



Projektbericht  
Studiengang : Machine Learning & Data Analytics

---

## **Extraktion eines Entity-Relationship-Modells aus einem Text mithilfe von Natural Language Processing**

von  
Vaihunthan Vyramuthu  
87761

Betreuer Professor: Prof. Dr. Gregor Grambow

Einreichungsdatum : 15. August 2023

# Eidesstattliche Erklärung

Hiermit erkläre ich, **Vaihunthan Vyramuthu**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

14.08.2023

---

Ort, Datum



---

Unterschrift

# Kurzfassung

In dieser Projektarbeit geht es um die Entwicklung eines Systems zur Erfassung von Entity-Relationship-Modellkomponenten aus deutschen Texten mithilfe von *Natural Language Processing*. Der wesentliche Mehrwert bei diesem Projekt liegt in der Ersparnis einer zeitaufwändigen Interpretation von textuellen Datenbankszenarien. Hierbei werden regelbasierte und modellbasierte Ansätze implementiert und getestet. Die Hauptextraktion übernimmt die regelbasierte Variante. Dabei werden die Textdaten vorverarbeitet, strukturiert, analysiert und transformiert. Für die Analyse der grammatischen Struktur von Sätzen wird hauptsächlich die umfangreiche *SpaCy*-Bibliothek verwendet. Diese Bibliothek enthält mächtige Methoden, sodass strategisch die Entitäten, Attribute, Beziehungen und Kardinalitäten identifiziert werden können.

Über zwei verschiedene neuronale Netze, die auf die ER-Extraktion trainiert wurden, können ER-Komponenten extrahiert werden. Diese Ergebnisse werden als Vergleich zu den regelbasierten Ergebnissen herangezogen, sodass eventuelle Änderungen vorgenommen werden können. Eines dieser neuronalen Netze basiert auf das Transformermodell von *SpaCy*, welches mit eigenen Labels bzw. Klassen trainiert wurde. Dieses Modell ist spezialisiert auf das sogenannte *Custom Named Entity Recognition*. Das zweite Netz beinhaltet ein LSTM-Modell und wurde durch die *Sequential-API* der *Tensorflow Keras*-Bibliothek aufgebaut. Ergänzend zu den Ergebnissen aus den regelbasierten Algorithmen sind diese Modelle in der Lage auch anspruchsvollere Extraktionen, von beispielsweise mehrwertigen Attributen oder ISA-Vererbungen, durchzuführen. Mithilfe eines externen Annotierungstool, werden die Textdaten annotiert, vorverarbeitet und zum Trainieren der Modelle verwendet. Dabei wurden Modellgenauigkeiten von bis zu 92.1% erreicht.

Durch das in *Jupyter Notebook*-Dateien und in der Programmiersprache *Python* umgesetzte Projekt, wird aus dem regelbasierten Teilsystem eine strukturierte *.json*-Datei erstellt, welche in das ER-Modelingtool von Simon Ruttman [1] hochgeladen werden kann, um das ER-Modell zu visualisieren und eventuell in ein relationales Modell zu transformieren.

# Inhaltsverzeichnis

<b>Eidesstattliche Erklärung</b>	i
<b>Kurzfassung</b>	ii
<b>Inhaltsverzeichnis</b>	iii
<b>Abbildungsverzeichnis</b>	vi
<b>Tabellenverzeichnis</b>	viii
<b>Quelltextverzeichnis</b>	ix
<b>Abkürzungsverzeichnis</b>	x
<b>1. Einleitung</b>	1
1.1. Hintergrund und Motivation . . . . .	1
1.2. Problemstellung und -abgrenzung . . . . .	2
1.3. Zielsetzung der Arbeit . . . . .	3
1.4. Überblick über die Vorgehensweise . . . . .	3
<b>2. Grundlagen</b>	6
2.1. Natural Language Processing . . . . .	6
2.1.1. Definition und Ziel . . . . .	6
2.1.2. Anwendungsgebiete . . . . .	7
2.1.3. Herausforderungen und Limitationen . . . . .	7

2.2. Entity-Relationship-Modell . . . . .	10
2.2.1. Grundprinzipien und Komponenten . . . . .	10
2.2.2. Bedeutung und Anwendung in der Datenmodellierung . . . . .	13
2.2.3. Erstellung eines ER-Modells . . . . .	14
<b>3. Anforderungsanalyse</b>	<b>17</b>
<b>4. Verwandte Arbeiten</b>	<b>20</b>
4.1. Überblick über bestehende Ansätze zur NLP-basierten ER-Generierung	20
4.2. Kritische Bewertung und Analyse der vorhandenen Methoden . . . . .	25
<b>5. Methodik</b>	<b>27</b>
5.1. Datenbeschaffung und -vorverarbeitung . . . . .	27
5.2. Textstrukturierung . . . . .	29
5.3. Textanalyse . . . . .	31
5.3.1. Tokenisierung und Part-of-Speech-Tagging . . . . .	32
5.3.2. Named Entity Recognition . . . . .	32
5.3.3. Dependency Parsing . . . . .	35
5.4. Extraktion des Entity-Relationship-Modells . . . . .	36
5.4.1. Identifikation von Entitäten . . . . .	37
5.4.2. Identifikation von Beziehungen . . . . .	37
5.4.3. Identifikation von Attributen . . . . .	38
<b>6. Implementierung</b>	<b>40</b>
6.1. Beschreibung der verwendeten Tools und Bibliotheken . . . . .	40
6.2. Architektur der implementierten Lösung . . . . .	41
6.3. Umsetzung der einzelnen Schritte in der Architektur . . . . .	43
<b>7. Evaluierung</b>	<b>59</b>

<b>8. Projektplanung und -verlauf</b>	<b>63</b>
<b>9. Zusammenfassung und Ausblick</b>	<b>65</b>
9.1. Erreichte Ergebnisse . . . . .	65
9.2. Ausblick . . . . .	66
<b>Literatur</b>	<b>68</b>
<b>A. Anhang A</b>	<b>72</b>
<b>B. Anhang B</b>	<b>73</b>
<b>C. Anhang C</b>	<b>74</b>
<b>D. Anhang D</b>	<b>75</b>
<b>E. Anhang E</b>	<b>76</b>
<b>F. Anhang F</b>	<b>77</b>

# Abbildungsverzeichnis

1.1.	Wachstum von strukturierter und unstrukturierter Daten von 2010 bis 2020 [3] . . . . .	2
1.2.	CRISP-DM [10] . . . . .	4
2.1.	Beziehung zwischen AI, ML, DL und NLP [13] . . . . .	6
2.2.	Notationen für die Kardinalität. Rot: Chen-Notation, Schwarz: ( <i>min, max</i> )-Notation. . . . .	12
2.3.	Beispiel für eine existenzabhängige Entität . . . . .	13
2.4.	Beispiel einer ER-Modellierung . . . . .	16
4.1.	allgemeiner Systemaufbau aus der Arbeit von Ghosh et al. [28] . . . .	20
4.2.	Module des Programms [30] . . . . .	22
4.3.	Ausschnitt eines Beispieltextes mit Annotation [30] . . . . .	22
4.4.	Evaluationsergebnisse der Metriken für verschiedene Klassifikatoren [30] . . . . .	24
4.5.	<i>Top-Down-Parsing-Tree</i> für einen Beispielsatz [35] . . . . .	24
5.1.	NLP-Workflow [3] . . . . .	27
5.2.	Aufteilung des Datensatzes in Training, Validation und Testing [40] . .	29
5.3.	Beziehung zwischen Häufigkeit und Rangordnung von Wörtern [3] . .	31
5.4.	Die Gesamtarchitektur des multimodalen variationalen Autoencoders für semi-supervised MNER [44] . . . . .	33
5.5.	Recurrent Neural Network [45] . . . . .	34
5.6.	Vergleich der Modellarchitektur von RNN und LSTM [46] . . . . .	34

6.1.	Gesamtlaufdiagramm der Implementierung . . . . .	42
6.2.	Visualisierung der grammatischen Satzstruktur . . . . .	47
6.3.	Struktogramm für SVO-Extraktion aus einem Satz . . . . .	47
6.4.	Beispielsatz für die Extraktion von Primärschlüssel . . . . .	49
6.5.	Beispielsatz für die Extraktion von Beziehungen . . . . .	51
6.6.	Beispielsatz mit dem Adverb „höchstens“ . . . . .	51
6.7.	Benutzeroberfläche des Labelingtools . . . . .	53
6.8.	Ergebnisse der <i>Reclassification</i> mit dem <i>SpaCy</i> -Modell. . . . .	54
6.9.	Architektur des entworfenen LSTM-Modells . . . . .	55
6.10.	Verteilung der Satzlängen im kleinen Datensatz zum Testen. Der Median liegt bei der Länge 11. . . . .	57
6.11.	Vergleich der Verlustfunktionen von <i>Mean Squared Error</i> und <i>Cross Entropy</i> . <i>Ground Truth</i> ist bei 0. [50] . . . . .	58
7.1.	Evaluationsvergleich des <i>SpaCy</i> -Modells über eine <i>Confusion Matrix</i> . . . . .	59
7.2.	Lernkurven vom Training des LSTM-Modells. . . . .	60
7.3.	Verteilung der Anzahl an Wortkategorien im Datensatz. . . . .	61
8.1.	vereinfachtes Gantt-Diagramm . . . . .	63
D.1.	Struktur der umgewandelten <i>csv</i> -Datei für das LSTM-Modell . . . . .	75
E.1.	Auflistung verschiedener Verlustfunktionen. [52] . . . . .	76
F.1.	Visualisierung von den Ergebnissen der Evaluationsmetriken während dem Training des <i>SpaCy</i> -Modells. . . . .	77

# Tabellenverzeichnis

1.1. Beispiele für strukturierte und unstrukturierte Daten [3] . . . . .	1
2.1. Einsatzgebiete und typische Fragestellungen in NLP [3] . . . . .	7
2.2. Stilarten der deutschen Sprache [3] . . . . .	8
2.3. Wortbasierte Probleme für NLP [3, 22] . . . . .	9
3.1. Kürzel und Anforderungsbereiche . . . . .	19
4.1. Modell- und regelbasierte Präzisions- und Recallwerte [34] . . . . .	26
5.1. Tabelle mit Dependency Auflösung gemäß <i>SpaCy</i> . . . . .	36
5.2. Determinatoren und ihre zugehörigen min/max-Kardinalitäten . . . .	38

# **Quelltextverzeichnis**

6.1.	Aufteilung des Textes in einzelne Sätze . . . . .	45
6.2.	Erfassung von Beziehungen . . . . .	50
6.3.	Terminal-Escape Befehle für das Trainieren des SpaCy-Modells . . . . .	54
6.4.	Modellaufbau über die Sequential-API . . . . .	56
6.5.	Modellaufbau über die Sequential-API . . . . .	57
A.1.	Erfassung und Übersetzung der Kardinalität . . . . .	72
B.1.	Auszug aus output.json . . . . .	73

