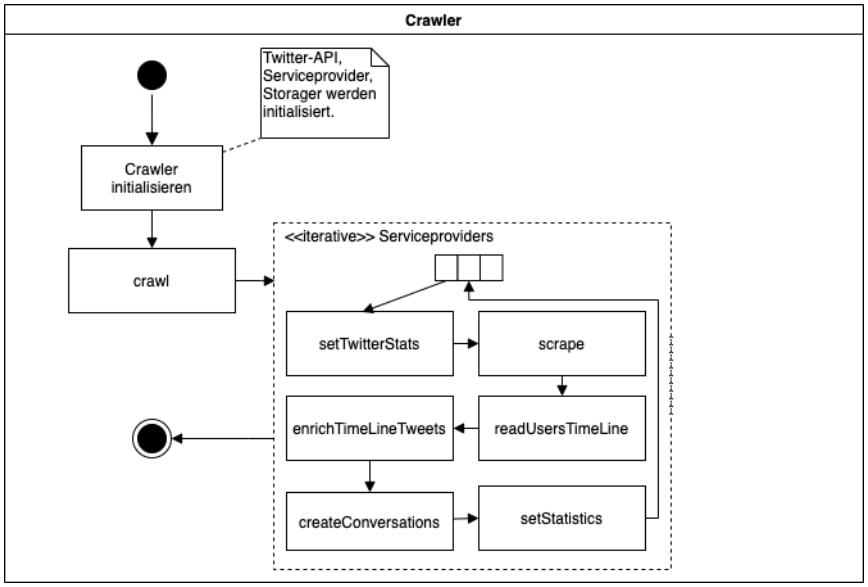


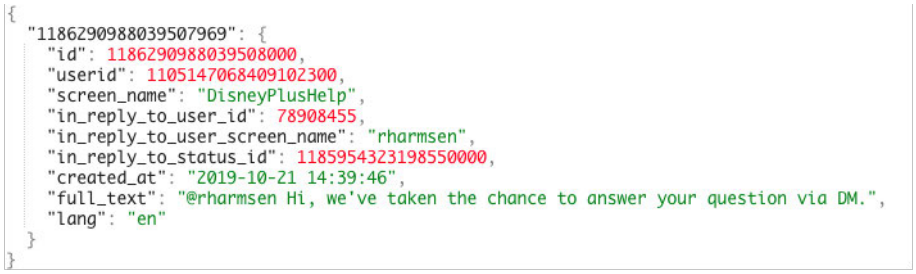
Abbildung 4.8: UML-Aktivitätendiagramm Crawler

4.3.2 Datenstrukturen

Da Twitter die Daten bereits im JSON-Format anbietet, wurden die gesammelten Tweets in der Python-Implementierung als Dictionary dargestellt. Die vorhandene Datenstruktur der Tweets wurde auf die benötigten Attribute gekürzt.

Tweet

Das Dictionary für die Tweets führt als Schlüssel die Tweet-ID und als Wert ein weiteres Dictionary mit den ausgewählten Tweet-Attributen, wie in Abbildung 4.9 zu sehen.

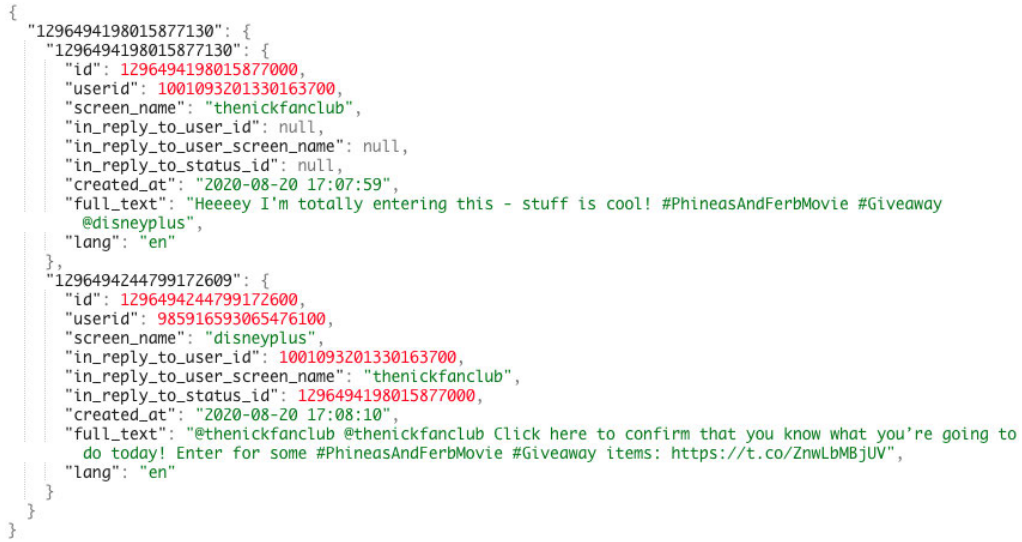


```
{
  "1186290988039507969": {
    "id": 1186290988039508000,
    "userid": 1105147068409102300,
    "screen_name": "DisneyPlusHelp",
    "in_reply_to_user_id": 78908455,
    "in_reply_to_user_screen_name": "rharmsen",
    "in_reply_to_status_id": 1185954323198550000,
    "created_at": "2019-10-21 14:39:46",
    "full_text": "@rharmsen Hi, we've taken the chance to answer your question via DM.",
    "lang": "en"
  }
}
```


Abbildung 4.9: JSON-Struktur Tweet

Unterhaltung

Für die spätere Weiterverarbeitung sortiert der Crawler die bereits ermittelten Tweets in Unterhaltungen ein. Diese werden im JSON-Format permanent gespeichert. Die Unterhaltungen sind in einem Dictionary gehalten, der Schlüssel ist die Kopf-Tweet-ID. Der Wert ist ein Dictionary von Tweets, die mit dem Kopf-Tweet und miteinander verkettet sind. Der Kopf jeder Unterhaltung ist ein Tweet ohne Wert in dem *in_reply_to_status_id*-Attribut (siehe Abbildung 4.10).



```
{
  "1296494198015877130": {
    "1296494198015877130": {
      "id": 1296494198015877000,
      "userid": 1001093201330163700,
      "screen_name": "thenickfanclub",
      "in_reply_to_user_id": null,
      "in_reply_to_user_screen_name": null,
      "in_reply_to_status_id": null,
      "created_at": "2020-08-20 17:07:59",
      "full_text": "Heeeey I'm totally entering this - stuff is cool! #PhineasAndFerbMovie #Giveaway
        @disneyplus",
      "lang": "en"
    },
    "1296494244799172609": {
      "id": 1296494244799172600,
      "userid": 985916593065476100,
      "screen_name": "disneyplus",
      "in_reply_to_user_id": 1001093201330163700,
      "in_reply_to_user_screen_name": "thenickfanclub",
      "in_reply_to_status_id": 1296494198015877000,
      "created_at": "2020-08-20 17:08:10",
      "full_text": "@thenickfanclub @thenickfanclub Click here to confirm that you know what you're going to
        do today! Enter for some #PhineasAndFerbMovie #Giveaway items: https://t.co/ZnwLbMBjUV",
      "lang": "en"
    }
  }
}
```


Abbildung 4.10: JSON-Struktur Conversation

Statistik

Für die Überprüfung der Arbeitsfortschritte wurden Statistiken ermittelt, die die gesammelten Tweets und grundsätzliche Informationen zu dem Serviceprovider liefern. Es handelt sich um ein Dictionary. Der Schlüssel wird durch das Tagesdatum gebildet, der Wert ist ein Dictionary der an diesem Tage ermittelten Werte (siehe Abbildung 4.11).


```
{
  "2020-08-12": {
    "tweets": 23985,
    "replies": 27934,
    "conversations": 20519,
    "status_count": 25168,
    "followers_count": 37744
  }
}
```


Abbildung 4.11: JSON-Struktur Statistik

4.4 Implementierung des Extractor

Der Extractor ist als Python-Klasse implementiert und verwendet die Serviceprovider-Klasse, mit der dazugehörigen Storager-Klasse. Diese verwaltet eine Liste von Serviceprovidern, erstellt aus den gesammelten Daten Truthsets und extrahiert aus diesen, mithilfe der Stichworte aus den *Pre-Defined Keyword Lists*, *Context Items*. Für diese Arbeitsschritte werden reguläre Ausdrücke und die Bibliothek *spaCy* verwendet. Die Extraktion bildet den Schwerpunkt dieser Arbeit und wird im Unterkapitel 4.4.4 genauer beschrieben und in Kapitel 5 ausgewertet. Die Erstellung der Pre-Defined Keyword Lists erfolgte hauptsächlich manuell und in Skripten, dies wird im Unterkapitel 5.2 beschrieben. Die Normalisierung und die Verwaltung der Tweets mit Context Items in Datenstrukturen für Truthsets, Tweets, Unterhaltungen und Bugreports sind grundsätzlich implementiert. Die Context Items Klärungsphase ist nicht implementiert.

4.4.1 Ablauf

Der Extractor normalisiert für einen oder mehrere Serviceprovider ermittelte Twitterdaten zu einem Truthset, aus welchem dann die Context Items extrahiert werden. Die extrahierten Context Items werden anschließend zu dem betreffenden Tweet in einer weiteren Datenstruktur abgelegt. Zur Weiterverarbeitung werden diese Tweets in Unterhaltungen zusammengefasst. Wenn diese Unterhaltungen eine bestimmte Menge Context Items aufweisen, werden diese mit allen verfügbaren Daten als Bugreport abgelegt.

4.4.2 Datenstrukturen

Für die Verarbeitung der relevanten Tweets werden diese gefiltert und in Truthset gespeichert, diese unterscheiden sich nicht von der Tweetstruktur (siehe Abbildung 4.9). Nach erfolgreicher Extraktion von Context Items erfolgt eine Speicherung der Tweets mit zusätzlichen Context Items-Attributen als *Tweet-CI*. Eine weitere Struktur erzeugt für jede Kopf-ID der Tweets mit Context Item ein Dictionary, in dem jeweils die unverarbeiteten und verarbeiteten Tweets enthalten sind. Diese werden bei der Erzeugung der Bugreports beigefügt. Der Bugreport bildet die letzte Struktur und enthält neben den Unterhaltungen die gesammelten Context Items und einen Link zur Unterhaltung.

