Technische Hochschule **Rosenheim**

Fakultät für Informatik

Studiengang Informatik

Social Media & Text Mining am Beispiel von Telegram

Master Thesis

von

Maximilian Bundscherer

Datum der Abgabe: 29.03.2021
Erstprüfer: Prof. Dr. Marcel Tilly
Zweitprüfer: Prof. Dr. Jochen Schmidt

Erklärung

Ich versichere, dass ich diese Arbeit selbstständig angefertigt, nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Rosenheim, den 29.03.2021

Maximilian Bundscherer

Abstract

Im Jahr 2019 ist die Infektionskrankheit Coronavirus Disease 2019 (COVID-19) ausgebrochen, die im darauffolgenden Jahr 2020 als Pandemie eingestuft wurde. Derzeit findet man sowohl in Print-Medien als auch in Online-Medien diverse Berichte über *Akteure* wie *Oliver Janich*, *Attila Hildmann*, *Eva Herman* und *Xavier Naidoo* sowie *Interessengruppen* wie *Querdenken*, wonach diese gegen Schutzmaßnahmen demonstrieren, COVID-19 verharmlosen oder dessen Existenz bestreiten, sich dabei rechtspopulistischer Aussagen bedienen, "Verschwörungstheorien" verbreiten und damit auch noch Geld verdienen. Den Berichten zufolge bauen diese Akteure insbesondere ein Netzwerk in dem Messenger *Telegram*[1] auf und benutzen dieses. Um zu validieren, ob die in den Medien häufig getätigten Aussagen korrekt sind, wurden diese *Akteure* und *Interessengruppen* auf *Telegram* ausfindig gemacht, anschließend die gefundenen Chats heruntergeladen und mit Social Media und Text Mining Methoden ausgewertet.

Die initiale explorative Vorgehensweise in der Arbeit, die in einem Data Science Prozess nicht unüblich ist, zeigt, dass es zunächst einfach ist, Social Media und Text Mining Methoden auf Daten anzuwenden, also damit die Chats aufzubereiten und zu modellieren bzw. zu visualisieren. Jedoch erfordert die vollständige Interpretation und Evaluation der Auswertungen einen umfangreicheren Blick als in dieser Arbeit gewährleistet werden kann. Daher wurde die *Telegram Analyse Plattform*[2] geschaffen, mit der es möglich sein soll, die implementierten Social Media und Text Mining Werkzeuge auf *Telegram*-Chats anzuwenden. So können Personen wie Data Scientists, die an Social Media bzw. Text Mining Methoden interessiert sind, und Personen wie Sozialwissenschaftler, die an Auswertungen und Analyse-Möglichkeiten besagter Gruppen interessiert sind, an diesem Vorhaben partizipieren.

Diese Arbeit möchte einen Überblick über die Felder Social Media und Text Mining am Beispiel von *Telegram* ermöglichen. Zunächst wird auf die Grundlagen aus diesen Feldern eingegangen. Analog zu dem generischen Data Mining Prozess *Cross Industry Standard Process for Data Mining (CRISP-DM)* beschreibt diese Arbeit die Vorgehensweise, die in den Phasen *Business Understanding* und *Data Understanding* stattgefunden hat. Die Stufen *Data Preparation* und *Modeling* wurden im Rahmen dieser Arbeit auf zwei Jupyter-Notebooks abgebildet, die ausführlich beschrieben werden.

Schlagworte: Social Media Mining, Text Mining, Telegram, COVID-19, Telegram Analyse Plattform, Social Graphs, Python, Jupyter, Data Science, PyTorch, Transformers, HanTa, Scikit, Docker

1 siehe `https://telegram.org/`
2 siehe `https://maxbundscherer.github.io/telegram-analysis/`

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Ziel der Arbeit . . . 1
- 1.3 Aufbau der Arbeit . . . 2

2 Grundlagen **3**
- 2.1 Data Science . . . 3
- 2.2 Data Mining Prozess - CRISP-DM . . . 4
- 2.3 Social Media Mining . . . 8
  - 2.3.1 Chancen . . . 8
  - 2.3.2 Herausforderungen . . . 9
- 2.4 Text Mining & Werkzeuge . . . 10
  - 2.4.1 Tokenization . . . 11
  - 2.4.2 Stopword removal . . . 11
  - 2.4.3 Bag-Of-Words (BOW) . . . 11
  - 2.4.4 N-grams/Bag-Of-N-grams . . . 13
  - 2.4.5 Lemmatization/Stemming . . . 13
  - 2.4.6 Part-of-Speech Tagging (POS-Tagging) . . . 14
  - 2.4.7 Named-entity recognition (NER) . . . 14
  - 2.4.8 Sentiment analysis . . . 15
  - 2.4.9 Topic Models . . . 15
- 2.5 Text Mining Metriken . . . 16
  - 2.5.1 Term Frequency (tf) . . . 16
  - 2.5.2 Inverse Document Frequency (idf) . . . 16
  - 2.5.3 Term Frequency & Inverse Document Frequency (tf-idf) . . . 17

3 Business & Data Understanding **19**
- 3.1 Telegram . . . 19
  - 3.1.1 Exportieren von Chats . . . 21
  - 3.1.2 Aufbau der heruntergeladenen Chats . . . 22
  - 3.1.3 Das Attribut Text im Detail . . . 24
- 3.2 Arbeitswerkzeuge/-Umgebung (Stack) . . . 24
  - 3.2.1 Komponenten aus dem Docker Image . . . 25
  - 3.2.2 Nachinstallierte Komponenten . . . 26
  - 3.2.3 Transformers . . . 28
- 3.3 Telegram Analyse Plattform . . . 29
  - 3.3.1 Ausgewertete Chats und Fragen . . . 29
  - 3.3.2 Aufbau und Einstieg . . . 31

4 Data Preparation & Modeling **33**
4.1 Arbeitsumgebung initialisieren . . . 34
4.1.1 Jupyter Notebook Parameter . . . 34
4.1.2 Arbeitsumgebung vorbereiten . . . 34
4.2 Chats laden und aufbereiten . . . 36
4.2.1 Aufbereitungsfunktionen für die deutsche Sprache . . . 37
4.2.2 Stufe 1: Chats laden . . . 39
4.2.3 Stufe 2: Chats aufbereiten . . . 39
4.2.4 Stufe 3: Nachrichten aufbereiten . . . 39
4.3 Social Media Mining . . . 43
4.3.1 Chats . . . 43
4.3.2 Social Graphs - Abbilden von Chats auf Features . . . 44
4.3.3 Social Graphs - Darstellung von Graphen . . . 49
4.3.4 Dimension Zeit . . . 53
4.4 Text Mining . . . 55
4.4.1 Verwertbarkeit von Textnachrichten . . . 55
4.4.2 Word Clouds . . . 56
4.4.3 N-grams . . . 57
4.4.4 POS-Tagging - Eigennamen . . . 58
4.4.5 Named-entity recognition (NER) . . . 59
4.4.6 Sentiment analysis . . . 60
4.4.7 Latent Dirichlet Allocation (LDA) . . . 62
4.5 Identifizierung von Autoren . . . 63
4.5.1 Chats aufbereiten . . . 64
4.5.2 Trainieren und Evaluieren . . . 64

5 Fazit und Ausblick **67**
5.0.1 Methoden . . . 68
5.0.2 Ausblick . . . 69

A Anhänge **71**
A.1 Dokumentation Telegram . . . 71
A.1.1 Autor-spezifische Attribute . . . 71
A.1.2 Nachrichten-spezifische Attribute . . . 72
A.1.3 Besondere Attribute in Textnachrichten . . . 73
A.2 Betrachtete Datensätze . . . 74
A.2.1 DataSet0 . . . 74
A.2.2 DataSet1 . . . 74
A.2.3 DataSet1a . . . 74
A.2.4 DataSet2 . . . 74
A.3 Skripte . . . 74
A.3.1 Ausführen des Telegram Notebooks mit Papermill . . . 74
A.3.2 Aufräumen des Arbeitsverzeichnisses . . . 75

Literaturverzeichnis **77**

Abbildungsverzeichnis

2.1 Darstellung der Begriffe Data Science, KI und ML als Venn-Diagramm 4
2.2 Visualisierung des CRISP-DM Prozessmodells 5
2.3 Visualisierung der Bag-Of-Words (BOW) Darstellung 12
2.4 Inverse Document Frequency (idf) 16

3.1 Darstellung der Knowledge-Pyramide 20
3.2 Exportieren von Telegram Chats 21

4.1 Übersicht der Chats im DataSet0 44
4.2 Visualisierung des Test Graphen 50
4.3 Social Graph: Top 25 Autoren (DataSet0) 52
4.4 Social Graph: Top 25 Autoren (DataSet1a) 52
4.5 Darstellung der Anzahl der Nachrichten in Chats aus DataSet0 im zeitlichen Verlauf 53
4.6 Erwähnungen von Parteien von Oliver Janich im zeitlichen Verlauf 54
4.7 Histogramm über die Anzahl der Zeichen in einer Nachricht 55
4.8 Eine Word Cloud über den gesamten Chat von Attila Hildmann 57
4.9 Top 25 der am häufigsten verwendeten Eigennamen von Oliver Janich 59
4.10 Zeitlicher Verlauf der Stimmung (DataSet0) 61
4.11 Visualisierung der Latent Dirichlet Allocation (LDA) mit PyLDAvis 62
4.12 Confusion matrix LinearSVC (DataSet0) 66

Tabellenverzeichnis

4.1 Parameter der Jupyter-Notebooks 35
4.2 CSV-Eingabeformat für zu verarbeitenden Chats 39
4.3 Features aus Stufe 3a 40
4.4 Features aus Stufe 3b 41
4.5 Features aus Stufe 3c 42
4.6 Features aus Stufe 3d 42
4.7 Formatierungs-spezifische statische Features 45
4.8 Autor-spezifische statische Features 46
4.9 Dynamische Features 47
4.10 Evaluation der trainierten Klassifikationsmodelle 65

A.1 Chats in DataSet0. 74
A.2 Chats in DataSet1a. 74
A.3 Chats in DataSet2. 75

Abkürzungsverzeichnis

COVID-19 Coronavirus Disease 2019

KI Künstliche Intelligenz

ML Maschinelles Lernen

CRISP-DM Cross Industry Standard Process for Data Mining

NLTK Natural Language Toolkit

NLP Natural Language Processing

LDA Latent Dirichlet Allocation

HanTa Hanover Tagger

POS-Tagging Part-of-speech Tagging

NER Named-entity recognition

BOW Bag-Of-Words

STTS Stuttgart-Tübingen-Tagset

1 Einleitung

Im Jahr 2019 ist die Infektionskrankheit *COVID-19* ausgebrochen, die im darauffolgenden Jahr 2020 als Pandemie eingestuft wurde.

Derzeit findet man sowohl in Print-Medien als auch in Online-Medien diverse Berichte über *Akteure* wie *Oliver Janich*, *Attila Hildmann*, *Eva Herman* und *Xavier Naidoo* sowie *Interessengruppen* wie *Querdenken*, wonach diese gegen Schutzmaßnahmen demonstrieren, *COVID-19* verharmlosen oder dessen Existenz bestreiten, sich dabei rechtspopulistischer Aussagen bedienen, "Verschwörungstheorien" verbreiten und damit auch noch Geld verdienen. Den Berichten zufolge bauen diese *Akteure* insbesondere ein Netzwerk in dem Messenger *Telegram*[1] auf und benutzen dieses. [Brü20] [Hur20] [Dür20] [Lin20] [Röb21] [Kuc20].

1.1 Motivation

Um zu validieren, ob die in den Medien häufig getätigten Aussagen korrekt sind, wurden diese *Akteure* und *Interessengruppen* auf *Telegram* ausfindig gemacht, anschließend die gefundenen Chats heruntergeladen und mit Social Media und Text Mining Methoden ausgewertet.

In dieser Arbeit wurde zunächst explorativ vorgegangen, wie es im Bereich Data Science nicht unüblich ist. Dabei wurde festgestellt, dass es zunächst einfach ist, Social Media und Text Mining Methoden auf heruntergeladene Chats anzuwenden, aber bereits die Auswahl der zu betrachtenden Chats einen umfassenderen Blick erfordert, als im Rahmen dieser Arbeit gewährleistet werden kann. Die Interpretation und Evaluation der Auswertungen hingegen stellte hingegen ein kompliziertes Unterfangen dar, das in dieser Arbeit nur kurz angeschnitten werden kann.

1.2 Ziel der Arbeit

Diese Arbeit möchte einen Überblick über die Felder Social Media Mining und Text Mining am Beispiel von *Telegram* ermöglichen und richtet sich zum einen an Personen wie Data Scientists, die an Social Media bzw. Text Mining Methoden interessiert sind, und zum anderen an Personen wie Sozialwissenschaftler, die an Auswertungen und Analyse-Möglichkeiten besagter Gruppen interessiert sind.

Im Rahmen dieser Arbeit wurde eine **Telegram Analyse Plattform**[2] geschaffen, mit der es möglich sein soll, Social Media & Text Mining Methoden auf *Telegram*-Chats anzuwenden. Diese Plattform soll ermöglichen, dass auch Personen ohne Kenntnisse aus der Informatik an dieser Arbeit partizipieren können.

1 siehe `https://telegram.org/`
2 siehe `https://maxbundscherer.github.io/telegram-analysis/`

1.3 Aufbau der Arbeit

Einführend wird daher zunächst auf die wichtigsten **Grundlagen** der Felder Social Media und Text Mining, vgl. Kapitel 2, eingegangen. Begriffe aus diesem Umfeld werden nicht immer einheitlich verwendet, daher werden die für die Arbeit wichtigen Begriffe aus diesem Feld vorgestellt, vgl. Abs. 2.1. Um sich besser vorstellen zu können, wie ein Data Mining Prozess allgemein aussieht, wird dieser Prozess an einem generischen Prozessmodell namens *Cross Industry Standard Process for Data Mining (CRISP-DM)* erläutert, vgl. Abs. 2.2. Anschließend wird das Feld *Social Media Mining*, vgl. Abs. 2.3, einführend beschrieben unter Einbeziehung der Chancen und Herausforderungen im Hinblick auf *Telegram*. Die in dieser Arbeit zur Auswertung von Chat-Nachrichten verwendeten *Text Mining* Methoden werden in Abs. 2.4 mit ihren relevanten Werkzeugen und Kennzahlen beschrieben.

In Anlehnung an die Phase **Business & Data Understanding** aus dem Prozessmodell *CRISP-DM* wird in Kapitel 3 die Vorgehensweise dieser Arbeit beschrieben. Dafür ist es zunächst erforderlich, sich näher mit dem Messenger *Telegram* und seinen Eigenheiten, vgl. Abs. 3.1, auseinanderzusetzen sowie mit dem Exportieren von Chats und dem Aufbau von Nachrichten. Ein möglicher Technologie-Stack, der in diesem Kontext eingesetzt werden kann, wird in Abs. 3.2 beschrieben. Im Rahmen dieser Arbeit wurde die *Telegram Analyse Plattform* geschaffen, die in Abs. 3.3 beschrieben wird.

Im darauffolgenden Kapitel 4 **Data Preparation & Modeling** werden verschiedene Social Media Mining und Text Mining Methoden auf die zuvor heruntergeladenen Chats angewendet, die im Kapitel *Grundlagen* zunächst nur in ihren Grundsätzen beschrieben wurden. Im Rahmen der Arbeit wurde ein Data Mining Prozess abgebildet auf zwei *Jupyter*-Notebooks [Bun21b] [Bun21a] implementiert. Dieser Prozess lässt sich in fünf Stufen unterteilen, die analog zu den Notebooks in dieser Arbeit beschrieben werden.

Im Kapitel **Fazit** wird abschließend das Vorgehen aus der Arbeit zusammengefasst und bewertet, vgl. Kapitel 5. Dabei wird auf die Vorgehensweise und auf verschiedene Implementierungen dieser Arbeit Bezug genommen.

2 Grundlagen

Um eine Vorstellung von den Feldern Social Media & Text Mining zu erlangen, wird in diesem Kapitel auf ihre Grundlagen eingegangen. Zunächst werden für die Arbeit relevante Begriffe aus diesen Feldern, die häufig unterschiedlich verwendet werden, vorgestellt, vgl. Abs. 2.1.

Um sich besser vorstellen zu können, wie ein Data Mining Prozess allgemein aussieht, wird dieser Prozess an einem generischen Prozessmodell erläutert, vgl. Abs. 2.2. Anschließend wird das Feld *Social Media Mining* einführend beschrieben, vgl. Abs. 2.3. Dabei wird sowohl auf die Chancen von Social Media Mining eingegangen, vgl. Abs. 2.3.1, als auch auf die damit verbundenen Herausforderungen, vgl. Abs. 2.3.2.

Die in dieser Arbeit zur Auswertung von Chat-Nachrichten verwendeten *Text Mining* Methoden werden in Abs. 2.4 mit ihren relevanten Werkzeugen und Kennzahlen beschrieben, vgl. 2.5.

2.1 Data Science

Da diese Arbeit Personen verschiedener Fachrichtungen ansprechen soll, ist es notwendig ein einheitliches Vokabular zu verwenden. Manche der in dieser Arbeit verwendeten Begriffe werden häufig mehrdeutig verwendet und lassen sich je nach Anwendungsfall und Fachgebiet des Lesers unterschiedlich interpretieren. Diese Arbeit versucht, die Begriffe so weit wie möglich zu beschreiben und abzugrenzen, verweist aber für tiefergehende Ausführungen mancherorts auf externe Literatur.

Die Begriffe **Data Science** und **Data Mining** sowie Begriffe aus diesem Umfeld werden häufig unterschiedlich, aber auch synonym verwendet, da viele Personen und Organisationen sich diese Begriffe gerne zu eigen machen und zu ihrem eigenem Vorteil nutzen [Pro13, S. 2]: So könnte beispielsweise ein Unternehmen damit werben, Produkte mit "künstlicher Intelligenz" zu verbessern, obwohl die meisten Produktoptimierungen durch "menschliche Intelligenz" und nicht durch künstliche Intelligenz in dem jeweiligen Unternehmen entstanden sind. Abstrakt gesprochen beschäftigt sich Data Science mit der Extraktion von Wissen, also relevanten Informationen, aus (großen) Datenmengen und wird meistens durch den Anwendungsfall konkreter definiert. Um zu verstehen, wie so ein Prozess allgemein funktioniert, wird ein Data Mining Prozessmodell im Abschnitt 2.2 beschrieben.

Der Begriff **Künstliche Intelligenz (KI)** wird in dieser Arbeit als Forschungsgebiet definiert, das sich damit beschäftigt, wie Computer selbstständig Herausforderungen lösen. So könnte beispielsweise *Google Duplex*[1], welches selbstständig telefonisch einen Friseurtermin

1 siehe `https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html`

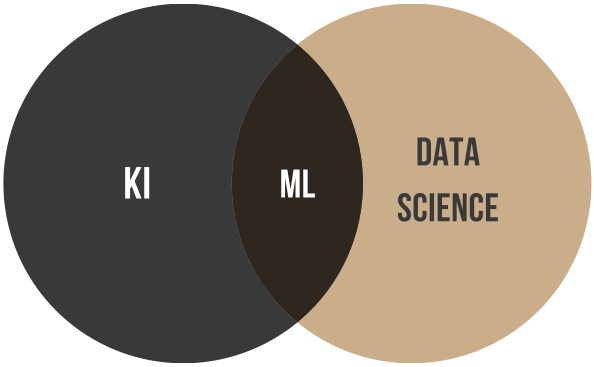

Abbildung 2.1 Darstellung der Begriffe *Data Science*, *Künstliche Intelligenz (KI)* und *Maschinelles Lernen (ML)* als Venn-Diagramm.

vereinbart, als *KI* betrachtet werden. **Maschinelles Lernen (ML)** wird in dieser Arbeit als Teilgebiet der *KI* gesehen und beschäftigt sich damit, wie ein Computer selbstständig Zusammenhänge aus Daten erkennen kann [Mül16, S. 1]. Das Verhältnis zwischen den Begriffen *Data Science*, *KI* und *ML* wird in Abbildung 2.1 als Venn-Diagramm dargestellt.

Der Begriff **Social Media Mining** lässt sich als Teilgebiet des *Data Minings* verstehen: Bei diesem Forschungsgebiet geht es darum, aus sozialen Netzwerken Informationen zu extrahieren. Mehr dazu im Abschnitt 2.3. Gleiches gilt für den Begriff **Text Mining**: Dieser beschreibt die Extraktion von Informationen aus Text, im Fall dieser Arbeit aus Chat-Nachrichten. Mehr dazu im Abschnitt 2.4.

Unter **Features** versteht ein Data Scientist **Merkmale**, die für einen Data Science Prozess von Nutzen sein könnten. Vereinfacht gesagt versucht man, in diesem Fall Text auf bestimmte *Merkmale* wie *Anzahl der Zeichen* oder *Rechtschreibfehler* abzubilden, mit denen später weiter gearbeitet werden kann.

2.2 Data Mining Prozess - CRISP-DM

Im vorherigen Abschnitt 2.1 wurde beschrieben, dass sich das Gebiet Data Mining abstrakt gesprochen damit beschäftigt, nützliche Informationen aus Daten zu extrahieren. Wie dieser Prozess im Detail aussieht, hängt direkt vom Ziel, zum Beispiel das der Wettervorhersage, und von der Ausgangslage, zum Beispiel der Datenlage, ab. Daneben können natürlich andere Herausforderungen wie unterschiedliche Sichtweisen oder Kompetenzen der Projektteilnehmer und weitere Faktoren, wie später im Abs. 2.3.2 beschrieben, das Vorhaben erschweren und damit den Prozess beeinflussen.

Um trotzdem ein allgemeines Verständnis über den Data Mining Prozess zu erlangen, kann auf ein generisches Prozessmodell namens **Cross Industry Standard Process for Data Mining (CRISP-DM)** zugegriffen werden. Nach diesem Modell wird ein Data Mining

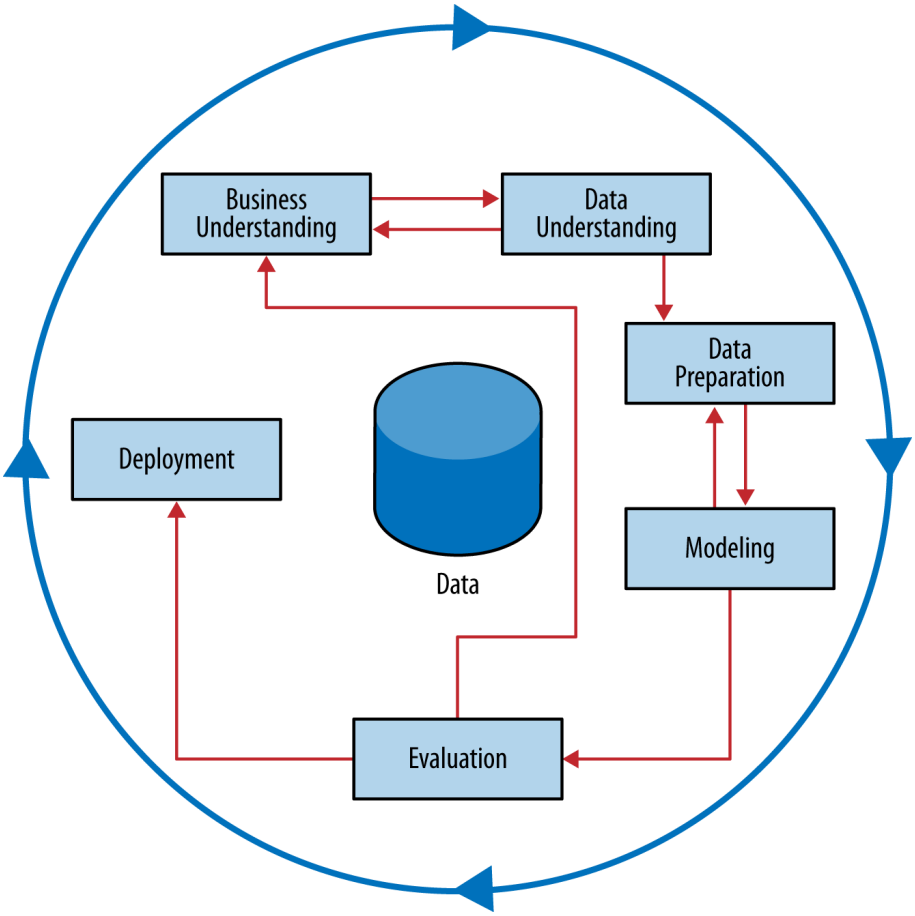

Abbildung 2.2 Visualisierung des *Cross Industry Standard Process for Data Mining (CRISP-DM)* Prozessmodells [Pro13, S. 27].

Prozess grundsätzlich in sechs Phasen unterteilt [Pro13, S. 27], die in diesem Abschnitt kurz für den Zweck dieser Arbeit beschrieben werden, da diese nicht Schwerpunkt dieser Arbeit sind und nur zum allgemeinen Verständnis beitragen sollen.

Hinweis: Abbildung 2.2 soll unter anderem verdeutlichen, dass dieser Prozess meist nicht linear stattfindet, d.h. einzelne Phasen können wiederholt werden oder sich überlappen, es kann zwischen den Phasen gewechselt werden, aber auch der ganze Prozess kann und sollte sogar noch einmal von vorne durchlaufen werden [Pro13, S. 27]. So kann beispielsweise die Phase *Evaluation* zeigen, dass die verwendeten Modelle die tatsächliche Situation nicht ausreichend abbilden. So ist es möglich, dass man von der Phase *Evaluation* wieder zur Phase *Business Understanding* zurückspringen muss, um das zu lösende Problem besser zu verstehen und damit besser modellieren zu können.

Business Understanding

In dieser Phase wird versucht, das zu lösende Problem und die Arbeitsgrundlage zu verstehen. Was sich zunächst trivial anhören mag, stellt eine große Herausforderung dar, vgl. Kapitel 3: Das zu lösende Problem oder die zu lösenden Probleme sind selten exakt bzw. tief genug für einen Data Mining Prozess beschrieben [Pro13, S. 28]. Häufig werden bereits in dieser Phase, insbesondere bei der ersten Iteration, falsche Annahmen getroffen, die das gesamte Vorhaben beeinflussen. Das Problem aus verschiedenen Sichten, zum Beispiel aus Sicht der Marketing-Abteilung oder aus Sicht der HR-Abteilung, zu betrachten ist von Vorteil, da das Problem so vollumfänglich beschrieben werden kann, was Fehlannahmen reduziert.

Die anfängliche Problembeschreibung ist häufig nicht final oder vollständig und wird häufig iterativ verbessert [Pro13, S. 28]. Ziel dieser Phase ist es also, generell die Problemstellung zu verstehen und einen groben Überblick über die künftigen Schritte zu erlangen und Ziele zu definieren.

Data Understanding

Im Rahmen dieser Phase versucht man, die zur Verfügung stehenden Daten zu verstehen und zu bewerten. Es ist wichtig, die Stärken und Grenzen der Daten zu verstehen, mit denen man versucht ein Problem zu lösen. So werden zum Beispiel Daten häufig für einen anderen Zweck gesammelt oder Daten mit unterschiedlichem Grad an Aussagekraft miteinander kombiniert [Pro13, S. 28]. So könnten zum Beispiel weniger aussagekräftige, evtl. sogar historische Marketing-Daten mit aussagekräftigeren konkreten Bestellungen kombiniert werden, um das Kaufverhalten von Kunden besser im Vorhinein abschätzen zu können.

Teil dieser Phase ist es auch abzuschätzen, welcher Aufwand, etwa in Form von Kosten oder Zeit dafür entsteht, bestimmte Daten zu sammeln [Pro13, S. 28]. So gibt es beispielsweise Daten, die ein Versandhändler bereits gesammelt hat, wie konkrete Kundenbestellungen. Andere Daten wie Marketing-Daten, zum Beispiel aus einer *Google Ads*[2]-Werbekampagne, müssen erst gesammelt werden. In dieser Arbeit muss der Aufbau der heruntergeladenen Chats ebenfalls verstanden werden, worauf im Kapitel 3 weiter eingegangen wird. Häufig

2 siehe `https://ads.google.com`

beeinflussen sich die Phasen *Business Understanding* und *Data Understanding* gegenseitig stark [Pro13, S. 27 u. 29].

Data Preparation

Werkzeuge aus dem Bereich Data Mining benötigen häufig Daten aufbereitet in einer bestimmten Struktur [Pro13, S. 29]. In dieser Phase werden die Daten so aufbereitet, dass diese mit Data Mining Werkzeugen verarbeitet werden können [Pro13, S. 30].

So könnte beispielsweise eine Textnachricht zu einem Eingabevektor für ein künstliches neuronales Netzwerk aufbereitet werden, aber auch das Normalisieren von numerischen Werten [Pro13, S. 30] oder das Entfernen von nicht signifikanten Datensätzen aus einer Menge von Daten kann als **Datenaufbereitung** betrachtet werden, vgl. Kapitel. 4.

Modeling

In dieser Phase wird versucht, die aufbereiteten Daten mit dem Verständnis über das zu lösende Problem auf ein Modell zu abstrahieren, mit dem ein Computer arbeiten kann [Pro13, S. 31]. So könnte beispielsweise ein *ML*-Verfahren wie *Support Vector Machines* den Autor einer Nachricht mit einer bestimmten Wahrscheinlichkeit identifizieren, vgl. Abs. 4.5.

In diese Phase werden also Data Mining Techniken auf die Daten angewendet [Pro13, S. 31]. Mit anderen Worten könnte man sagen, dass beispielsweise versucht wird eine statistische Gleichung aufzustellen, mit der ein Problem gelöst werden soll. Häufig beeinflussen sich die Phasen Data Preparation und Modeling gegenseitig stark [Pro13, S. 27].

Evaluation

Das Ziel dieser Phase ist es zu bewerten, ob die aufgestellten Modelle und Informationen, ob also Ableitungen aus den Daten und den Grundannahmen korrekt sind, also das generelle Vorgehen zu validieren [Pro13, S. 31].

Es ist zum Beispiel einfach, trainierte *ML*-Modelle wie neuronale Netzwerke unter Laborbedingungen an Daten auszuprobieren und daraus Rückschlüsse auf die Zuverlässigkeit des Modells über z.B. eine *Confusion Matrix*, vgl. Abs. 4.5.2, zu ziehen, ist aber nicht ratsam, falls das Modell zum Beispiel als Beratungssystem für Geschäftsentscheidungen eingesetzt werden soll [Pro13, S. 31].

Das gesamte System zu validieren, zum Beispiel mit der Durchführung und Auswertung mehrerer Feldversuche, ist unter Umständen sehr aufwendig, aber notwendig um zu vermeiden, dass zum Beispiel falsche Entscheidungen aufgrund des Einsatzes eines solchen Systems getroffen werden. Daher sollte ein solches System vor der Auslieferung und evtl. sogar noch nach der Auslieferung fortlaufend validiert werden. Im Rahmen dieser Arbeit können die Daten zwar aufbereitet und modelliert, aber nicht bewertet werden, vgl. Abs. 5.