# Abkürzungsverzeichnis

<b>ERM</b>	Entity-Relationship-Modellen	1
<b>NLP</b>	Natural Language Processing	1
<b>SQL</b>	Structured Query Language	1
<b>CRISP-DM</b>	Cross Industry Standard Procedure for Data Modelling	4
<b>NER</b>	Named Entity Recognition	5
<b>POS</b>	Part-of-Speech	5
<b>JSON</b>	JavaScript Object Notation	5
<b>ML</b>	Machine Learning	6
<b>DL</b>	Deep Learning	6
<b>DR</b>	Document Retrievel	7
<b>IR</b>	Information Retriever	7
<b>SVM</b>	Support Vector Machine	21
<b>SVO</b>	Subjekt-Verb-Objekt	25
<b>SMO</b>	Sequential minimal optimization	22
<b>VAE</b>	Variational Autoencoder	33
<b>LSTM</b>	Long short-term memory	33
<b>RNN</b>	Recurrent Neural Network	34
<b>API</b>	Application Programming Interface	54

# 1. Einleitung

## 1.1. Hintergrund und Motivation

In der heutigen datengetriebenen Welt werden in großen Maßen Texte generiert, welche wertvolle Informationen erhalten können. Beispiele hierfür sind z.B. Polizeiberichte aus Online-Presseportale, Twitter Kommentare oder Amazon Reviews. Hierbei sind verschiedene Analysen möglich, die zukünftige Trends, Bewertung bzw. Sentiment der Nutzer oder anderweitige Fragestellungen beantworten können. [2]

strukturierte Daten	unstrukturierte Daten
SQL-Tabellen	Freitext-Absätze (auch in SQL / XML / HTML / L <sup>A</sup> T <sub>E</sub> X)
XML	Scans
HTML	HTML-Dumps

Tabelle 1.1.: Beispiele für strukturierte und unstrukturierte Daten [3]

Die Tabelle 1.1 listet einige Beispiele für unstrukturierte und strukturierte Daten auf. Die Extraktion strukturierter Daten aus unstrukturierten Texten ist eine komplexe Aufgabe, die eine manuelle Verarbeitung erfordert. Die Automatisierung dieses Prozesses kann Zeit und Ressourcen sparen und die Effizienz der Datenanalyse verbessern. Die Abbildung 1.1 visualisiert den zunehmenden Anstieg der unstrukturierten Daten und zeigt somit die Notwendigkeit der Automatisierung für die Textverarbeitung. [4, 5]

Gleichzeitig ist die Nutzung von Datenbanksystemen zur Speicherung und Verwaltung von Daten weit verbreitet. Ein gängiger Ansatz zur Modellierung von Datenbanken ist die Verwendung von Entity-Relationship-Modellen (ERM), gefolgt von der Umwandlung in ein relationales Modell und schließlich in Structured Query Language (SQL)-Anweisungen, um die Daten in einer Datenbank abzubilden. [6]

Die Motivation dieser Projektarbeit liegt in der Entwicklung einer Methode zur Extraktion eines ERM aus unstrukturiertem Text mithilfe von Natural Language Processing (NLP). Durch die Entwicklung einer automatisierten Methode zur Extraktion eines ERM aus Texten wird die Hürde einer zeitintensiven Interpretation von langen Texten reduziert.

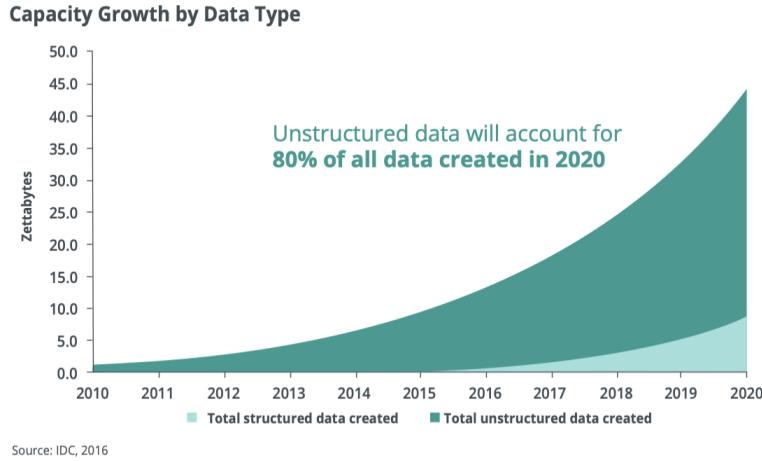


Abbildung 1.1.: Wachstum von strukturierter und unstrukturierter Daten von 2010 bis 2020 [3]

## 1.2. Problemstellung und -abgrenzung

Die Extraktion von Informationen aus unstrukturiertem Text ist eine komplexe Aufgabe, die linguistische Herausforderungen mit sich bringt. Die manuelle Verarbeitung großer Textmengen ist zeitaufwendig, fehleranfällig und nicht skalierbar. Daher besteht ein Bedarf an automatisierten Methoden, um diesen Prozess effizienter und so zuverlässig wie möglich zu gestalten. [7]

Vorhandene Ansätze zur automatisierten Generierung von ER-Modellen aus Texten weisen einige Einschränkungen auf. Sie sind gegebenenfalls nicht ausreichend robust, um mit der Vielfalt von Textstrukturen umzugehen, oder sie könnten Schwierigkeiten haben, semantische Zusammenhänge präzise zu erfassen. [8]

In den bestehenden Literaturen gibt es noch Forschungslücken im Bereich der automatischen Extraktion von ER-Modellen aus Texten mithilfe von NLP-Techniken. Einige Aspekte, die weiter untersucht werden müssen, umfassen die Integration von Kontextinformationen und die Berücksichtigung von Mehrdeutigkeiten. [8]

Diese Arbeit konzentriert sich auf die Extraktion von ER-Modellen aus technischen Anforderungsdokumenten für Datenbanken. Es werden keine anderen Formen, unstrukturierter Daten oder Textarten berücksichtigt. Darüber hinaus liegt der Fokus auf der Verwendung von NLP-Techniken zur Erkennung von Entitäten und Beziehungen sowie auf der Darstellung des ER-Modells in der klassischen Entity-Relationship-Notation.

Die Abgrenzung dieser Arbeit ermöglicht eine fokussierte Betrachtung der Kernproblematik und legt den Grundstein für die Entwicklung einer effektiven Lösung zur

Extraktion eines ERM aus Texten unter Berücksichtigung von NLP-Techniken.

### 1.3. Zielsetzung der Arbeit

Das Ziel dieser Projektarbeit ist es, die manuelle Datenextraktion zu automatisieren und dadurch den Prozess der Datenbankmodellierung und -integration zu vereinfachen. Die manuelle Datenextraktion umfasst dabei, den Text durchzulesen und korrekt zu interpretieren, sodass eine möglichst effiziente Datenbankmodellierung umgesetzt wird. Die Ergebnisse dieser Umsetzung gelten als erfolgreich, sofern die Diskrepanz zwischen der menschlichen und der softwaretechnischen Interpretation eines Textes hinsichtlich korrekter ERM-Generierung gering ist.

Die Projektarbeit legt den Schwerpunkt auf die Identifikation von Entitäten und Attribute sowie die Erfassung der Entitätsbeziehungen (inkl. Kardinalitäten). Die Analyse von semantischen Zusammenhängen und komplexen Textstrukturen, die über die Erfassung von Entitäten, Attribute und Beziehungen hinausgehen, ist nicht Teil dieser Arbeit. Die Implementierung des Systems konzentriert sich auf die Verarbeitung von Texten in deutscher Sprache.

Die Umwandlung des ERM in ein relationales Modell und die Generierung von SQL-Anweisungen werden als Folgeschritte betrachtet, sind jedoch nicht Gegenstand dieser Arbeit. Es wird davon ausgegangen, dass ein bestehendes Projekt oder Tool für diese Aufgaben zur Verfügung steht und das extrahierte ERM als Eingabe akzeptiert. Dementsprechend ist eine geeignete Schnittstelle zwischen diesen beiden Projekten zu schaffen.

Schließlich werden in dieser Arbeit keine spezifischen Implementierungsdetails des NLP-Systems betrachtet. Es wird davon ausgegangen, dass die verwendeten NLP-Techniken und -Tools bereits vorhanden oder verfügbar sind. Die Integration und Anpassung dieser Techniken an die spezifischen Anforderungen zur Textextraktion werden jedoch behandelt.

### 1.4. Überblick über die Vorgehensweise

Das Projekt gliedert sich in mehrere Phasen. Zunächst werden grundlegende Kenntnisse über NLP, ERM und Datenbankmodellierung vermittelt. Eine Anforderungsanalyse legt die erwarteten Funktionalitäten fest. Es wird ein Überblick über den aktuellen Stand der Forschung zu ähnlichen Ansätzen zur Extraktion von ERM aus Texten gegeben.

Anschließend werden Methoden besprochen, um Texte mithilfe NLP-basierter Tech-

nik zu analysieren und relevante Informationen zu identifizieren, die für die Erstellung eines ERM relevant sind. Dies umfasst die Identifikation von Entitäten, Attributen und die Beziehungen zwischen ihnen.

Die entwickelte Lösung wird mit dem bestehenden Projekt von Ruttman [1] verknüpft, welches die extrahierten Informationen des ERM in ein relationales Modell überführt und schließlich in SQL-Befehle umwandelt. Als Gesamtergebnis erhält man idealerweise ein Tool, welches ausgehend von einem textuellen Datenbankszenario, die notwendigen SQL-Anweisungen ableiten kann.

Abschließend werden die Ergebnisse anhand eines Beispieltextes evaluiert, mögliche Schwierigkeiten präsentiert sowie ein Ausblick auf zukünftige Entwicklungen und Verbesserungen gegeben.