Tweet-CI

Das Dictionary für die Tweet-CI ist analog zu der JSON-Struktur des Tweets aufgebaut, verfügt aber pro Context Item über ein weiteres Attribut (siehe Abbildung 4.12).


```
{
  "996385903156383745": {
    "id": 996385903156383700,
    "userid": 32615207,
    "screen_name": "BradBrisco",
    "in_reply_to_user_id": 497340309,
    "in_reply_to_user_screen_name": "SpotifyCares",
    "in_reply_to_status_id": 996385068229779500,
    "created_at": "2018-05-15 13:44:45",
    "full_text": "iphone 6 ios 11.3 spotify version 8.4.52.796 it's not app-breaking it's just odd that it
      happens maybe once or twice a day",
    "lang": "en",
    "SV": [
      "11.3"
    ],
    "AV": [
      "8.4.52.796"
    ],
    "DEV": [
      "iphone 6"
    ],
    "PF": [
      "ios"
    ],
    "EC": []
  }
}
```


Abbildung 4.12: JSON-Struktur Tweet-CI

Unterhaltung-CI

Die Unterhaltung-CI (siehe Abbildung 4.13) vereint für die spätere Verarbeitung die ursprüngliche Unterhaltung (*RAW*) mit jenen Tweets (*CI*), die ermittelte Context Items enhalten.

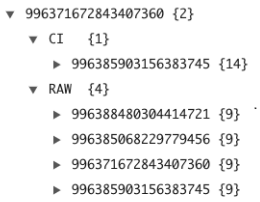


Abbildung 4.13: JSON-Baumstruktur Unterhaltung-CI

Bugreport

Der Bugreport (siehe Abbildung 4.14) vereint im letzten Schritt die Verlinkung zur Unterhaltung mit dem Attribut *LINK*, die gesammelten Context Items unter dem Attribut *BR*, sowie die Rohfassung der Unterhaltung und jene Unterhaltung mit enthaltenen Context Items.

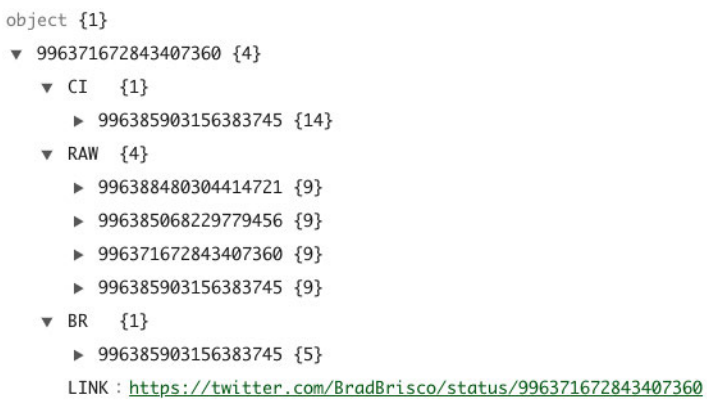


Abbildung 4.14: JSON-Baumstruktur Bureport-CI

4.4.3 Normalisierung

Die gesammelten Tweets werden normalisiert, indem zuerst alle Tweets des Serviceprovider und ihm zugehörigen Twitter-Accounts ausgefiltert werden. Die verbleibenden Tweets werden mit Hilfe von regulären Ausdrücken um Leerzeichen, Twitter-Erwähnungen, Hashtags und sonstigen Sonderzeichen, die nicht zu einem Wort gehören bereinigt. Abgelegt in einer Datenstruktur sind diese zur Weiterverarbeitung bereit.

4.4.4 Extraktion Context Items

Für die Extraktion der Context Items werden zuerst die zugehörigen Daten aus den *Pre-Defined Keyword Lists* und *spaCy*-Phrasematcher für die Gerätebezeichnungen, App- und Betriebssystemsversionen für den jeweiligen Serviceprovider geladen. Darauf folgend wird jeder Tweet des Truthsets auf enthaltene Context Items überprüft. Die Extraktion erfolgt in zwei Phasen. In Phase I werden die Texte analysiert und mögliche Context Items in spezifische Sets geschrieben. In Phase II werden Überschneidungen zwischen möglichen Versionen bereinigt.

Phase I analysiert alle im Tweet enthaltenen Wörter und sichert diese zur Überprüfung mit einer *Windowsize* von 1, Vorgänger und Nachfolger in einer allgemeine Liste mit möglichen Versions-Context Items (vgl. Algorithmus 1). Dazu werden die Wörter unterschiedlich verarbeitet. Numerische Werte werden mit Vorgänger und Nachfolger in einer Liste mit möglichen Versionen gespeichert. Nicht eindeutig numerische Wörter, die aus numerischen und nicht-numerischen Werte zusammengesetzt sind, werden mit einem regulären Ausdruck überprüft und ebenfalls mit Vorgänger und Nachfolger der Liste hinzugefügt.

Algorithm 1 Versionsmatching mögliche Versionen

```
1: PossibleMatches ← list(), outlist ← loadOutlist(), doc ← nlp(TweetText)
2: for Token in doc do
3:    if Token.is_num() then
4:       predecessor ← Token.nbor(−1), successors ← Token.nbor()
         {.nbor() yields string or empty string}
5:       PossibleMatches.append([predecessor, Token, successors])
6:    else
7:       for Match in re.findall(r'\w(\d + \. ∗ \d ∗ \. ∗ \d∗)', Token.Text) do
8:          predecessor ← Token.Text.split(Match)[: 1]
9:          successors ← Token.Text.split(Match)[1 :]
10:         PossibleMatches.append([predecessor, Match, successors])
11:      end for
12:   end if
13: end for
```

Diese möglichen Treffer werden im nächsten Schritt mit ihren Vorgänger und Nachfolgern gegen eine Ausschlussliste geprüft. Hierbei werden die Werte übersprungen, die auf Datum, Währungen, Zeit oder spezielle Wertangabe hindeuten. Damit soll unter anderem verhindert werden, dass z.B. *14.34 $* als Appversion missverstanden werden kann.

Die verbliebenen möglichen Betriebssystem- oder Appversionen werden auf direkte Übereinstimmungen mit den *Pre-Defined Keyword Lists* geprüft und sofort der spezifischen Sets hinzugefügt. Andere Versionen, die in Teilen mit einer bekannten Version übereinstimmen, werden entsprechend hinzugefügt. Eine teilweise Übereinstimmung bedeutet für eine solche Version, dass mindestens eine Übereinstimmung in den ersten beiden Teilen der Version vorliegt. Die mögliche Version 14.34 wird als Teilversion der Version 14.34.11 akzeptiert, die mögliche Version 14 hingegen nicht.

Es werden auch die Vorgänger und Nachfolger gegen mögliche Versionssynonyme geprüft und bei Übereinstimmung einer spezifischen Liste der möglichen Context Items hinzugefügt.

Die Sammlung der möglichen Versionsbezeichnungen, Betriebssysteme und die Gerätebezeichnungen wird mit *PhraseMatchern*, welche die genauen Bezeichnungen der Context Items enthalten, komplettiert. *PhraseMatcher* sind in der *spaCy*-Bibliothek enthalten und ermöglichen es einen Eingangstext gegen die im *PhraseMatcher* enthaltenen Stichworte zu prüfen. Dabei ist es möglich, Stichworte eindeutig zu identifizieren und miteinander zu vergleichen. Ist bspw. in einem Text die Gerätebezeichnung 'iphone 7' enthalten und

im *PhraseMatcher* befinden sich die Stichworte 'iphone' und 'iphone 7', die als potentielle Matches für die Gerätebezeichnung in Frage kommen, wird nur der größte Match übernommen (vgl. Algorithmus 2). Außerdem ist es möglich, nicht-numerische Versionsbezeichnungen, wie *MacOS: Sierra*, *Big Sur* und *Android: Cupcake*, *Ice Cream Sandwich* zu identifizieren.

Algorithm 2 Devicematching PhraseMatcher

1: $PossibleDEV \leftarrow PossibleDEV$, $Matcher \leftarrow PhraseMatcher()$
{Set der möglichen Gerätebezeichnungen, PhraseMatcher mit geladenen Gerätebezeichnungen}
2: $doc \leftarrow nlp(TweetText)$
3: $device\_matches = matcher(doc)$
4: **for** *matchdevice* in $spacy.util.filter\_spans(device\_matches)$ **do**
5: possibleDEV.add(matchdevice)
{filterspan yields longest Match}
6: **end for**

Während der Überprüfung der Betriebssystem- und Appversion werden die bereits gefundenen Token aus dem *PhraseMatcher* vor der Zuordnung in eine der spezifischen Sets darauf überprüft, ob sie ein Teil einer Gerätebezeichnung sind (siehe Algorithmus 3). In diesem Fall wird der betreffende Token aus der Liste der gefundenen Token und der möglichen Versionen entfernt. Sollte die Versionsbezeichnung erneut in dem Tweet vorkommen und nicht in einer Gerätebezeichnung enthalten sein, wird diese dann erneut hinzugefügt.

Algorithm 3 Versionmatching PhraseMatcher

```
1: PossibleSV ← PossibleSV, Matcher ← PhraseMatcher()
   {Set der möglichen Versionsbezeichnungen, PhraseMatcher mit geladenen Versionsbezeichnungen}
2: doc ← nlp(TweetText)
3: sv_matches = matcher(doc)
4: for start_devices, end_devices in device_matches do
5:    for start_av, end_av in sv_matches.copy() do
6:       if start_devices <= start_av < end_device then
7:          sv_matches.remove((start_av, end_av))
8:          if (start_av, end_av)inPossibleSV then
9:             possibleSV.remove((start_av, end_av))
10:         end if
11:      end if
12:   end for
13: end for
14: for matchav in spacy.util.filter_spans(av_matches) do
15:    possibleAV.add(matchav)
      {longest match of cleaned av_matches}
16: end for
```

Nach diesen Arbeitsschritten ist Phase I abgeschlossen.

Im letzten Schritt werden Konflikte in möglichen Betriebssystem- und Appversion, nach Möglichkeit, aufgelöst. Dieser Versuch Versionskonflikte aufzulösen ist in dem Algorithmus 4 dargestellt. Je nach Länge des zu überprüfenden Konfliktes und ermittelten Betriebssystemen wird eine Prüfliste erstellt. Es wird die längste Übereinstimmung pro Context Items gebildet und verglichen.