Deployment

In dieser Phase werden Modelle und Informationen aus der vorherigen Phase *Evaluation* in der “echten Welt” angewendet: “In deployment the results of data miningand increasingly

the data mining techniques themselvesare put into real use in order to realize some return on investment."[Pro13, S. 32].

Zum Beispiel werden die aufbereiteten Informationen in einer strategischen Geschäftsentscheidung berücksichtigt oder die aufgestellten und validierten Modelle als Grundlage für ein Empfehlungssystem genutzt. In Rahmen einer Webseite [Bun21c], vgl. Abs. 3.3, stellt diese Arbeit die Artefakte aus dem Kapitel. 4 bereit.

2.3 Social Media Mining

"Social Media Mining is the process of representing, analyzing, and extracting actionable patterns from social media data."[Zaf14, S. 2]. Mit dieser Aussage lässt sich Social Media Mining stark vereinfacht zusammenfassen: Das Forschungsfeld **Social Media Mining** beschäftigt sich vorrangig mit der Darstellung, Aufbereitung, Analyse und Extraktion von relevanten Informationen aus sozialen Netzwerken wie beispielsweise *Facebook*[3] oder *Instagram*[4]. In diesem Feld werden viele Disziplinen wie Data Science, *ML*, mathematische Gebiete wie Optimierung oder Topologie, und Sozialwissenschaften vereinigt [Zaf14, S. 2].

Da das Feld sehr umfangreich ist, möchte diese Arbeit erst auf die Chancen von Social Media Mining eingehen, vgl. Abs. 2.3.1, und anschließend noch auf die Herausforderungen, vgl. Abs. 2.3.2, in diesem Feld insbesondere im Hinblick auf den Messenger *Telegram*, vgl. Abs. 3.1.

2.3.1 Chancen

Die Kommunikationsstrukturen zwischen Menschen aus sozialen Netzwerken stellen eine Erweiterung der Kommunikationsstrukturen aus der Realität dar und die Grenze dazwischen verläuft fließend [Sza18, S. 41] [Zaf14, S. 1]. So ist es es beispielsweise nicht unüblich, nach einem persönlichen Treffen mit einer Person mit dieser weiterhin über soziale Netzwerke in Kontakt zu bleiben. Auch Personen mit gemeinsamen Interessen, zum Beispiel ein Auto kaufen, würden sich in der Realität mit anderen Personen darüber austauschen. In sozialen Netzwerken werden diese Interessengruppen häufig gesammelt abgebildet, beispielsweise in einer Gruppe in *Telegram*.

Viele Firmen bieten ihre Dienstleistungen oder Produkte häufig im Internet mit Funktionen an, die an soziale Netzwerke erinnern. So ist es zum Beispiel bei der Koch-Plattform *Chefkoch*[5] möglich, Rezepte zu teilen, bewerten und kommentieren. Auch Plattformen wie *Wikipedia*[6], *YouTube*[7] und *Discogs*[8], bieten häufig mindestens eine Kommentar-Funktion an. Damit generieren diese Firmen Daten, die mit Social Media Mining Methoden verarbeitet und damit analysiert werden können. Methoden dieser Arbeit sind daher auch auf andere Plattformen übertragbar.

3 siehe `https://www.facebook.com/`
4 siehe `https://www.instagram.com/`
5 siehe `https://www.chefkoch.de`
6 siehe `https://www.wikipedia.org/`
7 siehe `https://www.youtube.com/`
8 siehe `https://www.discogs.com/`

Um zu verstehen, was man sich von Social Media Mining erhofft bzw. damit möglich ist, lohnt es sich zu hinterfragen, wofür soziale Netzwerke eigentlich genutzt werden [Bon16, S. 10]: Zu erwähnen wäre da, um mit Freunden, Kollegen, Bekannten und Familien in Kontakt zu bleiben (*Facebook*) und nebenbei die aktuellen News auf *Twitter*[9] zu lesen, sich über den künftigen Arbeitgeber zu informieren (*Kununu*[10]), Antworten auf Fragen zu finden (*Stack Overflow*[11]) oder Gleichgesinnte kennenzulernen (*Parship*[12]).

Es gibt viele Anwendungsmöglichkeiten, die in zahlreichen Büchern wie [Kaz15] und [Rus13] beschrieben oder auf Konferenzen wie [Hua14] diskutiert werden. Mit den Social Media Mining Methoden könnte beispielsweise der Betreiber eines sozialen Netzwerkes seine Nutzer klassifizieren und so gezielt Werbung ausspielen. Aber auch Firmen aus der Lebensmittelindustrie könnten ihre Produktion im Vorhinein anpassen, indem diese Firmen Social Media Mining Methoden in Echtzeit auf die Daten aus einer Plattform wie *Chefkoch*, also die Kommentare, Aufrufe und Gefällt-mir-Angaben zu Rezepten, anwenden und auf die Erkenntnisse reagieren.

2.3.2 Herausforderungen

Bei der Extraktion von Wissen aus Sozialen Netzwerken gibt es viele Herausforderungen. Diese Arbeit beschreibt diese einführend im Hinblick auf den Messenger *Telegram*, vgl. Abs. 3.1, weitere Information zu dem Thema findet man in der jeweiligen erwähnten Literatur.

Es fängt bei **Privacy Restrictions** (dt. "Datenschutzeinschränkungen") an [Sza18, S. 42]: So bilden beispielsweise die heruntergeladenen *Telegram* Chats die Chats nicht vollständig ab, sondern nur aus Sicht des Nutzers, der den Chat heruntergeladen ab. So können beispielsweise Nachrichten von Nutzern nicht gesehen werden, die den Download-Nutzer blockiert haben.

Eine weitere bekannte Herausforderung stellt das **Big Data Paradox** dar: Soziale Netzwerke sammeln häufig extrem viele Daten. Bei dieser Datenmenge kann man den Begriff *Big Data* hier ruhig anführen. Oftmals hat etwa ein Empfehlungssystem für Produkte, das für eine Person Empfehlungen ausspricht, evtl. Zugriff auf viele (strukturierte) Daten, aber nicht ausreichend Informationen, um diese Person für eine exakte Empfehlung zutreffend einschätzen zu können [Zaf14, S. 2].

Es ist meistens nicht möglich zu validieren, ob man alle relevanten Daten zur Lösung eines Problems hat. So ist es beispielsweise in dieser Arbeit nicht möglich zu validieren, ob man alle relevanten *Akteure* innerhalb eines Netzwerks heruntergeladen hat, vgl. Abs. 4.3.2. Dieses Problem wird als **Obtaining Sufficient Samples** bezeichnet [Zaf14, S. 3].

Daten, insbesondere Daten von Text-Nachrichten können etwa Rechtschreibfehler enthalten. In klassischer Data Mining Literatur würde man die Herausforderung als **Noise reduction** (dt. "Rauschunterdrückung") bezeichnen [Zaf14, S. 3]: Stellt man eine Analogie zwischen Social Media Mining und Signalanalyse her, so kann man Daten aus sozialen Netzwerken als

9 siehe `https://twitter.com`
10 siehe `https://www.kununu.com/`
11 siehe `https://stackoverflow.com/`
12 siehe `https://www.parship.de/`

Signal interpretieren: Rechtschreibfehler und Eigenheiten der deutschen Sprache können z.B. Text Mining Methoden stören, vgl. Abs. 4.2.1. Diese könnten als Rauschen (engl. “noise”) interpretiert werden, was innerhalb der Datenaufbereitung unterdrückt (engl. “reduction”) wird, vgl. Abs. 4.2.

Das **Evaluation Dilemma** [Zaf14, S. 3]: Im Abschnitt 2.2 wurde innerhalb der Phase *Evaluation* beschrieben, dass es notwendig ist, die erarbeiteten Modelle und Informationen, zum Beispiel mit der Durchführung und Auswertung mehrerer Feldversuche, zu validieren. Im Rahmen dieser Arbeit können nur Social Media Mining Werkzeuge beschrieben, bereitgestellt und auf Beispiel-Daten angewendet werden. Die Evaluation konnte im Rahmen dieser Arbeit nicht durchgeführt werden, da ein Abgleich mit der Realität unmöglich oder nur sehr zeitaufwendig möglich wäre, vgl. Kapitel 5.

2.4 Text Mining & Werkzeuge

Die Extraktion von nützlichen Informationen aus Texten wird häufig als **Text Mining**, **Text Analytics**, oder **Machine learning from text** bezeichnet, abhängig von Anwendergruppe, Anwendungsfall und Zielgruppe [Agg18, S. 1], vgl. Abs. 2.1.

Im Buch “Fundamentals of Predictive Text Mining” wird in scherzhafter Weise unterschieden zwischen den **numbers guys** als Personen, die Prognosemodelle zum Beispiel für Wettervorhersagen auf Basis von Zeitreihenmodellen mit numerischen Werten (Temperatur), entwickeln, und den **text miners** als Personen, die direkt mit Text arbeiten und nicht auf numerische Werte angewiesen sind.[Wei15, S. 1]. Diese scherzhafte Differenzierung liegt nahe wenn man betrachtet, dass die meisten Data Mining Methoden Daten stark in numerischer Form aufbereitet benötigen, was eine Datenaufbereitung, vgl. Abs. 4.2, notwendig macht [Wei15, S. 1].

Daher ist es notwendig, die heruntergeladenen Text-Nachrichten erst aufzubereiten, was bereits schon in Abschnitt 2.3.2 angeschnitten wurde. Da die zu betrachtenden Texte also aus Nachrichten im Rahmen verschiedener “Telegram-Chats” von unterschiedlichen Menschen mit unterschiedlichem Hintergrund und Ziel geschrieben wurden, liegt es nahe, dass die Textnachrichten Rechtschreib- & Grammatikfehler, unterschiedliche Schreibweisen von Begriffen bzw. Eigennamen, Stilmittel wie **Emoticons** (z.B: **:)**) und vieles mehr enthalten, die bei vielen Text Mining Methoden eher störend als nützlich sind. Auch die deutsche Sprache hat Eigenheiten wie Umlaute, vgl. Abs. 4.2.1, in die diesem Kontext nicht von Vorteil sind. Diese Arbeit kann diese Eigenschaften nicht betrachten, auch wenn diese zum Beispiel bei der *Identifizierung von Autoren*, vgl. Abs. 4.5, nützlich sein könnten.

Vereinfacht gesagt muss man mit Text etwas tun, um diesen auswerten zu können, vgl. Abs. 2.4. In diesem Umfeld gibt es dafür verschiedene Werkzeuge. Generell ist aber das Ziel, den Text in irgendeiner Weise auf numerische Werte abzubilden, mit denen der Computer arbeiten kann.

Die Methoden aus dem Feld sind zahlreich und können im Rahmen dieser Arbeit nicht vollständig erläutert werden. In dieser Arbeit werden einige Werkzeuge zunächst einführend theoretisch beschrieben und später auf die heruntergeladenen Chats, vgl. Kapitel 4,

angewendet.

2.4.1 Tokenization

Die **Tokenization** (dt. "Tokenisierung") bezeichnet die Segmentierung, also die Unterteilung eines Textes in (aussagekräftige) Einheiten, die als **Token** bezeichnet werden [Wei15, S. 17], zum Beispiel in Wörter oder Satzzeichen [Bir09, S. 121], aber evtl. auch bestimmte *Such-Terme*, vgl. Abs. 4.3.4.

Ein sehr triviales Vorgehen wäre zunächst, den Text einfach anhand von Leerzeichen zu trennen [Bir09, S. 109]: So würde beispielsweise der Satz "Hallo, mein Name ist Max!" wie folgt zerlegt werden: `['Hallo,'; 'mein', 'Name', 'ist', 'Max!'].` Je nach Anwendungsfall und angewendetem *ML*-Verfahren könnten Satzzeichen vernachlässigbar oder sogar störend sein. Um dem entgegenzuwirken, könnte man z.B. Satzzeichen herausfiltern, was zumindest in diesem ausgedachten Fall zu einer Lösung führt.

Generell lässt sich zur *Tokenization* noch sagen, dass es hier kein allgemeines Vorgehen gibt. Das Vorgehen hängt vom Ziel ab, meistens werden diese Verfahren für die Datenaufbereitung für anschließende Text-Mining Methoden verwendet. So spielen beispielsweise Satzzeichen bei der *Latent Dirichlet Allocation (LDA)*, siehe Abschnitt 4.4.7, keine Rolle und können deswegen einfach heraus gefiltert werden.

Wie bereits im Abschnitt 2.4 beschrieben, sind beispielsweise Rechtschreibfehler in den Nachrichten enthalten. Diese können die *Tokenization* stark beeinflussen und es könnte zu fehlerhaften *Token* kommen.

In dieser Arbeit wird die *Tokenization* für die deutsche Sprache separat in Abschnitt 4.2.1 behandelt.

2.4.2 Stopword removal

Bei der **Stopword removal** (dt. "Stoppwort-Entfernung") entfernt man für den Anwendungsfall nicht relevante *Token* [Wei15], genannt **Stopwords**, die meist sehr häufig vertreten und nicht aussagekräftig sind, etwa Wörter wie "ein".

Welche Wörter zu den *Stopwords* gezählt werden, ist meistens eine subjektive Entscheidung und wird stark vom Anwendungsfall beeinflusst. So könnten beispielsweise Wörter wie "das" bei grafischen Ausgaben wie *Word Clouds*, vgl. Abs. 4.4.2, nicht zum Verständnis beitragen, aber wenn man zum Beispiel in einem Text nach einem Filmtitel wie "Das Boot"[13] sucht, könnte man den Film nicht finden, wenn man das Wort "das" vorher entfernt hat.

2.4.3 Bag-Of-Words (BOW)

Die **Bag-Of-Words (BOW)**-Repräsentation ist eine weit verbreitete und simple, aber gut dazu geeignete Repräsentation, Text für ein *ML*-Verfahren darzustellen [Mül16, S. 327].

Bei diesem Ansatz stellt man einen Text meist als Feature-Vektor für andere *ML*-Verfahren

13 siehe `https://de.wikipedia.org/wiki/Das_Boot_(Film)`

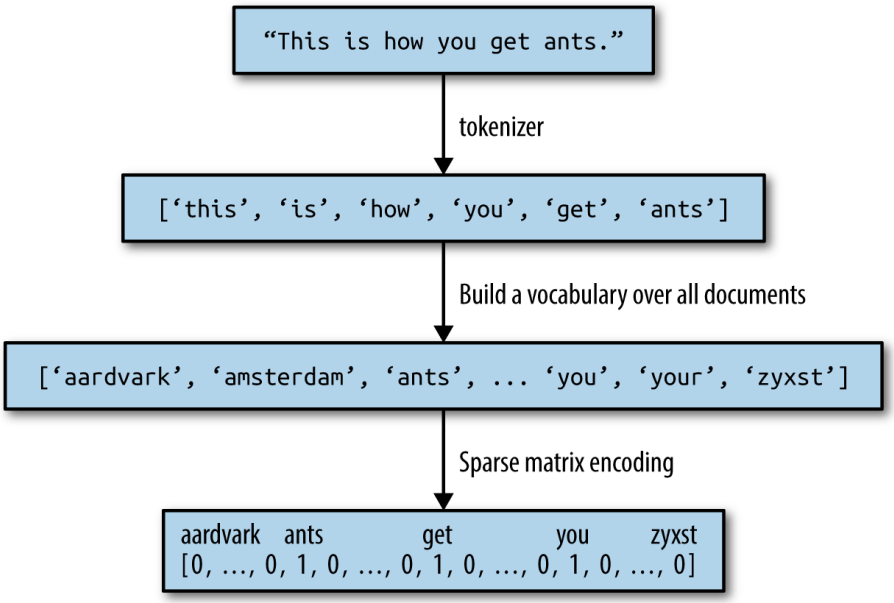

Abbildung 2.3 Visualisierung der *Bag-Of-Words (BOW)* Darstellung: Vom Text zum Feature-Vektor [Mül16, S. 328].

dar, vgl. Abs. 4.5. So würde beispielsweise der Text “Studenten sind faul und Studenten essen gerne Eis.”, zu folgender Repräsentation führen:
`{"Studenten":2, "sind":1, "faul":1 ,"und":1, "essen" [...]}`.

Wichtig zu verstehen ist bei diesem Ansatz, dass nur die *Token* eines Textes unabhängig von Reihenfolge, Grammatik und Rechtschreibung dargestellt werden [Pro13, S.252] [Har16, S. 354].

In der Praxis wird ein Text häufig auf eine **Sparse matrix**, also einer dünn bzw. schwach besetzten Matrix abgebildet. Grundsätzlich wird ein Text in drei Schritten auf eine solche abgebildet [Mül16, S. 328]:

1. **1: Tokenization**: Zunächst wird der Text auf *Token*, vergleiche Abschnitt 2.4.1 abgebildet.
2. **2: Vocabulary building**: Über alle Texte eines Dokuments wird ein gemeinsames *Vokabular* aus den *Token* aufgestellt. Das gemeinsame *Vokabular* repräsentiert die Struktur der Zielmatrix.
3. **3: Encoding**: Für jeden Text wird gezählt, welche *Token* wie häufig im gemeinsamen *Vokabular* vorkommen. Jeder zu verarbeitende Text kann so konkret auf die Zielmatrix abgebildet.

Die Abbildung 2.3 visualisiert dieses Vorgehen. In der Praxis können diese Matrizen durch diese Form der Abbildung (pro *Token* eine Spalte) ziemlich groß werden und sollten daher optimiert implementiert werden. Es empfiehlt sich, als Struktur der Zielmatrix nicht das vollständige Wörterbuch einer Sprache zu nehmen, da dies meist nicht besser zum Ziel führt, sondern nur den Rechenaufwand erhöht [Mül16, S. 330].

2.4.4 N-grams/Bag-Of-N-grams

Bei **N-grams** versucht man ähnlich wie bei der *BOW*-Repräsentation, einen Text auf verschiedene *Token* abzubilden. Der Unterschied besteht darin, dass die *Token* hier nicht nur einzelne Wörter sind, sondern ein *Token* auch (in der korrekten Reihenfolge) aus mehreren (daher das **n**) *Token* besteht bzw. bestehen kann. Das Verfahren berücksichtigt also die Reihenfolge der Wörter [Pro13, S. 263].

Das *n* steht für die Anzahl der Wörter in einem *Token*. Einige der *N-grams* haben einen eigenen Namen: So nennt man ein N-Gram mit `n=1` ein **Monogramm**, mit `n=2` **Bigramm**, mit `n=3` **Trigramm** [Mül16, S. 341].

So würde beispielsweise der Satz "Mir geht es heute gut!" wie folgt auf ein *Monogramm* abgebildet werden: `['Mir'; 'geht', 'es', 'heute', 'gut!']`, und wie folgt auf ein *Bigramm*: `['Mir geht'; 'geht es', 'es heute', 'heute gut']`, und wie folgt auf ein *Trigramm*:
`['Mir geht es'; 'geht es heute', 'es heute gut']`.

Manche Literatur ordnet die *N-grams* dem *BOW*-Ansatz zu [Pro13, S. 263] [Mül16, S. 339], da es zum Beispiel auch möglich ist, die *Token* ähnlich wie bei dem *BOW*-Ansatz als Eingabevektor, zum Beispiel als *Sparse matrix*, für ein angeknüpftes *ML*-Verfahren abzubilden, vgl. Abs. 2.4.3. Diese Arbeit sieht dies als verwirrende Ausdehnung der Definition und grenzt es daher mit dem Begriff **Bag-Of-N-grams** ab. In dieser Arbeit wurde noch weiter auf die *N-grams*, in Abs. 4.4.3 eingegangen.

2.4.5 Lemmatization/Stemming

An dieser Stelle ist zu beachten, dass manche Literatur die Begriffe *Lemmatization* (dt. "Lemmatisierung" oder kurz "Lemma") und *Stemming* (auch "Stammformreduktion") häufig synonym dafür benutzt, ein Wort auf eine einheitliche Grundform zu reduzieren [Wei15, S. 19] [Bir09, S. 107] [Agg18, S. 23]. Meist folgen diese Verfahren im Anschluss an die *Tokenization* [Wei15, S. 19].

Grundsätzlich sollte aber zwischen diesen beiden Vorgehensweisen differenziert werden, auch wenn diese als Normalisierungsverfahren für Text gesehen werden können [Bir09, S. 107]. Die **Lemmatization** nutzt auch den Kontext eines Worts, um dieses auf eine einheitliche Grundform abzubilden, während beim **Stemming** ausschließlich das Wort verwendet wird [Agg18, S. 24]. So könnte beispielsweise das Wort "Liebe" auf die Grundform des Substantivs "Die Liebe" oder auf die Grundform des Verbs "Ich liebe dich!" abgebildet werden. Einem *Stemming*-Verfahren, das zur Klassifikation genutzt würde, wäre es an dieser Stelle nicht möglich, eine korrekte Entscheidung zu treffen.

Lemmatization arbeitet meistens mit morphologischer Analyse und einem Vokabular (Dictionary) und hängt daher sehr stark von der verwendenten Sprache ab [Agg18, S. 24]. In dieser Arbeit wird auf die *Lemmatization* für die deutsche Sprache separat im Abschnitt 4.2.1 eingegangen.

Als berühmter *Stemming*-Algorithmus kommt z.B. der *Porter Stemmer* gerne zum Einsatz

[Bir09, S. 107]: Bei diesem Algorithmus werden bestimmte Abbildungsvorschriften zur Verkürzung eines Wortes solange auf ein Wort angewendet, bis dieses eine bestimmte Anzahl von Silben aufweist. Der Algorithmus wurde ursprünglich für die englische Sprache entwickelt, kann aber auch auf andere Sprachen portiert werden [Har16, S. 27].

2.4.6 Part-of-Speech Tagging (POS-Tagging)

Beim sogenannten **Part-of-speech Tagging (POS-Tagging)** werden *Token*, z.B. "essen", meistens einer Wortart, z.B. "Verb" zugeordnet. Das Tagging ist sehr stark abhängig von der Sprache. Unterschiedliche Sprachen besitzen unterschiedliche Wortarten, die meisten Sprachen haben zumindest zwei unterschiedliche Wortklassen: Substantive und Verben [Wei15, S. 31].

Die Zuordnung von Wortarten könnte beispielsweise über ein Wörterbuch (engl. "Dictionary") erfolgen. Ein Wörterbuch hat die Eigenschaft nie vollständig zu sein, es ist aber trotzdem möglich, bestimmte *Token* mit einer bestimmten Wahrscheinlichkeit aufgrund von bestimmen Merkmalen wie z.B. Groß-/Kleinschreibung und der Position im Satz zuzuordnen [Wei15, S. 31].

Ähnlich wie bei fast allen Text Mining Werkzeugen hängt die konkrete Umsetzung vom Ziel ab: So könnte man das *POS-Tagging* als Vorstufe zu einem weiteren Verfahren sehen, zum Beispiel als Datenaufbereitungsstufe für die sogenannte *Named-entity recognition (NER)*, vgl. Abs. 2.4.7 [Wei15, S. 31].

HanTa implementiert einen für diese Arbeit geeigneten *POS-Tagger* für die deutsche Sprache, vgl. Abs. 4.2.1.

Hinweis: *Stemming*, vgl. Abs. 2.4.5, sollte bei der Anwendung vom *POS-Tagging* vermieden werden, da die meisten *Stemming-Algorithmen* die *Token* stark modifizieren, was dazu führen könnte, dass das Tagging zu unterschiedlichen Ergebnissen kommt [Har16, S. 27].

2.4.7 Named-entity recognition (NER)

Bei der **Named-entity recognition (NER)** versucht man relevante Entitäten, zum Beispiel für den Text relevante Personen, Organisationen, Orte, Termine, Filmtitel, Events etc. aus einem Text zu identifizieren [Wei15, S. 33] [Har16, S. 42] [Bir09, S. 281] [Pro13, S. 264].

Die häufig verwendete deutsche Übersetzung "Eigennamen-Erkennung"[14] erscheint wenig geglückt, da bei dieser Übersetzung gerne übersehen wird, dass das Ziel nicht nur Namen von Personen sind. Ein besserer Vorschlag hierfür wäre "Entitäten-Erkennung", auch wenn dies nicht unmittelbar zum Verständnis des Verfahrens beiträgt.

Die Informationen aus einer *NER* können beispielsweise für ein System verwendet werden, was eingehende Support-Anfragen automatisch clustert, den Text auswertet und relevante Entitäten herausfiltert, um die Nachricht dem richtigen Mitarbeiter zuteilen zu können .[Bir09, S. 281], Auf die *NER* wird im Abs. 4.4.5 weiter eingegangen.

14 siehe z.B. `https://de.wikipedia.org/wiki/Named-entity_recognition`

2.4.8 Sentiment analysis

Bei der **Sentiment analysis** (dt. "Stimmungs-Analyse") versucht man, einen Text (oder Ausschnitte) einer bestimmten Stimmung, z.B. "positiv", "negativ" oder "neutral" zuzuordnen [Har16, S. 580].

Die Möglichkeiten für die Stimmungen beschränken sich nicht auf die oben genannten Stimmungen, sondern hängen vom Ziel ab [Wei15, S. 156]. So könnte man beispielsweise Film- oder Produkt-Rezensionen Werte zwischen 1 und 5 Sterne zuweisen.

Vereinfacht gesagt könnte man die *Sentiment analysis* als (Multi-Klassen) Klassifikationsproblem sehen. Die Auswahl der Klassen erfolgt nach Anwendungsfall, die Zuordnung von Text zu diesen ist subjektiv [Har16, S. 580].

Häufig greifen diese Verfahren auf ein Lexikon zurück, bei dem die Wörter einer bestimmten Stimmung zugeordnet sind. Es ist aber auch möglich, mit *NER*-Ausgaben, vgl. Abs. 2.4.7, oder mit *N-Gram*-Ausgaben, siehe Abschnitt 2.4.4, zu arbeiten [Har16, S. 585]. In dieser Arbeit wird weiter auf die *Sentiment analysis* in Abs. 4.4.6 eingegangen.

2.4.9 Topic Models

Ein Text kann von einem oder mehreren **Topics** (dt. "Themen") handeln. Es liegt nahe, diese *Topics* in einem Text oder Chat zu identifizieren. Die Verfahren, die einen Text auf einzelne *Topics* abbilden, werden **Topic Models** genannt [S. 264][Pro13].

Prominente Vertreter sind die *Latent semantic analysis (LSI)* und die *Latent Dirichlet Allocation (LDA)* [Pro13, S. 265]. Die Arbeit beschreibt die *LDA* nicht, es gibt aber eine gut verständliche Zusammenfassung [Sch14, S. 5], auch fundiert z.B. [Gel15, S. 203] und für die praktische Anwendung mit Python [Mül16, S. 348].

Stark didaktisch vereinfacht und damit auch teilweise inkorrekt kann man sich die *LDA* als Text-Generierungs-Maschine vorstellen: Diese Maschine generiert abhängig von verschiedenen Eingabe-Parametern, wie der *Anzahl der Topics* und der *Verteilungs-Parameter* der Multinomialverteilungen und Dirichlet-Verteilung, die vor allem die Text-Generierung beeinflussen, sowie einem *Referenztext* meist eine große Anzahl von Texten. Diese können als **Latent Information** gesehen werden. Diese generierten Texte ergeben einfach gesagt keinen Sinn, können aber dazu benutzt werden zu evaluieren, wie gut die Maschine gearbeitet hat, indem diese Texte mit dem Referenztext auf Ähnlichkeit verglichen werden.

Suchmaschinenbetreiber wie *Google* könnten die *LDA* als Dimensionsreduktions-Verfahren einsetzen, um mithilfe von *Latent Information* Suchanfragen schneller und gezielter beantworten zu können [Pro13, S. 303]. Dieses Vorgehen setzt allerdings den Zugriff auf sehr viele Daten voraus, in diesem Fall indexierte Webseiten, was für Firmen wie *Google* kein Problem darstellt.

In Abs. 4.4.7 wird mithilfe der *LDA* trotzdem versucht herauszufinden, über welche *Topics* in den Chat Gruppen diskutiert wird.

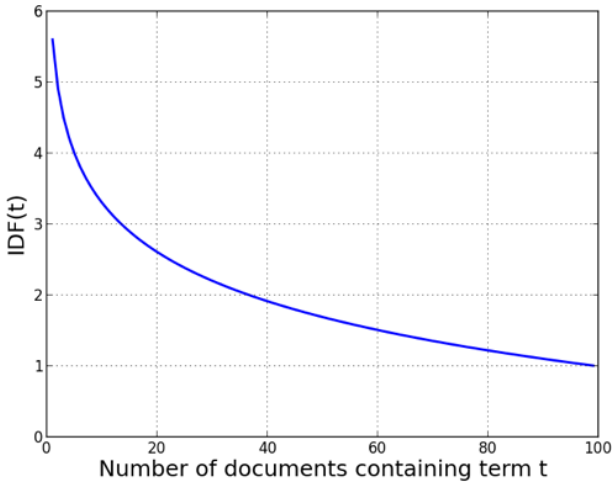

Abbildung 2.4 Inverse Document Frequency (idf) an einem abstrakten Beispiel [Pro13, S. 255].

2.5 Text Mining Metriken

In dem Feld Text Mining gibt es verschiedene Kennzahlen, die je nach Anwendungsfall unterschiedlich geeignet sind. Diese Arbeit wird drei Kennzahlen aus diesem Gebiet vorstellen. Es gibt noch andere [Wei15, S. 27] [Pro13, S. 256], es werden aber nur die Zahlen beschrieben, die für diese Arbeit wichtig sind.

An dieser Stelle wichtig zu verstehen ist, dass diese Kennzahlen (meist abbildbar auf Matrizen), z.B. auch als Datenaufbereitungsverfahren für ein neuronales Netzwerk, vgl. Abs. 4.5, verwendet werden können.

2.5.1 Term Frequency (tf)

Die **Term Frequency** (dt. "Vorkommenshäufigkeit") gibt an, wie oft ein bestimmter **Term** t in einem **Dokument** d vorkommt und kann wie folgt angegeben werden:

$$f(t,d)$$

So würde beispielsweise beim Satz "Studenten sind faul und Studenten essen gerne Eis!" die *Term Frequency* des Terms "Studenten", wie folgt angeben werden:

$$f(\text{“}Studenten\text{“}, \text{“}Studenten\ sind\ ...\ gerne\ Eis!\text{“}) = 2$$

Bei längeren Dokumenten ist diese Definition nicht geeignet, da meistens die gesuchten *Terme* unter den anderen *Termen* untergehen. Die *Term Frequency* wird daher häufig wie folgt verwendet und angeben [Har16, S. 68]:

$$tf(t,d) = 0,5 + \frac{0,5 \times f(t,d)}{max\{f(w,d) : w \in d\}}$$

2.5.2 Inverse Document Frequency (idf)

Die **Inverse Document Frequency** (dt. "Inverse Dokumenthäufigkeit"), gibt an, wie relevant ein *Term* für die Gesamtmenge der **betrachteten Dokumente** D ist, anders als die *Term*

Frequency tf, die sich nur auf ein Dokument (d) bezieht [Wei15, S. 25] [Pro13, S. 254].