Die Vorgehensweise während der Bearbeitung des Projektes orientiert sich hauptsächlich an Cross Industry Standard Procedure for Data Modelling (CRISP-DM). CRISP-DM ist ein weit verbreitetes, generisches Modell für Data-Mining-Prozesse, das einen umfassenden Rahmen für den Lebenszyklus von Data-Mining-Projekten bietet. Die Anwendung von CRISP-DM bringt eine Reihe von Vorteilen mit sich, darunter Kosten- und Zeitersparnis. Es bietet einen strukturierten und iterativen Ansatz, der sich aus sechs Hauptphasen zusammensetzt: *Business Understanding*, *Data Understanding*, *Data Preparation*, *Modeling*, *Evaluation* und *Deployment*. Jede Phase bietet klare Schritte und Aktivitäten, die zu einer erfolgreichen Umsetzung des Projekts führen. Ein Beispiel hierfür findet sich in Abbildung 1.2. [9]

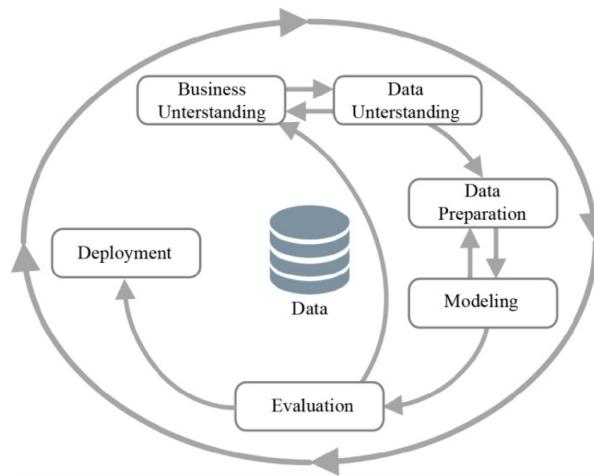


Abbildung 1.2.: CRISP-DM [10]

**Business Understanding** In dieser Phase liegt der Fokus darauf, das Geschäftsziel und die Anforderungen des Projekts zu verstehen. Es ist wichtig, die zugrunde liegenden Ziele und den Mehrwert der Extraktion eines ERM aus Texten zu iden-

tifizieren. Dazu können Anforderungen erfasst werden (siehe Kapitel 3). Zudem werden die zu erwartenden Ergebnisse und Erfolgskriterien definiert.

**Data Understanding** Hierbei steht die Erfassung und Analyse der verfügbaren Daten im Vordergrund. Es ist wichtig, den Textkorpus, auf dem das ERM basieren soll, genauer zu untersuchen. Dabei können verschiedene Textquellen und -formate berücksichtigt werden. Es werden Datenanalysetechniken angewendet, um einen Einblick in die vorhandenen Datenstrukturen und -inhalte zu gewinnen. Zudem können sprachspezifische Besonderheiten und Herausforderungen identifiziert werden, um die NLP-Verarbeitung entsprechend anzupassen (siehe Kapitel 2.1.3).

**Data Preparation** In diesem Schritt werden die Daten für die weitere Verarbeitung vorbereitet. Das beinhaltet die Datenbereinigung, bei der mögliche Fehler, Duplikate oder irreführende Informationen entfernt werden. Zudem können Textvorverarbeitungstechniken wie Tokenisierung, Lemmatisierung und Entfernung von Stoppwörtern angewendet werden, um den Text in eine geeignete Form für die NLP-Bearbeitung zu bringen (siehe Kapitel 5.1). [11]

**Modeling** Auf der Grundlage der bereinigten und vorbereiteten Daten werden Techniken angewendet, um Entitäten und Beziehungen im Text zu identifizieren. Es können verschiedene Algorithmen und Ansätze zur Named Entity Recognition (NER), Part-of-Speech (POS)-Tagging und *Dependency Parsing* verwendet werden (siehe Kapitel 5). [11, 12]

**Evaluation** In dieser Phase wird die Qualität des extrahierten ERM bewertet. Das ERM wird mit vorhandenen Textdaten validiert und mit manuell erstellten ERM verglichen, um die Genauigkeit und Vollständigkeit der Extraktion zu bewerten. Es können Metriken wie *Precision*, *Recall* und *F1-Score* verwendet werden, um die Leistung des Modells zu quantifizieren. Bei Bedarf werden Anpassungen vorgenommen, um die Extraktionsgenauigkeit zu verbessern. [12]

**Deployment** In dieser abschließenden Phase wird das extrahierte ERM in das gewünschte Zielformat, wie z.B. in eine JavaScript Object Notation (JSON)-Datei oder in ein anderes Dateiformat, überführt. Das fertige Projekt wird dokumentiert und für die weitere Nutzung bereitgestellt.

## 2. Grundlagen

### 2.1. Natural Language Processing

#### 2.1.1. Definition und Ziel

NLP ist ein wesentlicher Bereich der Künstlichen Intelligenz (siehe Abbildung 2.1), der sich mit der Anwendung von maschinellem Lernen und Computerlinguistik befasst. Komplexe Problemstellungen in NLP werden häufig durch neuronale Netze gelöst, welche auf entsprechende Machine Learning (ML) und Deep Learning (DL) Netzstrukturen basieren. [13]

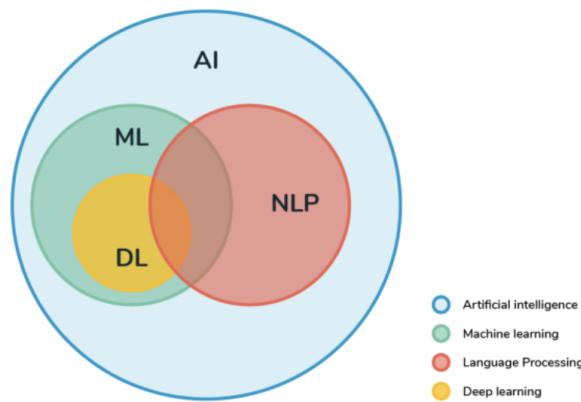


Abbildung 2.1.: Beziehung zwischen AI, ML, DL und NLP [13]

Das Hauptziel besteht darin, die Interaktion zwischen Mensch und Computer einfach und effizient zu gestalten. In NLP werden Computer darauf trainiert, die Syntax und Bedeutung menschlicher Sprache zu verstehen, sie zu verarbeiten und dem Benutzer entsprechende Ergebnisse zu liefern. [14]

### 2.1.2. Anwendungsbereiche

Die Verarbeitung natürlicher Sprache ist ein aktuell intensiv diskutiertes und erforschtes Thema. Die Fortschritte in der Verarbeitung natürlicher Sprache haben einen bedeutenden Durchbruch in den Bereichen Informatik und künstliche Intelligenz ermöglicht. Es handelt sich um eines der ältesten Forschungsfelder im Bereich des maschinellen Lernens und findet Anwendung in wichtigen Bereichen wie maschineller Übersetzung, Spracherkennung und Textverarbeitung. [14]

Die Textverarbeitung von unstrukturierten Texten wird vermehrt angewendet, z.B. zur Klassifizierung von Texten, zur Entdeckung von Themen und zur Sentiment-Analyse. Mithilfe der Informationsextraktion können relevante Informationen aus unstrukturierten Texten extrahiert werden, beispielsweise zur Extraktion von Informationen aus medizinischen Berichten [15] oder Finanzdokumenten [16]. Darüber hinaus ermöglicht NLP, die Entwicklung von Chatbots und virtuellen Assistenzten [17], welche natürliche Dialoge mit Benutzern führen können. Die Tabelle 2.1 demonstriert unterschiedliche Fragestellungen bzw. Tasks für die jeweiligen Einsatzgebiete der NLP.

Einsatzgebiet	Fragestellung/Task
Recherche	Finde alle Dokumente mit dem Wort „Datenbank“
Information Retrieval	Liste alle Seiten mit dem Wort Golf
Document Retrieval	Liste alle Seiten, die sich mit Golfsport befassen
Klassifikation	Ist es eine Fachliteratur oder Märchenbuch?
Bewertung	Sind die Reviews zu einem Thema positiv oder negativ?
Verständnis	Um was geht es in dem Text?

Tabelle 2.1.: Einsatzgebiete und typische Fragestellungen in NLP [3]

Die Recherche kann in die Bereiche Document Retrieval (DR) und Information Retrieval (IR) aufgeteilt werden. Während das DR eine Einzelwortsuche darstellt, wird in der IR nach komplexen Inhalten gesucht. Die Tabelle 2.1 verdeutlicht diesen Unterschied anhand der Fragestellungen. In der Praxis werden die beiden Begriffe häufig als Synonyme verwendet. [18]

### 2.1.3. Herausforderungen und Limitationen

Obwohl NLP große Fortschritte gemacht hat, gibt es immer noch Herausforderungen und Limitationen, mit denen diese Technologie konfrontiert ist.

**Semantik und Kontextverständnis** Die korrekte Interpretation der Bedeutung von Sätzen kann schwierig sein, da Wörter und Sätze je nach Kontext unterschiedliche Bedeutungen haben können. Die feine Differenzierung von Sprache und das Verständnis des Kontexts stellen eine Herausforderung für NLP-Systeme dar. Ambiguität und Mehrdeutigkeit erschweren die präzise Analyse von Texten. Folgendes Beispiel zeigt die mögliche Schwierigkeit der Interpretation eines Satzes [3]:

„Er sieht den Verkäufer mit dem Fernglas.“

**Seltene Sprachen und Dialekte** Die NLP-Sprachmodelle sind oft auf häufig verwendete Sprachen und Standarddialekte trainiert, während seltene Sprachen und regionale Dialekte möglicherweise nicht ausreichend abgedeckt sind. Das Fehlen von ausreichenden Trainingsdaten und Ressourcen für seltene Sprachen stellt eine Herausforderung dar, da NLP-Modelle möglicherweise nicht gut auf solche Sprachen übertragbar sind. Die Tabelle 2.2 kategorisiert drei verschiedene Stilarten der deutschen Sprache. [3]

Stil	Beispieltext
mündliches Deutsch	Also ich war gestern im Zoo gewesen. Echt coole Tiere gibt's da! Die Koalas, also die, die dieses grüne Zeug, Eukalyptus, fressen, die haben nur auf ihren Bäumen gesessen. Krass waren auch die Tiger, also die mit den Streifen.
schriftliches Deutsch	Gestern machte ich einen Ausflug in den Zoo. Dort sah ich viele interessante Tiere. Besonders beeindruckt haben mich die Koalas. Koalas sind grau, fressen Eukalyptus und sitzen auf Bäumen, wo man sie gut sehen kann.
Social-Media Deutsch	Zoo gewesen, viele interessante Tiere da. Koalas beeindrucken, Eukalyptus fressen auf Bäumen #Tiere#Zoo

Tabelle 2.2.: Stilarten der deutschen Sprache [3]

Oft müssen Wörter trotz Rechtschreibfehler korrekt erkannt und interpretiert werden. Hierbei gibt es mögliche Techniken, wie zum Beispiel die *Levenshtein-Distanz* [19], mithilfe dessen, die Wörter korrigiert werden können. Mehr dazu in Kapitel 5.1

**Annotierte Trainingsdaten** NLP-basierte Sprachmodelle erfordern große Mengen an annotierten Trainingsdaten, um gute Leistung zu erzielen. Das Sammeln und

Annotieren solcher Daten kann zeitaufwändig und ressourcenintensiv sein. Insbesondere für spezifische Domänen oder seltene Sprachen kann es schwierig sein, ausreichend geeignete Daten zu erhalten. Des Weiteren wird durch das falsche Annotieren der Textdaten, der *Ground Truth* verfälscht, sodass die allgemeine Genauigkeit des Sprachmodells sinkt. [20]