Algorithm 4 Auflösen von Versionskonflikten

```
1: PossibleSV, PossibleAV, PossiblePF
   {Sets der möglichen Betriebssystem-, Appversion und Betriebssystem}
2: SetofSystemversions, SetofAppVersions ← getPDKVersionsbyPF(PossiblePF)
   {Sets der hinterlegten PDK-Versionen abhängig von ermittelten Betriebssystemen}
3: SetofIntersections ← PossibleSV.intersection(PossibleAV)
4: for Version in SetofIntersection do
5:    longestMatchSV, longestMatchAV ← emptyString
6:    for SubVersion in Version do
7:       if Subversion in SetofSV and not in SetofAV then
8:          longestMatchSV ← Subversion
9:       end if
10:      if Subversion not in SetofSV and in SetofAV then
11:         longestMatchAV ← Subversion
12:      end if
13:   end for
14:   if longestMatchSV > longestMatchAV then
15:      possibleAV.remove(Version)
16:   else if longestMatchSV < longestMatchAV then
17:      possibleSV.remove(Version)
18:   end if
19: end for
```

Wenn eine eindeutige Übereinstimmung zwischen möglicher Version und Prüfliste hergestellt werden kann, wird die Version aus der anderen möglichen Liste entfernt. Kann keine eindeutige Zuordnung erfolgen, werden beide möglichen Versionen behalten.

5 Evaluation

Die verwendeten Daten in dieser Arbeit reichen von Herstellerangaben bezüglich der vorhandenen Betriebssystem- und Appversion, bis zu den unterstützten Geräten und den erhobenen Twitterunterhaltungen. Diese dienen als Trainingsdaten für die Ermittlung alternativer Bezeichnungen der Herstellerangaben und bilden die Grundlage aus der die relevanten Context Items erhoben werden sollen. Im weiteren Verlauf des Kapitels wird beschrieben, welche Context Items erhoben und welche Hersteller dafür herangezogen wurden.

Für die Validierung des extrahierten Kontextes wurde exemplarisch für zwei Gruppen der beschriebene Ansatz überprüft. Dazu wurden die Accounts *SpotifyCares*, *SpotifyHilft* und *NetflixHelps* für die Betriebssysteme *iOS* und *Android* in einer Kontrollgruppe Mobile gemeinsam betrachtet. Die zweite Gruppe Kontrollgruppe IDE bilden *intelliJsupport*, *Pycharm*, *webstormIDE*, *visualstudios*. Sie betreiben ihre Software für die Betriebssysteme *Linux*, *Windows* und *macOS*. Die Kontrollgruppe Mobile wurde zudem mit den Ergebnissen der Arbeit von Maleej und Martens [21] verglichen.

Die Verifikation erfolgte über die händische Annotation ausgewählter Tweets aus dem Truthset der jeweiligen Serviceprovider, die mit dem extrahierten Werten des Extractors verglichen wurde. Die Annotation erfolgte unter Verwendung des Tools *doccano*.

Zum Schluss werden im Unterkapitel Interpretation (5.7) die Ergebnisse allgemein und in ihren Besonderheiten beschrieben.

5.1 Vorverarbeitung der Daten

Um den Extractor mit sinnvollen *Pre-Defined Keyword Lists* zu versorgen, wurden pro Context Item Konfigurationsdateien erstellt und mit Herstellerangaben befüllt (vgl. Unterkapitel 5.2). Hierzu wurden Herstellerangaben zu Versionen übernommen und Geräte-

bezeichnungen gesammelt, aufbereitet und mit einem FastText-Modell (5.3) auf alternative Bezeichnungen trainiert. Pro Serviceprovider wurde zusätzlich noch ein Annotations-Set erstellt. Hierzu wurden Unterhaltungen untersucht, die mindestens ein gewähltes Stichwort enthielten (vgl. Unterkapitel 5.4).

5.2 Context Items

Für die Kategorien **Betriebssystem**, **Betriebssystemversion**, **Appversion** und **Gerätebezeichnung** wurden Konfigurationsdateien angelegt. Mögliche **Betriebssysteme** umfassen die Kategorien *iOS, Android, macOS, Linux* und *Windows*. Diese Kategorien verweisen dann auf weitere Produktbezeichnungen, bspw. *Ubuntu* für *Linux* oder *Win7* für *Windows*. Die **Betriebssystemversionen** werden unter den Betriebssystemen gebildet und umfassen alleine für *iOS* und *Android* 222 unterschiedliche Versionen. Da der Marktanteil der Betriebssysteme auf mobilen Geräten zu 99 % aus *iOS* und *Android* besteht, werden ausschließlich diese Betriebssysteme ermittelt. Für Desktopbetriebssysteme sieht es in der Aufteilung ähnlich aus, jedoch mit anderen Herstellern. Hier teilen sich *Windows* (76 %) und *macOS* (17 %). *Linux* ist mit (1 %) Marktanteil nicht sehr relevant, in der hier betrachteten Gruppe der IDE-Hersteller dennoch mit aufgeführt. [10][11]

Die **Appversionen** wurden pro Hersteller und Betriebssystem erfasst. Spotify betreibt bspw. seine App für *iOS, Android, macOS Windows, Windows Mobile, Linux, Amazon (Android)* und *Chromebook*. Wenn man die auf *Android* basierten Betriebssystem bei Spotify zusammenfügt, bleiben dennoch fünf unterschiedliche Betriebssysteme mit möglicherweise unterschiedlichen Appversionsverläufen.

Die **Gerätebezeichnungen** werden pro Betriebssystem erfasst und die Herstellerangaben mit dem Benutzer-Feedback in *Pre-Defined Keyword List with alternative Spellings* verarbeitet (vgl. Unterkapitel 4.2.4).

Der **Fehlercode** wurde durch Stichworte ermittelt, die sich im Truthset neben numerischen Token befanden und verfügt über keine Konfigurationsdatei.

Der hier gezeigte Ansatz bietet zwar grundsätzlich die Möglichkeit, sämtliche Variationen zu bearbeitet, dennoch wurde pro Hersteller eine Auswahl in den einzelnen Kategorien getroffen, um diese in Kontrollgruppen zu vergleichen.

5.2.1 Betriebssysteme

Die Context Items für die Betriebssysteme sind 1. *iOS*, 2. *Android*, 3. *macOS*, 4. *Linux*, 5. *Windows*. Sie stehen für Betriebssystemgruppen, für die weitere Variationen und Untergruppen vorhanden sind. Bspw. *Linux* für die Linux-Distributionen Ubuntu, Debian, Arch-Linux. Windows fasst die Betriebssystemvarianten Windows 7, Windows XP, Windows Vista, usw. zusammen. Die *Pre-Defined Keyword Lists* wurden manuell zusammen gestellt und enthalten weniger als 50 Einträge in den Untergruppen. Die Zusammenstellung orientiert sich am Serviceprovider. Jetbrains mit seinem Produkt IntelliJ bietet seine IDE für Linux, Windows und macOS an. Android ist bspw. ein Linux-Derivat, ist aber in der Unterkategorie von *Linux* nicht enthalten, da die IDE nicht auf *Android* eingesetzt wird. Ebenso wird auf sehr veraltete Versionen in anderen Kategorien verzichtet. (Wenn die Desktop-App von Spotify bislang nicht mit Windows XP kompatibel war, dann ist Windows XP nicht relevant.)

5.2.2 Betriebssystemversion

Für die Betriebssystemen *iOS, Android, macOS* wurden die Versionen manuell ermittelt und jeweils einer *Pre-Defined Keyword List* hinzugefügt. Die Listen für *Linux* und *Windows* wurden aufgrund ihrer großen Streuung der Unterkategorien hingegen nur vereinzelt gepflegt. Insgesamt wurden ca. 500 Betriebssystemversionen erfasst.

5.2.3 Appversion

Pro Serviceprovider wurde eine *Pre-Defined Keyword List* mit dem Betriebssystem und den darin eingesetzten Appversionen erstellt. Die Informationen wurden entweder direkt beim Hersteller oder bei Anbieter für Mobile-Marktinformationen *Appannie* zusammengestellt. [1][2][3][4][5]
Für 18 verschiedene Applikationen (die zwei erfassten Twitter-Accounts von Spotify bieten das selbe Produkt an) sind mehr als 6.500 Appversionen ermitelt worden. Dort wo es die Daten es erlaubten wurden die Einträge per CSV-Import eingelesen.

5.2.4 Gerätebezeichnung

Die Gerätebezeichnung als Context Item soll das eingesetzte Gerät ermitteln. Die *Pre-Defined Keyword Lists* für die Betriebssysteme *iOS, Android* und *macOS* sind gepflegt, da es eine begrenzte und offizielle Auflistung von kompatiblen Geräte gibt, die keine Überschneidungen aufweisen.

Für die Betriebssysteme Linux und Windows gibt es diese Zuordnung nicht direkt. Für diese Betriebssysteme wurden die *Pre-Defined Keyword Lists* mit einigen gängigen Bezeichnungen wie *notebook, pc, computer, thinkpad* gefüllt. Ein Rückschluss auf das Betriebssystem erfolgt in keinem Fall, da es auch möglich ist, ein *macOS*-Gerät mit einem anderen Betriebssystem zu betreiben.

5.2.5 Fehlercode

Die Identifizierung des Fehlercodes erfolgt hier ausschließlich über benachbarte Token. Der Fehlercode ist selbst nicht in einer Pre-Defined Keyword List realisiert.

5.3 Training Pre-Defined Keyword Lists

Für die Erkennung von abweichenden Bezeichnungen der unterstützen *iOS*- und *Android*-Geräte, wurde mit einem einfachen Python-Skript die bereinigten Tweets des Truthsets der betreffenden Seviceprovider eingelesen und verarbeitet. Ein *FastText*-Modell wurde mit 2.4 Millionen Tweets in 10 Epochen trainiert und gesichert. Mit diesem Modell konnte anschließend alternative Bezeichnungen für die Geräte ermittelt werden. Die Vorschläge aus dem *FastText*-Modell sind erneut maschinell gefiltert und anschließend händisch der Pre-Defined Keyword List with alternative Spellings hinzugefügt worden.