Die *Inverse Document Frequency* kann wie folgt angegeben werden [Pro13, S. 255]:

$$idf(t, D) = 1 + log(\frac{\text{“}total\ number\ of\ documents\text{“}}{\text{“}number\ of\ documents\ containing\ t\text{“}})$$

Wie man in Abbildung 2.4 sehen kann, wird der Wert für die *idf* sehr groß, wenn nicht viele Dokumente den gesuchten *Term* beinhalten, und nähert sich schnell dem Wert 1 an (gegeben durch die obige Formel).

2.5.3 Term Frequency & Inverse Document Frequency (tf-idf)

Ein beliebtes Vorgehen ist es, die Kennzahlen *Term Frequency (tf)* und *Inverse Document Frequency (idf)* zu einer **tf-idf** zu kombinieren, um zu einer aussagekräftigeren Kennzahl zu gelangen [Har16, S. 68] [Pro13, S. 256]:

$$tfidf(t, d, D) = tf(t, d) \times idf(t, D)$$

Häufig wird diese Kennzahl als Eingabe für weitere *ML*-Verfahren verwendet, zum Beispiel für Autorenklassifizierung in Abschnitt 4.5.

3 Business & Data Understanding

In dieser Arbeit wurde anfangs explorativ vorgegangen, um eine Strategie zu entwickeln, wie es im Bereich Data Science nicht unüblich ist. Dafür wurde sich zunächst mit dem sozialen Netzwerk *Telegram* und dessen Funktionen auseinandergesetzt und definiert, welche Chats heruntergeladen werden sollen, damit diese später ausgewertet werden können. Bereits diese Frage konnte im Rahmen der Arbeit nicht abschließend beantwortet werden. Während dieser explorativen Vorgehensweise wurde auch festgestellt, dass die Zielsetzung der Arbeit, vgl. Abs. 1.1, nicht ausreichend konkret für einen Social Media und Text Mining Prozess formuliert ist, vgl. Abs. 2.2.

Es wurden daher initial Chats ausgewählt, diese wurden heruntergeladen und anschließend mit Social Media und Text Mining Methoden ausgewertet.

Abbildung 3.1 stellt die **Knowledge-Pyramide** dar: Die Rohdaten (*DATA*) werden auf der untersten Stufe zunächst in Verbindung gebracht (*CONTEXT*). In diesem Kontext könnte man sagen, dass die heruntergeladen Chats zunächst aufbereitet, auf Features abgebildet und anschließend modelliert werden, vgl. Kapitel. 4.

Die Rohdaten werden also auf Informationen (*INFORMATION*) abgebildet. Durch die Abstraktion können Informationen verloren gehen, was später zu falschen Entschlüssen führen kann, vgl. Abs. 2.3.2. Falls man das nötige Wissen besitzt, diese Informationen zu interpretieren, kann man aus den Daten konkrete Aussagen (*KNOWLEDGE*) ableiten. Wie in Abs. 5 beschrieben, kann diese Arbeit nur den Übergang von der Stufe *DATA* zur Stufe *INFORMATION* sinnvoll gewährleisten, da der Verfasser nicht über ausreichend Wissen aus den sozialen Wissenschaften verfügt und das der zeitliche Rahmen einer Masterarbeit nicht zulässt.

Andere Ansichten, also z.B. von Sozialwissenschaftlern sind erforderlich, um diese Auswertungen aus der Arbeit zu interpretieren und das Vorgehen aus dieser Arbeit, zum Beispiel in Form eines Feldversuchs zu validieren. Daher wurde im Rahmen dieser Arbeit eine Plattform zur Verfügung gestellt, die das Mitarbeiten verschiedener Personen an dieser Idee ermöglichen soll, vgl. 3.3.

Bevor auf diese Plattform sinnvoll eingegangen werden kann, wird zunächst das soziale Netzwerk *Telegram*, vgl. 3.1, beschrieben und anschließend auf die Technologien hinter der Plattform eingegangen, vgl. Abs. 3.2.

3.1 Telegram

Telegram beschreibt sich selbst als eine kostenlose Messenger-App mit dem Schwerpunkt auf Geschwindigkeit und Sicherheit und zählt daher zu den sozialen Netzwerken, vgl. 2.3.

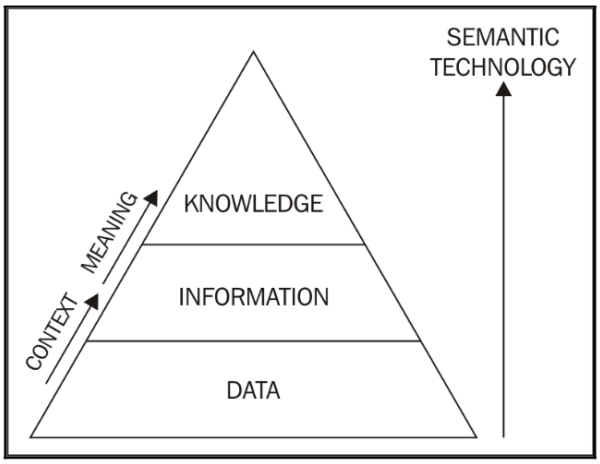

Abbildung 3.1 Darstellung der *Knowledge-Pyramide* [Bon16, S. 10].

Der Messenger kann sowohl auf dem Mobiltelefon als App[1] für z.B. iOS und Android, als auch auf dem Computer als Web App[2] oder klassische Desktopanwendung[3] benutzt werden. *Telegram* wurde das erste Mal am 14. August 2013 für das iPhone veröffentlicht und unterstützt Chat-Gruppen bis 200.000 Mitglieder[4].

Grundsätzlich gehört jede Nachricht in *Telegram* zu einem Chat. Der Messenger unterscheidet zwischen verschiedenen Chat-Arten[5], die im folgenden Abschnitt aufgelistet werden. Falls man an die Problematik **Privacy Restrictions**, vgl. Abs. 2.3.2, denkt, stellt man fest, dass diese bereits hier zum Tragen kommt: Nicht alle Nachrichten innerhalb des Chat sind zugänglich.

Private chats

Eine gewöhnlicher Chat zwischen zwei Benutzern oder mit sich selbst (es ist möglich, mit sich selbst zu schreiben, um beispielsweise Erinnerungen zu dokumentieren). Diese Art von Chats werden in dieser Arbeit nicht betrachtet, da diese schlicht nicht zugänglich sind.

Basic groups

Eine Chat-Gruppe von 0 bis zu 200 Teilnehmern. Jeder Teilnehmer hat eine Kopie der Nachrichten, so können neue Nutzer beim Eintreten in diese Gruppen-Art die den vergangenen Nachrichten-Verlauf nicht sehen. Diese Art von Chats werden in dieser Arbeit nicht betrachtet, da diese aufgrund der minimalen Teilnehmeranzahl nicht aussagekräftig wären.

Supergroups

Eine Chat-Gruppe von 0 bis zu 200.000 Teilnehmern. Im Vergleich zur normalen Gruppen können neue Nutzer beim Eintreten in diese Gruppen-Art den vergangenen Nachrichten

1 siehe `https://telegram.org/apps#mobile-apps`
2 siehe `https://telegram.org/apps#web-apps`
3 siehe `https://telegram.org/apps#desktop-apps`
4 siehe `https://telegram.org/faq`
5 siehe `https://core.telegram.org/tdlib/getting-started#tdlib-glossary`

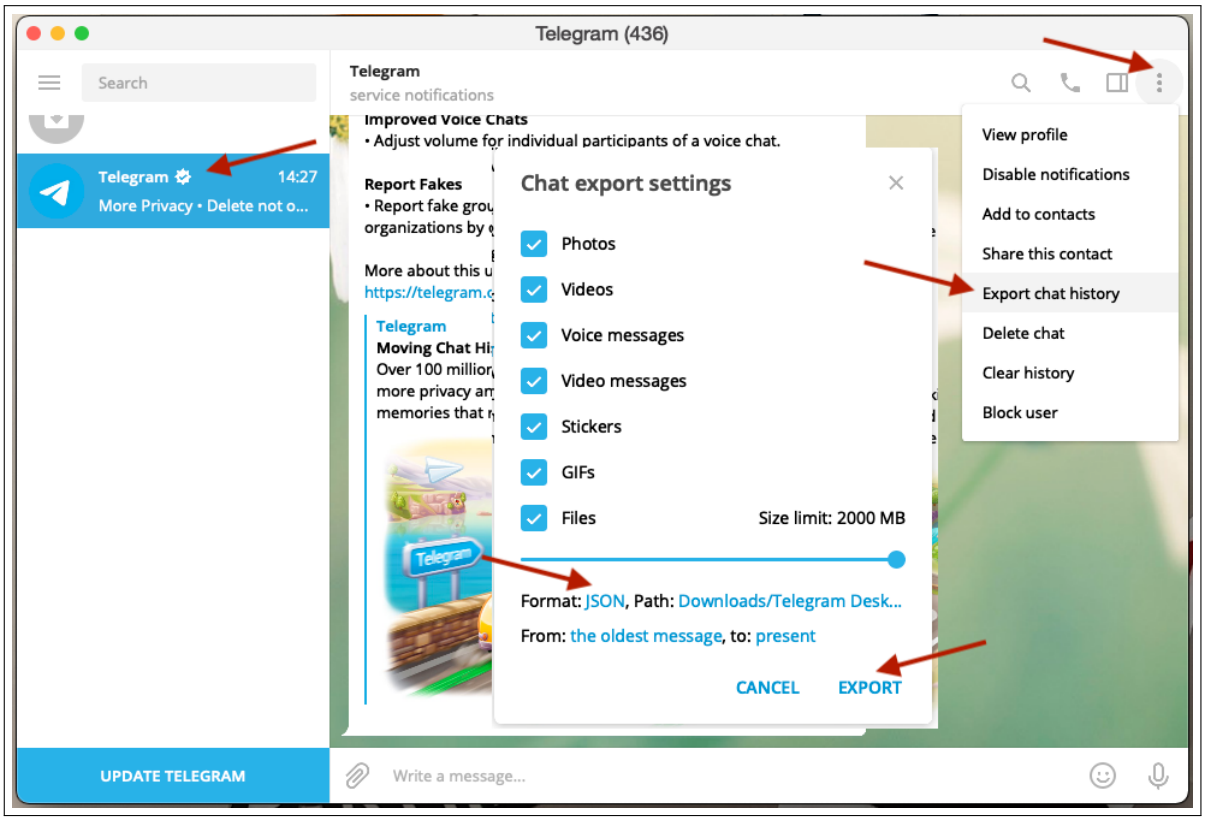

Abbildung 3.2 Anleitung: Exportieren eines Chats in der offiziellen Desktopanwendung von Telegram in eine JSON-Datei.

Verlauf einsehen. Dieses Verhalten kann aber vom Gruppen-Administrator beeinflusst werden. Es gibt eine spezielle Art von Supergruppen, den sogenannten **Channel** (dt. Kanal), der eine unbegrenzte Anzahl an Teilnehmern unterstützt, wobei aber nur der oder die *Channel*-Administrator(en) Nachrichten teilen dürfen. Der Kommunikationweg führt hier also nur in eine Richtung. Diese Art von Chats werden in dieser Arbeit hauptsächlich betrachtet.

Secret chats

Ein End-zu-End verschlüsselter Chat zwischen zwei Benutzern oder mit sich selbst. Durch die Verwendung von End-zu-End-Verschlüsselung können die Nachrichten nur auf dem Geräten gelesen werden, auf dem der Chat initiiert worden ist. D.h. Nachrichten können nicht über mehrere Geräte hinweg synchronisiert werden, was diese Art von Chat von den private chats unterscheidet. Diese Art von Chats werden in dieser Arbeit nicht betrachtet, da diese ebenfalls nicht zugänglich sind.

3.1.1 Exportieren von Chats

Telegram selbst bietet offiziell zwei APIs namens *Bot API* für die Anbindung von *Bots*, vgl. Abs. A.1.2, und *TDLib* an, um eigene *Telegram*-Clients entwickeln zu können. So haben

sich zahlreiche Open Source Projekte wie *Unigram*[6] oder *MadelineProto*[7] entwickelt. Auf nicht offizielle Clients geht diese Arbeit nicht weiter ein, da diese vom Funktionsumfang ähnlich sind wie die offiziellen Clients, da die APIs zumindest die Serverfunktionen vorgeben und die offiziellen Clients durch die starke Verbreitung am häufigsten benutzt werden.

Das Anbinden von APIs ist zunächst zeitaufwendig und konnte vermieden werden, da es in der offiziellen Desktopanwendung von *Telegram* seit Version 2.1.10 (vom 05.06.2020) die Möglichkeit gibt, Chats direkt in einem menschenlesbaren Format (HTML) oder maschinenlesbaren JSON-Format herunterzuladen[8]. Diese Arbeit beschränkt sich auf das JSON-Format, da dieses für die automatische Verarbeitung besser als HTML geeignet ist. Falls man an die Problematik **Privacy Restrictions**, vgl. Abs. 2.3.2, denkt, stellt man fest, dass diese auch hier zum Tragen kommt: Der Chat wird aus Sicht des jeweiligen Nutzers heruntergeladen. Abbildung 3.2 zeigt, wie man einen Chat im JSON-Format herunterlädt.

3.1.2 Aufbau der heruntergeladenen Chats

```
1 find ~/Downloads/Telegram\ Desktop/
2
3 /[...]/result.json
4 /[...]/files/2_5359602822964840494(6).pdf
5 /[...]/photos/photo_167@01-05-2020_16-43-23.jpg
6 /[...]/stickers/AnimatedSticker (1).tgs
7 /[...]/video_files/video_5@03-05-2020_17-02-10.mp4
8 /[...]/voice_messages/audio_81@30-05-2020_09-30-44.ogg
```

Auflistung 3.1 Gekürzte Datei-Übersicht nach dem Download eines Telegram-Chats.

6 siehe `https://github.com/UnigramDev/Unigram`
7 siehe `https://github.com/danog/MadelineProto`
8 siehe `https://desktop.telegram.org/changelog`

```
{
        "name": "Telegram",
        "type": "personal_chat",
        "id": 4295744296,
        "messages": [
        {
                "id": 184,
                "type": "message",
                "date": "2019-10-02T13:04:08",
                "from": "Telegram",
                "from_id": 4295744296,
                "text": [
                {
                        "type": "bold",
                        "text": "Anmeldecode:"
                },
                " 10795. ",
                {
                        "type": "bold",
                        "text": "Auf keinen Fall"
                },
                " diesen Code anderen geben, selbst wenn sie
                   behaupten zum Telegram-Team zu gehören!\n\
                   nDieser Code kann dazu benutzt werden, um sich
                   mit deinem Konto zu verbinden. [...], so kannst
                   du diese Nachricht einfach ignorieren."
                ]
        },

[...]
```

Auflistung 3.2 Anfang der Chat-Export-Datei im JSON-Format.

Telegram lädt die Chats in ein lokales Verzeichnis herunter, vgl. Abs. 3.1.1. Wie in Auflistung 3.1 ersichtlich, legt die Desktopanwendung verschiedene Ordner mit verschiedenen Dateien, abhängig vom Chat und Inhalt, an. Die Nachrichten und Meta-Informationen über den Chat werden auf die Datei `result.json` abgebildet.

In Auflistung 3.2 wird der Anfang der Datei `result.json` gezeigt. Die Datei bildet eine JSON-Struktur ab. Innerhalb dieser Datenstruktur befinden sich Meta-Informationen wie Titel des Chats und die Identifikationsnummer (ID), aber auch Chat-Historie; die Nachrichten des Chats befinden sich also in dieser Datei. Erfreulicherweise werden auch Inhalte wie Dateigröße oder Art des Inhaltes angeführt, die durch einen Filter, vgl. Abb. 3.2, herausgefiltert worden sind.

Die offizielle *Telegram* APIs Dokumentation über Nachrichten[9] [10] beschreibt nicht ansatzweise vollständig, welche Attribute eine Nachricht besitzen kann und beschreibt diese Attribute für das Vorgehen dieser Arbeit nicht ausreichend tief. Daher werden diese Attribute für diese Arbeit dokumentiert. Die Dokumentation im Anhang ist nicht vollständig und in Anlehnung an die offizielle Dokumentation entstanden.

In dieser Arbeit wird unterschieden. Zum einen gibt es Attribute, die direkt Aufschlüsse über Autoren zulassen, die als **Autor-spezifische Attribute**, vgl. Abs. A.1.1, bezeichnet werden, wie `from` und `forwarded_from`. Erwähnenswert hierbei ist, dass die IDs nicht immer mit dem Wert übereinstimmen. Beispielsweise könnte sich `from_id` einmal auf den Autor einer Nachricht oder auf den Herkunftschat zeigen. In dieser Arbeit muss daher immer direkt mit den Werten, in diesem Beispiel also `from` gearbeitet werden, worauf separat im Abs. 4.3.2, eingegangen wird.

Zum anderen sind da **Nachrichten-spezifische Attribute**, vgl. A.1.2, die im Gegensatz dazu nur Attribute sind, die direkt einer Nachricht zuordenbar sind, wie `text` oder `message_id`. Diese können über Umwege aber ebenfalls dazu genutzt werden, einen Autor zu identifizieren, vgl. Abs. 4.5. Im Rahmen dieser Arbeit konnte nur mit einer kleinen Teilmenge der Attribute gearbeitet werden, wie noch in Kapitel 5 beschrieben wird.

3.1.3 Das Attribut Text im Detail

In *Telegram* ist es möglich, innerhalb von Text-Chat-Nachrichten Formatierungen wie **fett**, *kursiv* oder unterstrichen zu verwenden. Falls man eine URL wie `https://telegram.org/` eintippt, werden diese besonders hervorgehoben. Das gleiche gilt für `#hashtags`. In den Daten wird das ersichtlich, falls das Attribut `text`, ein JSON-Objekt abbildet und keinen einfachen String wie `Hallo`, vgl. Auflistung. 3.2.

Es gibt zahlreiche besondere Attribute, die in *Telegram* verwendet werden können, aber aus Gründen der Übersicht hier nicht aufgezählt werden, ein davon Teil wird im Anhang beschrieben: A.1.3. Diese Attribute lassen sich auch als *Nachrichten spezifische Attribute* gesehen werden.

Es ist in *Telegram* nicht möglich, diese Formatierungen zu kombinieren, weshalb es zum Beispiel nicht möglich ist, die Formatierung **fett** zusammen mit einer URL-Adresse zu verwenden. Daher dürfen diese URLs nicht nur aus diesen Formatierungen bzw. Attributen genommen werden, sondern müssen aus dem Text geparst und evtl. kombiniert werden, vgl. Abs. 4.2.4.

3.2 Arbeitswerkzeuge/-Umgebung (Stack)

Im Abschnitt 1.2 wurde beschrieben, dass sich diese Arbeit an Data Scientists und an Social Media Mining und Text Mining interessierte Personen richtet, die an diesem Projekt partizipieren möchten. Daher wird in diesem Abschnitt der verwendete Technologie-Stack beschrieben, der auch in anderen Anwendungsfällen (evtl. modifiziert) anwendbar ist, vgl.

9 siehe `https://core.telegram.org/bots/api#message`
10 siehe `https://core.telegram.org/constructor/message`

Abs. 2.3.1.

Ein **Jupyter Notebook** ist eine interaktive Umgebung, um z.B. Python-Code im Browser laufen zu lassen und wird häufig von Data Scientists verwendet, um explorativ Daten zu analysieren [Mül16, S. 7]. Die Jupyter Notebooks unterstützen als Programmiersprache nicht nur Python, sondern auch Sprachen wie Scala[11] und viele weitere.

Jupyter Notebooks können nicht nur im Browser laufen, sondern auch als Client/Server-Anwendung betrachtet werden. So wurde in dieser Arbeit häufig auf die Entwicklungsumgebung **Visual Studio Code**[12] mit der **Jupyter-Erweiterung**[13] zurückgriffen, um übersichtlicher und effizienter mit den (großen) Jupyter Notebooks zu arbeiten. Die Entwicklungsumgebung hat sich hierbei mit einem externen Python-Kernel[14] verbunden. Ein **Kernel** ist in diesem Kontext ein Back-End, also zum Beispiel ein Server mit Laufzeitumgebung(en) und mit Zugriff auf die zu verarbeitenden Daten, der den Code aus Jupyter Notebooks Zellen ausführt.

Um den Kernel mit Abhängigkeiten wie Bibliotheken einfach zur Verfügung stellen und flexibler arbeiten zu können, wurde auf die Containerisierung-Lösung **Docker**[15] gesetzt. So können die Jupyter Notebooks nicht nur zum Entwickeln lokal in *Visual Studio Code* ausgeführt, sondern auch auf Servern automatisch verarbeitet werden. Dafür wurde **Papermill**[16] verwendet, ein Werkzeug um (parametrisierte) Jupyter Notebooks unabhängig vom einem Front-End wie einem Browser automatisch von der Shell auszuführen. Mehr dazu im Abs. 3.3.2.

3.2.1 Komponenten aus dem Docker Image

Als grundlegendes *Docker* Image kam das `datascience-notebook`-Image von dem offiziellen *GitHub*-Repository von *Jupyter* für vorbereitete *Docker*-Images[17] zum Einsatz (Stand 21.12.2020). Dieses beinhaltet unter anderem eine Laufzeitumgebung für Python und für diesen Einsatzfall geeignete Bibliotheken. Das Image ist z.B. bei x64-Architekturen lauffähig, jedoch nicht bei ARM-Architekturen.

Das vorbereitete Image bringt bereits einige Komponenten mit, die für diese Arbeit verwendet werden können. Die nachinstallierten Komponenten werden im darauffolgenden Abschnitt separat aufgelistet.

- **Pandas**: Eine Bibliothek mit verschiedenen Werkzeugen, um Daten zu verarbeiten und zu analysieren. Ein zentrales Element dieser Bibliothek stellt die eigene Datenstruktur, bezeichnet als *DataFrame*, dar. [Mül16, S. 10] [Har16, S. 126].

11 siehe `https://almond.sh/`
12 siehe `https://code.visualstudio.com/`
13 siehe `https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter`
14 siehe `https://jupyter-client.readthedocs.io/en/stable/kernels.html`
15 siehe `https://www.docker.com/`
16 siehe `https://papermill.readthedocs.io`
17 siehe `https://github.com/jupyter/docker-stacks`

- **NumPy**: Ist eines der fundamentalen Bibliotheken für wissenschaftliches Arbeiten mit Computern und beinhaltet beispielsweise mathematische Funktionen und Werkzeuge für das Arbeiten mit Matrizen [Mül16, S. 7] [Har16, S. 112].

- **Matplotlib**: Ist die "[...] primary scientific plotting library in Python." [Mül16, S. 9]. Mit dieser Aussage ist wahrscheinlich gemeint, dass *matplotlib* die relevanteste Zeichen- bzw. Plotter- Bibliothek für die wissenschaftliche Arbeit ist. Diese Aussage kann man nur mit dem Wissen treffen, dass viele "modernere" Plot-Bibliotheken wie **Seaborn**[18] auf *matplotlib* aufsetzen. *Seaborn* wird in dieser Arbeit verwendet, um vereinfacht mit *matplotlib* zu interagieren.

- **Scikit-learn**: Ist eine Open Source (BSD License) Implementierung vieler *ML*-Algorithmen, wie Modelle zur Klassifikation, Regression oder Clustering [Mül16, S. 5] [Har16, S. 163].

- **NetworkX**: Ist eine Python-Bibliothek für das Arbeiten wie z.B. Manipulieren von Strukturen, z.B. Graphen [Bir09, S. XIV]. Die Bibliothek enthält außerdem viele Standard-Algorithmen für das Arbeiten mit Graphen, wie z.B. den *A* Algorithmus*. Es eignet sich auch um mit Daten-Strukturen aus sozialen Netzwerken zu arbeiten [Har16, S. 154], falls diese auf einen Graphen abgebildet werden.

Hinweis: Diese Übersicht ist nicht vollständig, da dies nicht zum Verständnis beitragen würde. Vernachlässigt wurden Komponenten, die in der Praxis "triviale" Probleme lösen, wie das Abfragen der aktuellen Uhrzeit (`import time`), oder Bibliotheken die z.B. simple Zeichenersetzung (`import re`) implementieren. Auch Netzwerk-Bibliotheken (`import network`) oder OS-Bibliotheken wie (`import os`) werden vernachlässigt.

3.2.2 Nachinstallierte Komponenten

Ein Teil der verwendeten Komponenten konnte direkt in den gebauten Docker-Container vom `datascience-notebook`-Image über den Paketmanager **Conda**[19] installiert werden. Dafür musste das `Dockerfile` modifiziert werden, vgl. Abs. 3.3.2.

Leider konnten nicht alle Komponenten über *Conda* installiert werden, da es diesen schlicht nicht in den Standard-Paket-Quellen gab. Dafür konnten die Komponenten innerhalb der Jupyter Notebooks via dem alternativen Paketmanager **Pip**[20] installiert werden. In der folgenden Aufzählungen wurden alle Komponenten mit *Conda* installiert, falls nicht anders angegeben:

- **Demoji**[21]: Ist eine Bibliothek, um mit Python vereinfacht mit Emojis (z.B. `<3` - ASCII Darstellung gewählt) arbeiten zu können. Die Implementierung ermöglicht es beispielsweise, alle *Emoticons* aus einer Zeichenkette zu filtern oder diese auf einen Text, zB. auf `:heart`, abzubilden. Das Paket war in *Conda* nicht verfügbar und wurde deshalb direkt mit *Pip* installiert.

18 siehe `https://seaborn.pydata.org/`
19 siehe `https://docs.conda.io`
20 siehe `texthttps://pypi.org/project/pip/`
21 siehe `https://pypi.org/project/demoji/`

- **Gensim**[22]: Ist eine Python-Bibliothek, um unter anderem mit *Topic Models*, arbeiten zu können. Die Bibliothek umfasst beispielsweise eine Implementierung der *Latent Dirichlet Allocation (LDA)* [Har16, S. 91], siehe Abs. 4.4.7.

- **PyLDAvis**[23]: Ist eine Python-Bibliothek, um interaktiv (z.B. im Browser) mit Topic Modellen aus z.B. *Gensim* arbeiten zu können. Das Projekt bezeichnet sich selbst als Portierung der R-Bibliothek *LDAvis*[24].

- **NLTK**[25]: Das *Natural Language Toolkit (NLTK)* ist eine Sammlung von Text Mining Werkzeugen für Python [Agg18, S. 16], die zum Beispiel Implementierungen für das Erkennen von Funktionen einzelner Wörter im Satz (z.B. Subjekt, Verb, Objekt, etc.) enthält.

- **Wordcloud**[26]: Eine Python-Bibliothek, um *Word Clouds*, siehe Abschnitt 4.4.2, zu generieren. *Wordcloud* basiert auf *NumPy* und *Pillow*[27].

- **PyTorch**[28]: Ist eine *ML* Bibliothek für Python, entwickelt von Facebook. *PyTorch* setzt sich aus mehreren Bibliotheken zusammen[29] und beinhaltet unter anderem Funktionen, um neuronale Netzwerke zu trainieren und auf ein Produktivsystem auszuliefern.

- **Transformers**[30]: Die sogenannten *Transformers* von der Firma *Hugging Face*[31] verbinden Text Mining Methoden mit *PyTorch* oder *TensorFlow*[32]. Darauf wird separat in Abs. 3.2.3 eingegangen.

- **HanTa**[33]: Der *Hanover Tagger (HanTa)* ist ein "Versuch", mit heuristischen Verfahren und Markov Ketten, *POS-Tagging* und *Lemmatization*-Aufgaben für die deutsche Sprache zu lösen [War19], vgl. Abs. 4.2.1. Das Paket war in *Conda* nicht verfügbar, und wurde deshalb direkt mit *Pip* installiert.

- **Graphviz**[34]: Ist eine Open Source Visualisierungs-Software. Es unterstützt die Generierung von Bildern aus strukturierten Daten [Bir09, S. XIV], wie in dieser Arbeit Daten aus *NetworkX* zu Vektorgraphiken, siehe Abschnitt 4.3.3.

- **TextBlob**[35]: Implementiert ähnlich wie *NLTK* verschiedene Text Mining Funktionen. In dieser Arbeit werden beispielsweise Funktionen aus der *Sentiment analysis* [Har16, S. 554] verwendet, vgl. Abs. 4.4.6.

22 siehe `https://pypi.org/project/gensim/`
23 siehe `https://github.com/bmabey/pyLDAvis`
24 siehe `https://github.com/cpsievert/LDAvis`
25 siehe `https://www.nltk.org/`
26 siehe `https://pypi.org/project/wordcloud/`
27 siehe `https://pillow.readthedocs.io/en/stable/`
28 siehe `https://pytorch.org/`
29 siehe `https://pytorch.org/ecosystem/`
30 siehe `https://pypi.org/project/transformers/`
31 siehe `https://huggingface.co/`
32 siehe `https://www.tensorflow.org/`
33 siehe `https://github.com/wartaal/HanTa`
34 siehe `https://graphviz.org/`
35 siehe `https://textblob.readthedocs.io/`

3.2.3 Transformers

Transformers, auch bekannt als **pytorch-transformers**, stellen eine generische Architektur für Sprachmodelle wie *BERT*[36] von *Google* (2018) und *RoBERTa*[37] von *Facebook* (2019) für die Analyse von Text für Python bereit [Hug21]. *RoBERTa* basiert auf *BERT*. Neben verschiedenen Text Mining Methoden, wie die *NER* oder die *Sentiment analysis*, stehen auch Sprachmodelle bereit, die Text generieren können.

Vereinfacht gesagt ist es mithilfe von *Transformers* von der Firma *Hugging Face* möglich, diverse Sprachmodelle inklusive Datenbanken herunterzuladen und auf Text, in diesem Fall auf Text-Nachrichten anzuwenden. Die bereitgestellten Sprachmodelle stammen von verschiedenen Personen und Institutionen.

In dieser Arbeit wurden verschiedene Sprachmodelle benutzt, die hier nur sehr oberflächlich erklärt werden können, in den Fußnoten befinden sich dazu aber weitere Beiträge:

- **bert_de_ner**[38]: Ist ein Sprachmodell basierend auf *BERT*, das für die Durchführung einer *NER* (Entitäten-Erkennung), vgl. Abs. 2.4.7, für die deutsche Sprache trainiert worden ist.

- **xlm-roberta-large-finetuned-conll03-german**[39]: Wie *bert_de_ner* stellt dieses Sprachmodell eine Entitäten-Erkennung bereit. Dieses Modell basiert im Gegensatz zu dem vorherigen Modell auf dem erweiterten Ansatz *XLM-RoBERTa*[40] (eine *RoBERTa* Abwandlung), das neben der deutschen viele Sprachen unterstützt, was in diesem Kontext erfreulich ist, weil die Nachrichten von deutschen Autoren stammen, die sich manchmal an der englischen Sprache bedienen.