**Vorurteile und Fairness** Analog zu allgemeinen ML- oder DL-Modellen können auch Textverarbeitungsmodelle Vorurteile und Diskriminierungen aus den Trainingsdaten übernehmen und reproduzieren. Wenn die Trainingsdaten verzerrt sind oder unbeabsichtigte Vorurteile enthalten, können NLP-Systeme diese Vorurteile verstärken und zu unfairer Behandlung oder Diskriminierung führen. Die Gewährleistung von politischer Korrektheit und das Vermeiden von Fehlinterpretationen sind wichtige Herausforderungen, um ethisch verantwortliche Sprachmodelle zu entwickeln. [21]

**Wortbasierte Probleme** Polysemie bezieht sich auf Wörter, die mehrere Bedeutungen haben. Homonymie dagegen auf verschiedene Wörter mit ähnlicher Schreibweise oder Aussprache, die jedoch unterschiedliche Bedeutungen haben. Das richtige Verständnis und die korrekte Zuordnung solcher Äquivalenzen in einem gegebenen Kontext kann in einigen Fällen sehr schwierig sein. Die Tabelle 2.3 demonstriert verschiedene Wortarten, welche zu Schwierigkeiten in der NLP-basierter Interpretation führen können. [22]

Wortart	Bedeutung	Beispiele
Homonyme	gleich geschrieben; Bedeutung verschieden	Schale, Bank, Tau
Homophone	gleich gesprochen; Bedeutung verschieden	mahlen/malen leeren/lehren
Homographen	gleich geschrieben; verschieden gesprochen	Montage (Pl. Montag/ Zusammenbau)
Synonyme	unterschiedl. Bezeichnung für einen Begriff	essen, speisen, konsumieren
Komposita	Wortzusammensetzung	Kapitänsmütze
Abkürzungen	verkürzte Darstellungsform von Wörtern	u.s.w.
Eigennamen	individuelle Namen	Angela Merkel Apple corp.
Flexionen	grammatikalische Anpassung der Wörter	ich singe ich sang
Anglizismen	übernommene Wörter aus dem Englischen	surfen, boostern

Tabelle 2.3.: Wortbasierte Probleme für NLP [3, 22]

Neben diesen Wortarten können auch unterschiedliche Schreibweisen den Textverarbeitungsprozess erschweren. Beispiele hierfür sind Tür(e) oder Schi vs. Ski. Solche Abweichungen müssen im NLP-Prozess berücksichtigt und validiert werden. [3]

## 2.2. Entity-Relationship-Modell

Das ERM ist ein konzeptionelles Datenmodell, das zur Darstellung und Organisation von Daten in einer Datenbank verwendet wird. Dieses Datenmodell soll einen definierten Sachbereich der realen Welt so detailliert beschreiben, wie es für den Anwendungszweck ausreichend ist [23]. Gemäß Peter Chen wurde das ER-Modell im Jahr 1976<sup>1</sup> publiziert. In seinen verschiedenen Versionen, mit unterschiedlicher Syntax und einigen semantischen Variationen erhält das ERM sowohl in der Forschung als auch in der Industrie wachsende Beliebtheit. Dies liegt daran, dass das Modell einfach zu verstehen und anzuwenden ist. Durch ein ER-Diagramm erhält man einen übersichtlichen Überblick über das Datenbankdesign. Es stellt die Beziehungen zwischen Entitäten (Objekten) und deren Attributen in einer grafischen Form dar. [24]

### 2.2.1. Grundprinzipien und Komponenten

Das Entity-Relationship-Modell basiert auf einige grundlegende Prinzipien und Komponenten, darunter:

**Entität** Eine Entität E ist ein unterscheidbares Objekt aus einem Sachbereich, das durch eine Menge von Attributen  $A = \{a_1, \dots, a_n\}$  beschrieben werden kann. Die Kombination der Attributwerte differenziert ein Objekt von allen anderen. Sobald zwei identische Entitäten in der Datenbank existieren, gilt diese als defekt. Es gibt auch spezielle Arten von Entitäten, wie zum Beispiel schwache Entitäten und vererbte Entitäten. Des Weiteren ist wichtig zu unterscheiden, dass im ERM ausschließlich auf dem Entität-Typelevel gearbeitet wird. Ein Entität-Typ E ist eine Klasse von Entitäten mit gleichen Attributen. Im Gegensatz hierzu wird auf dem Instanzlevel, zu den Entität-Typen gehörenden Instanzen betrachtet. Folgende Beispiele verdeutlichen den Unterschied dieser Betrachtungsebenen. [23]

Typelevel:

Student = ({Matrikelnr: Zahl, Name: Zeichenkette, Vorname: Zeichenkette,  
Note:{1..6}}, ...)

---

<sup>1</sup>Chen, P. P. (1976). "The Entity-Relationship Model—Toward a Unified View of Data". ACM Transactions on Database Systems (TODS), 1(1), 9-36. doi: 10.1145/320434.320440.

Instanzlevel:

```

Student = {Max, Albert}
Attributmenge = {Matrikelnr, Name, Vorname, Note}
    Max = ({12345, Planck, Max, Note: 2})
    Albert = ({54321, Einstein, Albert, Note: 1})

```

**Attribut** Attribute sind Eigenschaften einer Entität und beschreiben bestimmte Merkmale oder Informationen. Sie werden verwendet, um die verschiedenen Aspekte einer Entität zu beschreiben und zu differenzieren. Jede Entität kann mehrere Attribute haben, die ihren Zustand oder ihre Charakteristika darstellen. Jedes Attribut hat einen Wertebereich (*Domain*) eines Attributs  $a$ :  $\text{dom}(a)$ . [23]

$$\text{dom}(\text{Note}) = \{1,2,3,4,5,6\}$$

Ergänzend zu den einfachen Attributen gibt es komplexe Attribute, welche in mehrwertig und zusammengesetzt unterteilt werden können. Mehrwertige Attribute können mehrere gleichartige Werte annehmen. Im ERM wird das Attribut als eine Ovale mit doppelter Umrandung dargestellt. [23]

Typelevel:

```
Student = ({Matrikelnr: Zahl, Name: Zeichenkette, Vorname: Zeichenkette,
           Note:{1..6}}, Kompetenzfächer: Kompetenzfach*, ...)
```

Instanzlevel:

$$\text{Max}(\text{Kompetenzfächer}) = \{\text{Optik}, \text{Physik}, \text{Photonik}, \text{Messtechnik}\}$$

Zusammengesetzte Attribute setzen sich aus mehreren Werten zusammen. Diese Werte können auch von verschiedenen Typen sein. Im ERM wird hierbei ein Attribut mit mehreren weiteren Attributen verbunden. [23]

Typelevel:

```
Student = ({Matrikelnr: Zahl, Name: (Nachname: Zeichenkette, Vorname:
           Zeichenkette), Note:{1..6}}, ...)
```

Instanzlevel:

$$\text{Albert} = ({54321, Name: (Nachname: Einstein, Vorname: Albert), Note: 1}, ...)$$

**Beziehung** Beziehungen zeigen die Verbindungen und Abhängigkeiten zwischen Entitäten an. Sie geben an, wie Entitäten miteinander in Beziehung stehen und wie sie miteinander interagieren können. Beziehungen können zusätzliche Attribute haben, um ihre Eigenschaften zu beschreiben. Eine Beziehung wird mithilfe folgender *Relationship-Deklaration* beschrieben [23]:

$$R = (E_1 \times \dots \times E_n, \text{Beziehungsattribute}) \\ (\text{Beziehungsattribute} = \{a_1, \dots, a_m\})$$

Konkretes Beispiel:

$$\text{bearbeitet} = (\text{Student} \times \text{Projekt}, \{\text{Wochenstunden: Zahl}\})$$

Gemäß der *Chen-Notation* zur Festlegung der Kardinalität von Entität-Typen können Beziehungen eins zu eins ( $1 : 1$ ), eins zu viele ( $1 : N$ ) oder viele zu viele ( $N : M$ ) sein. Hierbei sind keine minimale oder maximale Kardinalitätsangaben möglich. Eine Erweiterung bietet die  $(min, max)$ -Notation an. Hierbei sind beliebige Spezifikationen der Kardinalität von Entität-Typen in einer abstrakten Beziehung möglich. Die folgende Abbildung 2.2 illustriert anhand des vorherigen Beziehungsbeispiels „bearbeitet“ die zwei verschiedenen Notationen. Durch die  $(min, max)$ -Notation wird eingeschränkt, dass ein Student mindestens ein Projekt bearbeitet. Außerdem können an ein Projekt beliebig viele Studenten arbeiten. Beispielsweise sind Projekte ohne Studenten abgeschlossene Projekte. [23]

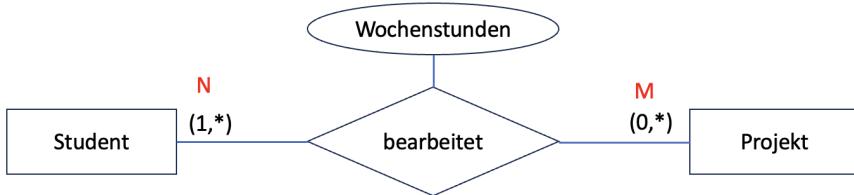


Abbildung 2.2.: Notationen für die Kardinalität. Rot: Chen-Notation, Schwarz:  $(min, max)$ -Notation.

**ISA-Beziehungen (Vererbungen)** ISA-Beziehungen (auch bekannt als Vererbungen) werden verwendet, um hierarchische Strukturen in einem Datenmodell darzustellen. Eine Entität kann eine spezialisierte Form einer anderen Entität sein und erbt ihre Attribute und Beziehungen. Dies ermöglicht die Modellierung von Klassenhierarchien und die Repräsentation von Generalisierung und Spezialisierung. Die Spezialisierungsbeziehungen selber haben keine Attribute. Die Notation für „Entität-Typ  $E_s$  spezialisiert Entität-Typ  $E$ “ lautet [23, 25]:

$$\begin{aligned} E_s &\text{ ISA } E \\ E_s &\subseteq E \end{aligned}$$

Alle Attribute von  $E$  werden von  $E_s$  geerbt.  $E_s$  erbt auch den Schlüssel von  $E$ .