Die Liste der *iOS*-Geräten ist mit 67 relativ übersichtlich. Aus ihr konnten 53 weitere Bezeichnungen wie bspw. *iphone6s+*, für die offizielle Bezeichnung *iphone 6s plus* ermittelt werden.

Die offizielle Liste der unterstützten *Android*-Geräte hat 30.431 Einträge. Diese Einträge sind aufgeteilt in vier Kategorien: *Retail Branding, Marketing Name, Device* und *Model*. Die Herstellerangaben sind allerdings so unterschiedlich, dass eine bestimmte Kombination der Kategorien nicht zu einem einheitlichen Ergebnis führt (siehe Tabelle 5.1). Darum

wurde zunächst jede Bezeichnung mit mindestens vier Ziffern der *Pre-Defined Keyword List* hinzugefügt. Anschließend wird das *Retailed Branding* mit den restlichen Kategorien, getrennt durch ein Leerzeichen, verbunden. Dieses Vorgehen resultiert in 95.055 Einträgen. Da einige Bezeichnungen in den Kategorien so allgemein sind, dass diese in natürlicher Sprache häufig vorkommen würden, müssen sie erneut bereinigt werden. Dazu zählen Jahreszahlen, Nomen, Adjektive und Verben, bspw: *2018, cherry, feeling, look, extrem.* Die Bereinigung erfolgte über eine Überprüfung aller Bezeichnungen mit einer einfachen Wortlänge. Die aufbereitete Liste verfügt über 93.683 Einträge.

Retail Branding	Marketing Name	Device	Model
Sony	Xperia 5	J8270	J8270
Redmi	K30 PRO	lmi	K30 Pro
QMobile	HD Plus	QMobile_HD_Plus	HD Plus
QMobile	J1	J1	QMobile J1
Echo Mobile	FEELING	FEELING	FEELING
Echo Mobile	FUSION	ECHO_FUSION	ECHO_FUSION
Explay	Gravity	T4728	Gravity
Echo Mobile	FEELING	FEELING	FEELING
Vivo	vivo 2015	2015	vivo 2015
Vivo	vivo 2018	2018	vivo 2018

Tabelle 5.1: Exemplarische Diversität der Herstellerangabe in unterstützten Android-Geräten

Um weitere Bezeichnungen zu ermitteln, wurden die Gerätebezeichnung ebenfalls mit FastText-Modell verarbeitet. Die resultierenden 2.306 Einträge wurden durch Filter verringert und letztendlich in alphabetischer Reihenfolge ausgegeben und per Hand auf 292 abweichende Gerätebezeichnungen festgelegt (siehe Tabelle 5.2).

Betriebssystem	PDK	PDKAS	+/-
iOS	67	120	+ 53
Android	93.683	93.775	+ 292

Tabelle 5.2: Trainingsresultat der Gerätebezeichnung

5.4 Annotation

Mit der Annonation der gefilterten Tweets soll später die Performance des Extractor gemessen werden. Ziel ist es pro Tweet die einzelnen Context Items zu markieren und

diese anschließend so aufzubereiten, dass die Ergebnisse des Extractor mit denen der Annotation verglichen werden. Für die Annotation in *doccano* wurde pro Serviceprovider ein Projekt erzeugt. In diesen Projekten wurden für jeden Tweet Textstellen mit Labeln des jeweiligen Context Items versehen. Hierzu wurden insgesamt 6.222 Tweets überprüft und 4.533 Label, teilweise mehrfach, vergeben. Das erfolgte, händisch, in zwei Läufen: Der erste Lauf zur Annotation und der zweite Durchlauf zur Kontrolle.

Für den Vergleich der Ergebnisse wurden Mehrfachlabel entfernt und alphabetisch sortiert. Die Ergebnisse der Klassifikation wurden automatisch ermittelt, händisch kontrolliert und fehlende oder falsch zugeordnete Label korrigiert.

Jedes Annotations-Set der Serviceprovider wurde aus Unterhaltungen erstellt, in denen bestimmte Stichworte mindestens einmal vorkommen (siehe Algorithmus 5). Für die Kontrollgruppe Mobile waren es die Stichworte *ios* und *android*, in der Kontrollgruppe IDE wurde mit den Stichworten *linux*, *windows* und *macos* gesucht.

Algorithm 5 Zusammenstellung Annotations-Sets

```
1: for Tweet in Truthset do
2:   if StichwortinTweet.Text then
3:     GefilterteKopftweets.add(Unterhaltung[GetKopfTweet(Tweet)]
4:   end if
5: end for
6: for Unterhaltung in GefilterteKopftweets do
7:   AnnotationsSetCounter=0
8:   if AnnotationsSetCounter <= 1000 then
9:     Antwortcounter=0
10:    for Antwort in Unterhaltung do
11:      Antwortcounter+=1
12:      if Antwort.Screen_Name! = Serviceprovider then
13:        OUTPUTFILE.add(Antwort)
14:        AnnotationsSetCounter+=1
15:      end if
16:    end for
17:  end if
18: end for
```

Zuerst wurden Tweets mit einem Stichwort aus dem Truthset ermittelt und die dazugehörigen Unterhaltungen in ein Dictionary gesichert. Pro Unterhaltung wurden maximal zehn Antworten überprüft. Nach 1.000 hinzugefügten Tweets werden keine neuen Unterhaltungen mehr verarbeitet. Pro passenden Tweet wurde der Text normalisiert und mit weiteren Meta-Daten in eine Datei geschrieben, um in *doccato* annotiert zu werden.

5.5 Validierung der Ergebnisse mit Extractor

Die Ergebnisse der Validierung zwischen Annotation und Extraktion wurden zum Teil maschniell vorgenommen. Anhand der Tweets aus dem Annotations-Set wurden die Ergebnisse des Extractors ermittelt und in eine CSV-Datei ausgespielt. Hier wurden pro Tweet der Link zum Tweet, der Text und pro Context Item jeweils eine Spalte mit dem Ergebnis der Annotation und der Extraktion exportiert. Während des Exports wurden auch die Ergebnisse verglichen und das Resultat der Klassifizierung zur Auswertung hinzugefügt. In vier Spalten pro Context Items wurde dann eine *1* im jeweiligen Feld (*TP,FP,TN,FN*) erfasst. Die Ergebnisse konnten so zügig erzeugt und effizient verglichen werden.

In der Kontrollgruppe IDE befinden sich Anbieter für die Anwendungsentwicklungssoftware. Ihre Betriebssystemarten sind macOS, Windows und Linux. Die Systemversionen sind sehr divers und zwischen den Betriebssystem- und Appversionen gibt es viele Übereinstimmungen.

SpotifyCares, *SpotifyHilft* und *NetflixHelps* bilden die zweite Kontrollgruppe Mobile. Ihre Anwendungen werden zwar für mehrere Betriebssysteme angeboten, in dieser Arbeit wurde allerdings nur die relevanten mobilen Betriebssystem konfiguriert.

5.6 Ergebnisse

Insgesamt wurden 19 Twitter-Accounts von Herstellern bearbeitet, die entweder vielversprechende Namen (-hilft, -support, -cares, -helps) enthielten oder populäre Software wie z.B. Browser betreiben. Bei einer weiteren Gruppe von erfassten Twitter-Accounts handelt es sich um Hersteller von IDE-Software. Insgesamt wurden 5.000.853 Tweets erfasst, aus denen 2.335.564 Unterhaltungen erstellt werden konnten (siehe Tabelle 5.3). Während der Ermittlung verknüpfter Tweets ist es vorgekommen, dass Tweets nicht gelesen werden konnten, weil diese entweder gelöscht waren oder die Privatsphäreneinstellungen eine Abfrage untersagten. Insgesamt konnten 488.503 Tweet-IDs nicht abgefragt werden. Die gesammelten Daten wurden in ein Truthset mit 2.443.058 Tweets normalisiert. Für die Normalisierung wurden leere Tweets und Tweets der Hersteller (inklusiver zugehöriger Accounts, wie z.B. @*Spotify*) entfernt. Die Normalisierung der Tweet-Texte erfolgte so, dass Sonderzeichen, Twitter-Erwähnungen (@*NetflixDE*), Hashtags (*#Hilfe*), Leerzeichen, Absätze entfernt und sämtliche Großbuchstaben in Kleinbuchstaben umgewandelt

wurden. Aus dem Truthset konnten anschließend 227.210 Tweets mit Context Items extrahiert werden.

Die Ermittlung der Daten erfolgte über den implementierten Crawler. Das große Tweetaufkommen der Twitter-Accounts *SpotifyCares* und *NetflixHelps* kommt aufgrund einer Verarbeitung eines Data-Sets aus der Arbeit von Maleej und Martens [21] zustande. Hier wurden die Tweet-IDs der gesammelten Daten importiert und durch den Crawler angereichert.

Twitter-Account	Tweets	Unterhaltungen	Truthset	Tweets-CI	Tweet-UR
@DisneyPlusHelp	75.767	27.725	43.217	6.098	1.672
@NetflixHelps	1.633.486	796.394	836.509	76.573	161.877
@ParallelsCares	42.300	12.632	20.777	4.819	2.418
@SCsupport	62.573	24.605	30.120	1.757	4.046
@SlackHQ	109.521	36.840	58.300	6.126	5.450
@Snapchatsupport	167.770	69.422	80.305	3.853	21.872
@SpotifyCares	2.475.674	1.217.617	1.142.781	108.125	264.156
@SpotifyHilft	42.860	15.214	21.242	2.613	3.304
@TwitchSupport	49.469	15.174	27.686	974	2.668
@TwitterSupport	36.588	15.803	19.709	628	5.943
@Zoom_us	37.778	13.818	22.390	1.002	1.075
@Firefox	56.577	20.344	31.302	2.646	3.315
@GoogleChrome	54.084	15.650	23.228	1.120	2.015
@MicrosoftEdge	33.787	15.155	21.375	2.126	3.521
@Opera	36.606	12.469	19.313	2.491	2.544
@IntellisSupport	3.457	1.124	1.724	256	41
@Pycharm	11.126	3.750	6.325	652	354
@VisualStudio	47.382	15.328	23.737	4.237	1.551
@WebStormIDE	24.048	6.473	13.018	1.114	681
Gesamt: 19	5.000.853	2.335.564	2.443.058	227.210	488.503

Tabelle 5.3: Gesamtübersicht gesammelter Daten

5.6.1 Kontrollgruppe IDE

Während der Annotation wurden insgesamt 1.774 Context Items mit einem Label versehen. Diese Label beinhalten auch Mehrfachnennungen pro Tweet, so dass die Nennung pro Tweet gruppiert wurden. Die bereinigte Menge beläuft sich auf 1.579 Label (siehe Tabelle 5.4). Das am stärksten vertretene Context Item, neben dem Betriebssystem, ist das der Appversion. Für das Betriebssystem wird insgesamt eine Precision von 79 % und ein Recall von 99,5 % erreicht. Es wurden 803 der enthaltenen Context Items erkannt. Für die Appversion wird eine Precision von 68,5 % und ein Recall von 87,8 % erreicht. Hier wurden 267 der enthaltenen Context Items richtig erkannt. Die Precision in der Gerätebezeichnung ist mit 34 %, abgesehen vom Fehlercode, der niedrigste Wert. Hier stehen 166 erkannte, 317 falsch erkannte Context Items gegenüber. Die Performance der einzelnen Context Items sind in der Tabelle 5.5 aufgelistet.