- **bert-base-multilingual-uncased-sentiment**[41]: Ist ein Sprachmodell basierend auf *BERT*, das für die Durchführung einer *Sentiment analysis* für die Sprachen, Englisch, Deutsch, Französisch, Niederländisch, Spanisch und Italienisch trainiert worden ist. Das besondere an diesem Modell ist, dass dieses ursprünglich an Produktbewertungen trainiert worden ist, weshalb es einen *Score* zwischen $[1;5]$ vergibt, was trotzdem im Rahmen dieser Arbeit an Chat-Nachrichten ausprobiert wird.

- **german-gpt2**[42]: Ist ein Sprachmodell, das Text generieren kann. Das Modell basiert auf dem Sprachmodell *BERT* und wird von der bayrischen Staatsbibliothek bereitgestellt. Es gehört zu der Klasse der *OpenAI GPT2*[43] Sprachmodellen.

- **german-gpt2-faust**[44]: Ist eine Abwandlung des Sprachmodells *german-gpt2*, das Text generieren kann, der klingt als hätte die deutsche Sagenfigur Faust diesen gesprochen und wird ebenfalls von der bayrischen Staatsbibliothek bereitgestellt.

36 siehe `https://github.com/google-research/bert`
37 siehe (gekürzte Adresse) `https://bit.ly/310641z`
38 siehe `https://huggingface.co/fhswf/bert_de_ner`
39 siehe `https://huggingface.co/xlm-roberta-large-finetuned-conll03-german`
40 siehe `https://huggingface.co/transformers/model_doc/xlmroberta.html`
41 siehe `https://huggingface.co/nlptown/bert-base-multilingual-uncased-sentiment`
42 siehe `https://huggingface.co/dbmdz/german-gpt2`
43 siehe `https://huggingface.co/transformers/model_doc/gpt2.html`
44 siehe `https://huggingface.co/dbmdz/german-gpt2-faust`

3.3 Telegram Analyse Plattform

Im Rahmen dieser Arbeit wurde eine Plattform namens **Telegram Analysis**[45] [Bun21c] geschaffen, die es ermöglicht, dass Personen wie Data Scientists und Sozialwissenschaftler strukturiert Telegram Chats inklusive Verbindungen auswerten können. Langfristig soll so ermöglicht werden, die definierte Zielsetzung zu erfüllen.

Neben zwei Jupyter-Notebooks mit implementierten Social Media und Text Mining Werkzeugen für das praktische Arbeiten mit diesen, vgl. Abs. 3.3.2, stellt die Plattform eine Webseite bereit, die die aufbereiteten Auswertungen aus den Notebooks zugänglich macht [Bun21c]. Diese Webseite richtet sich an Personen, die kein tiefes Informatikwissen vorweisen können, die aber die Auswertungen mit einer anderen Perspektive betrachten und evtl. interpretieren und evaluieren können, vgl. Kapitel 5.

Um diese Arbeit möglichst optimal mit den Jupyter-Notebooks vergleichen zu können, wird auf die verwendete Zitierweise zurückgegriffen [Bun21b] [Bun21a]. Über *GitHub* können die Notebooks direkt im Browser ohne weitere Installationen angesehen werden.

3.3.1 Ausgewertete Chats und Fragen

Wie bereits in Abschnitt 3 beschrieben, war es im Rahmen der Arbeit nicht möglich letztgültig zu definieren, welche Chats zu betrachten sind, um die Zielsetzung der Arbeit erfüllen zu können. Um trotzdem Social Media bzw. Text Mining Methoden ausprobieren zu können, wurden initial Annahmen getroffen, die in diesem Absatz weiter beschrieben werden.

Chats

Es war möglich, mit Suchmaschinen wie *Google*[46] oder direkt Telegram-Suchmaschinen wie *Telegram Analytics*[47] einen Teil der in den Artikeln häufig zitierten Chat-Gruppen und Personen im sozialen Netzwerk *Telegram* zu finden. Diese Datensätze dienen zunächst als Ausgangsbasis für die Plattform. Eine Vollständigkeit kann nicht gewährleistet werden, vgl. *Suchmuster für Chats* in Abs. 4.3.2.

Durch die Datenmenge war es notwendig, diese Chats in **DataSets** (engl. "Datensätze") zu unterteilen. Diese *DataSets* werden im Anhang beschrieben, vgl. A.2, und lassen sich kurz zusammenfassen:

- `DataSet0`: *Oliver Janich*, *Attila Hildmann*, *Eva Herman* und *Xavier Naidoo* stellten eine Ausgangbasis dieser Arbeit dar. Es wurden zunächst ausschließlich `channels` betrachtet. In diesem Datensatz befinden sich also ausschließlich *Akteure*.
- `DataSet1` und `DataSet1a`: Zunächst wurde `DataSet0` ausgewertet. Mit einem Suchmuster, wurden verschiedene Gruppen identifiziert, anschließend heruntergeladen und auf `DataSet1` abgebildet. Aus zeitlichen Gründen ist es nicht möglich, das vollständige `DataSet1` zu betrachten, weshalb in der Arbeit nur die `supergroups` betrachtet wurden. Dieses Subset wird hier als `DataSet1a` bezeichnet.

45 siehe `https://maxbundscherer.github.io/telegram-analysis/`
46 siehe `https://google.de`
47 siehe `https://tgstat.com/`

- `DataSet2`: Umfasst Chat-Gruppen der Firma bzw. Bewegung *Querdenken*. [48] [49] In diesem Datensatz befinden sich also *Interessensgruppen* und stellen auch die Ausgangsbasis dieser Arbeit dar.

Fragen

Wie ebenfalls bereits in Abs. 3 beschrieben, war die Zielsetzung zunächst nicht konkret genug für einen Social Media Mining Prozess. Daher wurden zunächst konkrete Fragen erarbeitet, die im Laufe der Arbeit erweitert wurden. Es wurde versucht, diese zu beantworten. Diese Fragen dienen zunächst als Ausgangsbasis für die Plattform. Eine Vollständigkeit kann nicht gewährleistet werden und die Fragen können in dieser Arbeit nicht beantwortetet werden, da eine Evaluation und Interpretation der Auswertungen im Rahmen dieser Arbeit nicht möglich war, vgl. Kapitel 5.

- **Gibt es nicht vielleicht wenige *Akteure*, die große Gruppen "lenken"?**: Mit Meta-Attributen von Nachrichten und Eigenheiten wie Schreibstil, d.h. verwendete Formatierungen, Rechtschreibfehler und verwendete *Emoticons*-Kombinationen, lassen sich die Nachrichten der Autoren voneinander unterscheiden. Es wird vermutet, dass es einflussreiche *Akteure* und *Interessengruppen* in diesem Netzwerk gibt.

- **Worüber wird in diesen Chat-Gruppen diskutiert?**: Mit Text Mining Methoden wird versucht, die Chat-Text-Nachrichten auszuwerten. Mit diesen Werkzeugen können zwar Text-Nachrichten ausgewertet werden, aber im Rahmen dieser Arbeit können diese Auswertungen nicht evaluiert und interpretiert werden.

- **Wie beeinflussen sich die Gruppen gegenseitig?**: Es wird vermutet, dass diese Gruppen sich gegenseitig beeinflussen. Um diese Frage beantworten zu können, gilt es zunächst zu definieren, worüber in den Chats diskutiert wird. Anschließend können verschiedene Chats miteinander zeitlich verglichen werden.

- **Gibt es zeitlich korrelierende Effekte, z.B. Zeitungen schreiben über ein Thema, die Gruppen schreiben über dieses Thema (und verzerren es evtl.)?**: Vermutet wird, dass Zeitungsartikel und politische Ereignisse die Gruppen beeinflussen. Um diese Vermutung zu untersuchen, können beispielsweise *Sitemaps*[50] von Zeitungen oder Twitter-Tweets von Nachrichten-Portalen herangezogen werden.

- **Wieviel Prozent der Gruppen ist aktiv oder passiv?**: Die Abonnenten-Zahl eines Chats sind wenig aussagekräftig, da vermutet wird, dass ein (großer) Teil der Mitglieder passiv ist, d.h. die Mitglieder sind nicht aktiv an der Diskussion beteiligt.

- **Handelt es sich bei den Nachrichten eher um Aussagen, Fragen, Antworten oder Aufrufe?** Für diese Frage kann diese Arbeit keine Lösung liefern, es würde sich aber evtl. lohnen, sich weiter mit den *Transformers*, vgl. Abs. 3.2.3, auseinanderzusetzen.

- **Aus welchen Quellen kommen die Inhalte bzw. wie werden diese belegt?** Dafür könnten zitierte Webseiten und andere *Telegram*-Teilnehmer von Relevanz sein.

48 siehe `https://querschenken.company/`
49 siehe `https://netzpolitik.org/2020/querdenken-der-geschaeftige-herr-ballweg/`
50 siehe `https://de.wikipedia.org/wiki/Sitemap`

- **Welche Stilmittel werden in den Gruppen häufig verwendet?** Das hängt davon ab, wie man Stilmittel definiert; so können bereits Formatierungen als Stilmittel betrachtet werden.

- **Worum geht es in den vielen geteilten Videos?** In den Daten wurde festgestellt, dass häufig auf YouTube-Videos verwiesen wird. Im Rahmen der Arbeit konnten nur Video-Titel aufgelöst werden, vgl. *dynamisches Auflösen* Abs. 4.3.2.

3.3.2 Aufbau und Einstieg

Dieser Abschnitt widmet sich vor allem an Personen wie Data Scientists, die an dieser Arbeit direkt partizipieren möchten und eher weniger an Personen, die sich gerne die Auswertungen ansehen und diese evtl. evaluieren bzw. interpretieren möchten. Für Personen mit nicht-technischem Hintergrund bietet sich eher die Webseite [Bun21c] mit den aufbereiteten Auswertungen an, vgl. Abs. 3.3.

Bevor sich mit der Plattform auseinandergesetzt werden kann, wird empfohlen, sich mit den Grundlagen von Social Media und Text Mining zu beschäftigen, vgl. Kapitel 2, sowie sich mit *Telegram*, vgl. Abs. 3.1, und dem empfohlenen Technologie-Stack, vgl. Abs. 3.2, auseinanderzusetzen. Initial wurden Annahmen über die *Akteure* und *Interessengruppen* getroffen und Forschungsfragen definiert, vgl. Abs. 3.3.1.

Abbildung der Plattform auf GitHub

Als Hoster der Plattform wurde *GitHub*[51] benutzt. Für die Plattform wurde ein eigenes Repository[52] angelegt.

Ziel ist es zum einen, die Chat Auswertungen aufbereitet darzustellen, wofür auf die Technologie **GitHub-Pages** gesetzt wurde, mit der ein normales *GitHub*-Repository schnell eine Webseite bereitstellen kann, vgl. Abs. 3.3 [Bun21c].

Neben der Webseite stellt das GitHub-Repository zwei Jupyter-Notebooks bereit, die im Laufe dieser Arbeit dokumentiert werden bzw. werden die wichtigsten Ideen und Auswertungen zusammengefasst, vgl. Kapitel. 4.

Mit diesen zwei Notebooks soll weitergearbeitet werden. Davor sollte sich aber erst einmal mit dem grundlegenden Aufbau dieses Repositorys und damit auch der Plattform beschäftigt werden, weshalb hier kurz der Aufbau beschrieben wird:

- `docker/`: In diesem Ordner befinden sich unter anderem das modifizierte `Dockerfile` und eine Anleitung `README.md`, die erklärt, wie man das Docker Image zum laufen bringt (*Docker Container*).

- `docs/`: In diesem Ordner befinden sich der Quellcode und ein Teil der bereitgestellten Grafiken der Projekt-Webseite (*GitHub-Pages*) [Bun21c].

51 siehe `https://github.com/`
52 siehe `https://github.com/maxbundscherer/telegram-analysis`

- `notebooks/`: In diesem Ordner befinden sich unter anderem die zwei zu dieser Arbeit gehörenden Jupyter-Notebooks `Telegram.ipynb` [Bun21b] und `Classifier.ipynb` [Bun21a]. Diese Notebooks werden im Kapitel 4 detailliert mit ihren grundlegenden Ideen beschrieben. Außerdem werden über die Datei `inputFiles.csv` die zu verarbeitenden Chats definiert. Über die Datei `additionalStopwords.txt` ist es möglich, eigene *Stopwords* zu definieren, welche später mit *Stopwords* aus *NLTK* (deutsch und englisch) kombiniert werden.
- `notebooks/cache/`: Hier werden die verschiedenen Cache-Files gespeichert, die dafür sorgen, dass bestimmte Schritte nicht jedes mal neu berechnet werden. Mit anderen Worten werden hier die in der Arbeit erzeugten *DataFrames* auf `Pickle`-Dateien[53] abgebildet.
- `notebooks/data`: Hier werden die heruntergeladen *Telegram* Chats abgelegt. Die Struktur wird dabei vom *Telegram* Client für Desktop vorgegeben, vgl. Abs. 3.1.2.
- `notebooks/output`: Hier werden die vom Jupyter-Notebook generierten Auswertungen abgelegt, zum Beispiel Vektorgrafiken oder HTML-Dateien.

Ausführen der Notebooks

Für die lokale Entwicklung empfiehlt es sich, mit *Visual Studio Code* und dem modifizierten *Docker*-Image, vgl. Abs. 3.2, zu arbeiten. Dafür ist es zunächst erforderlich, das Repository auf den lokalen Rechner zu klonen (`git clone`) und anschließend die Schritte in der Docker-Beschreibung (`docker/README.md`) zu befolgen. Die zu verarbeitenden Chats und zusätzlichen *Stopwords* werden in den Dateien `inputFiles.csv` und `additionalStopwords.txt` konfiguriert, wie bereits zuvor beschrieben. Gearbeitet werden können mit den Notebooks `Telegram.ipynb` [Bun21b] und `Classifier.ipynb` [Bun21a], welche ausführlich mit Ihren Ideen im Kapitel 4 beschrieben werden.

Da die implementieren Methoden einige Rechenzeit in Anspruch nehmen und eine vollständige Ausführung unter Umständen, abhängig von der Größe der zu betrachtenden Chats, sehr lange dauern kann, gibt es die Möglichkeit für die lokale Entwicklung auf Stichproben zu arbeiten, vgl. Abs. 4.1.1. Eine vollständige (evtl. sogar automatische) Ausführung wird daher auf einem Server empfohlen. Dafür empfiehlt sich der Einsatz von *Papermill*, vgl. Abs. 3.2. Dafür wurden zwei Skripte, vgl. Anhang A.3.2 und A.3.1, angehängt.

53 siehe `https://pandas.pydata.org/docs/reference/api/pandas.read_pickle.html`

4 Data Preparation & Modeling

Im Rahmen dieser Arbeit wurden zwei Jupyter-Notebooks implementiert, die auch auf der *Telegram Analyse Plattform* zur Verfügung gestellt werden [Bun21b] [Bun21a]. Die Auswertungen dieser Notebooks werden auf der Webseite bereitgestellt [Bun21c]. Die Plattform wurde in Abschnitt 3.3 beschrieben.

In Abs. 2 wurde eine Übersicht über die Felder Social Media und Text Mining ermöglicht. Diese Notebooks implementieren Methoden aus diesen Feldern und wenden diese auf die heruntergeladenen Chats an, vgl. 3.3.1. Forschungsfragen wurden in diesem Abschnitt ebenfalls definiert.

Dieser Abschnitt beschreibt das in dieser Arbeit angewendete Vorgehen und greift dabei unterstützend auf die zwei Jupyter Notebooks [Bun21b] und [Bun21a] zurück. In dieser Arbeit wird immer direkt auf die ausgeführten *Zellen* in den Jupyter-Notebooks verwiesen, wofür die bisher verwendete Zitierweise (z.B. [Bun21b, 0]) genutzt wird. Es empfiehlt sich, neben dieser Arbeit gleichzeitig die zitierten Jupyter-Notebooks zu vergleichen. Die Überschriften aus dieser Arbeit findet man in den Notebooks wieder, um eine Vergleichbarkeit zu ermöglichen.

In dieser Arbeit werden die wichtigsten Schritte, Ideen und Erkenntnisse aus den Notebooks beschrieben. Generell lassen sich die Implementierungen in den Notebooks auf fünf Stufen abbilden, die analog zur Reihenfolge der Jupyter-Notebooks beschrieben werden:

1. **Arbeitsumgebung initialisieren** (Abs. 4.1): In dieser Stufe wird die Arbeitsumgebung initialisiert, d.h. es werden unter anderem die Lauf-Konfigurationen angewendet, Bibliotheken dem Kernel zugänglich gemacht und weitere Abhängigkeiten wie *Stopwords*-Datenbanken heruntergeladen. In den beiden Jupyter-Notebooks ist dieser Vorgang fast identisch, daher wird in dieser Arbeit das umfassendere Vorgehen aus dem Notebook `Telegram.ipynb` beschrieben [Bun21b, 1-16].

2. **Chats laden und aufbereiten** (Abs. 4.2): In dieser Stufe werden die heruntergeladenen *Telegram* Chats geladen und aufbereitet. In diesem Abschnitt wird zunächst auf Text-Aufbereitungsfunktionen für die deutsche Sprache eingegangen, anschließend werden die heruntergeladenen Chats auf verschiedene *DataFrames*, d.h. auf Features abgebildet [Bun21b, 17-43].

3. **Social Media Mining** (Abs. 4.3): In dieser Stufe werden zunächst die heruntergeladenen Chats auf Meta-Ebene betrachtet und anschließend auf weitere Features abgebildet, die sodann mithilfe von Graphen visualisiert werden. In dieser Stufe werden die Chats auch im zeitlichen Verlauf betrachtet [Bun21b, 44-94].

4. **Text Mining** (Abs. 4.4): In dieser Stufe werden Text Mining Methoden auf die Text-Nachrichten aus Chats angewendet. Zunächst wird sich damit auseinandergesetzt,

welche Nachrichten für diese Methoden überhaupt in Frage kommen. Anschließend werden verschiedene Verfahren auf die Datensätze angewendet und die Auswertungen beschrieben [Bun21b, 95-165].

5. **Identifizierung von Autoren** (Abs. 4.5): In dieser Stufe werden verschiedene *ML*-Modelle für den Zweck der Zuordnung einer Text-Nachricht zu einem Autor trainiert und evaluiert. Diese Stufe ist in einem eigenem Jupyter-Notebook `Classifier.ipynb` [Bun21a] abgebildet und benötigt die bereits aufbereiteten Daten aus Stufe *Daten laden und aufbereiten* in Form einer `Pickle`-Datei.[1]

4.1 Arbeitsumgebung initialisieren

Zuerst muss die Arbeitsumgebung initialisiert werden, d.h. es werden Lauf-Konfigurationen auf die Jupyter-Umgebung angewendet, zusätzliche Bibliotheken installiert und Datenbanken wie *Stopwords*-Datenbanken oder Sprachmodelle von Transformers heruntergeladen und damit den Kernel zugänglich gemacht.

4.1.1 Jupyter Notebook Parameter

Das Jupyter-Notebook, also die Läufe, lassen sich über verschiedene Parameter beeinflussen. Einige dieser Parameter sind technisch notwendig, da diese z.B. die Arbeitsverzeichnisse für die Jupyter-Notebooks definieren, andere hingegen dienen nur dazu, den Lauf für Entwicklungszwecke bzw. Automatisierungszwecke zu beschleunigen. Diese Parameter werden in der Tabelle 4.1 detailliert beschrieben und befinden sich am Anfang des Notebooks [Bun21b, 1]. Das Einlesen der zu verarbeitenden Chats wird im Abs. 4.2.2 beschrieben.

4.1.2 Arbeitsumgebung vorbereiten

Die Schritte im folgenden Absatz werden analog zum Ablauf des Jupyter-Notebooks erläutert. Es empfiehlt sich daher, diesen Abschnitt mit den Notebook [Bun21b, 2-16] zu vergleichen. Die Überschrift aus dieser Arbeit finden sich in den Notebooks wieder.

Bibliotheken und Abhängigkeiten laden

Zunächst werden die Komponenten aus dem Docker Image und technisch notwendige Bibliotheken importiert und damit dem Kernel zugänglich gemacht. Anschließend werden weitere Komponenten über den Paketmanager *Pip* installiert und ebenfalls importiert. Die Komponenten und das Vorgehen werden in Abschnitt 3.2 beschrieben [Bun21b, 2-6].

Stoppuhr bereitstellen

Eine simple Stoppuhr wird definiert, um später verschiedene Durchlaufzeiten für ausgewählte Social Media Mining bzw. Text Mining Methoden zu messen [Bun21b, 7].

1 siehe `https://pandas.pydata.org/docs/reference/api/pandas.read_pickle.html`

Tabelle 4.1 Parameter der Jupyter-Notebooks.

Bezeichner	Beispiel	Beschreibung
C_LOCAL	True	Setze diesen Wert auf "True", falls eine externe Verbindung zum Kernel besteht, zum Beispiel wenn in Visual Studio Code gearbeitet wird. Setze auf "False", falls direkt im Browser gearbeitet wird. Definiert technisch betrachtet die lokalen Arbeitsverzeichnisse.
C_LOAD_DATASETS	set1, set2	Definiert, welche *DataSets* betrachtet werden sollen, vgl. Abs. 3.3.1.
C_SHORT_RUN	False	Setze diesen Wert auf "True", falls ein reduzierter Lauf durchgeführt werden soll. Damit werden bestimmte (Unter-) Stufen übersprungen.
C_NUMBER_SAMPLES	1000	Falls "C_SHORT_RUN" auf "True" gesetzt ist, gültig. Um die Entwicklungszeiten weiter zu verkürzen, kann nur auf einem Teil der Datenmenge operiert werden. Dieser Wert definiert die Anzahl der Nachrichten pro Chat.
C_RESOLVE_NEW_URLS	True	Definiert, ob dynamisches Auflösen von Social Graph Features durchgeführt werden soll, zum Beispiel ob Titel von YouTube-Videos oder `bitly.com`-URLs aufgelöst werden, was ebenfalls sehr lange dauern kann, vgl. Abs. 4.3.2.
C_TIME_PLOT_FREQ	1D	Definiert die Zeitspanne bzw. Intervalle der Plots aus Abs. 4.3.4.
C_LOAD_TRANSFORMERS	True	Definiert, ob die Sprachmodelle (Transformers) geladen werden soll, vgl. 3.2.3. Das Herunterladen kann abhängig von den Sprachmodellen sehr lange dauern, deshalb kann dies übersprungen werden.
C_TRANSFORMERS_DATASETS	set1, set2	Falls "C_LOAD_TRANSFORMERS" auf "True" gesetzt ist, gültig. Definiert, auf welchen Chats die Sprachmodelle (Transformers) angewendet werden.
C_NEW_CACHE_FILE	file.pkl	Dieser Wert soll gesetzt werden, falls ein neues `Pickle`-File, erzeugt werden soll und definiert den Dateinamen. Mit diesem Cache-File ist es möglich, die Datenaufbereitung aus *Stufe 3*, vgl. Abs. 4.2.4, zu überspringen.
C_USE_CACHE_FILE	file.pkl	Dieser Wert soll gesetzt werden, falls ein bestehendes `Pickle`-File verwendet werden soll und definiert ebenfalls den Dateinamen.

Modelle und Datenbanken bereitstellen

Teil dieser Stufe ist auch das Herunterladen und Bereitstellen von Modellen und Datenbanken [Bun21b, 8-11]. Hierbei werden auch die *Transformers*-, *NLTK*- und *Demoji* Arbeitsdateien heruntergeladen. Die lokalen Daten werden immer mit den aktuellen Daten der jeweiligen Bibliotheken abgeglichen. So werden *Transformers* Modelle, vgl. Abs. 3.2.3, immer mit den neuesten Modellen von der Plattform *Hugging Face* abgeglichen. *NLTK* lädt z.B. *Stopwords*-Datenbanken und *Demoji*-Datenbanken *Emoticon*-Datenbanken herunter.

Das Herunterladen von Transformers kann über einen Parameter übersprungen werden, da dieser Vorgang abhängig von den Sprachmodellen sehr lange dauern kann, vgl. Abs. 4.1.1. Die *Stopwords* werden zunächst von *NLTK* für die Sprache Deutsch und Englisch heruntergeladen und mit der Liste `additionalStopwords.txt`, vgl. Abs. 3.3.2, kombiniert und bereitgestellt.

Konfigurationen auf Umgebung anwenden

Abschließend werden noch umgebungsspezifische Konfigurationen wie Ausgabekonfigurationen angewendet, das lokale Arbeitsverzeichnis auf Dateisystemebene definiert und der Cache für das dynamische Auflösen, vgl. Abs. 4.3.2, wird initialisiert, falls nicht bereits vorhanden [Bun21b, 12-16].

4.2 Chats laden und aufbereiten

Die zuvor heruntergeladenen Chats aus *Telegram* müssen erst für anschließende *ML*-Verfahren aufbereitet werden. Die Schritte werden in diesem Abschnitt ebenfalls analog zum Ablauf des Jupyter-Notebooks erläutert [Bun21b, 17-43]. Diese Stufen lassen sich zunächst auf drei weitere Stufen aufteilen:

1. **Chats laden** (Abs. 4.2.2): In dieser Stufe werden die Chats geladen. Die zu ladenden Chats werden über eine `CSV`-Datei definiert. Die Chats hierfür müssen auf *DataSets* abgebildet werden [Bun21b, 22-23].

2. **Chats aufbereiten** (Abs. 4.2.3): In dieser Stufe werden die Chats aufbereitet, d.h. es werden Meta-Informationen aus den Chats extrahiert [Bun21b, 24-26].

3. **Nachrichten aufbereiten** (Abs. 4.2.4): In dieser Stufe werden die Nachrichten der Chats auf *DataFrames*, d.h. auf Features abgebildet [Bun21b, 27-43].

Stichprobenartig wurde überprüft, in welchen Sprachen die Nachrichten vorliegen. Wenig überraschend war, dass die meisten Nachrichten in deutscher Sprache formuliert worden sind. Einige der Nachrichten wurden auch in englischer Sprache formuliert, diese Unterscheidung spielt aber für diese Arbeit keine Rolle, da die meisten implementierten Verfahren Bibliotheken nutzen, die sich primär auf die englische Sprache ausrichten und auch die deutsche Sprache unterstützen. In diesem Kontext sind diese Sprachen also sehr ähnlich und können daher ähnlich auf Features abgebildet werden.

4.2.1 Aufbereitungsfunktionen für die deutsche Sprache

Die deutsche Sprache hat aber Eigenschaften, die für die verwendeten Bibliotheken (Python-Implementierungen), vgl. Abs. 3.2, eher "störend" als nützlich sind. Auf diese muss gesondert eingegangen werden.

Es stellt sich auch die Frage, welche Implementierungen für die Aufbereitungen der (primär) deutschen Sprache geeignet sind, daher werden diese in diesem Abschnitt separat betrachtet.

Hinweis: Für manche Verfahren können diese Implementierungen unvorteilhaft sein, ähnlich der *Stemming-Problematik* beim *POS-Tagging*, vgl. Abs. 2.4.6, und der Problematik bei der *Tokenization*, vgl. Abs. 2.4.1.

Deutsch-spezifische Buchstaben aus einem String ersetzen

```
1 gloReplaceGermanChars("ö ä ü Ö Ä Ü")
2
3 'oe ae ue Oe Ae Ue'
```

Auflistung 4.1 Beispiel: Deutsche Buchstaben aus einer Zeichenkette entfernen.

Umlaute wie `ä` und Buchstaben wie ßstören manche Python-Implementierungen, da diese nicht in vielen Standard-Zeichensätzen wie ASCII enthalten sind. Die Implementierung in dem Notebook reagiert auf diese Problemstellung, indem diese durch zum Beispiel `ae` und `ss` ersetzt werden. Gezeigt in Auflistung 4.1 [Bun21b, 17].

Tokenization über NLTK

```
1 list(getTokenFromText("Hallo Leser! Das ist ein Test."))
2
3 ['Hallo', 'Leser', '!', 'Das', 'ist', 'ein', 'Test', '.']
```

Auflistung 4.2 Beispiel: *NLTK* Standard *Tokenizer* für die deutsche Sprache.

Erfreulicherweise stellt *NLTK* bereits einen für diesen Anwendungsfall geeigneten *Tokenizer* für die deutsche Sprache bereit, vgl. Abs. 2.4.1, und *NLTK* implementiert verschiedene *Tokenizer*. Diese Arbeit geht jedoch nur auf die Standard-Implementierung ein. Ein Beispiel dafür findet man in Auflistung 4.2 [Bun21b, 18].

Lemmatization & POS-Tagging über HanTa

```
sampleText = "Sie lesen gerade einen kurzen Beispielsatz!"
nltk.pos_tag(getTokenFromText(sampleText))

[('Sie', 'NNP'),
('lesen', 'PRP'),
('gerade', 'VBD'),
('einen', 'JJ'),
('kurzen', 'JJ'),
('Beispielsatz', 'NN'),
('!', '.')]
```

Auflistung 4.3 Beispiel: *NLTK* englisches *POS-Tagging* an einem deutschen Satz.

NLTK implementiert kein deutsches *POS-Tagging*, vgl. Abs. 2.4.6. Trotzdem wurde versucht, das englische *POS-Tagging* von *NLTK* an einem Beispielsatz auszuprobieren, nach der *Tokenization* durch den deutschen *NLTK-Tokenizer* [Bun21b, 19-20].

In der Auflistung 4.3 wird ersichtlich, dass das eher schlecht als recht funktioniert: Die verschiedenen *Token* werden jeweils einer Klasse zugeordnet. In dem Beispiel sind folgende Klassen vertreten: `NNP` (noun, proper, singular), `PRP` (Personalpronomen - pronoun, personal), `VBD` (Verb past tense), `JJ` (Adjektiv numeral ordinal), `NN` (Noun, singular - Eigenname) und `.` (Satzzeichen) [Bir09, S. 183]. Die meisten *Token* (`Sie`, `lesen`, `gerade`, `einen` und `kurzen`) werden also den falschen Wortarten zugeordnet.

```
getLemmaAndTaggingFromText(sampleText)

[('Sie', 'sie', 'PPER'),
('lesen', 'lesen', 'VVFIN'),
('gerade', 'gerade', 'ADV'),
('einen', 'einen', 'ART'),
('kurzen', 'kurz', 'ADJA'),
('Beispielsatz', 'Beispielsatz', 'NN'),
('!', '--', '$.')]
```

Auflistung 4.4 Beispiel: *Lemmatization* und *POS-Tagging* implementiert durch *HanTa* angewendet an einem deutschen Satz.

Für die Implementierung von *POS-Tagging* und der *Lemmatization*, vgl. Abs. 2.4.5, wurde in dieser Arbeit daher auf *HanTa*, vgl. Abs. 3.2.2, zurückgegriffen. Die *Tokenization* erfolgt hierbei weiterhin über *NLTK* [Bun21b, 21].