**Schwacher Entität-Typ** Schwache Entitäten sind Entitäten, die ihre eigene Existenz nicht unabhängig von anderen Entitäten haben können. Sie sind von einer anderen Entität abhängig und können nur durch ihre Beziehung zu dieser Entität identifiziert werden. Die Kardinalität in der Beziehung zwischen einem „starken“ und „schwachen“ Entität-Typ ist meistens  $1 : N$  (Chen-Notation). In Abbildung 2.3 ist ein Beispiel für eine existenzabhängige Entität gegeben. [23]

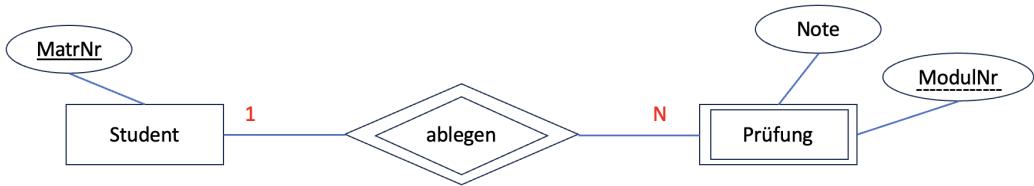


Abbildung 2.3.: Beispiel für eine existenzabhängige Entität

**Schlüssel** Als Schlüssel wird eine Menge  $S \subseteq A$  von Attributen eines Entitätstyps  $E = (A, K)$  beschrieben, die alleine ausreicht, um ein Tupel eindeutig zu bestimmen. Ist diese Attributmenge  $A$  eindeutig und minimal, so wird der Schlüssel als Primärschlüssel bezeichnet. Es wird häufig die folgende Deklaration angewendet [23]:

Student = {MatrNr, Name, Vorname, Note}

Das Entity-Relationship-Modell verwendet diese Komponenten, um eine klare und strukturierte Darstellung von Daten zu ermöglichen.

### 2.2.2. Bedeutung und Anwendung in der Datenmodellierung

Das Entity-Relationship-Modell ist ein wichtiges Instrument in der Datenmodellierung und wird in verschiedenen Phasen des Datenbankentwicklungsprozesses eingesetzt [26]:

Das ERM wird verwendet, um das Datenmodell auf konzeptioneller Ebene zu entwerfen. Es hilft bei der Festlegung der Tabellen, der Attribute und der Beziehungen

zwischen den Entitäten. Dadurch wird eine solide Grundlage für die effiziente Speicherung und Verwaltung von Daten geschaffen.

Zudem erleichtert es die Integration und Migration von Daten zwischen verschiedenen Systemen. Durch die klare Darstellung der Entitäten und Beziehungen können Daten aus verschiedenen Quellen effektiv zusammengeführt und harmonisiert werden.

Darüber hinaus unterstützt die ER-Modellierung die Datenanalyse und ermöglicht komplexe Abfragen, um Informationen aus der Datenbank abzurufen. Durch die klare Darstellung der Beziehungen können komplexe Abfragen (*JOINS*) über mehrere Tabellen hinweg erstellt werden, um spezifische Informationen zu extrahieren.

### 2.2.3. Erstellung eines ER-Modells

Bei der manuellen Erstellung des ER-Modells gibt es keine eindeutige Vorgehensweise. Jedoch erleichtert sich die Generierung, wenn man gemäß folgender Suchkriterienreihenfolge, den Text analysiert:

1. Suche nach Entitäten
2. Suche nach Beziehungen
3. Suche nach Attribute (für Entitäten & Beziehungen)
4. Bestimmung der Kardinalitäten

Es sei der folgende Text gegeben, indem ein Datenbankszenario für ein Geschäft beschrieben ist [27]:

*Ein Geschäft hat viele Filialen. Jede Filiale darf von höchstens einem Filialleiter geführt werden. Ein Filialleiter darf höchstens zwei Filialen leiten.*

*Ohne eine Filiale wird kein Filialleiter bestimmt. Die Filiale bietet viele Produkte an. Das Produkt wird von vielen Filialen angeboten. Die Filiale beschäftigt viele Mitarbeiter. Ein Mitarbeiter darf höchstens einen Verkauf bearbeiten. Im Verkauf können viele Produkte involviert sein. Jedes Produkt enthält Preis, Produktnam e und eindeutige Produkt ID. Es gibt zwei spezielle Produkttypen: Neuwaren, bei denen die Verpackungsnummer registriert ist. Bei Gebrauchtwaren wird die Anzahl der Vorbesitzer mitgeführt. Eine Filiale wird durch die Filial ID eindeutig identifiziert. Die Filiale hat einen Namen, eine Adresse und eine Telefonnummer. Verkauf enthält Datum, Uhrzeit und Betrag.*

*Jeder Mitarbeiter hat einen Namen bestehend aus Vor- und Nachname, eine oder mehrere Adressen und eine Telefonnummer. Die Mitarbeiter und der Filialleiter werden durch eine ID identifiziert.*

Durch iterative Analyse des Textes werden die hervorgehobenen ERM-Komponenten ermittelt. Diese Vorgehensweise wird entsprechend vom automatisierten NLP-System erwartet. Darüber hinaus kann überprüft werden, ob Vererbungshierarchien, schwache Entitäten/Beziehungen oder zusammengesetzte bzw. mehrwertige Attribute existieren. Ergänzend zu den vier Hauptkriterien kommen demnach weitere vier Kriterien hinzu:

5. Bestimmung der Primärschlüssel
6. Anzeichen für schwache Entitäten/Beziehungen
7. IsA-Vererbung
8. zusammengesetzte/mehrwertige Attribute

Die entsprechenden Textpassagen sind ebenfalls gekennzeichnet, sodass der Text nun folgendermaßen aussieht:

*Ein Geschäft hat viele Filialen. Jede Filiale darf von höchstens einem Filialleiter geführt werden. Ein Filialleiter darf höchstens zwei Filialen leiten.*

*Ohne eine Filiale wird kein Filialleiter bestimmt. Die Filiale bietet viele Produkte an. Das Produkt wird von vielen Filialen angeboten. Die Filiale beschäftigt viele Mitarbeiter. Ein Mitarbeiter darf höchstens einen Verkauf bearbeiten. Im Verkauf können viele Produkte involviert sein. Jedes Produkt enthält Preis, Produktnname und eindeutige Produkt ID. Es gibt zwei spezielle Produkttypen: Neuwaren, bei denen die Verpackungsnummer registriert ist. Bei Gebrauchtwaren wird die Anzahl der Vorbesitzer mitgeführt. Eine Filiale wird durch die Filial ID eindeutig identifiziert. Die Filiale hat einen Namen, eine Adresse und eine Telefonnummer. Verkauf enthält Datum, Uhrzeit und Betrag.*

*Jeder Mitarbeiter hat einen Namen bestehend aus Vor- und Nachname, eine oder mehrere Adressen und eine Telefonnummer. Die Mitarbeiter und der Filialleiter werden durch eine ID identifiziert.*

Um die Effizienz der Datenbankmodellierung zu steigern, müssen so wenige Tabellen wie möglich erzeugt werden. Oftmals ist es besser, gewisse Anpassungen zu treffen, sodass gewisse Entitäten als Beziehung modelliert werden. Dies hängt jedoch von der Sichtweise auf das Problem ab. Somit können sich für einige Datenbankszenarios mehr als eine ERM-Lösung ergeben.

Die grafische Darstellung eines ERM besteht grundsätzlich aus Knoten und Kanten. Die Knoten, welche Entität-Typen (Rechtecke mit Namen), Attribute (Ovale mit Namen) oder abstrakte Beziehungen (Rauten mit Namen) repräsentieren, werden durch Kanten verbunden. Zwei Entität-Typen oder Beziehungen sind nicht direkt verbindbar. [23]

Die Abbildung 2.4a und 2.4b veranschaulichen diese zwei verschiedenen Möglichkeiten der Modellierung aus dem vorherigen Text, wobei Abbildung 2.4b aus der Datenbanksicht günstiger ist. Im späteren Verlauf nach der Umsetzung in das relationale Modell spielt es jedoch keine Rolle mehr, für welche Variante man sich entschieden hat.

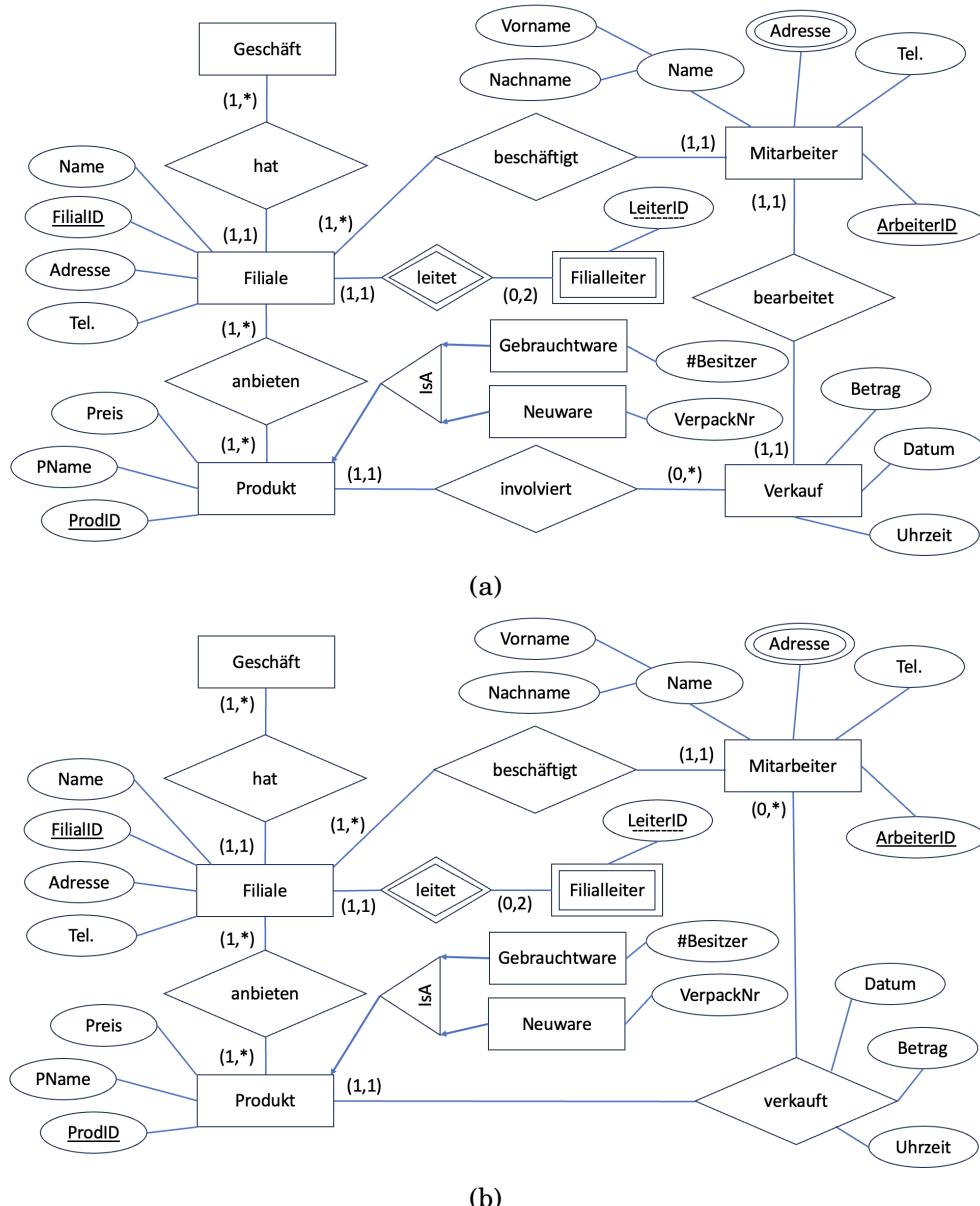


Abbildung 2.4.: Beispiel einer ER-Modellierung (a) „Verkauf“ als Entität. (b) „verkauft“ als Beziehung.