Kontrollgruppe	CI	Annotiert	Bereinigt	Extrahiert	Diff
Kontrollgruppe IDE	Betriebssystem	939	841	803	-38
	BS-Version	195	134	105	-29
	Appversion	442	413	267	-146
	Gerätebezeichnung	195	188	166	-22
	Errorcode	3	3	2	-1
	Gesamt	1.774	1.579	1.343	-239

Tabelle 5.4: Unterschied Annotation und Extraktion Kontrollgruppe IDE

CI	Account	CI	TP	TN	FP	FN	Prec	Rec	Acc
	IntellijSupport	70	70	106	16	0	0.814	1.000	0.917
	Pycharm	201	200	765	39	1	0.837	0.995	0.960
Betriebssystem	VisualStudio	366	363	515	121	3	0.750	0.992	0.876
	WebStormIDE	170	170	797	38	0	0.817	1.000	0.962
	Gesamt	807	803	2183	214	4	0.790	0.995	0.932
	IntellijSupport	7	7	183	2	0	0.778	1.000	0.990
	Pycharm	33	28	968	4	5	0.875	0.848	0.991
BS-Version	VisualStudio	20	19	976	6	1	0.760	0.950	0.993
	WebStormIDE	53	51	936	16	2	0.761	0.962	0.982
	Gesamt	113	105	3063	28	8	0.789	0.929	0.989
	IntellijSupport	17	17	173	2	0	0.895	1.000	0.990
	Pycharm	84	79	916	4	5	0.940	0.940	0.990
Appersion	VisualStudio	102	98	796	104	4	0.485	0.961	0.892
	WebStormIDE	101	73	892	12	28	0.859	0.723	0.960
	Gesamt	304	267	2777	123	37	0.685	0.878	0.950
	IntellijSupport	18	17	158	16	1	0.515	0.944	0.911
	Pycharm	35	34	924	46	1	0.425	0.971	0.953
Gerätebez.	VisualStudio	59	59	747	196	0	0.231	1.000	0.804
	WebStormIDE	57	56	889	59	1	0.487	0.982	0.940
	Gesamt	169	166	2396	317	3	0.344	0.982	0.900
	IntellijSupport	2	1	189	1	1	0.500	0.500	0.990
	Pycharm	0	0	1003	2	0	n/a	1.000	0.998
Fehlercode	VisualStudio	1	1	1000	1	0	0.500	1.000	0.999
	WebStormIDE	0	1	1004	1	0	n/a	1.000	0.999
	Gesamt	3	3	3196	5	1	0.286	0.667	0.998

Tabelle 5.5: Extractorperformance Kontrollgruppe IDE

5.6.2 Kontrollgruppe Mobile

In der Kontrollgruppe Mobile wurden aus 3.018 Tweets im Annotations-Set 2.759 Context Items mit Labeln versehen (siehe Tabelle 5.6). Das am stärksten vertretene Context Items, neben dem Betriebssystem, ist die Gerätebezeichnung. Für das Betriebssystem wird insgesamt eine Precision von 98,3 % und ein Recall von 97,8 % erreicht. Es wurden 1.393 der enthaltenen Context Items richtig erkannt. Für die Gerätebezeichnung wird eine Precision von 80,9 % und ein Recall von 94,3 % erreicht. Hier wurden 480 der enthaltenen Context Items richtig erkannt. Die Precision der Appversion ist bei NetflixHelps im Verhätnis mit 46,7 % zwar viel geringer, als in den beiden anderen Serviceprovider, hat aber aufgrund der geringen Menge der Context Items keinen großen Einfluss auf die

Gesamtperformance in dieser Kategorie (vgl. Tabelle 5.7). Die Erkennung des Betriebssystem bei dem Account von *NetflixHelps* weist einen im Verhältnis zu den anderen Accounts einen viel höheren Wert der *FP* auf (87,5 % der gesamten *FP*).

Kontrollgruppe	CI	Annotiert	Bereinigt	Extrahiert	Diff
Kontrollgruppe Mobile	Betriebssystem	1.452	1.386	1.363	-23
	BS-Version	439	436	369	-67
	Appversion	180	160	149	-31
	Gerätebezeichnung	662	656	480	-176
	Errorcode	26	26	12	-14
	Gesamt	2.759	2.684	2.373	-311

Tabelle 5.6: Unterschied Annotation und Extraktion Kontrollgruppe Mobile

CI	Account	CI	TP	TN	FP	FN	Prec	Rec	Acc
Betriebssystem	SpotifyCares	589	575	415	0	14	1.000	0.976	0.986
	SpotifyHilft	432	421	576	2	11	0.995	0.975	0.987
	NetflixHelps	373	368	610	21	5	0.946	0.987	0.974
	Gesamt	1393	1363	1601	24	30	0.983	0.978	0.982
BS-Version	SpotifyCares	172	169	805	27	3	0.862	0.983	0.970
	SpotifyHilft	126	119	815	69	7	0.633	0.944	0.925
	NetflixHelps	83	81	899	22	2	0.786	0.976	0.976
	Gesamt	380	374	2517	121	6	0.758	0.969	0.957
Appersion	SpotifyCares	82	75	916	6	7	0.926	0.915	0.987
	SpotifyHilft	76	67	925	9	9	0.882	0.882	0.982
	NetflixHelps	8	7	988	8	1	0.467	0.875	0.991
	Gesamt	166	149	2829	23	17	0.866	0.898	0.987
Gerätebez.	SpotifyCares	193	182	787	24	11	0.883	0.943	0.965
	SpotifyHilft	162	154	811	37	8	0.806	0.951	0.955
	NetflixHelps	154	144	798	52	10	0.735	0.935	0.938
	Gesamt	509	480	2396	113	29	0.809	0.943	0.953
Fehlercode	SpotifyCares	1	0	1002	1	1	n/a	n/a.	0.998
	SpotifyHilft	1	1	1009	0	0	1.000	1.000	1.000
	NetflixHelps	22	11	981	1	11	0.917	0.500	0.988
	Gesamt	24	12	2992	2	12	0.857	0.500	0.995

Tabelle 5.7: Extractorperformance Kontrollgruppe Mobile

CI	Account	CI	TP	TN	FP	FN	Prec	Rec
Betriebssystem	NetflixHelps	311	303	701	0	8	n/a	0.97
	Snapchatsupport	416	403	615	0	13	n/a	0.97
	SpotifyCares	204	204	801	0	0	n/a	1.00
	Gesamt	931	910	2,117	0	21	n/a	0.98
BS-Version	NetflixHelps	56	47	948	1	8	0.98	0.85
	Snapchatsupport	130	116	876	0	14	1.00	0.89
	SpotifyCares	109	88	898	2	19	0.98	0.82
	Gesamt	295	251	2.722	3	41	0.99	0.86
Appersion	NetflixHelps	11	7	994	3	1	0.70	0.88
	Snapchatsupport	26	17	977	9	0	0.65	1.00
	SpotifyCares	89	74	918	11	4	0.87	0.95
	Gesamt	126	98	2.889	23	5	0.81	0.95
Gerätebez.	NetflixHelps	168	140	845	8	20	0.95	0.88
	Snapchatsupport	164	130	840	12	22	0.92	0.86
	SpotifyCares	156	116	859	18	22	0.87	0.84
	Gesamt	488	386	2.544	38	64	0.91	0.86

Tabelle 5.8: Performance Maleej und Martens [21]

Im Vergleich zu der Performance des Versuches von Maleej und Martens [21] (vgl. Tabelle 5.8) in Hinblick auf die Gesamtperformance der enthaltenen Twitter-Accounts, sind die Ergebnisse sehr ähnlich. Sie unterscheiden sich hauptsächlich in der Erkennung der Betriebssystemversion und Gerätebezeichnung (siehe Tabelle 5.9). Für die Angabe des Wertes *n/a* in der Precision des Betriebssystems wurde für den Vergleich ein Wert von 1.00 angenommen. (Die Autoren werteten die Precision der Betriebssystemversion für den Account *SnapchatSupports* in der Tabelle genau so, bei gleichen Konfiguration.)

CI	Prec	Rec	Prec-MM	Rec-MM	Diff. Prec	Diff. Rec
Betriebssystem	0.98	0.98	n/a	0.98	- 0.02	0.00
BS-Version	0.76	0.97	0.99	0.86	- 0.23	+ 0.13
Appversion	0.87	0.90	0.81	0.95	+ 0.06	- 0.05
Gerätebezeichnung	0.81	0.94	0.91	0.86	- 0.10	+ 0.08

Tabelle 5.9: Vergleich Performance Mobile & Maleej und Marten [21]

5.7 Interpretation

Ungefähr die Hälfte der gesammelten Tweets konnten in Truthsets überführt werden, die Kommunikation wird als zu fast gleichen Teilen von Benutzer und Anbieter verursacht.