In der Auflistung 4.4 wird ersichtlich, dass das besser funktioniert: Die verschiedenen *Token* werden jeweils einer Klasse zugeordnet. In dem Beispiel sind folgende Klassen vertreten: `PPER` ((nicht-reflexives) Personalpronomen), `VVFIN` (finites Vollverb), `ADV` (Adverb),

ART (Artikel), ADJA (attributives Adjektiv) und NN (normales Nomen). *HanTa* weist die Klassen nach dem *TIGER Morphologie Annotationsschema* [War19, S. 44] zu, einer Erweiterung des *Stuttgart-Tübingen-Tagset (STTS)* [Cry05, S. 4], und bildet ein *Token* immer im Infinitiv ab, also auf die Grundform eines Wortes [War19, S. 42].

4.2.2 Stufe 1: Chats laden

Tabelle 4.2 CSV-Eingabeformat für zu verarbeitenden Chats.

Bezeichner	Beispiel	Beschreibung
inputName	Attila Hildmann	Titel des Chats
inputPath	ChatExport_2021-01-05/	Relativer Pfad zum heruntergeladenen Chat
inputType	public_channel	Art des Chats
inputId	10034163583	ID des Chats
inputLabel	dataSet0	Filter-Label
inputDownloadType	onlyText	Download-Filter

Die Verarbeitungsaufträge werden von einer CSV-Datei inputFiles.csv gelesen, vgl. Abs. 3.3.2, und auf den *DataFrame* dfInputFiles abgebildet, siehe Tabelle 4.2 [Bun21b, 22].

Der Jupyter Notebook Parameter C_LOAD_DATASETS, vgl. Abs. 4.1.1, filtert die zu betrachtenden Chats. Diese Arbeit betrachtet Chats aus Abschnitt 3.3.1 [Bun21b, 23].

4.2.3 Stufe 2: Chats aufbereiten

In dieser Stufe werden die heruntergeladenen Chats jeweils einzeln geparst und auf einen *DataFrame* abgebildet. Wie in Abschnitt 3.1.2 beschrieben, beinhaltet die Datei result.json neben den Nachrichten (messages) auch die Chat-Meta-Informationen wie Titel des Chats (name), Art des Chats (type) und die ID des Chats (id) [Bun21b, 24-26].

Im Gegensatz zum dem *DataFrame* dfInputFiles werden die Informationen nicht durch den Entwickler bzw. Data Scientist definiert, sondern direkt aus den heruntergeladenen Chats geparst. Diese *DataFrames* werden zunächst mithilfe eines *Python Dictionary* nach ihrem Dateipfad indexiert (dictMeta), um mit diesen Daten später weiter arbeiten zu können.

4.2.4 Stufe 3: Nachrichten aufbereiten

In dieser Stufe werden die Nachrichten aufbereitet, d.h. auf den *DataFrame* abgebildet (dfAllDataMessages), in dem sich alle Nachrichten aller Chats befinden. Um vereinzelt auf die Nachrichten eines Chats zuzugreifen, werden die Chats ähnlich wie bei der vorherigen Stufe mithilfe eines *Python Dictionary* nach ihrem Dateipfad indexiert (dictMessages).

Das Aufbereiten von Nachrichten lässt sich auf die folgenden drei Unterstufen abbilden:

- **Stufe 3a Nachrichten parsen**: In dieser Unterstufe werden die Nachrichten erst geparst, anschließend (optional) auf eine Stichprobe reduziert und bereits initial auf technisch notwendige Features abgebildet [Bun21b, 27-29].

- **Stufe 3b Text- und Metainformationen extrahieren**: In dieser Unterstufe wird sich mit den Texten innerhalb der Nachrichten auseinandergesetzt. Diese Texte werden also zunächst auf weitere Features abgebildet [Bun21b, 30-33].
- **Stufe 3c Query Features**: In dieser Unterstufe werden die Nachrichten auf sogenannte *Query Features* abgebildet. Mithilfe dieser Features ist es später möglich, vereinfacht Abfragen über die Nachrichten durchzuführen [Bun21b, 34-36].
- **Stufe 3d Fortgeschrittene Text Mining Modelle**: In dieser Unterstufe werden verschiedene Sprachmodelle wie beispielsweise *BERT* und *RoBERTa*, vgl. Abs. 3.2.3, auf die Nachrichten angewendet [Bun21b, 37-39].

Deren eigentliche Anwendung auf die heruntergeladenen Chats wird durch die Implementierung bedingt erst nach den drei Stufen durchgeführt *Aus dem Cache laden oder Cache erzeugen* [Bun21b, 40-43]: Es ist möglich, die dritte Stufe zu überspringen und auf die bereits vorher erzeugte `Pickle`-Datei zurückzugreifen, um so die Datenaufbereitung auf zum Beispiel externe Server auszulagern, vgl. 4.1.1, und lokal nur noch mit den zuvor aufbereitenden Daten zu arbeiten, was viel Zeit einsparen kann.

Stufe 3a: Nachrichten parsen

Tabelle 4.3 Features aus Stufe 3a.

Bezeichner	Beispiel	Beschreibung
`ftFilePath`	`ChatExport_2021-01-05/`	Relativer Pfad zum heruntergeladen Chat
`ftChatType`	`public_channel`	Art des Chats
`ftIsJsonFormatted`	`True`	Beinhaltet die Text-Nachricht Formatierungen, vgl. Abs. 3.1.3 [Bun21b, 29]

In dieser Stufe werden zunächst die Nachrichten aus den indexierten *DataFrames* in `dictMeta` (`message`) geparst, vgl. Abs. 4.2.3 [Bun21b, 27].

In Abschnitt 4.1.1 wurde beschrieben, dass die Läufe reduziert auf einer Stichprobe laufen können, um den Lauf zu Entwicklungszwecken abzukürzen. In dieser Unterstufe wird nach dem Parsen eine Stichprobe genommen, falls die Stichproben konfiguriert sind. Um die Attribute der Nachrichten von *Telegram* kennenzulernen, werden diese in Abschnitt 3.1.2 erläutert und in Zelle [28][Bun21b] noch einmal dargestellt.

Die von *Telegram* gegebenen Attribute werden 1:1 übernommen, aber es wird aus Gründen der Schichtentrennung nur mit "eigenen" Features weitergearbeitet [Bun21b, 40], die immer mit dem Präfix `ft` beginnen. Diese technisch-notwendigen Features werden in Tabelle 4.3 erläutert.

Stufe 3b: Text und Meta Informationen extrahieren

Der Text von Nachrichten beinhaltet einige bestimmte Merkmale, die von Interesse sein könnten. Dazu zählen beispielsweise URLs, *Hashtags* oder Formatierungsinformationen wie Fettdruck von Passagen. Diese Merkmale werden nun vereinfacht als **Text-Metainformationen**

Tabelle 4.4 Features aus Stufe 3b.

Bezeichner	Beispiel	Beschreibung
`ftTdText`	`Hallo! ä :) <3`	Das (nicht aufbereitete) Text-Attribut von *Telegram*
`ftTdUrls`	`[g.de, t.de]`	Array von URLs aus dem Text
`ftTdHashtags`	`[#h1, #h2]`	Array von *Hashtags* aus dem Text
`ftTdBolds`	`[t1, t2]`	Array von **fetten Passagen** aus dem Text
`ftTdItalics`	`[t1, t2]`	Array von *kursiven Passagen* aus dem Text
`ftTdUnderlines`	`[t1, t2]`	Array von unterstrichenen Passagen aus dem Text
`ftTdEmails`	`[test@test.de]`	Array von E-Mail-Adressen aus dem Text
`ftTdEmojis`	`[:) <3]`	Array von verwendeten *Emoticon*-Kombinationen aus dem Text
`ftTdEmojisDesc`	`[:smile:heart]`	Array von verwendeten *Emoticon*-Kombinationen aus dem Text, abgebildet auf einen Text
`ftTdCleanText`	`Hallo! ae`	Nachrichten-Text aus `ftTdText` ohne *Emoticons* und ohne deutsche Zeichen und Zeilenumbrüche
`ftTdSafeText`	`Hallo ae`	Nachrichten-Text aus `ftTdCleanText` ohne Sonderzeichen wie Punkt
`ftTdSafeLowerText`	`hallo ae`	Nachrichten-Text aus `ftTdSafeText`, abgebildet auf Text ohne Groß- und Kleinschreibung
`ftTdTextLength`	`9`	Anzahl der Zeichen in `ftTdCleanText`

bezeichnet. Ein Teil dieser Features lässt sich direkt aus der JSON-Datenstruktur ableiten [Bun21b, 30]. Leider ist durch die Datenstruktur des Attributes `text` gegeben, dass geschachtelte Formatierungen nicht korrekt erkannt werden können, weil diese auf einen JSON-Pfad abgebildet werden, vgl. Abs. 3.1.3.

So können beispielsweise URLs (`link` und `text_link`) und *Hashtags* (`hashtag`) nicht korrekt erkannt werden, falls diese innerhalb einer fett-formatierten Passage verwendet werden. Daher werden diese Features zusätzlich aus dem Text geparst und mit diesen kombiniert und anschließend aus dem Text entfernt [Bun21b, 31-33].

Im Text sind auch *Emoticons* enthalten, die nach der Nachbearbeitung zusätzlich mit der Bibliothek `demoji`, vgl. Abs. 3.2.2, extrahiert und anschließend aus dem Text entfernt werden. Aus Gründen der Schichtentrennung beginnen alle diese textbezogenen Features mit dem Präfix `ftTd`. Die verschiedenen Features aus dieser Stufe werden in Tabelle 4.4 beschrieben.

Stufe 3c: Query Features

Tabelle 4.5 Features aus Stufe 3c.

Bezeichner	Beispiel	Beschreibung
`ftQrIsValidText`	`True`	Ist der Text für die Auswertung geeignet, vgl. Abs. 4.4.1?
`ftQrIsEdited`	`False`	Ist die Nachricht editiert worden?
`ftQrIsForwarded`	`True`	Ist die Nachricht weitergeleitet worden?
`ftQrCoPhotos`	`False`	Enthält die Nachricht Fotos?
`ftQrCoFiles`	`True`	Enthält die Nachricht Dateien?
`ftQrCoUrls`	`False`	Enthält die Nachricht URLs?
`ftQrCoHashtags`	`True`	Enthält die Nachricht *Hashtags*?
`ftQrCoBolds`	`False`	Enthält die Nachricht **fett** formatierte Passagen?
`ftQrCoItalics`	`True`	Enthält die Nachricht *kursiv* formatierte Passagen?
`ftQrCoUnderlines`	`False`	Enthält die Nachricht unterstrichen formatierte Passagen?
`ftQrCoEmails`	`True`	Enthält die Nachricht E-Mail Adressen?
`ftQrCoEmojis`	`False`	Enthält die Nachricht *Emoticons*?

Um später schnell und einfach Datenabfragen (engl. "Query") durchführen zu können, werden Nachrichten ebenfalls auf hier bezeichnet als **Query Features** abgebildet. Aus Gründen der Schichtentrennung beginnen alle diese Features mit dem Präfix `ftQr`. Diese Features werden in Tabelle 4.5 beschrieben.

Diese Features greifen wiederum auf Features zurück, die in der vorangegangenen Stufen bereits abgebildet worden sind. So wird beispielsweise ausgewertet, ob ein Text für die Text-Auswertung geeignet ist, mithilfe des Features `ftTdTextLength` [Bun21b, 34-36].

Stufe 3d: Fortgeschrittene Text Mining Modelle

Tabelle 4.6 Features aus Stufe 3d.

Bezeichner	Beschreibung
`ftTrNerRoberta`	*NER* durch *xlm-roberta-large-finetuned-conll03-german*
`ftTrNerBert`	*NER* durch *bert_de_ner*
`ftTrSenBert`	*Sentiment analysis* durch *bert-base-multilingual-uncased-sentiment*
`ftSenTb`	*Sentiment analysis* durch *TextBlob*

In dieser Stufe werden verschiedene Sprachmodelle wie *BERT* und *RoBERTa* angewendet, vgl. 3.2.3, für den Zweck einer *Sentiment analysis*, vgl. Abs. 2.4.8, und *NER*, vgl. Abs. 2.4.7 [Bun21b, 37-39].

Die verschiedenen Sprachmodelle bilden die Klassen unterschiedlich ab: So bildet beispielsweise *BERT* die erkannten Personen auf `I-PER` ab, im Gegensatz zu *RoBERTa*, das diese auf `B-PER` abbildet. Daher werden diese unterschiedlichen Schemen auf ein für diese Arbeit einheitliches Schema abgebildet (`per`) [Bun21b, 37] und greifen dabei auf den Text `ftTdCleanText` zu.

Diese werden in Tabelle 4.6 beschrieben und beginnen, je nach eingesetztem Sprachmodell und Zweck, zum Beispiel mit dem Präfix `ftTr` für Sprachmodelle aus den Transformers. `ftSenTb` stellt eine Ausnahme dar, die auf die Komponente *TextBlob* zurückgreift. Die Auswertung dieser Sprachmodelle wird gesondert in Abs. 4.4.5 und 4.4.6 betrachtet.

4.3 Social Media Mining

Nach dem Aufbereiten der Chats, also der Meta-Informationen über den Chat und der Nachrichten selbst zu Features, vgl. Abs. 4.2, kann man sich nun mit dem Feld Social Media Mining beschäftigen, vgl. Abs. 2.3.

In Abschnitt 3.3.1 wurden Fragen definiert, die mithilfe von Social Media Mining Methoden beantwortet werden sollen. In dieser Arbeit wurde zunächst explorativ vorgegangen. Die folgenden Abschnitte beschreiben die wichtigsten Gedanken und die Vorgehensweise wieder anhand des Jupyter Notebooks [Bun21b, 44-94].

Es ist mögliche die Strukturen von sozialen Netzwerken auf Graphen abzubilden, die anschließend dazu genutzt werden können, diese Strukturen zu visualisieren. Im Rahmen dieser Arbeit ist es nicht möglich, in angemessener Tiefe auf die Graphentheorie einzugehen. Im Abs. 4.3.2 wird weiter darauf eingegangen. Um mehr über Graphen und die Graphentheorie zu erfahren, empfehlen sich die Bücher [Tur10] [Tit11] und [Pot19], die auch diese Arbeit beeinflusst haben.

4.3.1 Chats

Zunächst ist es erforderlich, ein grobes Verständnis über den Aufbau von *Telegram*-Chats zu erlangen. Es werden die Chats aus Abs. 3.3.1 betrachtet. Folglich wurden 7 *Supergroups* und 54 *Channels* aufbereitet [Bun21b, 44], vgl. Abs. 3.1.

Da die Chat-Namen Zeichen wie *Emoticons* und Formatierungen enthalten, mit denen viele Python-Bibliotheken nicht umgehen können, vgl. Abs. 4.2.1, und mit diesen später weitergearbeitet wird, werden diese Namen auf einen eindeutigen Bezeichner abgebildet, der sich ausschließlich des ASCII-Zeichensatzes bedient [Bun21b, 45].

Um zu verstehen, welche Features zunächst zur Auswertung in Betracht gezogen werden, wurden Datenabfragen (Queries) über die Chats definiert [Bun21b, 46-51], beispielsweise wie viele Nachrichten in einem Chat Formatierungen enthalten (`Formatted Messages`), vgl. Abs. 3.1.3, wie viele Nachrichten editiert worden sind (`Edited Messages`), aber auch verschiedene Arten von Datei-Anhängen wurden betrachtet. Im Jupyter Notebook werden diese Auswertungen zusammenfassend auch als Tabelle dargestellt [Bun21b, 52].

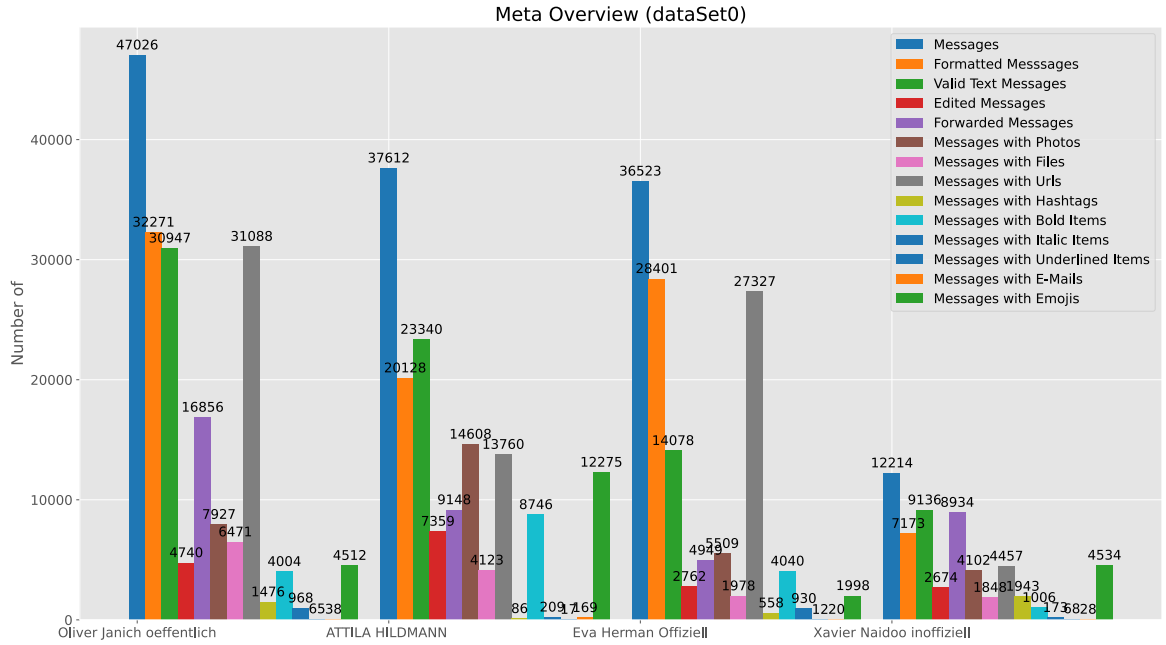

Abbildung 4.1 Übersicht der Chats im *DataSet0*.

Die Abbildung 4.1 stellt diese Auswertungen über den `DataSet0` als Säulendiagramm dar: Wenig überraschend ist, dass die Text-Nachrichten einen großen Teil der Chats ausmachen. So enthalten beispielsweise 30.947 von 47.026 (ca. 66%) der Nachrichten von *Oliver Janich* Text, der für diese Arbeit mit Text Mining Methoden ausgewertet werden kann (nach dem in der Arbeit definierten Kriterium, vgl. 4.2.4). Weitere Auswertungen sind im Notebook [Bun21b, 53-57] einsehbar.

4.3.2 Social Graphs - Abbilden von Chats auf Features

Diese Stufe beschäftigt sich nicht direkt mit dem Text aus den Nachrichten, sondern mit den Features aus diesen, vgl. 3.1.3, also den *Nachrichten-spezifischen Attributen* und außerdem den *Autor-spezifischen Attributen*, vgl. Abs. 3.1.2, und versucht diese auf einen Graphen abzubilden, um die Kommunikationsstrukturen in sozialen Netzwerk zu visualisieren. Diese Graphen werden im Rahmen der Arbeit als **Social Graphs** bezeichnet [Bun21b, 58-70].

Abweichungen von Chat und Nutzernamen

```
1 - Add changed attribute in actor (prev=['Victor Schurk']/new=['
    Victor Schurk', 'Heinrich Gruber'])
2 [...]
3 - Add changed attribute in from (prev=['ME Night']/new=['ME Night',
    'Dario Zaza'])
4 - Add changed attribute in from (prev=['RJ']/new=['RJ', 'Rosa'])
```

Auflistung 4.5 Messungen von Abweichungen *from* und *actor* über die Zeit.

In *Telegram* ist es möglich, seinen Nutzernamen und den Namen eines Chats unbegrenzt oft zu ändern. Das erschwert die Auswertbarkeit dieser Chats.

Erfreulicherweise übermittelt *Telegram* abhängig von der Chat-Art für jede Nachricht immer den Namen des Autors (`from`) und die ID des Autors (`from_id`). Gleiches gilt für Service-Nachrichten, also z.B. wenn ein Administrator den Namen eines Chats geändert hat (`actor` und `actor_id`). Interessanterweise zeigen die aktuellen *Telegram*-Clients (Stand 25.03.2020) immer den aktuellen Namen eines Autor zu einer in der Vergangenheit erstellten Nachricht, bei der der Autor noch anders hieß.

Mithilfe dieser Daten ist es also nicht nur möglich, Abweichungen von Nutzernamen in der Vergangenheit festzustellen, sondern auch Abweichungen von Chatnamen festzustellen, da abhängig von Chat-Art der Chat-Name im Feld `from` übermittelt wird [Bun21b, 58-59]. Die stark gekürzte Ausgabe dieses Verfahrens wird in Auflistung 4.5 dargestellt.

Es wurde über alle Datensätze gemessen, wie viele Abweichungen nach Features (`from` und `actor`) insgesamt aufgetreten sind (`from` 0,08%, `actor` 0,01%). Da die Abweichungen für dieses Vorhaben nicht signifikant sind, werden diese nicht berücksichtigt [Bun21b, 59].

Extrahieren von Features und dynamisches Auflösen

Durch die Datenaufbereitung von Chats, vgl. Abs. 4.2.4, stehen uns nun verschiedene Features für Chats zu Verfügung. Um diesen Chat auf einen Graphen abzubilden, werden diese Features weiter aufbereitet und lassen sich sprachlich als **Social Graph Features** abgrenzen.

Diese Features lassen sich unterscheiden in **statische Features** [Bun21b, 60-61], die sich direkt aus den Daten ableiten lassen, und **dynamische Features** [Bun21b, 62-64], die einer dynamischen Auflösung bedürfen. Im Rahmen der Arbeit kann nur eine Teilmenge aus allen Features betrachtet werden. Die Teilmenge kann und soll erweitert werden, siehe Abs. 5.

Die *statischen Features* werden weiter unterschieden: Einerseits sind da **Formatierungs-spezifische statische Features**, die durch Features aus den *Nachrichten-spezifischen Attributen*, vgl. Abs. 3.1.2, abgeleitet werden können. Diese werden in Tabelle 4.7 beschrieben.

Tabelle 4.7 Formatierungs-spezifische statische Features.

Bezeichner	Beschreibung
`fSgListHashtag`	Eine Liste aus verwendeten *Hashtags* in einem Chat.
`fSgListEmoji`	Eine Liste mit verwendeten Kombinationen von *Emoticons* in Nachrichten in einem Chat.

Andererseits gibt es **Autor-spezifische statische Features**, die sich direkt aus den *Autoren spezifischen Attributen*, vgl. Abs. 3.1.2, ableiten lassen. Diese werden in Tabelle 4.8 beschrieben.

Tabelle 4.8 Autor-spezifische statische Features.

Bezeichner	Beschreibung
`fSgListFrom`	Eine Liste von Autoren in einem Chat.
`fSgListForFrom`	Eine Liste von Autoren, von denen weitergeleitet worden ist, in einem Chat.
`fSgListActor`	Eine Liste von Administratoren in einem Chat.
`fSgListMember`	Eine Liste von Benutzern, die beispielsweise explizit in den Chat eingeladen worden sind.
`fSgListSavedFrom`	Eine Liste von Benutzern, die Nachrichten in den Chats "gesichert haben".

In den Daten wurde festgestellt, dass eine große Anzahl von Nachrichten auf YouTube-Videos und Webseiten referenziert. Dabei wurde auch festgestellt, dass ein Teil der URLs mit dem Dienst `bitly.com`[2] gekürzt worden ist. Diese externen Abhängigkeiten erschweren die Aufbereitung von Chats, daher werden diese in diesem Notebook aufgelöst, was als **dynamisches Auflösen** bezeichnet wird [Bun21b, 62-64]. Die Features werden als *Dynamische Features* bezeichnet und über die Attribute ebenfalls aus den *Nachrichten-spezifischen Attributen*, vgl. Abs. 3.1.2, abgeleitet. Diese Features benötigen jedoch externe Abhängigkeiten wie Webserver, um bei der **dynamischen Auflösung** immer auf die aktuelle Datenlage zurückgreifen zu können. Diese Features werden in Tabelle 4.9 beschrieben. Im Rahmen dieser Arbeit wurde die dynamische Auflösung fortlaufend durchgeführt, da bei jedem Lauf die aufgelösten Daten, also die URLs und YouTube-Titel, in eine Datei geschrieben wurden, vgl. Abs. 3.3.2.

Top-n Darstellung

```
1  Analyse now >>DS-05-01-2021/ChatExport_2021-01-05-janich<<
2  [...]
3  - Top hashtags -
4  ('StopLeftDystopia', 108)
5  ('Berlin', 42)
6  ('links', 42)
7  ('Messer', 35)
8  ('Einreise', 32)
9  ('Kosten', 32)
10 ('Justiz', 32)
11 ('Netzfund', 31)
12 ('CoronaKrise', 21)
13 ('Orwell', 19)
```

Auflistung 4.6 Top-10 verwendete *Hashtags* von *Oliver Janich* (*DataSet0*).

2 siehe `https://bitly.com/`

Tabelle 4.9 Dynamische Features.

Bezeichner	Beschreibung
`fSgListUrl`	Die URLs werden durch die Verwendung von *urlparse*, vgl. Abs. 3.2, auf ein einheitliches Format abgebildet, z.B. `https://example.org/` [Bun21b, 63]. *Bitly*-Adressen werden hierbei auf die richtigen URLs abgebildet [Bun21b, 62].
`fSgListRef`	In den gefunden URLs kommen häufig *Telegram*-Referenzen z.B. auf andere Chats vor. Diese URLs enthalten also die Zeichenkette `t.me` und werden hier aufgelistet [Bun21b, 63].
`fSgListHost`	Die Abbildung von URLs auf ein einheitliches Format, vgl. `urlList`, ist zwar praktisch, beinhaltet aber viele Informationen, die später bei der Abbildung auf einen Graphen eher störend als nützlich sein könnten, wie Query-Parameter (`&key=value`) und Routing-Informationen wie das Protokoll (`https`) und Pfade wie (`/path`). Daher wird hier zusätzlich nur der Hostname abgebildet, z.B. `example.org` [Bun21b, 63].
`fSgListYouTube`	Falls es sich bei der URL um ein YouTube Video handelt, wird es über YouTube und Netzwerk-Bib zu einem Titel aufgelöst [Bun21b, 64].

```
1  Analyse now >>DS-05-01-2021/ChatExport_2021-01-05-hildmann<<
2  [...]
3  - Top emojis -
4  (':wolf', 1605)
5  (':play button', 693)
6  (':cross mark', 573)
7  (':eagle', 259)
8  (':grinning squinting face', 250)
9  (':grinning squinting face:grinning squinting face:grinning
       squinting face', 212)
10 (':cross mark:wolf:crossed swords', 180)
11 (':red exclamation mark:red exclamation mark:wolf', 171)
12 (':crossed swords', 151)
13 (':green heart', 125)
```

Auflistung 4.7 Top-10 verwendete *Emoticons* von *Attila Hildmann* (*DataSet0*).

Die Chats können also auf die zuvor definierten *Social Graph Features* abgebildet und mithilfe von Graphen visualisiert werden, vgl. 4.3.3. Diese Features können aber nicht nur über einen Graphen visualisiert, sondern auch als Top-n Liste ausgegeben werden. Das n steht hierbei für die Anzahl der Einträge, so könnte beispielsweise $n = 10$ für eine Top-10-Liste stehen.

Mithilfe dieser Listen ist es möglich, mehr über die "Vernetzungen" der Chats zu erfahren, zum Beispiel auf welche Webseite häufig referenziert wurde oder welche *Akteure* besonders einflussreich sind. Es ist aber auch möglich, schnell zu erkennen, welche Stilelemente verschiedene *Akteure* häufig verwenden. Dazu gehören beispielsweise `#hashtags`, vgl. Abs. 4.6, oder verwendete *Emoticon*-Kombinationen innerhalb einer Nachricht, vgl. Abs. 4.7. Um aber ein aussagekräftiges Ergebnis zu erhalten, muss die Anzahl der Verwendungen von bestimmten Merkmalen in Relation zu der Anzahl der Nachrichten in einem Chat gestellt werden. Im Notebook wird das vollständige `DataSet0` betrachtet, aus Gründen der Übersicht ist hier nur ein Auszug möglich [Bun21b, 67-68].

Suchmuster für Chats

```
[...]
Analyse Chat (Forwarded From) >>Oliver Janich oeffentlich<<

1:  (downloaded=True)  (refs=1211)      >>Kulturstudiotv<<
2:  (downloaded=True)  (refs=1140)      >>Jouwatch<<
3:  (downloaded=True)  (refs=974)       >>Eva Herman Offiziell<<
4:  (downloaded=True)  (refs=786)       >>Der Waldgang<<
5:  (downloaded=True)  (refs=759)       >>Einzelfallinfos<<
6:  (downloaded=True)  (refs=661)       >>QlobalChange <<
7:  (downloaded=True)  (refs=635)       >>ExpressZeitung<<
8:  (downloaded=True)  (refs=577)       >>1984  Das Magazin<<
9:  (downloaded=False) (refs=554)       >>Oliver Janich Premium<<
10: (downloaded=True)  (refs=421)       >>ForscherGeist<<
[...]
```

Auflistung 4.8 Verifizierung der Vollständigkeit von heruntergeladenen Chats (*Suchmuster - DataSet0*).

In Abschnitt 2.3.2 wurde beschrieben, dass es nicht immer einfach ist zu gewährleisten, ob man alle notwendigen Daten heruntergeladen hat (*Obtaining Sufficient Samples*). Auch hier hat man das praktische Problem, nicht gewährleisten zu können, dass alle notwendigen Chats in betrachtet wurden.

`DataSet0` und `DataSet2` stellen die Ausgangsmenge in dieser Arbeit dar, vgl. Abs. 3.3.1.

Wie im Abs. 2.3.2 beschrieben, sind nicht immer ausreichend Daten vorhanden, um ein soziales Netzwerk vollständig abbilden zu können. Falls man beispielsweise Chats oder Autoren in einem Graphen darstellen möchte, muss man Annahmen über diese treffen, z.B. dass *Akteure*, von denen oft weitergeleitet wurde, für einen Chat einflussreicher sind.

Um diese *Akteure*, also Autoren in Chats, in einem Graphen darstellen zu können, ist es notwendig, diese auch noch (teilweise) herunterzuladen. Dafür ist es vorher notwendig, diese Chats zu identifizieren, also sich ein Muster zu überlegen, wie man relevante Chats erkennt. Dieses Muster wird in der Arbeit als **Suchmuster für Chats** bezeichnet [Bun21b, 69].

Im Rahmen dieser Arbeit wurden die Chats manuell aus *Telegram* exportiert, vgl. Abs.