### 3. Anforderungsanalyse

In einer Anforderungsanalyse für das Projekt zur Extraktion eines ERM aus einem Text mithilfe von NLP sollten die spezifischen Anforderungen an das System und die erwarteten Funktionalitäten festgelegt werden. Dies ist ein kritischer Schritt, um die Ziele und Funktionalitäten des Systems klar zu definieren und sicherzustellen, dass es den Erwartungen der Benutzer entspricht. Eine sorgfältige Hinführung und Begründung für die Anforderungen ist entscheidend, um sicherzustellen, dass das System effektiv, effizient und benutzerfreundlich ist. Die folgenden Anforderungen sind entsprechend ihrer Priorität geordnet.

**A1: Schnittstellenanknüpfung** Das Projekt sollte in der Lage sein, einen unstrukturierten Text als Eingabe zu akzeptieren. Dieser Text kann entweder als Datei oder direkt über eine Benutzerschnittstelle eingegeben werden. Im Falle der Implementierung einer Benutzerschnittstelle im Zielsystem von Ruttman [1] sollte diese Schnittstelle benutzerfreundlich sein, die es Benutzern ermöglicht, Texte einzugeben. Die Schnittstelle sollte intuitiv und leicht verständlich sein, um eine effektive Nutzung des Systems zu ermöglichen. Alternativ kann der unstrukturierte Text extern verarbeitet werden, sodass ein Upload in das Zielsystem von Ruttman [1] über eine JSON-Datei möglich ist. Dies bedeutet, dass eine geeignete JSON-Datei erstellt werden muss, welche vom bereits im Zielsystem vorhandenen Uploadfunktion akzeptiert wird. Um einen zeitsparenden Mehrwert aus der automatisierten Extraktion zu erhalten, ist die fehlerfreie Einführung der Textdaten bzw. einer JSON-Datei besonders wichtig. Erst dann ist eine Visualisierung der Endergebnisse möglich, sodass es die Überprüfung und Korrektur des Gesamtprozesses erleichtert. Die JSON-Datei sollte alle wichtigen Metainformationen der Komponenten erhalten, sodass nur noch wenige optische Verfeinerungen (wie z.B. das Verschieben der Modellblöcke im Editor) manuell erledigt werden müssen. Beide Varianten der Schnittstellen (Rohtexteingabe im Editor oder JSON-Input) besitzen ihre Vor- und Nachteile. Durch die Auswahl von zwei Optionen für diese Anforderung, wird die Wahrscheinlichkeit erhöht, dass im Falle von zeitlichen oder technischen Hürden während dem Projekt, mindestens eine funktionierende Schnittstellenanknüpfung sichergestellt wird.

**A2: korrekte Informationsextraktion** Das System sollte NLP-Strategien verwenden, um den eingegebenen Text zu analysieren und relevante Informationen zu

extrahieren. Dazu gehören die Identifikation von Entitäten und die Beziehungen zwischen ihnen. Außerdem muss darauf geachtet werden, dass zusammengehörige Entitäten als eine Entität zusammengefasst werden, sodass keine redundanten Tabellen entstehen. Des Weiteren ist zu beachten, dass die Attribute zu den korrekten Entitäten zugeordnet werden. Fehlende Primärschlüssel oder Attribute und nicht gefundene Entitäten (vorausgesetzt: keine vorhandene Verbindung zu Beziehungen) stellen kein Problem dar, da diese leicht in den Editor hinzugefügt werden können. Der Fokus liegt daher auf die Erstellung von höchstwahrscheinlich korrekten ER-Komponenten. Die qualitative Informationsextraktion soll über der quantitativen Extraktion stehen, da das Hinzufügen von Komponenten deutlich zeitsparender ist als das Ersetzen und Korrigieren der Endergebnisse.

**A3: Performance** Der Algorithmus sollte effizient arbeiten und in der Lage sein, große Textmengen schnell und korrekt zu verarbeiten. Es sollte skalierbar sein und auch bei umfangreichen Texten eine angemessene Laufzeit bieten. Obwohl der wesentliche Mehrwert in dieser Projektarbeit in der Ersparnis der eigenständigen ER-Modellierung aus einem Text liegt, sollte dabei die benötigte Zeit nicht vernachlässigt werden. Eine eindeutige Festlegung der Grenzzeit ist nicht möglich, da diese anwendungsfallspezifisch sein kann. Dennoch sollte man den zeitlichen Vorteil gegenüber der manuellen Vorgehensweise erkennen. Die korrekte Informationsextraktion und eine geeignete Schnittstellenanknüpfung haben im Vergleich zur Performance eine höhere Priorität, da die Performance erst nach der Erfüllung der beiden höher geordneten Anforderung zum Tragen kommt. Es sei angemerkt, dass für die Performance die Zeit, welche für das Training des Modells benötigt wird, ausgenommen ist.

**A4: Validierung und Fehlerbehandlung** Es sollten Mechanismen zur Validierung der extrahierten Informationen bereitgestellt werden, um Fehler nach dem Extraktionsprozess vorzubeugen. Eine manuelle Korrektur des ERM nach der Verarbeitung ist im Tool von Ruttman [1] möglich. Aufgrund der vorhandenen Unsicherheit der Ergebnisse ist es von Vorteil, die Ergebnisse mithilfe einer anderen Methode zu überprüfen. Somit wird sichergestellt, dass man ein möglichst korrektes Ergebnis erhält, ohne eigene Interpretationsleistung auf den unstrukturierten Text anzuwenden. Aufgrund dieser doppelten Absicherung der Ergebnisse wird dieser Anforderung eine niedrigere Priorität zugeordnet.

**A5: Erweiterbarkeit** Des Weiteren sollte das System flexibel und erweiterbar sein, um die Integration neuer Analysen und Verbesserungen zu ermöglichen. Es sollte in der Lage sein, neue Sprachmodelle zu integrieren bzw. auszutauschen, um die Extraktionsgenauigkeit und -effizienz zu verbessern. Es sollte die Möglichkeit bieten, die Sprache der Eingabe anzupassen, um eine angemessene Textverarbeitung

durchzuführen. In erster Linie liegt jedoch der Fokus in der Informationsextraktion aus der deutschen Sprache.

**A6: Dokumentation** Die verwendete Textverarbeitung sollte über eine umfassende Dokumentation bzw. Beschreibung verfügen, die dem Benutzer Informationen, Anleitungen und Hilfestellungen zur Nutzung und Konfiguration der notwendigen Installationsbibliotheken bietet. Diese Anforderung erleichtert den weiteren Nutzern im späteren Verlauf, gewisse Anpassung in der Implementierung vorzunehmen. Diese Anforderung verhilft der weiteren Bearbeitung in der Zukunft und besitzt wie die Anforderung A5 eine niedrigere Priorität.

Diese Anforderungen dienen als Ausgangspunkt für die Entwicklung und Implementierung des Systems zur Extraktion eines ERM aus Texten. Während der Evaluierung des entwickelten Systems im späteren Verlauf können diese Anforderungen bewertet und abgeglichen werden. Zur besseren Übersicht werden diese Anforderungen in der Tabelle 3.1 festgehalten.

Kürzel	Anforderungsbereiche
A1	Schnittstellenanknüpfung
A2	korrekte Informationsextraktion
A3	Performance
A4	Validierung & Fehlerbehandlung
A5	Erweiterbarkeit
A6	Dokumentation

Tabelle 3.1.: Kürzel und Anforderungsbereiche

# 4. Verwandte Arbeiten

Das vierte Kapitel dieser Arbeit widmet sich der Untersuchung von verwandten Arbeiten im Bereich der NLP-basierten ER-Generierung. In diesem Kapitel wird ein Überblick über bestehende Ansätze zur NLP-basierten ER-Generierung gegeben. Dabei werden verschiedene Techniken und Methoden betrachtet, um die Herausforderungen im Bereich der Informationsextraktion effektiv anzugehen. Im Anschluss wird eine kritische Bewertung und Analyse der vorhandenen Methoden durchgeführt. Es wird die Leistungsfähigkeit der verschiedenen Ansätze, z.B. hinsichtlich der Genauigkeit, untersucht. Zudem wird auf etwaige Grenzen dieser Methoden eingegangen, um ein umfassendes Verständnis ihrer Anwendbarkeit zu erlangen.

## 4.1. Überblick über bestehende Ansätze zur NLP-basierten ER-Generierung

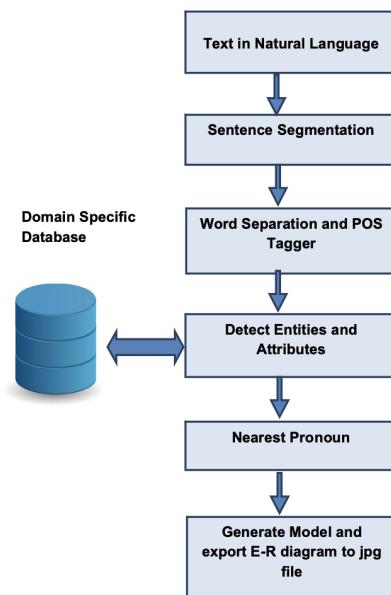


Abbildung 4.1.: allgemeiner Systemaufbau aus der Arbeit von Ghosh et al. [28]

In der Arbeit von Ghosh et al. [28] wird eine Methode vorgeschlagen, die grammatisches Wissensmuster sowie lexikalische und syntaktische Analysen von Anforderungstexten zur Erstellung von ERM verwendet. Dieses System setzt für eine korrekte Informationsextraktion voraus, dass der Eingabetext nur aus einfachen Subjekt-Prädikat-Objekt-Sätzen besteht. Durch diesen Satzbau ist es möglich, Entitäten (Subjekt), Beziehungen (Verb) und Attribute oder weitere in Beziehung stehende Entitäten (Objekt) zu detektieren. Die Gesamtarchitektur in Abbildung 4.1 demonstriert die Vorgehensweise von der Eingabe des unstrukturierten Textes bis hin zur Generierung des ERM. [28]

Bis zu dem Schritt, bevor die domainspezifische Datenbank für die Identifikation der ERM-Komponenten zum Einsatz kommt, werden ausschließlich Techniken der NLP, wie zum Beispiel *Sentence Segmentation*, *Word Separation (Tokenization)* oder *POS-Tagging* verwendet. [28]

Die Segmentierung wurde in dieser Arbeit von Ghosh [28] so vorgenommen, dass Fälle im Englischen, wie zum Beispiel, „Mr. Mustermann“, bei denen (.) nach Mr als Satzende erscheint, nicht erkannt werden. Beim *POS-Tagging* werden den einzelnen Wörtern, die entsprechenden Wortarten, wie z.B. Nomen, Verb, Pronomen, Adjektiv, Präposition etc., zugeordnet. In dieser Veröffentlichung wird die Erkennung von Entitätsattributen und Beziehungen mithilfe einer Datenbank und des Support Vector Machine (SVM)-Klassifikators durchgeführt. [28]

Es werden sechs Tabellen (Wort & Synonym für jede ERM-Komponente) implementiert, die bereichsspezifische Wörter und Synonyme enthalten. Um zusammengehörige Entitäten zu erfassen, werden die Synonyme eines Wortes aus der Tabelle mit derselben ID versehen. Damit wird verhindert, dass redundante SQL-Tabellen erstellt werden. Die Pronomen wie beispielsweise „er“, „sie“ oder „es“ werden bereits in der POS-Tagging-Phase erkannt. In dieser Phase wird das Pronomen anhand der nächstgelegenen vorherigen Entität identifiziert, die entweder im selben Satz oder in den vorherigen Sätzen vorhanden ist. Der textverarbeitungsspezifische Fachbegriff hierfür lautet *Coreference Resolution*. [28, 29]

In der Veröffentlichung von Kashmira et al. [30] werden die ERM-Komponenten über neuronale Netze erfasst. Es werden drei Hauptmodule, nämlich das Preprocessing-Modul, das Modul für maschinelles Lernen und das ER-Modellierungsmodul präsentiert. Die Abbildung 4.2 veranschaulicht den gesamten Programmablauf.