Aus den Truthsets konnten anschließend in 9 % der Fälle Context Items extrahiert werden, 10 % der Tweets waren unlesbar. Die von dem Crawler ermittelten Daten bestehen aus für den Scraper erreichbaren, in dem Replication Package von Maleej und Martens [21] enthaltenen und der kostenlosen Standard-API von Twitter zugänglichen Tweets. Ob es sich bei weiteren oder vollständigen Daten anders verteilen würde, ist nicht zu sagen. Die Ergebnisse der einzelnen Serviceprovidern sind ebenfalls unterschiedlich, diese lassen nicht eindeutig auf ein unterschiedliches Verhalten oder eine besonders hohe Antwortrate schließen.

Die **Aufbereitung** der Daten für die *Pre-Defined Keyword Lists* in dieser Arbeit war abhängig von der Zugänglichkeit und der Menge der Daten für die einzelnen Context Items.

Die Versionsrecherche war von öffentlich zugänglichen Daten abhängig und ist nicht vollständig. Ein Softwarehersteller sollte eine komplette Liste der möglichen Appversionen besitzen.

Die Anzahl der verfügbaren Stichworte für eine *Pre-Defined Keyword* List kann ebenfalls eine Herausforderung sein. Bei 30.431 unterstützen Android-Geräten ist die manuelle Überprüfung nicht sinnvoll zu erbringen. Die Performance in der Kontrollgruppe Mobile liegt mit dem hier angewandten Ansatz bei 80,9 % in der Precision und hat einen Recall von 94,3 %. Die fehlerhaften Positiven stammen aus zu allgemeinen Bezeichnungen wie *samsung* oder *lg g4* und an sonstigen Überschneidungen wie z.B. 'warner bros', das als Name eines Filmverleihs in einem Tweet angegeben ist, aber auch ein Marketing-Name eines Android-Gerätes ist.

Wenn man die Ergebnisse der **Annotation** und der Extraktion vergleicht, schneidet der Mensch in beiden Gruppen besser ab. Was die Ergebnisse nicht zeigen ist, dass es durchaus den Fall geben kann, in dem der Extractor eine Version oder eins der 30.431 Geräte erkennt, die der Mensch nicht kennt und nicht annotiert hat. Das Annotations-Set wurde in so einem Fall korrigiert, da es als *Vergleichsliste* für die Extractor-Ergebnisse dient. Trotz allem ist zu bemerken, dass die Performance des Extractors im Allgemeinen zufriedenstellend ist. Das Context Item Fehlercode ist zu vernachlässigen, denn von 4.533 annotierten Tweets wurden nur 29 Fehlercode-Label vergeben.

Während der Annotation aller Serviceprovider wurde am häufigsten das Betriebssystem ermittelt, was nicht weiter verwunderlich ist, da die Tweets in dem Annotations-Set aufgrund der Bezeichnungen der Betriebssysteme gefiltert wurden. Wenn man hier die

Kontrollgruppe Mobile betrachtet, fällt die relativ häufigen fälschlich negativen Klassifizierungen auf. Das ist durch eine Besonderheit in der zusammengesetzten Angabe, hauptsächlich der iOS-Betriebssysteme, zu begründen. Benutzer gaben in diesen Fällen *ios14* oder seltener *android7* an. Der Algorithmus zum Versionsmatching (vgl. Algorithmus 1) trennt aus dieser Angabe die Version heraus und ignoriert das Betriebssystem. Der PhraseMatcher überprüft diese Angaben später nicht.

Zudem fällt in der Kontrollgruppe Mobile die Erkennung der Betriebssystemversion auf. Hier wurden insgesamt 121 Betriebssystemversionen als *FP* klassifiziert. Mehr als die Hälfte davon speziell im Serviceprovider *SpotifyHilft*. Hier wird besonders häufig *windows 10* angegeben. Das führt dazu, dass die enthaltene *10* als Betriebssystemsversion von Android erkannt wird. Eine Überprüfung der Überschneidung mit anderen Betriebssystemversionen ist nicht implementiert. Diese Überprüfung funktioniert in diesem Beispiel ohnehin nicht, da für die Serviceprovider in dieser Gruppe nur mobile Betriebssysteme konfiguriert sind.

Der zuvor erwähnte hohe Wert der *FP* bei *NetflixHelps* ist in der Erkennung des Gerätes 'android tv' begründet. Zwischen erkannten Geräte und Betriebssystemen findet keine Prüfung auf Überschneidung statt, wie bei der Überprüfung der Geräte und Betriebssystem-/Appversion (vgl. Algorithmus 3).

Ein weiterer Grund für die fehlerhaften Treffer, ist die nicht ausreichend gefüllte Liste der Betriebssystem- und Appversionen. Die Angabe einer Version in einem Tweet als *Version 6.3.0.882* führt dazu, dass diese zur Überprüfung in die spezifischen Sets der möglichen Versionen hinzugefügt wird. Der Algorithmus zur Konfliktbeseitigung (vgl. Algorithmus 4) findet keine Übereinstimmung in irgendeiner Version und bereinigt kein Set. Das führt zu vier *FP* in der Betriebssystemversion und vier *TP* in der Appversion. Das Ergebnis könnte mit der Pflege von weiteren Versionen noch einmal stark verbessert werden. Es gibt aber auch weitere Überschneidungen, die nicht immer gefiltert werden können. Die Versionen des *Dolby Surround Sound 5.1* und *Stereo 2.1* überschneiden sich leider mit vorhandenen Appversionen. Die Filterung mit einer Ausschlussliste funktioniert aufgrund der *Windowsize* von 1 nicht, falls es sich um die bspw. *Stereo Version 2.1* handelt.

Im Hinblick auf die Ergebnisse der Kontrollgruppe Mobile und denen der Performance von Maleej und Martens (siehe Tabelle 5.9) sind die Ergebnisse vergleichbar und zeigen nur geringe Abweichungen auf. Die Annoations-Sets sind unterschiedlich, die Serviceprovider bis auf eine Ausnahme identisch. Im Fall der Ausnahme handelt es sich mit *SnapchatSupport* ebenfalls um einen Anbieter von Software für mobile Endgeräte. Der

Unterschied im Betriebssystem ist auf die bereits erwähnte Ausnahme des Begriffes 'android tv' zurückzuführen. Die Erkennung der Betriebssystemversion ist unpräziser, aber im Recall etwas stärker als die der Autoren Maleej und Martens. Durch die großzügige Konfliktbeseitigung der Versionen und der nicht ausreichend gepflegten *Pre-Defined Keyword List* der Versionen erkennt der hier entwickelte Ansatz häufiger Versionsaufkommen (richtig oder falsch). Dadurch ist die Zahl der *FP* höher und die der *FN* niedriger, was sich in der Ausprägung der Precision und im Recall wiederfindet. In der Gerätebezeichnung findet sich ein ähnliches Bild. Vermutlich erkennt der hier gezeigte Ansatz ebenfalls mehr Geräte (richtig oder falsch) und ist deswegen wieder unpräziser, aber stärker im Recall. Die Anzahl der Gerätebezeichnung für *iOS*- und *Android*-Geräte in der *Pre-Defined Keyword Lists with alternative Spelling* ist mit 93.895 Einträgen viel höher als die der beiden Autoren. Ihre Grundlage zum Training der Wortvektoren betrug 56 *iOS*-Geräte und 896 aus ursprünglich 23.387 *Android*-Geräten. Nach manueller Überprüfung der Ergebnisse dieser Ermittlung wurden 22 Bezeichnungen für *iOS*- und 66 für *Android*-Geräte hinzugefügt.

In der Kontrollgruppe IDE werden die Gerätebezeichnungen und Betriebssysteme besonders schlecht erkannt. Eine Precision in der Gerätebezeichnung von 34,4 % und 214 *FP* in der Erkennung des Betriebssystems. Das hat insbesondere mit zwei Begriffen zu tun, die schlecht geeignet sind in dem Szenario Betriebssysteme und Geräte zu erkennen. Der Begriff *windows* bezeichnet einmal das Betriebssystem, ist aber im Rahmen einer IDE auch häufig im Sinne des Bearbeitungsfenster gemeint. Zusätzlich kommt es vor, dass in Tweets von *windows* im Sinne der Bearbeitungsfenster gesprochen wird:
'i'm using visual studio mac. how can i create 'windows form application' on that?'
Alleine beim Serviceprovider *Visualstudio* ist das 82 Mal der Fall.

Ähnlich negativ wirkt sich der Begriff *mac* auf die Erkennung aus. Er deutet auf ein Gerät und gleichzeitig auf ein Betriebssystem hin. In den Tweets war von dem Gerät *'on my Mac'*, oder aber auf das Betriebssystem *'on mac'* zu lesen. Dem Betriebssystem macOS ist der Begriff *mac* als Stichwort dem Betriebssystem, aber auch der verbundenen Geräte zugeordnet.

Die falsch erkannten Appversionen ergeben sich hauptsächlich aus zu allgemeinen Versionsbezeichnungen. Die typische Bezeichnung der Produkte aller Hersteller besteht aus mehreren numerischen Werten, getrennt durch einen Punkt. Bei den Produkten *IntelliJ, Pycharm und Webstorm* bspw. *2020.3.2*. Gaben die Nutzer hier an, Probleme mit der *neuen* Version 2020 oder generell in *Webstorm 11* zu haben, hat der Extractor diese

Nennung ignoriert. Für die Plausibilitätsprüfung von numerischen Versionswerten wurden nur Werte übernommen, bei denen die ersten beiden Werte, getrennt durch einen Punkt, übereinstimmten. In der Abbildung 5.1 ist die Schwierigkeit mit einem einzelnen numerischen Wert zu erkennen. Es ist nicht eindeutig, ob der Nutzer von der Version 2019 oder von dem Jahr 2019 schreibt. Der Extractor erkennt die Version *2020.1*, ignoriert aber 2019, da es nicht auszumachen ist.

Abbildung 5.1: Twitter-Statusmeldung Probleme bei der Versionserkennung [47]

Ein Problem bei mehrdeutigen Context Items zeigt das Beispiel des erkannten Gerätes *Notebook* bei einem Tweet, welcher von einem *Junyper Notebook* handelt. Es hat nichts mit einem echten Gerät zu tun, wird aber vom Extractor als solches erkannt.