3.1.1, und in die Datei `inputFiles.csv` eingetragen, vgl. Abs. 3.3.2. Auch das Herunterladen der Chats hat einige Zeit in Anspruch genommen, da die Chats zunächst vollständig (ohne Download-Filter heruntergeladen worden sind.

Durch diese Umstände bedingt wurden die Top-10 referenzierten *Telegram*-Chats (`fSgListRef`) und Chats, von denen weitergeleitet worden ist (`dSgListForFrom`), in Betracht gezogen, da die Chat-Art `channel` des `DataSet0` nur diese beiden Attribute bereitstellt, vgl. Abs. 3.1. Diese werden als `DataSet1` und `DataSet1a` bezeichnet.

In dem Notebook wird überprüft, ob alle erforderlichen Chats geladen worden sind: Wie in Auflistung 4.8 ersichtlich, konnten nicht alle erforderlichen Chats heruntergeladen werden, was daran lag, dass Chats wie `Oliver Janich Premium`, nicht öffentlich zugänglich sind. Die Mehrheit der Chats konnte aber manuell recherchiert und heruntergeladen werden [Bun21b, 70].

4.3.3 Social Graphs - Darstellung von Graphen

Wie in Abschnitt 4.3.2 beschrieben, lassen sich die heruntergeladenen Chats auf *Social Graph Features* abbilden. Diese Features wiederum lassen sich auf Graphen abbilden, die sich anschließend visualisieren lassen.

Dafür muss aber zunächst weiter auf Graphen im Hinblick auf die Visualisierung von Kommunikationsstrukturen und Größenverhältnissen von Chatgruppen eingegangen werden [Bun21b, 71-76].

Visualisierung-Möglichkeiten von Graphen

Zur Visualisierung von Graphen wird auf die Bibliotheken *NetworkX* und *Graphviz* zurückgegriffen, vgl. Abs. 3.2. Diese Bibliotheken stellen verschiedene Funktionen bereit, um mit Graphen zu arbeiten, also zum Beispiel diese zu manipulieren oder zu visualisieren.

Zunächst stellt sich aber die Frage, was der Graph visualisieren soll: In der Arbeit wurde sich Folgendes überlegt:

- Die **Knoten** `V` (engl. “vertex”) des Graphen sollen die Chats bzw. Chat-Teilenehmer abhängig vom jeweiligen Graphen repräsentieren. Die Größe des Knotens soll bei der Visualisierung die Größe des Chats, also zum Beispiel anhand der Anzahl der Nachrichten repräsentieren. Nicht alle Informationen stehen für die jeweiligen Graphen zur Verfügung. Daher werden später die Knoten, über die alle Informationen zur Verfügung gestellt werden, *blau* gefärbt und die “geschätzten” Knoten *orange* gefärbt.
- Die **Kanten** `E` (engl. “edge”) des Graphen sollen die Verbindungen zwischen den Chats bzw. Chat-Teilnehmern abhängig vom jeweiligen Graphen repräsentieren. Falls zwei Chats (*Knoten*) enger miteinander verwandt sind, sollen diese bei der Visualisierung näher beieinander liegen. Bei der Visualisierung repräsentiert also die Kantenlänge die Stärke der Verwandtschaft von Chats. Längere Kanten sollen also bedeuten, dass die Chats wenig miteinander zu tun haben.

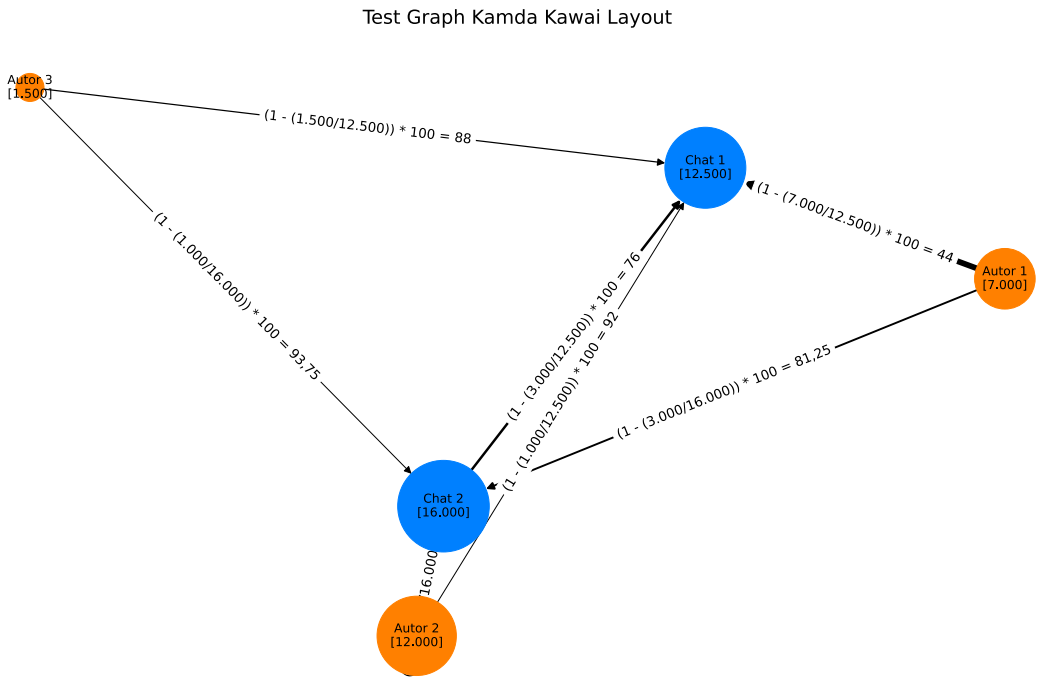

Abbildung 4.2 Visualisierung des Test Graphen (*Kamda Kawai Layout*).

Für die Visualisierung, siehe Abb. 4.2, einer solchen Darstellung wurde sich folgendes **Szenario** [Bun21b, 73] überlegt, in dem vier *Telegram* Entitäten betrachtet werden. Das folgende Szenario ist an die Datenlage für diese Arbeit angepasst. So kann beispielsweise der Betreiber von *Telegram* ein deutlich detaillierteres Modell entwickeln.

Chat 1 und **Chat 2** repräsentieren *Channels* mit verschiedenen Nachrichten von verschiedenen Autoren **Autor 1** und **Autor 2**, also *Telegram*-Benutzern. Wichtig an dieser Stelle ist, dass *Chat 2* selbst ein Autor werden kann und Nachrichten an *Chat 1* schreiben, was in *Telegram* einer Weiterleitung entsprechen würde.

Die Chats *Chat 1* und *Chat 2* wurden vollständig heruntergeladen und sind Ausgangspunkte für die Abbildung auf Graphen. Die Autoren *Autor 1* und *Autor 2* können nur über die heruntergeladenen Chats identifiziert werden. Da gibt es also keine Daten, daher werden die im Gegensatz zu den Chats *orange* gefärbt, nicht *blau*.

Durch die heruntergeladenen Daten von *Chat 1* und *Chat 2* ist bekannt, welcher Nachrichten-Verkehr aus Sicht der Chats stattgefunden hat:

- *Autor 1* an *Chat 1*: 7.000 Nachrichten
- *Autor 2* an *Chat 1*: 1.000 Nachrichten
- *Autor 3* an *Chat 1*: 1.500 Nachrichten
- *Chat 2* an *Chat 1*: 3.000 Nachrichten
- *Autor 1* an *Chat 2*: 3.000 Nachrichten

- *Autor 2* an *Chat 2*: 12.000 Nachrichten
- *Autor 3* an *Chat 2*: 1.000 Nachrichten

Folglich besteht der *Chat 1* aus 12.500 Nachrichten (7.000 + 1.000 + 1.500 + 3.000). Möchte man nun den Einfluss von Autor *Autor 1* und dem *Chat 1* in einem Graphen darstellen, mathematisch als Kantenlänge eW betrachtet, also die Anzahl der Nachrichten eines Autors in einem Chat n_a in Relation zur Anzahl der Nachrichten in einem Chat n_t setzen, könnte man sich der folgenden Formel bedienen:

$$eW(n_a, n_t) = (1 - \frac{n_a}{n_t}) * 100$$

Wichtig hierbei ist, dass die Kantenlänge auf Werten zwischen $[0; 100]$ abgebildet wird, da dies für die numerische Verarbeitung von Vorteil ist. In dem Beispiel des Einflusses von *Autor 1* auf *Chat 2* wäre die Kantenlänge $eW(7.000, 12.500) = 44LE$ lang und bei *Autor 2* $eW(1.000, 12.500) = 92LE$.

Die Knoten sollen die Chats und ihre Größe visualisieren. Die Knotengröße repräsentiert bei *Chat 1* und *Chat 2* die Anzahl der Nachrichten, da diese Informationen, bedingt durch die Datenstruktur, vorhanden sind. Daher könnten diese Knoten auch *blau* eingefärbt werden. Als Formel lässt sich die Knotengröße vW wie folgt angeben:

$$vW(n_t) = n_t$$

Die Autoren werden ebenfalls durch Knoten repräsentiert. Nun stellt sich die Frage, wie die Knotengröße in diesem Fall berechnet werden kann, da hier keine direkten Informationen über die Autoren vorliegen, sondern diese aus den Nachrichtenverläufen abgeleitet werden. Für dieses Vorgehen wurde sich überlegt, die größte Anzahl an Nachrichten eines Autors an einen Chat zu nehmen ($\max(n_a)$), da Nachrichten von Autoren doppelt an Chats versendet werden und das die graphische Aufbereitung verzerren könnte. Diese Knoten müssten nach diesem Szenario *orange* eingefärbt werden, um sie besser unterscheiden zu können.

Hinweis: Die Visualisierung der Graphen wird mithilfe von den Bibliotheken *NetworkX* und *Graphviz* gewährleistet, vgl. Abs. 3.2. Diese Bibliotheken implementieren verschiedene Layouts (z.B `Kamda Kawai Layout`, `Spring Layout` und `Graphviz Layout`) und Funktionen zur Visualisierung von Graphen und Berechnungen auf diesen, auf die in dieser Arbeit nicht eingegangen werden kann. Vereinfacht gesagt hat man in der Praxis das Problem, einen solchen Graphen nicht perfekt visualisieren zu können, falls die oben definierten Formeln eingesetzt werden. Mithilfe dieser Layouts ist es möglich, diese Graphen dennoch für den 2-Dimensionalen Raum zu visualisieren. Die Graphen in dieser Arbeit wurden mithilfe des `Spring Layouts` visualisiert. Das oben definierte Szenario wurde jedoch im Notebook [Bun21b, 71-76] auch noch mithilfe des `Kamda Kawai Layout` und des `Graphviz Layout` visualisiert.

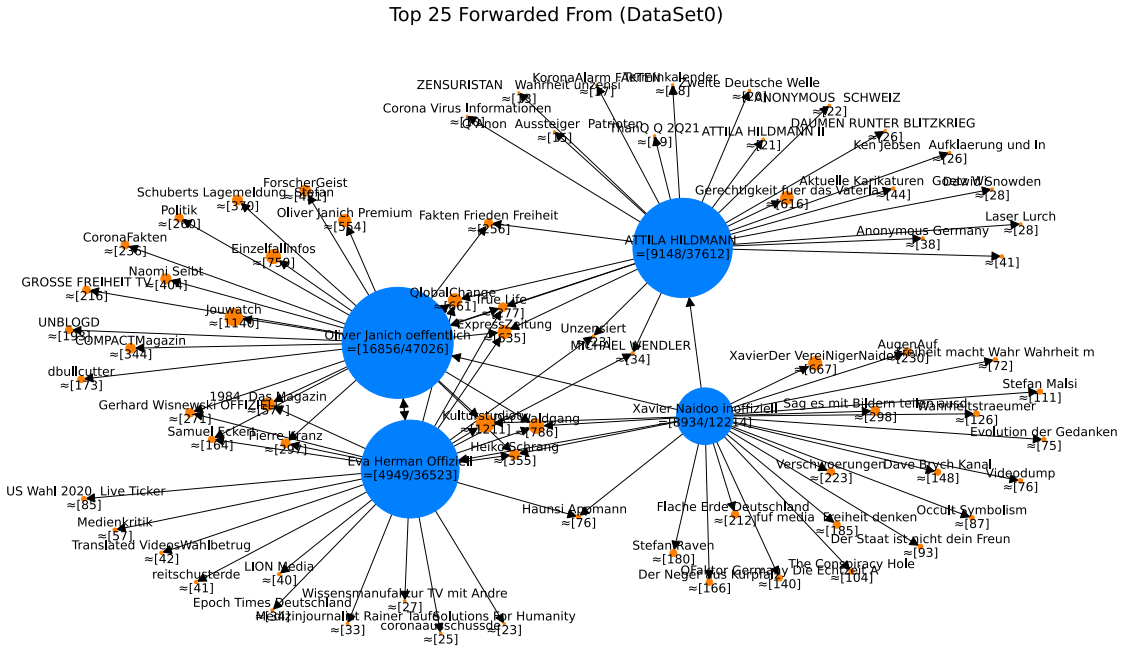

Abbildung 4.3 Social Graph: Top 25 der Autoren, von denen weitergeleitet worden ist (*DataSet0*).

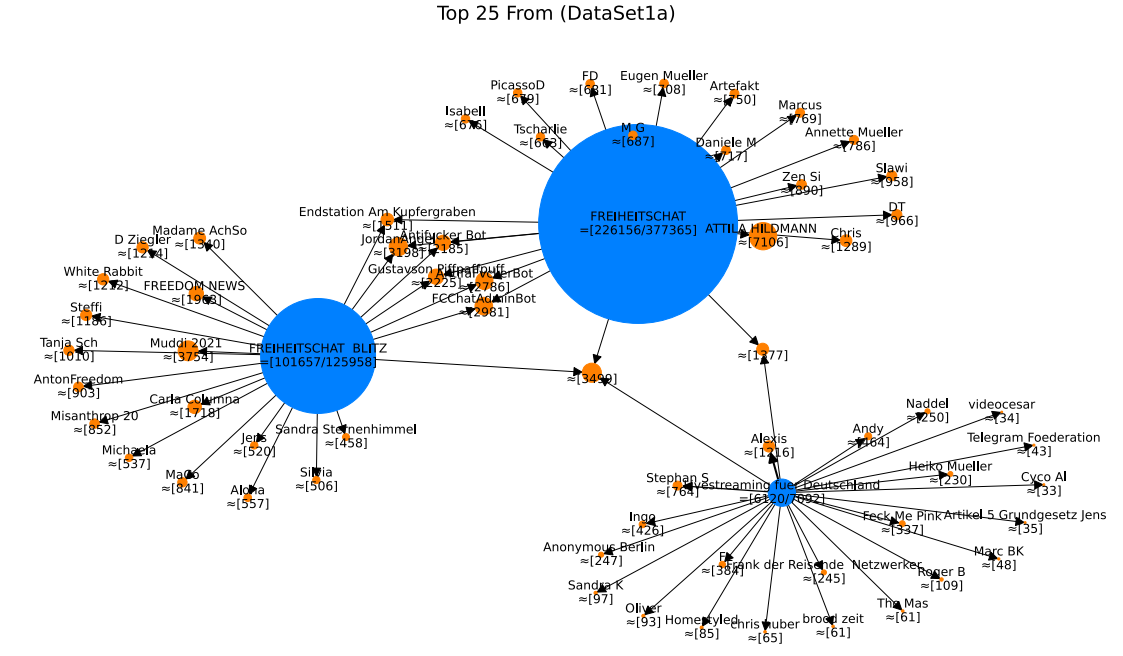

Abbildung 4.4 Social Graph: Top 25 Autoren (*DataSet1a*).

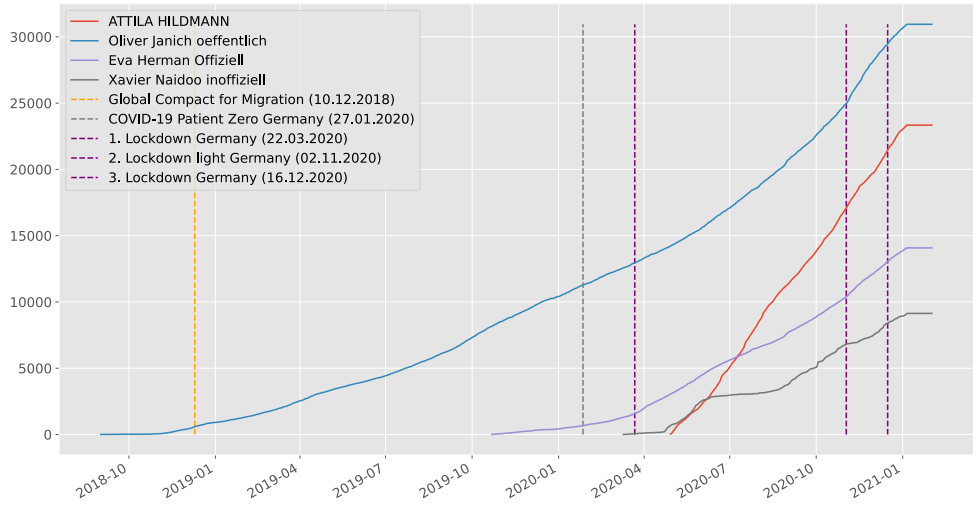

Abbildung 4.5 Darstellung der Anzahl der Nachrichten in Chats aus *DataSet0* im zeitlichen Verlauf.

Visualisierung-Möglichkeiten von heruntergeladenen Chats

Im vorherigen Abschnitt 4.3.2 wurden die sogenannten *Social Graph Features* beschrieben, mit denen es möglich ist, Chats auf Graphen abzubilden. Im Notebook wird mit den Features `fSgListForFrom`, `fSgListHashtag`, `fSgListEmoji`, `fSgListFrom` weitergearbeitet [Bun21b, 77-78].

Die Art des Graphs und damit die Darstellung hängt direkt von der Chat-Art ab, vgl. Abb. 3.1, so können Autoren von Nachrichten (`from`) nur innerhalb von `groups` ermittelt werden, da jene nur bei diesen von *Telegram* übermittelt werden.

In dieser Arbeit werden nur die Features `fSgListForFrom` aus Sicht das `DataSets0` (nur `channels`), vgl. Abb. 4.3, und `fSgListFrom` aus Sicht des `DataSet1a` (nur `groups`) betrachtet, vgl. Abb. 4.4. Unter dem Knoten-Namen befinden sich Angaben über die Knotengröße, die vom gewählten Feature abhängt, in Relation zur gesamten Anzahl der Nachrichten in einem Chat.

Hinweis: Auf die detaillierte Implementierung kann im Rahmen dieser Arbeit nicht eingegangen werden, diese befindet sich aber im Notebook [Bun21b, 77-83] und wird dort dokumentiert. An dieser Stelle ist wichtig zu verstehen, dass die benutzten Parameter starke Auswirkungen auf die Graphen haben. Es sollte sich also vor der Interpretation dieser mit den Parametern und der Implementierung auseinandergesetzt werden, vgl. Kapitel 5.

4.3.4 Dimension Zeit

Da *Telegram* zu jeder Nachricht den Zeitpunkt ihrer Erstellung (`date`) übermittelt, ist es möglich, diese Nachrichten im zeitlichen Verlauf darzustellen. So kann beispielsweise unter-

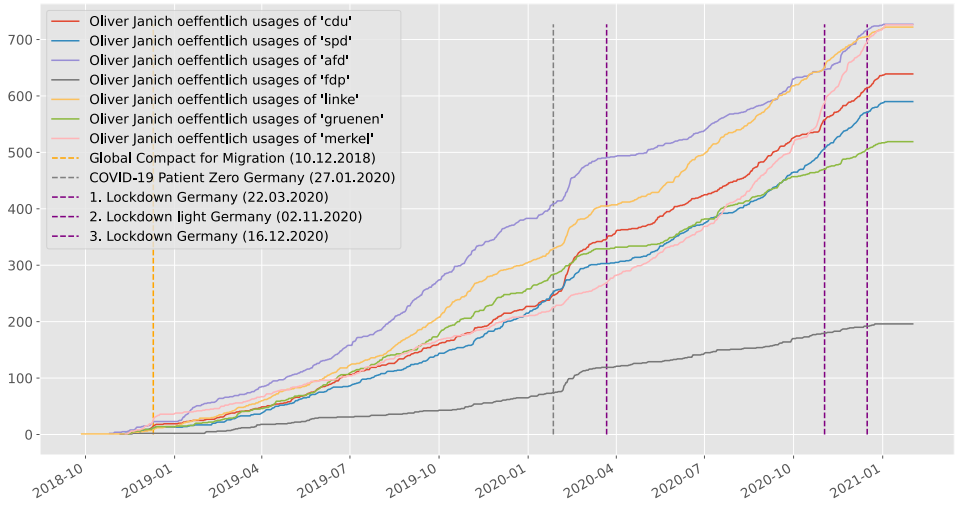

Abbildung 4.6 Erwähnungen von Parteien von *Oliver Janich* im zeitlichen Verlauf.

sucht werden, ob ein politisches Ereignis einen Chat beeinflusst hat.

Um sich dieser Thematik zu nähern, werden in Abb. 4.5 die Anzahl der Nachrichten der Chats in `DataSet0` im zeitlichen Verlauf dargestellt. Im Notebook befinden sich die Auswertungen zu allen betrachteten Datensätzen [Bun21b, 84-89]. In diesem zeitlichen Verlauf wurden für *COVID-19* relevante Termine markiert, da sich in den Daten zeigte, dass häufig darüber gesprochen wird. Außerdem wurde der Zeitpunkt des Migrationspaktes eingetragen, weil sich später in den automatischen *Word Clouds* zeigen wird, vgl. Abs. 4.4.2, dass *Oliver Janich* häufiger davon spricht.

In den Daten wurde ersichtlich, dass Parteien häufig erwähnt werden, daher wurde die Anzahl der Verwendungen der wichtigsten Parteien zunächst zeitlich dargestellt. Die Vorgehensweise hierbei war ziemlich trivial [Bun21b, 90] und ist damit nicht aussagekräftig, aber zunächst vielleicht aufschlussreich. In der Abbildung 4.6 wurde diese beispielhaft für *Oliver Janich* dargestellt. Im Notebook befinden sich die Auswertungen zu allen betrachteten Datensätzen [Bun21b, 91-94].

Hinweis: Die Anzahl der Abonnenten können mit diesen Daten nicht dargestellt werden, da in den Daten keine Informationen darüber enthalten sind. Es gibt aber beispielsweise Webseiten wie *Telegram Analytics*[3], die Telegram Gruppen permanent auswerten und die Anzahl der Abonnenten im zeitlichen Verlauf darstellen können.

3 siehe `https://tgstat.com/channel/@oliverjanich`

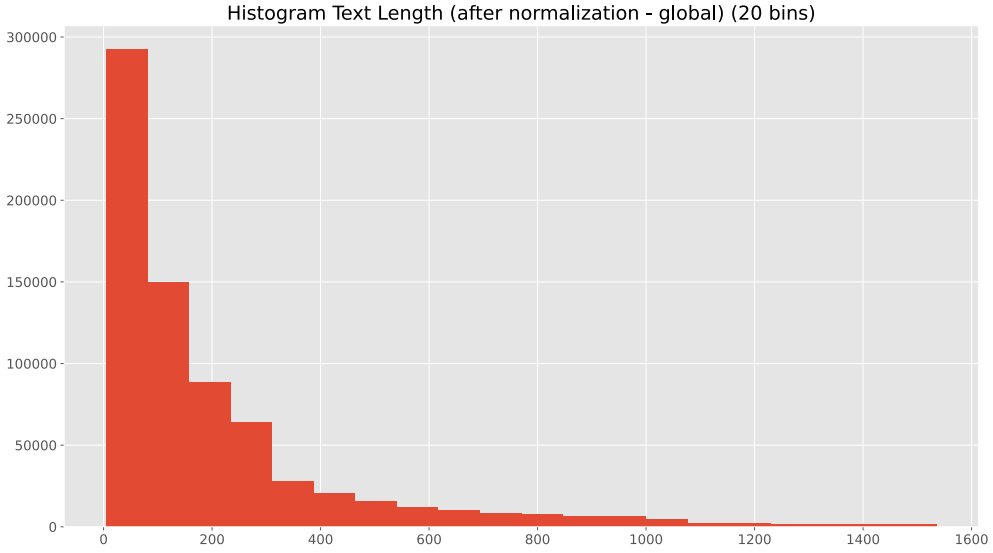

Abbildung 4.7 Histogramm über die Anzahl der Zeichen in einer Nachricht über alle Datensätze.

4.4 Text Mining

Um Text Nachrichten aus den Chats auswerten zu können, verwendet diese Arbeit Text Mining Methoden, die einführend in Abs. 2.4 beschrieben wurden. Die meisten implementierten Text Mining Methoden eigenen sich, um die Text Nachrichten aus den Chats für andere *ML*-Verfahren aufzubereiten und werden daher eher als Datenaufbereitungs-Methoden verstanden.

Einige dieser Verfahren können aber auch eingesetzt werden, um ein zunächst grobes Verständnis über die Inhalte in den Chats zu erlangen. Es ist möglich, mit diesen Methoden Themen zu extrahieren, über die diskutiert wird, aber es ist schwer zu beurteilen, worum es bei der inhaltlichen Diskussion geht und wie die Wertungen der einzelnen Autoren zu bestimmten Themen sind. Eine Evaluation der Auswertungen ist ebenfalls extrem schwierig, vgl. Abs. 5.

4.4.1 Verwertbarkeit von Textnachrichten

Initial sollte darauf eingegangen werden, dass nicht alle *Telegram*-Nachrichten einen Text beinhalten, der für die Auswertung mit den implementierten Text Mining Methoden geeignet ist. Wie bereits einführend in Abs. 4.2.1 beschrieben, können Umlaute wie ä oder Buchstaben wie ß manche Python-Implementierungen stören, da diese nicht in vielen Standard-Zeichensätzen wie ASCII enthalten sind.

Erschwerend kommt hinzu, dass diese Text Mining Methoden auf *Telegram* Nachrichten angewendet werden. In vielen sozialen Netzwerken, wie auch in *Telegram*, ist es üblich *Emoticons* wie `:)` zu benutzen. Auf diese verwendenden *Emoticons* und auf Formatierungen

wie **fette Passagen** in einer Nachricht kann im Rahmen dieser nicht eingegangen werden, da dies sonst den Rahmen sprengen würde, vgl. Kapitel 5.

Um mit dem Text aus den heruntergeladenen Chats einfach arbeiten zu können, stehen uns mehrere *Aufbereitungsstufen* zur Verfügung, vgl. *Stufe 3b* in Abs. 4.2.4, die sich wie folgt zusammenfassen lassen:

- `ftTdCleanText`: Nachrichten-Text aus `ftTdText` ohne Emoticons und ohne deutsche Zeichen und Zeilenumbrüche.
- `ftTdSafeText`: Nachrichten-Text aus `ftTdCleanText` ohne Sonderzeichen wie Punkte.
- `ftTdSafeLowerText`: Nachrichten-Text aus `ftTdSafeText`, abgebildet auf Text ohne Groß- und Kleinschreibung.

Es stellt sich bereits bei der Berechnung der Anzahl der Zeichen (`ftTdTextLength`) in einer Text-Nachricht die Frage, welche Abbildung für diese verwendet wird.

Die Abb. 4.7 visualisiert die Text-Nachrichten-Länge als Histogramm. Ein Histogramm ist eine Darstellung der Häufigkeitsverteilung und eignet sich in diesem Fall dazu, einen Eindruck über die Anzahl von Zeichen (Nachrichtenlänge) in *Telegram* Nachrichten zu gewinnen. Um ein aussagekräftigeres Ergebnis zu erhalten, wurden sogenannte Ausreißer entfernt, d.h. es es wurden keine Nachrichten betrachtet, deren Nachrichtenlänge um mehr als das Dreifache der Standardabweichung von dem Mittelwert abweichen [Bun21b, 95-97].

In den Notebooks werden abhängig vom Anwendungsfall verschiedene Text-Abbildungen (`ftTdCleanText`, `ftTdSafeText` und `ftTdSafeLowerText`) verwendet. Generell lässt sich aber sagen, dass das Kriterium, ob ein Text in spezifischen Nachrichten ausgewertet wird, über das *Query-Feature* `ftQrIsValidText`, vgl. *Stufe 3c* Abs. 4.2.4, gesteuert wird. Technisch beschrieben muss `ftTdTextLength` berechnet auf `ftTdCleanText` immer `>3` sein, falls nicht der gesamte Text in einem Chat betrachtet wird [Bun21b, 34].

4.4.2 Word Clouds

Um sich dem Chat-Inhalt [Lan13] zu nähern, wurden alle Texte aus den Nachrichten in einem Chat aggregiert, d.h. die *Telegram* Chats wurden jeweils auf einen kompletten Text abgebildet [Bun21b, 98].

Diesen Texte kann man nun als **Word Cloud** darstellen, also einer Art Schlagwortwolke. Auf die genaue Implementierung [Bun21b, 98-99] der in der Arbeit generierten *Word Clouds* kann im Rahmen dieser Arbeit nicht eingegangen werden, wohl aber auf die Interpretation. In der Diplomarbeit [Lan13] kann mehr dazu nachgelesen werden. Um *Word Clouds* zu generieren, wurde auf die gleichnamige Python-Bibliothek *WordCloud*, vgl. 3.2.2, zurückgegriffen.

Word Clouds sollen wichtige *Token* in einem Text hervorheben. Häufig verwendete *Token* werden in diesen Schlagwortwolken fetter dargestellt als weniger häufig verwendete *Token*. In der Abbildung 4.8 wird eine solche *Word Cloud* über den Chat von Attila Hildmann

Abbildung 4.8 Eine *Word Cloud* über den gesamten Chat von *Attila Hildmann*.

dargestellt. In den Notebooks werden alle Chats aus `DataSet0` als *Word Clouds* dargestellt [Bun21b, 100-104]. Es empfiehlt sich, auch die Auflistungen der 20 häufigsten *Token* anzusehen, da diese auch die Häufigkeit mit angeben.

Ihm Rahmen dieser Arbeit wurde festgestellt, dass eine *Word Cloud* über einen gesamten Chat nicht aussagekräftig ist, da sich zu viele Informationen in einem gesamten Chat befinden und diese sich schwer strukturiert abbilden lassen. Auf der Plattform wurden daher *Word Clouds* bereitgestellt, die Chats im zeitlichen Verlauf visualisieren, d.h. es wurden *Word Clouds* jeweils für ein Intervall (1 Monat) aus dem Text generiert [Bun21b, 105-111]. Auf der Plattform ist es so möglich, die Chats interaktiv über *Word Clouds* nachzuverfolgen, vgl. Abs. 5. Auf die *Java-Script Implementierung* kann im Rahmen dieser Arbeit nicht eingegangen werden, diese steht aber auf der Plattform unter dem Namen **Auto Word Clouds** zur Verfügung.