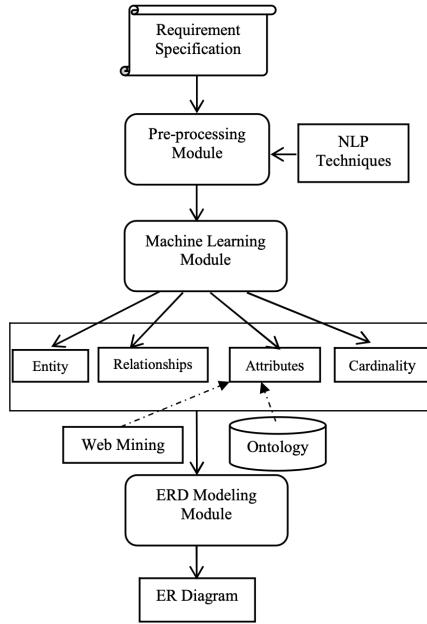


Abbildung 4.2.: Module des Programms [30]

Im ersten Schritt wird das Preprocessing-Modul unter Verwendung von *NLTK* implementiert, um den Text vorzuverarbeiten. Dazu gehören Schritte wie die Umwandlung des Textes in Kleinbuchstaben, die Tokenisierung in Sätze usw. Anschließend wird das Modul für maschinelles Lernen mithilfe *Supervised Learning* implementiert. [30]

Dieses Modul wird mit einem englischen Datensatz trainiert, bei dem die Wörter in verschiedene Kategorien eingeteilt werden, darunter Entität, Unterentität, Attribut und irrelevante Kategorie. Ein Beispiel für eine annotierte Anforderungsspezifikation ist in Abbildung 4.3 dargestellt, wobei grün für Entität, orange für Unterentität und rosa für Attribut steht. Beim Training des Modells werden vier Klassifikatoren berücksichtigt: *Random Forest*, *Naive Bayes*, *Decision Table* und *Sequential minimal optimization (SMO)*. [30]

there are **lecturers** who teach **courses** **instructors** who assist in doing tutorials and practical and projects under taken by **lecturers** a **lecturer** can be employed as either **professor** **senior lecturer** or **probationary lecturer** each **lecturer** has an **id** **name** **gender** **year of birth** **salary** **educational qualifications** a **professor** has a **rank** a **senior lecturer** has a **grade** and **probationary lecturer** has a **period** a **course** has a **course code** **title** and **credit value** a **course** can be either **undergraduate** or **post graduate** an **under graduate course** can have a **type** while **post graduate course** has a **level** an **instructor** has an **id** and **name** each **course** is assigned to **lecturer** and **instructor** a **course** is assigned to one or many **lecturers** and one or many **instructors** an **instructor** is assigned for one or many **course** and one or many **lecturers** a **lecturer** can undertake zero or many **research projects** for a **project** one or many **lecturers** can be involved each **project** has a **title** **grant name** **duration** and **budget**

Abbildung 4.3.: Ausschnitt eines Beispieltextes mit Annotation [30]

Um das Problem der Attributauswahl für Entitäten in einem ER-Diagramm anzugehen, nutzt das vorgeschlagene Modell eine Kombination aus *Ontologie* und *Web Mining*. Durch den Einsatz von *Ontologie* wird versucht, relevante Attribute aus den extrahierten Entitäten zu filtern. Zusätzlich wird *Web Mining* eingesetzt, um weitere Informationen aus dem Web zu gewinnen, die bei der Bestimmung der Attribute hilfreich sein können. [30]

Für die Bewertung der Genauigkeit der Algorithmen werden die folgenden Formeln<sup>2</sup> für *Precision* und *Recall* verwendet [30]:

$$\text{Precision} = \frac{TP}{(TP + FP)} \quad (4.1)$$

$$\text{Recall} = \frac{TP}{(TP + FN)} \quad (4.2)$$

mit:  $TP$  = korrekt identifizierte Entitäten

$FP$  = eigentliches Attribut oder Beziehung wurde als Entität identifiziert

$FN$  = eigentliches Entität wurde als Attribut oder Beziehung identifiziert

Diese Evaluationsmetriken dienen dazu, die Leistung der Algorithmen zu quantifizieren.  $TP$  steht dabei für *True Positives*, also die Anzahl der korrekt vorhergesagten positiven Ergebnisse.  $FP$  steht für *False Positives*, also die Anzahl der fälschlicherweise als positiv vorhergesagten Ergebnisse.  $FN$  steht für *False Negatives*; die Anzahl der fälschlicherweise als negativ vorhergesagten Ergebnisse. [32, 33]

Die *Precision* gibt an, wie viele der vorhergesagten positiven Ergebnisse tatsächlich korrekt sind. Ein hoher Präzisionswert zeigt an, dass die Vorhersagen des Algorithmus zuverlässig sind. Der Recall beschreibt, wie viele der tatsächlich positiven Ergebnisse vom Algorithmus korrekt erkannt wurden. Ein hoher Recall-Wert deutet darauf hin, dass der Algorithmus in der Lage ist, die relevanten positiven Ergebnisse effektiv zu identifizieren. Durch die Berechnung von *Precision* und *Recall* können die Algorithmen verglichen und ihre Leistung bewertet werden. Die folgende Abbildung 4.4 stellt die Ergebnisse der Evaluation des Modells dar. [32, 33]

---

<sup>2</sup>Diese Formel beschreiben Precision bzw. Recall für die Entitäten. Analog hierzu kann man die Bedeutung der Formel auch für Attribute und Beziehungen anpassen. [31]

Algorithm	Class	Precision	Recall
Random Forest	Entity	0.954	0.906
	Attribute	0.989	0.743
	Sub Entity	0.84	0.691
	Irrelevant	0.822	1.000
Naive Bayes	Entity	0.94	0.923
	Attribute	0.985	0.743
	Sub Entity	1.000	0.164
	Irrelevant	0.822	1.000
Decision Table	Entity	0.958	0.822
	Attribute	0.98	0.618
	Sub Entity	0.272	0.704
	Irrelevant	0.846	0.935
SMO	Entity	0.956	0.904
	Attribute	0.975	0.748
	Sub Entity	0.849	0.664
	Irrelevant	0.854	0.999

Abbildung 4.4.: Evaluationsergebnisse der Metriken für verschiedene Klassifikatoren [30]

Bei der Evaluierung und dem Vergleich der Genauigkeit der oben genannten Algorithmen zeigt sich, dass alle Algorithmen eine bemerkenswerte Genauigkeit bei der Identifizierung von Entitäts-, Attributs- und irrelevanter Klassen erreichen [30]. Dies ist ein vielversprechendes Ergebnis und deutet darauf hin, dass das Modell in der Lage ist, die automatisierte Generierung von ER-Diagrammen aus natürlichsprachlichem Text effektiv zu unterstützen.

Auch die Publikation von Habib [34] verfolgt zu Beginn ähnliche Vorverarbeitungsschritte, wie Ghosh [28] und Kashmira et al.[30]. Nach dem *Parsing*-Prozess erhält man den grammatischen Satzaufbau, sodass die Komponenten des ERM regelbasiert ermittelt werden können. Wie in Abbildung 4.5 gezeigt, werden die Wörter in eine Parse-Baum-Struktur umgewandelt, um zu verstehen, wie die einzelnen Satzteile miteinander in Beziehung stehen. Durch entsprechende Regeln für Satzstrukturen können Entitäten, Attribute, Kardinalitäten und Beziehungen bestimmt werden.

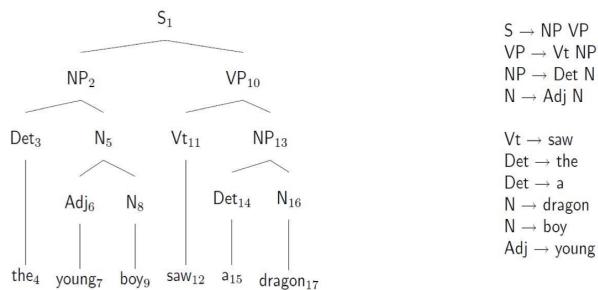


Abbildung 4.5.: Top-Down-Parsing-Tree für einen Beispielsatz [35]

**Literatur in ähnlichem Zusammenhang** Es gibt einige weitere Veröffentlichungen, die in eine ähnliche Richtung gehen und die Thematik der automatischen ER-Generierung im Kontext von NLP näher untersuchen. Genannt sei hier eine Veröffentlichung um die Autoren Omar et al. [36]. Diese Literatur beschreibt heuristikbasierte Analysemöglichkeiten zur ER-Modellgenerierung.

Dahingegen verfolgen Omar und Abdulla [37] den Ansatz, ein Machine Learning Modell zu trainieren, dass die Entitäten aus dem Text extrahieren kann. Abhängig von der Komplexität des Eingabetextes und dem Trainingsumfang ergeben sich für das Modell Präzisionswerte von bis 85 %.

Die Ergebnisse von Btoush und Hammad [38] lassen sich auch in einen ähnlichen Kontext einordnen. Hierbei wird ein Verfahren vorgestellt, das wie [34], bestimmte Regeln für die Informationsextraktion aus Texten definiert und anwendet. Für die Identifikation von Entitäten können bestimmte Regeln verwendet werden, welche in Kapitel 5.4 näher beschrieben werden.

## 4.2. Kritische Bewertung und Analyse der vorhandenen Methoden

Es kann festgehalten werden, dass für die ER-Modellgenerierung grundsätzlich zwei methodische Ansätze eingesetzt werden. Die ERM-Komponenten werden entweder regelbasiert oder mithilfe neuronalen Netzen bzw. künstlicher Intelligenz ermittelt.