6 Fazit und Ausblick

Zum Abschluss wird in diesem Kapitel ein Fazit das erstellte System und die Ergebnisse gezogen und schließt mit einem Ausblick auf mögliche Verbesserungen ab. Das System zur Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen wurde erstellt und konnte in artverwandten Kontrollgruppen die Ergebnisse der Autoren Maleej und Marten aus ihrem Konferenzpapier *Extracting and Analyzing Context Information in User-Support Conversationson Twitter* bestätigen. Die untersuchte Erweiterbarkeit des Ansatzes in einer Kontrollgruppe für Hersteller von IDE-Software kommt zu keinem eindeutigen Ergebnis. Während die Ermittlung von Betriebssystem- und Appversion ähnlich gut funktioniert, offenbart die Erweiterung dennoch Schwächen durch Mehrdeutigkeiten und Überschneidungen in einzelnen Context Items. Die Mehrdeutigkeit einzelner Wörter und Überschneidungen von Context Items führte in der Konsequenz zu sehr vielen FP in der Erkennung des Betriebssystems und in der Gerätebezeichnung. Zur Konfliktbeseitigung dieser Hindernisse besteht die Notwendigkeit weitere Verbesserungen und Techniken, die zukünftig erarbeitet und geprüft werden müssen.

6.1 Fazit

Die Implementierung des **Crawlers** ermöglicht es, die Kommunikation eines oder mehrerer Serviceprovider zu erfassen und zu analysieren. Die hier entwickelte Struktur lässt es zudem zu, weitere Attribute hinzuzufügen. Grundsätzlich benötigt dieser Crawler auch nur eine Liste von IDs, die dieser abarbeiten kann. Es wurde gezeigt, dass verschiedene Quellen und Technologien simpel zu verwenden sind. Die Abfragelimits der Twitter-API konnte auch aufgrund der *Tweepy*-Attribute *wait_on_rate_limit* eingehalten werden. Eine Grenze bildet allerdings die hier verwendeten Speicherung der Daten in JSON-Dateien. 2.4 Millionen Tweets eines Serviceproviders in verschiedenen Dictionaries zur Laufzeit zu halten, war eine Herausforderung für die eingesetzte Maschine.

Die **Annotation** der beiden Kontrollgruppen hat den Ausschlag in der Entwicklung des Extractors gegeben. Durch eine automatisierte Auswertung konnte einfach festgestellt werden, welche Regel gut oder nicht gut funktioniert und welche überflüssig ist. Die Auswahl der Tweets für die Annotationssets war gut gewählt, da sie auch Tweets enthielten, die keine Context Items beinhalteten. Es ist zu vermuten, dass eine Prüfung gegen ausschließlich mit Context Items gefüllten Tweets, nicht ausreichend ist, da nicht sichergestellt ist dass dieser Ansatz TN ebenfalls erkennt.

Die Annotation alleine durchzuführen machte es notwendig, das Annotations-Set mehrfach zu durchlaufen, was sehr zeitintensiv war (Für 6.000 Tweets benötigte man ca. 8 Stunden). Beim Performancetest war es dann immer wieder nötig, das Set in Teilen zu korrigieren. *doccano* verwaltet die Daten strukturiert und so es war möglich in diesen Fällen, in denen bspw. ein Label falsch zugeordnet war, einfach nach der Tweet-ID zu suchen und diese dann direkt zu korrigieren. Ein weiterer manueller Aufwand war die Ermittlung der verschiedenen Appversionen für die *Pre-Defined Keyword Lists*. Pro Hersteller eine Liste für ein oder mehrere Betriebssysteme zu ermitteln war aufwendig und nur durch die aufbereiteten Daten des Anbieters *AppAnnie* möglich. Da die meistens Serviceprovider auch regelmäßig ihre Apps aktualisieren, war der händische Aufwand immerhin soweit begrenzt, als das auch der Zeitraum der Erhebung begrenzt war. Die Automatisierung der Ermittlung hätte diesen händischen Aufwand sicherlich überstiegen.

Das hier entwickelte System zeigt, dass die technische Extraktion von Context Items möglich ist. Für unvermeidbaren Missverständnisse, die in einem Konstrukt aus mehreren Context Items auftreten können, wird dennoch ein menschlicher Bearbeiter benötigt. Die relativ freie Form von Tweets können einen versteckten oder tieferen Sinn enthalten, der nicht durch eine Regel erschlossen oder für dessen Erfassung widersprüchliche Regeln erforderlich wäre. Grundsätzlich können in Tweets Context Items enthalten sein, die sich nicht auf ein konkretes Problem beziehen, sondern einfach nur Lob und Tadel aussprechen.

Die **Extraktion** der Context Items aus den Tweets, in Kombination mit möglichen Context Items anhand der Beschaffenheit der Wörter, sowie die Suche nach festen Begriffen aus den *Pre-Defined Keyword Lists* funktioniert im Allgemeinen bereits gut. Eine vollständige Lösung ist bei dem diversen Feedback und seiner unterschiedlichen Form eher undenkbar. So greifen Benutzer auch mal auf Screenshots oder Videosequenzen zurück, um Probleme zu schildern. Die größten Hindernisse sind, soweit es im Rahmen der Evaluation (vgl. Kapitel 5) dieser Arbeit festzustellen war, lauteten:

1. die Muster einzelner Wörter
2. die Unvollständigkeit von *Pre-Defined Keyword Lists*
3. die doppelte Erkennung einzelner Context Items
4. die Überschneidungen zwischen einzelnen Context Items
5. die Mehrdeutigkeit einzelner Wörter

Während der Justierung der Regeln und Filter läuft man Gefahr, in ein Overfitting bzw. Underfitting zu geraten. Sind die Regeln zu spezifisch, gelten diese nur für die betrachteten Context Items (Overfitting). Sind diese Regeln zu allgemein verfasst, werden Daten übernommen, die nicht den gesuchten Context Items ähneln, ihnen aber nicht entsprechen (Underfitting). Die Extrahierung mit den PhraseMatchern sind so spezifisch, dass diese nur enthaltene Begriffe ausfindig machen. Das Versionsmatching mit numerischen Werten zieht auch teilweise übereinstimmende Werte mit in Betracht.

6.2 Ausblick

Das hier entwickelte System bestätigt grundsätzlich die Ergebnisse von Maleej und Martens [21]. Die Erweiterung des Ansatzes zeigt allerdings auch eine grundlegende Schwäche in der Erkennung und Interpretation von einzelnen Wörtern auf, wenn diese mehrdeutig sind. Hier bedarf es weiterer Konfigurationen und Techniken zur Verbesserung des Ergebnisses.

Eine mögliche Verbesserung für den **Crawler** ist, die Daten in einer Datenbankstruktur zu speichern und mit einem Datenbanken-Managementsystem zu verwalten. Um Tweets als Unterhaltungen durch Abfragen oder in Sichten darzustellen, Vergleiche zwischen Serviceprovidern mit den Tools der entsprechenden Datenbanktechnologie zu erstellen und auch mit Hinblick auf eine Parallelisierung der Crawling-Aktivitäten, bieten Datenbanken-Managementsysteme eine Erleichterung an.

Während der **Annotation** ist vermehrt aufgefallen, dass Benutzer sich auf das *letzte* oder *aktuellste* Update beziehen. Eine Analyse von einigen diesbezüglichen Stichworten *update, recent, acutal, updating, letztes* und *aktuellsten* auf ihr Vorkommen in den annotierten Tweets ergab, dass in 201 Fällen eines dieser Wörter fiel, ohne das eine explizite Versionsangabe erfolgte. Es bietet sich an für diese Stichworte in Kombination mit dem

Tweet-Datum die jeweilige aktuelle Version der App oder des Betriebssystems als Vorschlag hinzuzufügen.

Zur besseren Verarbeitung der fünf bereits erwähnten Hindernisse in der **Extraktion** gibt es unterschiedliche Ansätze: Für das erste Hindernis ist es denkbar, Filtern und Verarbeitung einzelner Wörter nach und nach soweit anzupassen und zu testen, so dass die Performance sich stetig verbessert. Das zweite Hindernis, die Unvollständigkeit der *Pre-Defined Keyword List*, ist etwas, das als Hersteller bezüglich der eigenen Appversionen viel einfacher zu lösen ist oder mit weiterer Recherche ebenfalls verbessert werden kann. Zur Verbesserung beider Hindernisse kann z.B. für ein annotiertes Tweetset in einem Kreislauf Regeln zur Verarbeitung einzelner Wörter und die Vervollständigung der *Pre-Defined Keyword List* stattfinden. Regeln und Stichwörter werden angepasst, angewendet, Ergebnisse ausgewertet, Regeln und Stichwörter erneut angepasst, usw.

Das dritte Hindernis ist durch eine Erweiterung der Abhängigkeiten zwischen den Context Items zu erreichen. Die Erkennung eines Gerätes ist gegen die doppelte Erkennung eines Versions-Context Item gesichert, nicht aber gegen die Erkennung eines Betriebssystems. Es wäre zu untersuchen, ob ein Ausschluss wie in dem Algorithmus 3 für diesen Fall oder eine Propagierung von erkannten Context Items auf weitere Context Items (*Android*-Geräte unterstützen nur das Betriebssystem *Android*) das Ergebnis verbessert. Beides ist denkbar und muss anhand der vorhandenen Context Items überprüft werden.

Bei dem vierten Hindernis, den Überschneidungen zwischen einzelner Context Items, wird es schwieriger. Hier birgt die Struktur der einzelnen *Pre-Defined Keyword Lists* und die Implementierung teilweise die Möglichkeit, aufgrund von bereits ermittelten Context Items, mögliche Werte schon vorher auszuschließen. Vermutlich wird es aber dabei bleiben, dass Überschneidungen stets zum Teil zu falschen Klassifikationen führt.

An Grenzen stößt das System, bei dem fünften Hindernis, den einzelnen Wörtern, die mehrfach interpretiert werden können. Begriffe, die gleichzeitig in verschiedenen Context Items gültig sind, müssen mit einer fortführenden Konfliktbeseitigung-Technik geprüft werden, um herauszufinden um welchen Fall es sich handelt. Aufgetreten ist in dieser Arbeit der Fall, dass es sich entweder um das Betriebssystem Windows oder das Bearbeitungsfenster handelte und ob es sich um Problem mit dem Mac als Gerät oder auf dem Mac als Betriebssystem handelt. Bei wachsender Anzahl von Context Items oder anders zusammengestellter Hersteller, ist es sehr wahrscheinlich, dass weitere dieser Konflikte auftreten.