Hinweis: Unter den vielen *Token*, die in einem Text vorkommen, sind meistens nur wenig für die Form der Darstellung relevante *Token* enthalten. Diese zu identifizieren und nicht relevante *Token* aus den Daten zu entfernen, ist ein komplexeres Verfahren, worauf in dieser Arbeit nicht ausreichend eingegangen werden kann. In den Notebooks wurden subjektiv *Stopwords* entfernt, vgl. 5, was aber für diese Darstellung im zeitlichen Rahmen technisch notwendig war, da häufige nicht relevante *Token* wie *ein* oder *das* die *Word Clouds* sonst stark verzerren würden.

4.4.3 N-grams

In Abs. 2.4.4 wurde bereits erläutert, was *N-grams* sind und das unter anderem die Reihenfolge der *Token* in einem Text von Relevanz ist. In dieser Arbeit wird auf die *N-gram-*

Implementierung von *NLTK* zurückgriffen, vgl. Abs. 3.2.2. Die *N-gram* Abbildung benötigt einen Text aufbereitet zu *Token*, vgl. *Tokenization* Abs. 4.2.1.

Mit *N-grams* ist in dem Social Media und Text Mining Kontext viel möglich. So könnte man beispielsweise Autoren von Nachrichten, vgl. Abs. 4.5, zuordnen, ohne das man diese kennt. Dadurch, dass man meistens nicht viele Daten über einen Autor hat, bieten sich vor allem *Monogramme*, *Bigramme* und *Trigramme* an, vgl. Abs. 2.4.4. Daher werden diese für das `DataSet0` in dem Jupyter Notebook [Bun21b, 112-116] dargestellt.

Im Rahmen dieser Arbeit kann nicht weiter auf *N-grams* eingegangen werden, die daher als Datenaufbereitungsverfahren für anschließende *ML*-Verfahren betrachtet werden. *N-grams* eignen sich aber generell dazu, Muster in Texten zu finden, da diese die Reihenfolge der *Token* berücksichtigen. Erwähnenswert an dieser Stelle ist, dass einen Text auf *Token* abzubilden kein einfaches Unterfangen darstellt, vgl. Abs. 2.4.1, und dabei viele Fehler durch falsche Annahmen gemacht werden können.

Falls das Ziel beispielsweise sein sollte, Muster in einem Text zu finden, kann es nicht sinnvoll sein, *Stopwords*, vgl. Abs. 2.4.2, aus dem Text herauszufiltern, da das Muster dann nicht mehr über die *N-grams* Abbildung abbildbar ist. In dieser Arbeit wurden daher zusätzlich *8-grams* betrachtet, da sie sich beim explorativen Vorgehen als interessant herausstellten. Wie in Auflistung 4.9 ersichtlich, könnte man auch mit dieser Abbildung nach Mustern, also häufigen Aussagen in einem Text, suchen, falls man diese der Häufigkeit nach sortiert ausgibt. Im Notebook befinden sich weitere *N-gram*-Repräsentationen der Text Nachrichten der jeweiligen Chats [Bun21b, 117-122].

```
1 Analyse now >>DS-05-01-2021/ChatExport_2021-01-05-hildmann<<
2 (('5', 'gutschein', 'fuer', 'alle', 'die', 'zum', 'ersten', 'mal'),
     148)
3 (('gutschein', 'fuer', 'alle', 'die', 'zum', 'ersten', 'mal', 'bei
    '), 148)
4 (('fuer', 'alle', 'die', 'zum', 'ersten', 'mal', 'bei', 'mir'),
    148)
5 (('alle', 'die', 'zum', 'ersten', 'mal', 'bei', 'mir', 'bestellen')
    , 128)
6 (('die', 'zum', 'ersten', 'mal', 'bei', 'mir', 'bestellen', 'pc'),
    125)
7 (('zum', 'ersten', 'mal', 'bei', 'mir', 'bestellen', 'pc', '
    topbioneu'), 125)
```

Auflistung 4.9 *8-grams* über den gesamten Chat von *Attila Hildmann*.

4.4.4 POS-Tagging - Eigennamen

Was *POS-Tagging* ist, wurde bereits in Abschnitt 2.4.6 erläutert. Wie das *POS-Tagging* in einer solchen Anwendung für die deutsche Sprache aussehen könnte, wurde bereits in Abschnitt 4.2.1 beschrieben. In dieser Arbeit wurde der POS-Tagger *HanTa*, vgl. Abs. 3.2.2, verwendet.

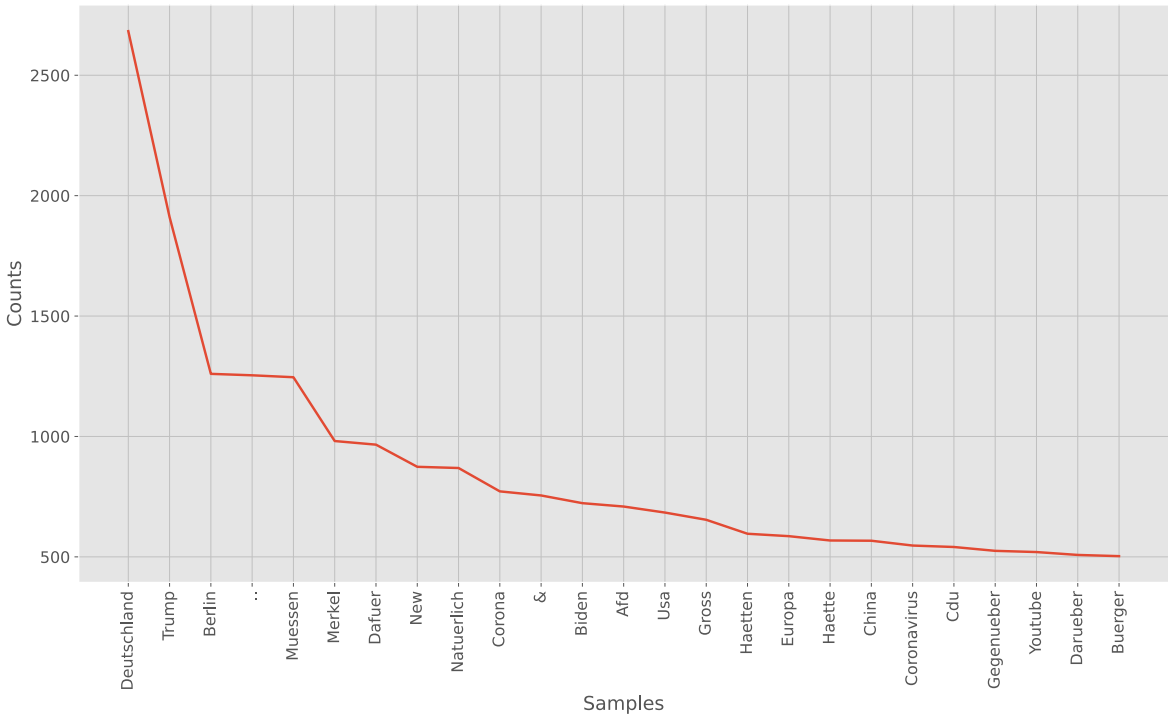

Abbildung 4.9 Top 25 der am häufigsten verwendeten Eigennamen von *Oliver Janich*. Einschließlich von Fehlerkennungen von *HanTa*.

HanTa ist in der Lage, Eigennamen in einem Text zu identifizieren. Diese könnten von Interesse sein, falls man beispielsweise daran interessiert ist, welche Themen, Personen oder Firmen in dem Text erwähnt werden.

Im Notebook wurden die Chats wie bei den *N-grams* und den *Word Clouds* zunächst auf einen Text abgebildet. Der Text wird über den *NLTK Tokenizer* zunächst auf *Token* abgebildet, die anschließend über den *HanTa* POS-Tagger getaggt werden, vgl. *HanTag* 4.2.1.

Da *HanTa* in der Lage ist, Eigennamen (Klasse `NE`) in einem Text zu identifizieren, ist es möglich, die häufigsten Verwendungen von Eigennamen in einem Chat über diese Methode ausfindig zu machen. Abbildung 4.9 stellt die 25 am häufigsten verwendeten Eigennamen von *Oliver Janich*, identifiziert über *HanTa* als Plot dar [Bun21b, 123-128].

4.4.5 Named-entity recognition (NER)

Ziel der Named-entity recognition (*NER*)ist es, relevante Entitäten in einem Text zu identifizieren, vgl. Abs. 2.4.7.

Bereits mit dem *POS-Tagging*-Verfahren *HanTa* ist es möglich, zumindest Eigennamen zu erkennen, vgl. Abs. 4.4.4. In diesen Notebooks wurden aber zusätzlich noch Sprachmodelle (Transformers) wie *BERT* und *RoBERTa* herangezogen, die auch für den Zweck einer *NER* trainiert worden sind, vgl. Abs. 3.2.3.

Wie in *Stufe 3d* in Abs. 4.2.4 bereits beschrieben, bilden unterschiedliche Sprachmodel-

le verschiedene Arten von Entitäten (Klassen) ab. Diese wurden bei der vorangegangen Datenaufbereitung auf ein einheitliches Format abgebildet.

Um den Rahmen nicht zu sprengen, kann in dieser Arbeit nicht auf die vollständigen *NER*-Auswertungen über die Text-Nachrichten in den Chats eingegangen werden, diese finden sich aber im Notebook [Bun21b, 129-134].

Auflistung 4.10 zeigt eine Ausgabe von den *NER*-Sprachmodellen *BERT* und *RoBERTa*, vgl. Abs. 3.2.3, angewendet an dem Satz "Hallo, mein Name ist Maximilian Mustermann und ich lebe in Deutschland in Europa."

```
'Hallo, mein Name ist Maximilian Mustermann und ich lebe in
   Deutschland in Europa.'

- BERT (bert_de_ner)

{'per': ['maxim', '##ilian', 'mus', '##ter', '##mann'],
'misc': [],
'org': [],
'loc': ['deutsch', '##land', 'europa']}

- RoBERTa (xlm-roberta-large-finetuned-conll03-german)

{'per': ['Maxim', 'ili', 'an', 'Must', 'mann'],
'misc': [],
'org': [],
'loc': ['Deutschland', 'Europa']}
```

Auflistung 4.10 *NER BERT* und *RoBERTa* an einem Beispielsatz.

4.4.6 Sentiment analysis

Wie bereits in Abschnitt 2.4.8 beschrieben, handelt es sich bei der *Sentiment analysis* (dt. "Stimmungs-Analyse") um ein "subjektives" Klassifikationsverfahren. Bei diesem man versucht man, einem Text eine bestimmte Stimmung wie "positiv" oder "negativ" zuzuordnen.

In dem Notebook wurden die Python Bibliothek *TextBlob*, vgl. Abs. 3.2.2, und das Sprachmodell *bert-base-multilingual-uncased-sentiment*, vgl. Abs. 3.2.3, verwendet. *TextBlob* wurde verwendet, da *NLTK* für die deutsche Sprache keine *Sentiment analysis* zur Verfügung stellt.

Da die verwendeten Komponenten die *Stimmungen* in unterschiedliche Klassen einteilen, müssen diese separat betrachtet werden. Da aber beide Komponenten die Stimmungen als numerischen Wert mit unterschiedlichen Werte-Bereichen ausdrücken, ist es möglich, die Stimmungen, die den Nachrichten zugeordnet sind, in einem zeitlichen Verlauf ähnlich wie in Abs. 4.3.4 darzustellen. Um die Stimmungen anschaulicher (aber nicht korrekter), zu visualisieren, wurden diese mithilfe des *Gleitenden Mittelwerts* geglättet. In Abbildung 4.10 wird die Stimmung, klassifiziert durch *BERT*, von `DataSet0` in einem zeitlichen Verlauf

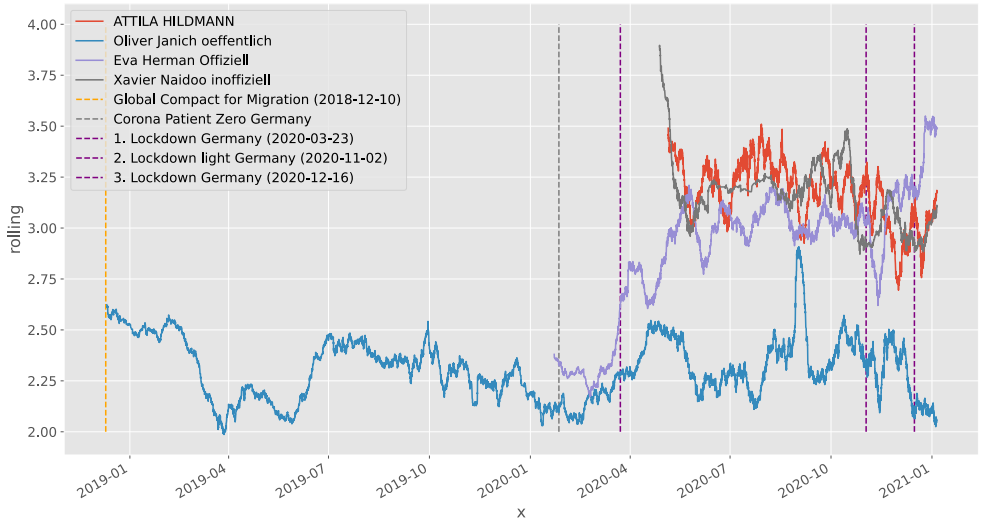

Abbildung 4.10 Zeitlicher Verlauf der Stimmung. Klassifiziert durch *BERT* (*DataSet0*).

dargestellt [Bun21b, 135-146].

TextBlob

```
'Heute ist ein toller Tag. Ich freue mich hier zu sein!'
{'polarity': 0.5, 'subjectivity': 0.0}

'Heute war ein furchtbarer Tag. Ich hasse alles.'
{'polarity': -0.5, 'subjectivity': 0.0}
```

Auflistung 4.11 *TextBlob Sentiment analysis* angewendet an zwei Beispielsätzen.

TextBlob gibt für die *Sentiment analysis*, vgl. Abs. 3.2.1, zwei Metriken an: die **polarity** (dt. "Polarität", auch "Gegensätzlichkeit") zwischen $[-1;1]$, wobei -1 für eine negative Nachricht und 1 für eine positive Nachricht steht, 0 wäre demnach neutral; und die **subjectivity** (dt. "Subjektivität") zwischen $[0;1]$, wobei 0 sehr objektiv wäre und 1 sehr subjektiv.

In der Auflistung 4.11 wird die *Sentiment analysis* von *Text Blob* an zwei Beispielsätzen angewendet [Bun21b, 136-137].

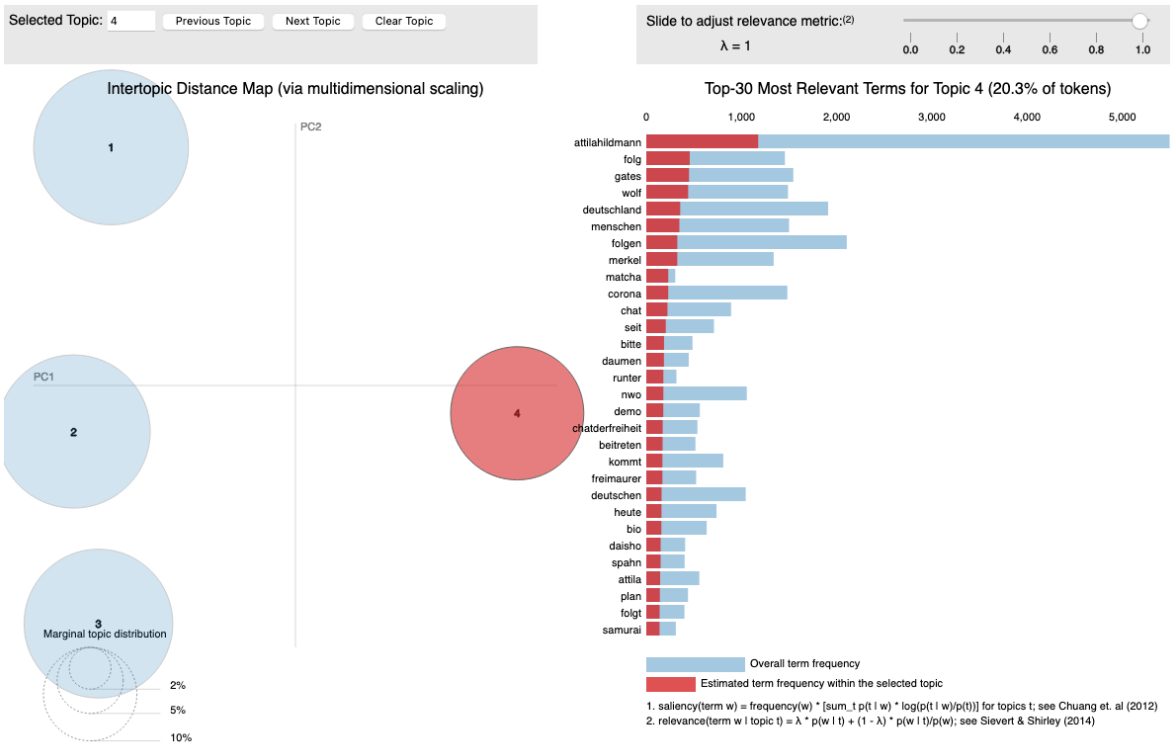

Abbildung 4.11 Visualisierung der *Latent Dirichlet Allocation (LDA)* durch *Gensim* mit *PyLDAvis* (*Attila Hildmann* - 4 *Topics*).

Transformers

```
'Das ist toll. Ich würde es mir wieder kaufen!'
5

'Das ist toll. Ich würde es aber nicht mehr kaufen!'
3

'Das funktioniert nicht.'
1
```

Auflistung 4.12 *BERT Sentiment analysis* angewendet an an Beispielsätzen.

Wie bereits in Abschnitt 3.2.3, ist das Besondere an dem eingesetzten Sprachmodell *bert-base-multilingual-uncased-sentiment*, dass dieses an Produktbewertungen trainiert worden ist und daher einen *Score* zwischen $[1;5]$ vergibt. In Auflistung 4.12 wurde dies an drei Beispielsätzen demonstriert [Bun21b, 142-144].

4.4.7 Latent Dirichlet Allocation (LDA)

Wie bereits in Abs. 2.4.9 beschrieben, bedient sich diese Arbeit der *LDA* als Vertreter eines *Topic Model* Verfahrens.

Um die *LDA* sinnvoll anwenden zu können, ist es zunächst notwendig, sich mit den Parametern und der Funktionsweise dieses Verfahrens auseinanderzusetzen. In dieser Arbeit wurde für die Implementierung der *LDA* auf die Python-Bibliothek *Gensim* zurückgriffen, da diese im Gegensatz zu *TextBlob* und *NLTK* die *LDA* implementiert hat [Bun21b, 147-164].

Die durch *Gensim* implementierte *LDA* benötigt wenig überraschend die *Anzahl der Topics*. In dem Notebook wurde daher abhängig von der Chat-Art mit verschiedenen Anzahlen experimentiert [Bun21b, 158-164].

Die Auswertungen von *Gensim* können durch die Python-Bibliothek *PyLDAvis*, vgl. Abs. 3.2.2, auf eine Webseite (HTML) abgebildet werden [Bun21b, 156]. Verschiedene Auswertungen stehen auf der Plattform zur Verfügung [Bun21c].

Es empfiehlt sich, *Stopwords*, vgl. Abs. 2.4.2, aus den zu betrachtenden Texten zu entfernen [Bun21b, 149]. Neben der *Anzahl an Topics* benötigt *Gensim* einen Text aufbereitet zu einer *BOW*-Darstellung, vgl. 2.4.3, und ein *Dictionary*, um die IDs der *Token* bei der Visualisierung wieder auf *Token* zurückzuführen. Um sich die *BOW*-Darstellung und das *Dictionary* besser vorstellen zu können, werden in dem Notebook zwei Beispielnachrichten auf diese abgebildet [Bun21b, 150-154].

4.5 Identifizierung von Autoren

Telegram übermittelt abhängig von der Chat-Art die Autoren einer Nachricht, vgl. Abs. 3.3. Bei *Channels* ist es beispielsweise nur möglich, den Autor einer Nachricht zu identifizieren, wenn ein Administrator das eingestellt hat.

Die *Social Graphs* haben gezeigt, dass die *Akteure* von `DataSet0` aus vielen *Channels* weiterleiten, die den Autor einer Nachricht nicht bekannt geben, vgl. Abs. 4.3.2. Daher stellt sich die Frage, ob es trotzdem möglich ist, den Autor einer Nachricht zu identifizieren, auch wenn keine direkten Meta-Informationen über diesen vorliegen. Eine Möglichkeit sich dieser Thematik zu nähern wäre, die Texte aus den Nachrichten auf Features abzubilden und schließlich mithilfe eines oder mehreren *ML*-Klassifikationsmodellen zu versuchen, diese auf Autoren abzubilden. Diese Modelle müssten allerdings vorher erst (evtl. lange, abhängig von Daten und Verfahren) trainiert und evaluiert werden. Es stellt sich wie bei allen Text Mining Methoden zunächst allerdings die Frage, wie der Text auf Features abgebildet werden soll, also was damit später erreicht werden soll.

In dieser Stufe wurden die bereits implementierten *ML*-Klassifikationsmodelle der Python-Bibliothek *Scikit-learn* verwendet, vgl. Abs. 3.2.1.

Hinweis: Diese Stufe ist ein eigenes Jupyter-Notebook [Bun21a]; es empfiehlt sich erneut das Notebook heranzuziehen. Die Abschnitte *Arbeitsumgebung initialisieren* und *Chats laden und aufbereiten* werden in diesem Abschnitt übersprungen, da diese bereits ausführlich in Abs. 4.1 und Abs. 4.2 beschrieben wurden und das Notebook mit bereits vorverarbeiteten Daten arbeiten kann [Bun21a, 13-16].

4.5.1 Chats aufbereiten

Die Texte aus den Nachrichten müssen gesondert für die Klassifikationsmodelle aus *Scikit-learn* aufbereitet werden. Dafür stehen verschiedene Implementierungen zur Verfügung, diese Arbeit verwendet dabei den `CountVectorizer` und den `TfidfTransformer`:

- `CountVectorizer`: Bei dieser Darstellung wird der Text auf die *Bag-Of-N-grams* Repräsentation abgebildet, vgl. 2.4.4. In dieser Arbeit wurde mit *Monogrammen*, *Bigrammen* und *Trigrammen* experimentiert [Bun21a, 17-21].
- `TfidfTransformer`: Die sogenannten *Count Matrizen* von `CountVectorizer` werden durch dieses Verfahren auf *tf-idf* Matrizen abgebildet, vgl. Abs. 2.5.3, also auf Matrizen, die die häufigsten *N-grams* von Autoren abbilden [Bun21a, 22-24].

Da bei dieser Vorgehensweise mit *N-grams* gearbeitet wird, wurden keine *Stopwords* entfernt, vgl. 2.4.4. Da *Emoticons* und deutsche Umlaute manche Python-Implementierungen stören können, da diese in vielen Standard-Zeichensätzen wie ASCII nicht enthalten sind, wurden diese entfernt bzw. ersetzt. Dabei wurde auf das Feature `ftTdCleanText`, vgl. *Stufe 3b* Abs. 4.2.4 zurückgegriffen.

Die zu betrachtenden Chats sind unterschiedliche "groß", d.h. die Anzahl an Textnachrichten, die je nach Autor verarbeitet werden können, variiert stark [Bun21a, 25-26]. Das könnte bei dem Trainieren und Evaluieren von verschiedenen Klassifikationsmodellen *Scikit-learn* problematisch werden. Daher wurden im Notebook zufällig (`getSamples(df, k=9103)`) Strichproben aus den Nachrichten genommen, um gewährleisten zu können, dass die Autoren gleichmäßig in den Daten vertreten sind [Bun21a, 27-30].

Ein Computer kann nur mit numerischen Werten arbeiten, daher müssen die Autoren auf ein sogenanntes *Dictionary* abgebildet werden, d.h. den Autoren werden IDs zugewiesen. Dieses *Dictionary* wird später beim Evaluieren dazu genutzt, die ausgegebenen IDs wieder einem Autor zuzuordnen [Bun21a, 31-32].

4.5.2 Trainieren und Evaluieren

Das Trainieren und Evaluieren von solchen Klassifikationsmodellen ist ein langwieriger Prozess, auf den im Rahmen dieser Arbeit nur einführend eingegangen werden kann, vgl. Abs. 2.2 und Kapitel 3.

Scikit-learn implementiert verschiedene *ML*-Modelle, in dieser Arbeit kann nur mit einem Teil dieser experimentiert werden. Im Rahmen dieser Arbeit kann nicht auf die Parameter und Funktionsweise dieser Modelle eingegangen werden, daher wurde auf Standard-Konfigurationen zurückgegriffen, die sich einführend eignen, verschiedene Verfahren gegenüberzustellen.

Zunächst ist es erforderlich, die Daten einerseits in eine **Trainingsmenge** zu zerlegen, um mithilfe dieser Daten die Modelle trainieren zu können, und andererseits in eine **Testmenge**, um die trainierten Modelle evaluieren zu können. Dabei wurden einfach 20% der Daten als *Testmenge* definiert. Es wurde darauf geachtet, dass Autoren in beiden Mengen gleichmäßig vertreten sind [Bun21a, 33-36].

Tabelle 4.10 Evaluation der trainierten Klassifikationsmodelle.

Bezeichner	Score %	Beschreibung
`LinearSVC`	70	Support Vector Machine[a]
`MultinomialNB`	63	Naive Bayes classifier[b]
`LogisticRegression`	68	Logistic regression[c]
`MLPClassifier`	69	Multi-layer Perceptron classifier[d]
`DecisionTreeClassifier`	55	Decision Tree classifier[e]
`RandomForestClassifier`	63	Random Forest classifier[f]
`DummyClassifier`	25	Dummy classifier[g]

a `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`
b `https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html`
c `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`
d `https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html`
e `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`
f `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`
g `https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html`

Es wurden verschiedene Modelle trainiert [Bun21a, 37-44], die in Tabelle 4.10 mit ihrem jeweiligen *Score* dargestellt werden. Ein **Score** ist in diesem Kontext eine Kennzahl, um einschätzen zu können, wie "erfolgreich" ein Modell auf der *Testmenge* bei der Identifizierung von Autoren war. Der `DummyClassifier` ist kein Klassifikationsmodell, das in einer Produktivumgebung eingesetzt werden sollte, da dieses Modell aufgrund von trivialen Regeln entscheidet, was mit "Raten" vergleichbar ist. Dieses Modell wird aber gerne als Referenzmodell herangezogen.

Auf den *Score* wird nicht weiter eingegangen, da diese Kennzahl in diesem Kontext nicht ausreichend aussagekräftig ist, da anhand dieser beispielsweise nicht erkannt werden kann, ob das trainierte Modell dazu neigt, einen bestimmten Autor häufiger zu erkennen oder zu verwechseln. Um ein solches Modell sinnvoller evaluieren zu können, bietet sich die sogenannte **Confusion matrix** an: Bei dieser Visualisierung ist es möglich abzulesen, welche Autoren jeweils richtig und falsch zugeordnet wurden. Außerdem ist es möglich herauszufinden, welcher Autor mit welchem Autor häufig verwechselt worden ist. Abbildung 4.12 visualisiert eine solche *Confusion matrix*.

In dem Notebook hat das Klassifikationsmodell *Support Vector Machine* für diesen Anwendungsfall am besten abgeschnitten und ist Vergleich zu einem neuronalen Netzwerk *Multi-layer Perceptron* schneller zu trainieren und benötigt weniger Daten.

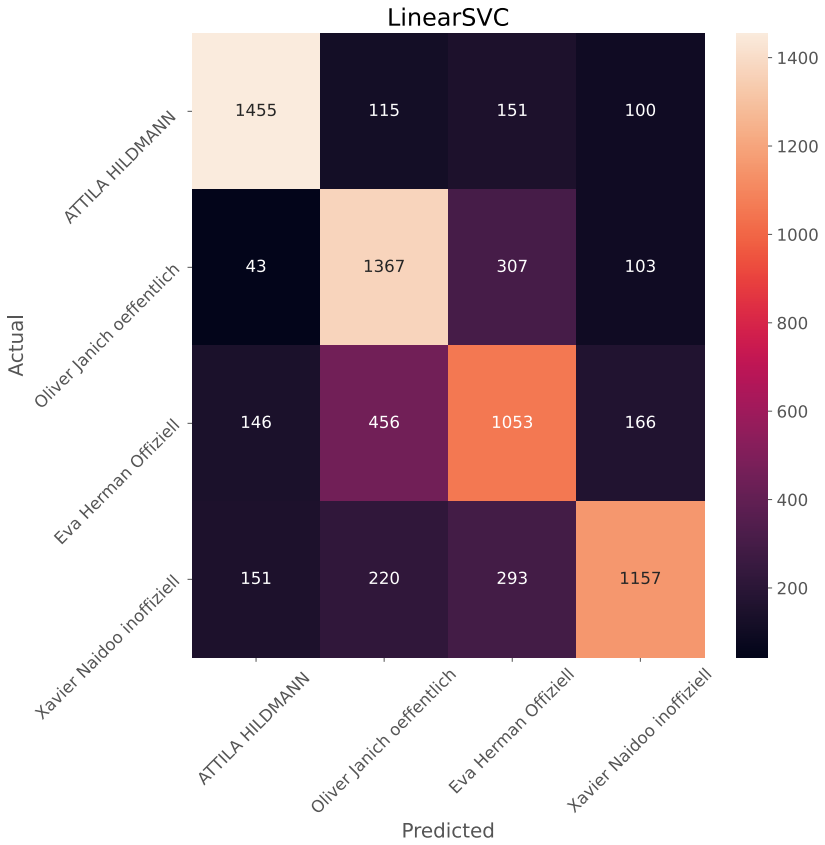

Abbildung 4.12 *Confusion matrix LinearSVC (DataSet0).*

5 Fazit und Ausblick

In dieser Arbeit wurden verschiedene Social Media und Text Mining Methoden auf zuvor heruntergeladene Chats angewendet. Bereits die Auswahl der zu betrachtenden Chats stellt eine große Herausforderung dar, da man nicht gewährleisten kann, alle notwendigen *Akteure* und *Interessengruppen* herangezogen zu haben. In dieser Arbeit wurde daher eine Ausgangsbasis und ein zunächst triviales Suchmuster definiert. Die einleitende Zielsetzung ist nicht ausreichend konkret definiert für einen Data Mining Prozess. Daher wurden verschiedene Fragen definiert, die mithilfe der implementierten Werkzeuge beantwortet werden sollen, vgl. Abs. 3.3.1.