Entscheidend ist, in welchem Szenario eine automatisierte Erhebung erfolgen soll. Die erhobenen Daten können dazu verwendet werden, Tweetaufkommen an Tagen von veröffentlichten Updates zu analysieren, um schneller auf Probleme mit einem Stand zu reagieren. Erhobene Context Items können auch in einem Support-Szenario dazu dienen, Tweets vorab zu kategorisieren. Ein Bearbeiter könnte sortierte Supportfälle anschließend entgegen nehmen und Zugriff auf alle verfügbaren Daten erhalten. Eine Rückmeldung des Benutzers, ob die Context Items richtig erkannt wurden, kann wiederum dazu verwendet werden die das Ergebnis der Extraktion zu verbessern. Bei einem stetig erhöhten Tweetaufkommen würde dann die Erhebung dafür sorgen, dass anfallende Datenmenge für den Benutzer zur einfachen Weiterverarbeitung aufbereitet wird.

Zusammenfassend ist zu sagen, dass eine automatische Erhebung von relevanten Kontextinformationen aus zuvor gesammelten Twitterunterhaltungen praktisch möglich ist, sich aber mit steigender Anzahl der gewählten Context Items in seiner Komplexität erhöht. Das hier entwickelte System mit dem Fokus auf Support-Accounts kann die beschriebenen Hindernisse, insbesondere der Hindernisse vier und fünf (Überschneidung einzelner Context Items und möglichen Mehrfachinterpretationen einzelner Wörter), nicht vollständig lösen und benötigt weiterhin einen Bearbeiter. Um diese Hindernisse aufzulösen müssen zusätzlich zu dem hier entwickelten System weitere Techniken der *NLP* zur Kontexterkennung untersucht werden.

Literaturverzeichnis

[1] : *IntelliJ Versions*. – URL jetbrains.com/idea/download/other.html. – Zugriffsdatum: 2020-09-10

[2] : *Pycharm Versions*. – URL https://www.jetbrains.com/de-de/pycharm/download/other.html. – Zugriffsdatum: 2020-09-10

[3] : *VisualStudio Versions*. – URL https://docs.microsoft.com/de-de/visualstudio/install/visual-studio-build-numbers-and-release-dates. – Zugriffsdatum: 2020-09-10

[4] : *WebStorm Versions*. – URL https://www.jetbrains.com/webstorm/download/other.html. – Zugriffsdatum: 2020-09-10

[5] Annie, App: *App Annie -Data platform*

[6] Arifanto, R. ; Asnar, Y. D. W. ; Liem, M. M. I.: Domain Specific Language for Web Scraper Development. In: *2018 5th International Conference on Data and Software Engineering (ICoDSE)*, 2018, S. 1–6

[7] Bal, S. K. ; Geetha, G.: Smart distributed web crawler. In: *2016 International Conference on Information Communication and Embedded Systems (ICICES)*, 2016, S. 1–5

[8] Bojanowski, Piotr ; Grave, Edouard ; Joulin, Armand ; Mikolov, Tomas: Enriching Word Vectors with Subword Information. In: *arXiv preprint arXiv:1607.04606* (2016)

[9] Crockford, Douglas: *Einführung in JSON*. – URL https://www.json.org/json-de.html. – Zugriffsdatum: 2020-07-02

[10] GlobalStats, Statcounter: *Operating-System Marketshare Desktop*. – URL https://gs.statcounter.com/os-market-share/desktop/worldwide. – Zugriffsdatum: 2020-11-20

[11] GlobalStats, Statcounter: *Operating-System Marketshare Mobile.* – URL https://gs.statcounter.com/os-market-share/mobile/worldwide. – Zugriffsdatum: 2020-11-20

[12] Google: *Classifcation: True vs. False and Positive vs. Negative.* – URL https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative. – Zugriffsdatum: 2020-11-17

[13] Google: *Classification: Accuracy.* – URL https://developers.google.com/machine-learning/crash-course/classification/accuracy. – Zugriffsdatum: 2020-11-17

[14] Google: *Classification: Precision and Recall.* – URL https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall. – Zugriffsdatum: 2020-11-17

[15] Gupta, A. ; Anand, P.: Focused web crawlers and its approaches. In: *2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE)*, 2015, S. 619–622

[16] Honnibal, Matthew ; Montani, Ines: *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.* 2017. – To appear

[17] Johnson, J. N. ; Dubois, P. F.: Issue tracking. In: *Computing in Science Engineering* 5 (2003), Nr. 6, S. 71–77

[18] Klawonn, Thilo: *Grenzen des "Web Scrapings".* – URL https://www.forschung-und-lehre.de/recht/grenzen-des-web-scrapings-2421/. – Zugriffsdatum: 2020-08-10

[19] Koster, Martijn: *The Web Robots Pages.* – URL http://www.robotstxt.org/. – Zugriffsdatum: 2020-08-09

[20] Mahto, D. K. ; Singh, L.: A dive into Web Scraper world. In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, S. 689–693

[21] Martens, Daniel ; Maalej, Walid: Extracting and Analyzing Context Information in User-Support Conversations on Twitter. In: *2019 IEEE 27th International Requirements Engineering Conference (RE)* (2019), Sep. – URL https://ieeexplore.ieee.org/document/8920599. ISBN 978-1-7281-3912-8

[22] Mezouar, Mariam E.: Are tweets useful in the bug fixing process? An empirical study on Firefox and Chrome. In: *Empirical software engineering : an international journal* 23 (2018), Nr. 3, S. 1704. – URL http://dx.doi.org/10.1007/s10664-017-9559-4

[23] N, R. ; C, D. Y. R. ; K, D. J. S.: Crawler Framework for Category Search Engine. In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, 2020, S. 1–5

[24] Nakashe, S. M. ; Kolhe, K. R.: Smart Approach to Crawl Web Interfaces Using a Two Stage Framework of Crawler. In: *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, 2018, S. 1–6

[25] Nakayama, Hiroki ; Kubo, Takahiro ; Kamura, Junya ; Taniguchi, Yasufumi ; Liang, Xu: *doccano: Text Annotation Tool for Human*. 2018. – URL https://github.com/doccano/doccano. – Software available from https://github.com/doccano/doccano

[26] Partalidou, E. ; Spyromitros-Xioufis, E. ; Doropoulos, S. ; Vologiannidis, S. ; Diamantaras, K. I.: Design and implementation of an open source Greek POS Tagger and Entity Recognizer using spaCy. In: *2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 2019, S. 337–341

[27] Pfaffenberger, Fabian: *Twitter als Basis wissenschaftlicher Studien*. 01 2016

[28] Python: *JSON encoder and decoder*. – URL https://docs.python.org/3/library/json.html. – Zugriffsdatum: 2020-07-02

[29] Řehůřek, Radim ; Sojka, Petr: Software Framework for Topic Modelling with Large Corpora. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta : ELRA, Mai 2010, S. 45–50. – http://is.muni.cz/publication/884893/en

[30] Roesslein, Joshua: *Tweepy Authentication Tutorial*. – URL http://docs.tweepy.org/en/latest/auth_tutorial.html#auth-tutorial. – Zugriffsdatum: 2020-06-15

[31] Roesslein, Joshua: *Tweepy Documentation*. – URL http://docs.tweepy.org/en/latest/index.html. – Zugriffsdatum: 2020-06-15

[32] Roesslein, Joshua: *Tweepy Documentation*. – URL http://docs.tweepy.org/en/latest/getting_started.html?highlight=ModelsReference#models. – Zugriffsdatum: 2020-06-15

[33] Roesslein, Joshua: *Tweepy Twitter API wrapper*. – URL http://docs.tweepy.org/en/latest/api.html#tweepy-api-twitter-api-wrapper. – Zugriffsdatum: 2020-06-15

[34] Taspina, Ahmet: *Twitterscraper*. – URL https://github.com/taspinar/twitterscraper. – Zugriffsdatum: 2020-06-26

[35] Twitter: *API-Get-Usertimeline*. – URL https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline. – Zugriffsdatum: 2020-06-10

[36] Twitter: *API-Rate-Limiting*. – URL https://developer.twitter.com/en/docs/basics/rate-limiting. – Zugriffsdatum: 2020-06-10

[37] Twitter: *API-Rate-Limiting*. – URL https://developer.twitter.com/en/docs/basics/rate-limits. – Zugriffsdatum: 2020-06-10

[38] Twitter: *API-Reference-Index*. – URL https://developer.twitter.com/en/docs/api-reference-index. – Zugriffsdatum: 2020-06-10

[39] Twitter: *Get started with the Twitter developer platform*. – URL https://developer.twitter.com/en/docs/basics/getting-started#get-started-app.html. – Zugriffsdatum: 2020-07-01

[40] Twitter: *Retweet objects*. – URL https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json. – Zugriffsdatum: 2020-07-01

[41] Twitter: *Tweet Counting Characters*. – URL https://developer.twitter.com/en/docs/counting-characters. – Zugriffsdatum: 2020-07-01

[42] Twitter: *Twitter Allgemeine Geschäftsbedingung*. – URL https://twitter.com/de/tos. – Zugriffsdatum: 2020-08-09

[43] Twitter: *User object.* – URL https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/user-object. – Zugriffsdatum: 2020-07-01

[44] Van Rossum, Guido ; Drake, Fred L.: *Python 3 Reference Manual.* Scotts Valley, CA : CreateSpace, 2009. – ISBN 1441412697

[45] Wang, H. ; Wang, T. ; Yin, G. ; Yang, C.: Linking Issue Tracker with Q A Sites for Knowledge Sharing across Communities. In: *IEEE Transactions on Services Computing* 11 (2018), Nr. 5, S. 782–795

[46] Wesbos: *TweetFileabugreport.* – URL https://twitter.com/wesbos/status/1266376779733721088. – Zugriffsdatum: 2020-06-15

[47] YazanAlaboudi: *TweetVersionserkennung.* – URL https://twitter.com/YazanAlaboudi/status/1253159496085626881. – Zugriffsdatum: 2020-11-25

[48] Yin, F. ; He, X. ; Liu, Z.: Research on Scrapy-Based Distributed Crawler System for Crawling Semi-structure Information at High Speed. In: *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 2018, S. 1356–1359

A Anhang

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Automatisierte Erhebung relevanter Kontextinformationen aus Twitterunterhaltungen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original