Die initiale explorative Vorgehensweise in der Arbeit, die in einem Data Science Prozess nicht unüblich ist, zeigt, dass es zunächst einfach ist Social Media und Text Mining Methoden auf Daten anzuwenden, also damit die Chats aufzubereiten und zu modellieren bzw. zu visualisieren. Jedoch erfordert die vollständige Interpretation und Evaluation der Auswertungen einen umfangreicheren Blick als in dieser Arbeit gewährleistet werden kann. Daher wurde die **Telegram Analyse Plattform** [Bun21c] geschaffen, mit der es möglich sein soll, die implementierten Social Media und Text Mining Werkzeuge auf Telegram-Chats anzuwenden, vgl. Abs. 3.3. So können Personen wie Data Scientists, die an Social Media bzw. Text Mining Methoden interessiert sind, und Personen wie Sozialwissenschaftler, die an Auswertungen und Analyse-Möglichkeiten besagter Gruppen interessiert sind, an diesem Vorhaben mitarbeiten. Langfristig soll so ermöglicht werden, die definierte Zielsetzung zu erfüllen.

Folglich richtet sich diese Masterarbeit vor allem an Personen, die an dem Projekt mitarbeiten möchten bzw. an den Auswertungen und Analyse-Möglichkeiten interessiert sind. So möchte diese Arbeit einen Überblick über Social Media und Text Mining Methoden am Beispiel von *Telegram* bieten und geht daher erst auf die **Grundlagen** und auf den Messenger *Telegram* ein. Analog zu dem generischen Data Mining Prozess *CRISP-DM*, vgl. Abs. 2.2, beschreibt diese Arbeit die initiale Vorgehensweise, die aus den Phasen **Business Understanding** und **Data Understanding**, vgl. Kapitel 3, entstand. Die Stufen **Data Preparation** und **Modeling** wurden in dieser Arbeit auf zwei Jupyter-Notebooks abgebildet, die in Kapitel 4 ausführlich beschrieben werden.

Dem "findigen Leser" fällt sofort auf, dass nach diesem Prozess das Kapitel **Evaluation** fehlt. Aus den oben genannten Gründen ist es im Rahmen dieser Arbeit nicht möglich, die Auswertungen ausreichend zu interpretieren und zu evaluieren. So war es also nicht möglich, die definierten und weiterentwickelten Fragen im Rahmen einer wissenschaftlichen Arbeit beantworten zu können. Die Phase **Deployment** wird in diesem Kontext auf die *Telegram Analyse Plattform* abgebildet.

Da diese Arbeit verschiedene Personen ansprechen soll, war es nicht immer einfach, einen Kompromiss zwischen fachlicher Tiefe und Verständlichkeit herzustellen. Daher wurde

in dieser Arbeit häufig auf weiterführende Literatur verwiesen und die verwendeten Methoden mit deutlichem Fokus auf *Telegram* beschrieben. Die Struktur des Kapitels 4 wurde stark durch die Strukturen der implementierten Jupyter-Notebooks [Bun21b] [Bun21a] beeinflusst, die in einer explorativen Vorgehensweise entstanden und im Rahmen dieser Arbeit stark strukturiert worden sind (*Refactoring*).

5.0.1 Methoden

In einem solchen Data Mining Prozess ist es notwendig, Annahmen über die Daten und das Problem zu treffen und sich zu überlegen, wie man die Daten aufbereitet, modelliert, visualisiert und evaluiert. Wie in einem Data Mining Prozess ebenfalls nicht unüblich, wurde zunächst explorativ vorgegangen, um so das Problem und die zu Verfügung stehenden Daten zu verstehen.

In dieser Arbeit wurden verschiedene Methoden, Modelle und Visualisierungs-Möglichkeiten betrachtet und implementiert, die stark von ihrem Anwendungsfall, also von ihrem Problem und Ziel, abhängen. Bereits bei der Aufbereitung der deutschen Sprache mit ihren Eigenheiten und dem Auswerten von Nachrichten mit *Emoticons* und bestimmten Formatierungen, vgl. Abs. 4.2.1, können viele Fehler gemacht werden, die später die Auswertungen maßgeblich beeinflussen werden. Auch welche *Stopwords* entfernt werden, welcher *Tokenizer*, *Lemmatizer* und welcher *POS-Tagger* verwendet wird, hängt maßgeblich vom Ziel ab, wie in dieser Arbeit beschrieben.

Viele der implementierten Data Mining Methoden sind schwer ausreichend tief zu beschreiben, über viele dieser Methoden könnte man eigene Abschlussarbeiten schreiben. Diese Arbeit konzentriert sich daher mehr auf die Umsetzung, Einschränkungen und Implementierung verschiedener Methoden und soll dazu anregen, sich weiterführende Gedanken zu diesen Verfahren zu machen. Über die Wahl der Parameter und die Implementierungen in den Notebooks konnte im Rahmen dieser Arbeit nicht ausreichend eingegangen werden. Nicht alle implementierten Methoden sind wissenschaftlich betrachtet aussagekräftig, was aber auch der Zielsetzung und Idee hinter dieser Arbeit nicht im Wege steht.

In Data Mining Prozessen ist es ebenfalls üblich, meist mathematische Verfahren zweckzuentfremden und mit diesen zu experimentieren. So wurde beispielsweise die *LDA* auf die Chats angewendet, obwohl diese eigentlich einem ganz anderen Zweck dient, vgl. 2.4.9. Auf bestimmte Auswertungen konnte im Rahmen dieser Arbeit nicht weiter eingegangen werden, da diese in einem wissenschaftlichen Kontext nicht aussagekräftig sind, falls nicht ihre Referenzmodelle betrachtet und evaluiert werden, was der Rahmen dieser Arbeit nicht sinnvoll zulässt. Das trifft beispielsweise auf *NER*-Auswertungen der Sprachmodelle (Transformers), vgl. 4.4.5, und auf die *Sentiment analysis* von *TextBlob* zu, vgl. Abs. 4.4.6.

Bei den heruntergeladenen Chats wurde darauf geachtet, dass die Chats im gleichen Zeitraum heruntergeladen worden sind. Es ist aber wichtig zu verstehen, dass diese Chats aus Sicht eines zufälligen *Telegram* Benutzers abgebildet werden und daher nicht auf alle Nachrichten zugegriffen werden kann, da dieser beispielsweise auf Inhalte aus privaten *Telegram*-Gruppen nicht zugreifen kann, vgl. Abs. 3.1. Das sollte beim Betrachten der Auswertungen im Hinterkopf behalten werden.

5.0.2 Ausblick

```
1 'Hallo, mein Name ist Max und ich esse gerne Eis. Ich schreibe
     gerade an meiner Masterarbeit und teste neue Verfahren. Ich
     komme aus dem Grossraum München und bin Informatiker.'
2
3 - gpt2 cMaxLength = 100
4 'Hallo, mein Name ist Max und ich esse gerne Eis. Ich schreibe
     gerade an meiner Masterarbeit und teste neue Verfahren. Ich
     komme aus dem Grossraum München und bin Informatiker.\nWir haben
      ein Haus für Kinder mit 2 oder 3 Jahren.\nEin Teil davon wird
     in eine katholische Kirche in der Nähe von München umziehen, wo
     sie eine Kirche namens Gittarnier errichten wird.\nSie ist sehr
     gut gefüllt, in einem sehr hellen und freundlichen Raum (in
     diesem Haus ist Platz für ca.'
5
6 - gpt2-faust cMaxLength = 100
7 'Hallo, mein Name ist Max und ich esse gerne Eis. Ich schreibe
     gerade an meiner Masterarbeit und teste neue Verfahren. Ich
     komme aus dem Grossraum München und bin Informatiker.
     Mephistopheles.O schaffe einen freien Raum, hebe dich nach dem
     Sinn der Welt auf. Ich geniesse die Welt und lass dich in ihr
     bestechen.Faust.Ich mag für meine Kunst kein Zauberwort, Und
     eine Welt, die jedem vertraut ist, Ist für mich kein Selbstzweck
     '
```

Auflistung 5.1 Beispiele: Textgenerierung mit *gpt2* und *gpt2-faust*.

Methoden aus dieser Arbeit können auf andere Felder übertragen werden, vgl. Abs. 2.3.1. Daher könnte es sich lohnen, einige von ihnen weiterzuentwickeln.

Erwähnenswert wären beispielsweise die *Social Graphs*, mit denen Kommunikationsstrukturen aus sozialen Netzwerken visualisiert werden können, 4.3.3. Die in der Arbeit gewählte 2D-Darstellung mit den gewählten Parametern kann diese Strukturen nicht ausreichend visualisieren. Eine interaktive 3D-Visualisierung würde diese Strukturen zum Beispiel exakter visualisieren. Stark vereinfacht ausgedrückt hat man das Problem zwischen einer korrekten Darstellung, also einer Darstellung, die die Realität möglichst genau abbildet, und einer übersichtlichen Darstellung, also einer Darstellung, die die Realität annähert, aber dafür leserlich ist. Es empfiehlt sich daher eher, auf die *Top-n* Darstellungen zurückzugreifen und die Werte von diesen in Relation zur Nachrichtenanzahl in einem Chat zu setzen. Dafür eigenen sich auch die Plots aus Abs. 4.3.1.

Word Clouds eignen sich dazu, einen Chat im zeitlichen Verlauf darzustellen ebenso wie die zeitlichen Darstellungen, vgl. Abs. 4.4.2 und 4.3.4. Die *Word Clouds* im zeitlichen Verlauf über die Chats können auf der Webseite der Plattform eingesehen werden [Bun21c].

In der Arbeit hat sich gezeigt, dass der *POS-Tagger HanTa* dafür geeignet ist, Eigenna-

men aus deutschen Texten zu erkennen, vgl. Abs. 4.2.1. Die in der Arbeit aufgezeigte *NER* von den Sprachmodellen (Transformers) ist ebenfalls in der Lage, Eigennamen aus deutschen Texten zu erkennen, vgl. 2.4.7. Es würde sich lohnen, diese beide Arten gegenüberzustellen. Gleiches gilt für die *Sentiment analysis* ausgeführt von den *Transformers* im Vergleich zu *TextBlob*, vgl. 4.4.6.

Das Suchmuster, das in der Arbeit dazu verwendet wurde, weitere Chats zu identifizieren, um diese anschließend auswerten zu können, geht technisch gesehen mehr in die Breite als in die Tiefe, was im Rahmen dieser Arbeit technisch notwendig war, vgl. *Suchmuster* Abs. 4.3.2, was die Auswertungen aber erheblich beeinflussen kann. Um das Vorhaben weiter zu verfolgen würde es sich daher lohnen, die *APIs* von *Telegram*, vgl. Abs. 3.1.1, anzubinden, um anschließend ein besseres Suchmuster einsetzen zu können. Im Rahmen dieser Arbeit konnte nur ein Teil der Attribute von *Telegram* betrachtet werden, vgl. Abs. 3.1.2, weitere Attribute sollten herangezogen werden. Erwähnenswert an dieser Stelle ist beispielsweise das Attribut `sticker_emoji`, vgl. A.1.2, bei dem *Telegram* die häufig versendeten "Sticker"[1] auf ASCII-*Emoticons* abbildet. Mithilfe dieses Attributs könnte beispielsweise noch die *Sentiment analysis* erweitert werden.

Die Identifizierung von Autoren aus Abs. 4.5 konnte nur einführend behandelt werden. So könnte man beispielsweise noch weitere Features, wie Rechtschreibfehler oder *Emoticon*-Kombinationen der Autoren heranziehen. Die Klassifikationsmodelle müssen noch weiter evaluiert und verbessert werden, um diese produktiv einzusetzen zu können; so müsste beispielsweise noch berücksichtigt werden, dass keiner der betrachteten Autoren die Nachricht geschrieben haben könnte.

Sprachmodelle wie *Transformers* können neben der *NER* und der *Sentiment analysis* auch Text generieren, wie in dem Notebook [Bun21b, 166-169] gezeigt, vgl. Auflistung 5.1, aber auch dazu verwendet werden, falsche Nachrichten (*Fakenews*) von echten Nachrichten zu unterscheiden. Auf Plattformen wie *Kaggle* stehen dafür Trainingsdaten bereit[2], häufig aber nur für den englischsprachigen Raum.

Um abschließend auf die definierten Fragen, vgl. 3.3.1, und auf das initiale Ziel der Arbeit zurückzukommen: Mithilfe der *Telegram Analyse Plattform*, dieser Arbeit und umfassenderen Ansichten sollen diese künftig evtl. beantwortet werden. Dabei sollten iterativ die *Ausgangschats*, *Fragen*, *Stopwords*, *Methoden*, *Implementierungen* aus dieser Arbeit erweitert und evaluiert werden.

1 siehe `https://tlgrm.eu/stickers`
2 siehe `https://www.kaggle.com/c/fake-news`

A Anhänge

A.1 Dokumentation Telegram

Wie in Abschnitt 3.1.2 beschrieben, lassen sich die Attribute von den heruntergeladenen Nachrichten in zwei Klassen unterscheiden.

A.1.1 Autor-spezifische Attribute

- `from` und `from_id` (z.B. `Max Mustermann` und `123`): Bezieht sich auf den In-Umlauf-Bringer der Nachricht. Im *Channel* ist es der *Channel*-Titel selbst. Dieses Attribut ist für diese Arbeit mit das Wichtigste.

- `forwarded_from` (z.B. `Max Mustermann` oder `Meine Chatgruppe`): Gesetzt, falls Nachricht weitergeleitet worden ist. Besonderheit: Zeigt bei Weiterleitung aus Gruppen immer auf den Gruppen-Namen und nicht auf den Autor der Nachricht. Dieses Attribut ist für diese Arbeit ebenfalls mit das Wichtigste.

- `author` (z.B. `Max Mustermann`): Falls ein *Channel* eingestellt hat [1], den Namen des Autors (in diesem Fall den des Admins) mitzusenden. Kommt selten vor, daher nicht beachtet.

- `actor` und `actor_id` (z.B. `Max Mustermann` und `123`): Falls Nachricht eine Service-Nachricht ist. Beschreibt Initiator der Aktion.

- `via_bot` (z.B. `CommentsBot`): Falls die Nachricht über einen *Bot*[2] versendet worden ist. *Bots* können in *Telegram* auch benutzt werden, um Reaktionen auf Nachrichten, wie "Gefällt mir" (`like`) abzugeben. Ein raffinierter *Bot* müsste aber nicht angeben, dass er ein *Bot* ist.

- `performer` (z.B. `Max Mustermann`): Gesetzt bei Sprachnachrichten und beschreibt den Autor. Kommt selten vor, daher nicht beachtet.

- `saved_from` (z.B. `Meine Chatgruppe`): Falls die Funktion "Speichern" verwendet und die Nachricht anschließend weitergeleitet wurde. Kommt selten vor, daher nicht beachtet.

- `inviter` (z.B. `Max Mustermann`): Gesetzt z.B. bei Service-Nachricht und `action` gleich `join_group_by_link` zeigt an, wer der "Einlader" war. Kommt selten vor, daher nicht beachtet.

1 siehe `https://telegram.org/blog/channels-2-0#admin-signatures`
2 z.B. `https://botostore.com/c/dooxbot/` oder `https://t.me/like`

A.1.2 Nachrichten-spezifische Attribute

- `id` (z.B. `0`): Eine eindeutige Identifizierungsnummer der Nachricht innerhalb des Chats.
- `type` (z.B. `message`): Art der Nachricht. Hier wird unterschieden zwischen `message`, einer gewöhnlichen Nachricht, und `service`, einer Service-Nachricht, vgl. Attribut `action`.
- `action` (z.B. `pin_message`): Gesetzt, falls es sich um eine Service-Nachricht, vgl. Attribut `type`, handelt. Definiert die Art der Service-Nachricht, z.B. `pin_message`, um dem Client mitzuteilen, eine Nachricht immer oben angepinnt anzuzeigen, oder `invite_members`, um anzuzeigen, welche Teilnehmer von den Administratoren eines Chats hinzugefügt worden sind.
- `date` (z.B. `2020-03-09T19:48:03`): Zeitstempel der Erstellung der Nachricht.
- `title` (z.B. `Meine Chatgruppe`): Gesetzt, falls es sich um eine Service-Nachricht z.B. vom `action` gleich `create_group` handelt. Definiert den Gruppentitel, also den angezeigten Gruppen-Namen.
- `text` (z.B. `Hallo, wie geht es dir?`): Beschreibt den Text-Inhalt einer Nachricht, wird in Abschnitt 3.1.3 aufgrund der Komplexität separat beschrieben, vgl. Abs. 3.1.3.
- `edited` (z.B. `2020-03-09T19:48:03`): Zeitstempel der letzten Veränderung der Nachricht: Es ist in *Telegram* möglich, Nachrichten nach dem Absenden der Nachricht zu verändern, die Version vor der Veränderung geht jedoch verloren.
- `file` (z.B. `stickers/sticker (161).webp`): Gesetzt, falls eine Datei, z.B. PDFs, Videos, Gif Dateien oder Sticker, aber nicht Fotos, vgl. Attribut `photo`, angehängt wurde und beschreibt den Dateipfad zur lokalen Datei. Besonderheit: Wird auch gesetzt, falls Dateien nicht wirklich heruntergeladen worden sind.
- `thumbnail` (z.B. `video_files/IMG_7539.MP4_thumb.jpg`): Gesetzt, falls ein Foto, vgl. Attribut `photo`, oder ein Video, vgl. Attribut `file`, und beschreibt den Dateipfad zur lokalen Vorschaubild (engl. Thumbnail) Datei. Besonderheit: Wird auch gesetzt, falls Dateien nicht wirklich heruntergeladen worden sind.
- `media_type` (z.B. `voice_message`): Gesetzt, falls ein Text-Inhalt nicht mit übermittelt worden ist, z.B. `video_file`, `animation`, `voice_message`.
- `mime_type` (z.B. `application/pdf`): Gesetzt, falls eine Datei, vgl. Attribut `file`, übermittelt worden ist und beschreibt den MIME-Type[3], z.B. `audio/ogg` oder `video/mpeg` einer Datei.
- `duration_seconds` (z.B. `20.0`): Gesetzt, falls ein Video oder eine Animation mit übermittelt wurde und beschreibt die Länge des Videos oder der Animation in Sekunden.
- `width` und `height` (z.B. `1280` und `720`): Gesetzt, falls ein Video oder Foto übermittelt wurde und beschreibt die Breite und Höhe eines Videos oder Fotos.

3 siehe `https://wiki.selfhtml.org/wiki/MIME-Type/`

- `photo` (z.B. `photos/photo_1@11-03-2020_19-39-39.jpg`): Gesetzt, falls ein Foto übermittelt worden ist, vgl. Attribut `file`, und beschreibt den Dateipfad zur lokalen Datei. Besonderheit: Wird auch gesetzt, falls Dateien nicht wirklich heruntergeladen worden sind.
- `reply_to_message_id` (z.B. `63211`): Gesetzt, falls auf eine Nachricht innerhalb des Chats direkt geantwortet wurde. Bezieht sich auf die `id` innerhalb des Chats.
- `poll`: Eine Umfrage (ignoriert in dieser Arbeit aus Zeitgründen). JSON-Objekt umfasst `question`, z.B. die Frage `Wie geht es dir heute?` und die Antworten `answers.text[]`, z.B. `Gut`.
- `sticker_emoji` (z.B. `<3` - ASCII Darstellung gewählt): Gesetzt, falls die Nachricht einen Sticker beinhaltet. *Telegram* abstrahiert auf ein einfaches Emoji (für z.B. Desktop-Clients, die Sticker nicht unterstützen oder für PUSH-Nachrichten-Vorschau).
- `message_id` (z.B. `63`): Gesetzt z.B. bei Service-Nachricht und `action` gleich `pin_message`, um auf die Nachricht zu referenzieren.
- `location_information`: Gesetzt, falls Standort-Daten an Nachricht geknüpft, JSON-Objekt umfasst z.B. `latitude` und `longitude`. Nicht weiter wichtig, extrem selten, aber evtl. nützlich.
- `members` (z.B. `[Max Mustermann, Erika Musterfrau]`): Gesetzt, falls es sich um eine Service-Nachricht z.B. vom `action` gleich `invite_members` handelt. Zeigt die eingeladenen Teilnehmer an.
- `game`: Ein Spiel (nicht relevant für Arbeit). JSON-Objekt umfasst z.B. `title`, den Spieltitel wie z.B. `LumberJack` und den Link zum Spiel `link`.
- `contact`: Angehängte Kontaktinfos, extrem selten. JSON-Objekt umfasst z.B. `vcard`, den Kontakt als vCard-Datei und z.B. `fist_name`.

A.1.3 Besondere Attribute in Textnachrichten

Die Liste ist lang, hier werden einige aufgezählt. Der Rest kann im Notebook [Bun21b] eingesehen werden:

- `bold`: **fett**
- `italic`: *kursiv*
- `underline`: <u>unterstrichen</u>
- `email`: Eine E-Mail Adresse. Werden besonders hervorgehoben.
- `link`: Eine URL Adresse. Werden besonders hervorgehoben.
- `text_link`: Eine URL Adresse mit Text. Werden besonders hervorgehoben.
- `hashtag`: Ein Hashtag `#baum`.
- `mention`: Eine Erwähnung `@MaxMustermann`.
- `phone`: Telefonnummern.
- `bank_card`: Bank-Informationen.

A.2 Betrachtete Datensätze

Stand der Abonnenten und Name: 21.03.2021

A.2.1 DataSet0

Stellte die Ausgangsbasis dar. Es wurden zunächst ausschließlich `channels` betrachtet, vgl. Tabelle A.1.

Tabelle A.1 Chats in DataSet0.

Vermutete Person	Name	Art	Abonnenten
Oliver Janich	"Oliver Janich öffentlich"	`channel`	164.328
Attila Hildmann	"ATTILA HILDMANN"	`channel`	118.514
Eva Herman	"Eva Herman Offiziell"	`channel`	176.646
Xavier Naidoo	"Xavier Naidoo (inoffiziell)"	`channel`	111.992

A.2.2 DataSet1

Stellen Ableitungen aus `DataSet0` dar. In diesem Datensatz werden nur `channels` betrachtet. Diese Datensätze spielen in dieser Arbeit eine untergeordnete Rolle. Daher werden aus Gründen der Übersicht diese Datensätze nur auf der Plattform beschrieben, vgl. Abs. 3.3.2.

A.2.3 DataSet1a

Stellen Ableitungen aus `DataSet0` dar. In diesem Datensatz werden nur `supergroups` betrachtet, also Diskussionsgruppen, vgl. Tabelle A.2.

Tabelle A.2 Chats in DataSet1a.

Name	Art	Abonnenten
"FREIHEITS-CHAT"	`supergroup`	33.420
"FREIHEITSCHAT - BLITZ"	`supergroup`	8.785
"Livestreaming für Deutschland, Österreich, ..."	`supergroup`	10.719

A.2.4 DataSet2

`DataSet2` umfassen Querdenken-Chats, vgl. Tabelle A.3.

A.3 Skripte

A.3.1 Ausführen des Telegram Notebooks mit Papermill

Beschreibt die Datei `notebooks/run-notebook.sh` (wurde für die bessere Lesbarkeit angepasst), vgl. Abs. 3.3.2:

Tabelle A.3 Chats in DataSet2.

Name	Art	Abonnenten
“Querdenken (089 - MÜNCHEN)	Diskussion & Austausch - Wir für das Grundgesetz”	supergroup	4.830
“QUERDENKEN (591 - Emsland) - INFO-Kanal”	channel	196
“QUERDENKEN (773 - HEGAU)	Diskussion & Austausch - Wir für das Grundgesetz”	supergroup	305
“QUERDENKEN (773 - HEGAU) - INFO-Kanal”	channel	139
“Querdenken (711 - STUTTGART) - Wir für das Grundgesetz”	supergroup	13.314
“QUERDENKEN (711 - STUTTGART) - INFO-Kanal”	channel	66.173
“Querdenken (69 - FRANKFURT)	Diskussion & Austausch - Wir für das Grundgesetz”	supergroup	1.763
“Querdenken 69 Frankfurt / Info Channel”	channel	1.791

```
# First install papermill
# pip install papermill (in running docker container)

echo "- Run notebook now"

touch log.txt

docker exec custom-ds-docker /bin/sh -c "papermill work/notebooks/
   Telegram.ipynb work/notebooks/Telegram-out.ipynb" > log.txt

echo "- Copy Stuff now (into github-pages-dir)"
cp output/pyLDAvis/*.html ../docs/topics

echo "[Finished]" >> log.txt
```

A.3.2 Aufräumen des Arbeitsverzeichnisses

Beschreibt die Datei `notebooks/clean-notebook.sh`, vgl. Abs. 3.3.2.

```
rm Telegram-out.ipynb
rm log.txt

cd output
rm -rf *
mkdir autoWordCloud pyLDAvis
touch autoWordCloud/.gitkeep
touch pyLDAvis/.gitkeep
cd ..

rm -rf ../docs/topics/*
```

Literaturverzeichnis

[Agg18] C. C. Aggarwal. *Machine Learning for Text*. Springer, Berlin, Heidelberg, 2018.

[Bir09] S. Bird, E. Klein und E. Loper. *Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, Inc.", Sebastopol, 2009.

[Bon16] M. Bonzanini. *Mastering Social Media Mining with Python*. Packt Publishing Ltd, Birmingham, 2016.

[Brü20] J. Brühl. Das Telegram-Prinzip. *Süddeutsche Zeitung*, 2020. [Abgerufen am 20.03.2021 auf `https://www.sueddeutsche.de/digital/telegram-corona-covid-naidoo-hildmann-1.4913092`].

[Bun21a] M. Bundscherer. *Jupyter Notebook - Identifizierung von Autoren*. 2021. [`https://github.com/maxbundscherer/telegram-analysis/blob/master/notebooks/Classifier.ipynb`].

[Bun21b] M. Bundscherer. *Jupyter Notebook - Telegram Mining*. 2021. [`https://github.com/maxbundscherer/telegram-analysis/blob/master/notebooks/Telegram.ipynb`].

[Bun21c] M. Bundscherer. *Telegram Analyse Plattform*. 2021. [`https://maxbundscherer.github.io/telegram-analysis/`].

[Cry05] B. Crysmann. *TIGER Morphologie-Annotationsschema*. Universität Stuttgart, 2005. [Abgerufen am 13.03.2021 auf `https://www.ims.uni-stuttgart.de/documents/ressourcen/korpora/tiger-corpus/annotation/tiger_scheme-morph.pdf`].

[Dür20] J. Dürrholz. Spiel mir das Lied vom Wendler. *Frankfurter Allgemeine Zeitung*, 2020. [Abgerufen am 20.03.2021 auf `https://bit.ly/3eZfaDW` (gekürzte URL)].

[Gel15] A. Gelbukh. *Computational Linguistics and Intelligent Text Processing - 16th International Conference, CICLing 2015, Cairo, Egypt, April 14-20, 2015, Proceedings, Part I*. Springer, Berlin, Heidelberg, 2015.

[Har16] N. Hardeniya, J. Perkins, D. Chopra, N. Joshi und I. Mathur. *Natural Language Processing: Python and NLTK*. Packt Publishing Ltd, Birmingham, 2016.

[Hua14] H.-Y. Huang, L. Jun und H.-P. Zhang. *Social Media Processing - Third National Conference, SMP 2014, Beijing, China, November 1-2, 2014, Proceedings*. Springer, Berlin, Heidelberg, 2014.

[Hug21] HuggingFace. *Offizielle Dokumentation: Transformers*. 2021. [Abgerufen am 21.03.2021 auf `https://huggingface.co/transformers/`].

[Hur20] S. Hurtz. Messenger bieten einen gefährlichen Nährboden für Desinformation. *Süddeutsche Zeitung*, 2020. [Abgerufen am 20.03.2021 auf `https://www.sueddeutsche.de/digital/whatsapp-telegram-desinformation-1.4962195`].

[Kaz15] P. Kazienko und N. Chawla. *Applications of Social Media and Social Network Analysis*. Springer, Berlin, Heidelberg, 2015.

[Kuc20] F. Kuckuk. Gegenwind für Verschwörer: Wie sich Twitter, Whatsapp und Facebook in der Corona-Krise schlagen. *Frankfurter Rundschau*, 2020. [Abgerufen am 20.03.2021 auf `https://www.fr.de/wirtschaft/platz-aufklaerung-verschwoerung-13781910.html`].

[Lan13] S. Lange. *Interaktiver Ansatz für die visuelleAnalyse von Textdokumentenbasierend auf der Word-Cloud-Visualisierungstechnik*. Diplomarbeit, Universität Stuttgart, 2013. [Abgerufen am 26.03.2021 auf `ftp://ftp.informatik.uni-stuttgart.de/pub/library/medoc.ustuttgart_fi/DIP-3352/DIP-3352.pdf`].

[Lin20] J. Lindern. Wo Zweifler sich in Rage chatten. *Zeit Online*, 2020. [Abgerufen am 20.03.2021 auf `https://bit.ly/3r4nvII` (gekürzte URL)].

[Mül16] A. C. Müller und S. Guido. *Introduction to Machine Learning with Python - A Guide for Data Scientists*. O'Reilly Media, Inc.", Sebastopol, 2016.

[Pot19] L. Pottmeyer. *Diskrete Mathematik - Ein kompakter Einstieg*. Springer-Verlag, Berlin Heidelberg New York, 2019.

[Pro13] F. Provost und T. Fawcett. *Data Science for Business - What You Need to Know about Data Mining and Data-Analytic Thinking*. O'Reilly Media, Inc.", Sebastopol, 2013.

[Röb21] S. Röbel. Polizei sucht mit Haftbefehl nach Attila Hildmann. *Spiegel*, 2021. [Abgerufen am 20.03.2021 auf `https://bit.ly/2PeToB1` (gekürzte URL)].

[Rus13] M. A. Russell. *Mining the Social Web - Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More*. O'Reilly Media, Inc.", Sebastopol, 2013.

[Sch14] M. Schoeneberg. *Ansaetze zur Trenderkennung in Texten*. Hochschule für Angewandte Wissenschaften Hamburg (HAW), 2014. [Abgerufen am 04.03.2021 auf `https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2014-aw2/schoeneberg/bericht.pdf`].

[Sza18] G. Szabo, G. Polatkan, P. O. Boykin und A. Chalkiopoulos. *Social Media Data Mining and Analytics*. John Wiley a. Sons, New York, 2018.

[Tit11] P. Tittmann. *Graphentheorie - eine anwendungsorientierte Einführung*. Fachbuchverl. Leipzig im Carl-Hanser-Verlag, München; Wien, 2011.

[Tur10] V. Turau. *Algorithmische Graphentheorie*. Oldenbourg Verlag, München, 2010.

[War19] C. Wartena. *A Probabilistic Morphology Model for German Lemmatization*. Hochschule Hannover, 2019. [Proceedings of the 15th Conference on Natural Language Processing (KONVENS 2019) - Abgerufen am 04.03.2021 auf `https://doi.org/10.25968/opus-1527`].

[Wei15] S. M. Weiss, N. Indurkhya und T. Zhang. *Fundamentals of Predictive Text Mining*. Springer, Berlin, Heidelberg, 2015.

[Zaf14] R. Zafarani, M. A. Abbasi und H. Liu. *Social Media Mining - An Introduction*. Cambridge University Press, Cambridge, 2014.