Beide Ansätze haben Vor- und Nachteile. Der größte Vorteil der regelbasierten Extraktion ist die zeiteffizientere Implementierung, da im Gegensatz zu neuronalen Netzen keine Datenvorverarbeitung und Training notwendig ist. Des Weiteren ist durch die eindeutige Definition der Regeln, eine hundertprozentige Extraktionswahrscheinlichkeit gegeben. In neuronalen Netzen bleibt immer eine Restgenauigkeit übrig. Im Kontrast hierzu ist die modellbasierte Extraktion nicht auf einige Regeln beschränkt, sondern kann komplexe und verschachtelte Satzstrukturen für die Ermittlung der ERM-Komponenten einlernen. Diese Satzstrukturen können linguistische Variationen und Mehrdeutigkeiten enthalten, welche durch neuronale Netze einfacher erkannt werden können als durch festgelegte Heuristiken. Der heuristische Ansatz ist eher für kleine Anwendungsbereiche geeignet und kann seine Effektivität nicht in großen Anwendungsbereichen aufrechterhalten. Darüber hinaus sind regelbasierte Ansätze nur effektiv, sobald man sich auf eine gewisse Satzstruktur (z.B. Subjekt-Verb-Objekt (SVO)-Sätze) des vermeintlich „unstrukturierten Textes“ beschränkt. Ähnliches gilt auch für den modellbasierten Ansatz; hierbei wird das Modell nur auf eine spezielle Teilmenge der relevanten Domänen trainiert, wodurch die Flexibilität zur Analyse der nicht berücksichtigten Anwendungsbereiche sinkt.

In der Arbeit von Habib [34] werden verschiedene Szenarien mit sowohl einem regelbasierten als auch einem probabilistischen Modul getestet. Für das Training werden 500 Sätze verwendet und für den Test werden 100 Sätze genutzt. Die Ergebnisse werden anhand eines regelbasierten Mappings und eines probabilistischen Modell-Mappings überprüft. Die Ergebnisse der Tabelle 4.1 legen nahe, dass probabilistische Modelle vielversprechender sind.

Kategorie	Präzision	Recall
regelbasiert	86.9	86.3
modellbasiert	91.3	89.0

Tabelle 4.1.: Modell- und regelbasierte Präzisions- und Recallwerte [34]

# 5. Methodik

In diesem Kapitel werden verschiedene Methoden besprochen, um in einem unstrukturiertem Text die ERM-Komponenten zu extrahieren. Diese Methoden werden für regelbasierte und modellbasierte Ansätze, welche in Kapitel 4 besprochen wurden, angewendet. Diese Methoden kommen in unterschiedlichen Phasen des NLP-Workflows vor. Gemäß der Abbildung 5.1 wird die Textverarbeitung in vier verschiedene Phasen unterteilt.

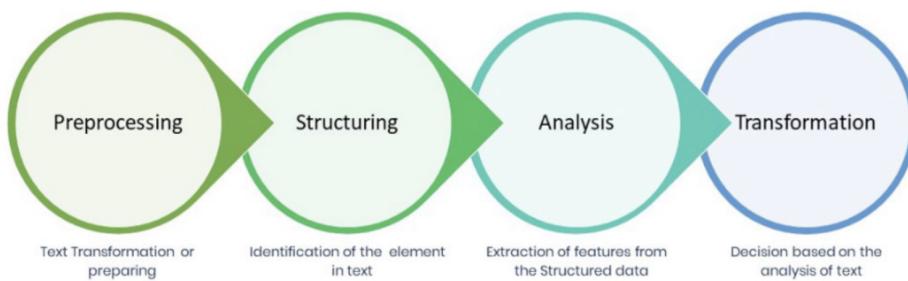


Abbildung 5.1.: NLP-Workflow [3]

## 5.1. Datenbeschaffung und -vorverarbeitung

**Datenbeschaffung** Um das Sprachmodell zu trainieren und es daraufhin mit Testdaten zu validieren, werden textuelle Datenbankszenarien benötigt. Auch im Falle einer regelbasierten Analyse werden diese Daten für die Evaluation hilfreich sein. Für diese Projektarbeit werden hauptsächlich Texte aus verschiedenen Quellen<sup>3</sup> wie Webseiten, Fachartikeln, Übungssammlungen von verschiedenen Universitäten oder anderen schriftlichen Materialien beschaffen.

**Datenvorverarbeitung** Die Datenvorverarbeitung (*eng. Data Preprocessing*) beinhaltet die Bereinigung und Transformation der Rohdaten, um sicherzustellen, dass

---

<sup>3</sup>Die Quellen, welche für dieses Projekt verwendet wurden, sind in *trainData\_source.txt* unter dem Ordner *literature* im Git-Repository vermerkt.

diese für die ER-Generierung geeignet sind. Zuerst soll eine Datenbereinigung stattfinden. Hierbei werden unnötige oder fehlerhafte Daten entfernt, um die Qualität der Daten zu verbessern. Dies kann das Entfernen von Duplikaten, das Korrigieren von Tippfehlern oder das Handhaben fehlender Werte umfassen.

Die Levenshtein-Distanz ist ein Konzept aus der Informatik, das zur Berechnung der Ähnlichkeit zwischen zwei Zeichenketten verwendet wird. Diese Metrik ist nach dem sowjetischen Mathematiker Wladimir Levenshtein benannt, der dies im Jahr 1965 eingeführt hat. Die Levenshtein-Distanz misst die Anzahl der Operationen, die erforderlich sind, um eine Zeichenkette  $w_1$  in eine andere Zeichenkette  $w_2$  zu transformieren. Somit kann diese Technik, dafür eingesetzt werden, um gewisse Rechtschreibfehler aus Wörtern zu beheben. Die zugelassenen Operationen sind das Einfügen, Löschen und Ersetzen eines einzelnen Zeichens. Jede Operation hat einen Kostenwert, der üblicherweise eins beträgt. In Sonderfällen können die Kosten individuell angepasst werden, um bestimmte Editieraktionen unterschiedlich zu gewichten. Um einen Wert im Intervall [0,1] zu erhalten, wurde die Levenshtein-Formel entsprechend angepasst. [39]

$$\text{Levenshtein - Ähnlichkeit}(w_1, w_2) = 1 - \frac{\text{Levenshtein - Distanz}(w_1, w_2)}{|w_1|}$$

Die Levenshtein-Distanz zwischen den Wörtern  $w_1 = \text{Raum}$  und  $w_2 = \text{Schaum}$  beträgt 3. Die Levenshtein-Ähnlichkeit berechnet sich demnach wie folgt:

$$\text{Levenshtein - Ähnlichkeit}(\text{Raum}, \text{Schaum}) = 1 - \frac{3}{4} \approx 0.25$$

Dahingegen berechnet sich die Levenshtein-Ähnlichkeit für die Wörter  $w_1 = \text{Finanzwesen}$  und  $w_2 = \text{Finanzbesen}$  zu:

$$\text{Levenshtein - Ähnlichkeit}(\text{Finanzwesen}, \text{Finanzbesen}) = 1 - \frac{1}{11} \approx 0.91$$

Bei großen Datenmengen kann es hilfreich sein, die Daten in Trainings- und Validierungsdaten aufzuteilen, sodass das Sprachmodell mit den Trainingsdaten trainiert und anschließend mit den Validierungs- bzw. Testdaten validiert werden kann. Dadurch wird sichergestellt, dass die Modellevaluationsparameter (*Precision* bzw. *Recall*) repräsentativ und aussagekräftig sind, da die Validierungsdaten für das Modell unbekannt sind. Die Testdaten werden erst im letzten Schritt angewendet, nachdem die Hyperparameter des Modells optimiert und somit das bestmögliche Modell gefunden wurde (siehe Abbildung 5.2).

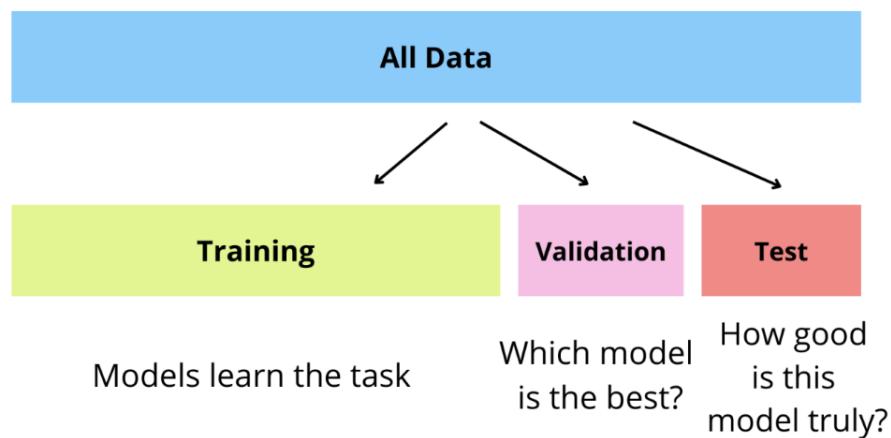


Abbildung 5.2.: Aufteilung des Datensatzes in Training, Validation und Testing [40]

Darüber hinaus müssen die Daten in das gewünschte Sprachformat gebracht werden. Falls die Texte von der deutschen Sprache ins Englische übersetzt werden, müssen dabei die grammatischen und linguistischen Abweichungen, wie z.B. Umlaute etc., in der Analyse mitberücksichtigt werden.

## 5.2. Textstrukturierung

Wenn die Textdaten aus verschiedenen Quellen stammen, müssen sie ggf. bearbeitet werden, um eine einheitliche Datenbasis zu schaffen. Dies kann z.B. die Anpassung der Satzstruktur an SVO beinhalten. Folgende Beispielverdeutlichung zeigt diesen Schritt:

*„Ein Unternehmen kann mehrere Abteilungen haben, wobei jede Abteilung von einem oder mehreren Mitarbeitern geleitet wird und spezifische Aufgaben und Verantwortlichkeiten in Bezug auf die Unternehmensziele übernimmt.“*

Nach der Umwandlung in die SVO-Form und der Filterung von irrelevanten Informationen, wird der Satz nun in die folgenden Sätze unterteilt.

*„Ein Unternehmen hat mehrere Abteilungen. Jede Abteilung wird von einem oder mehreren Mitarbeitern geleitet.“*

Eine weitere wichtige Methode, die zu der Textstrukturierung gehört, sind die Konzepte der *Term Frequency* (TF) und *Document Frequency* (DF). Diese Konzepte

können bei der Identifizierung von ERM-Komponenten und Filterung von relevanten Sätzen verhelfen.

Die *TF* wird verwendet, um die Häufigkeit eines bestimmten Begriffs in einem Textdokument zu berechnen. *TF* misst, wie oft ein Begriff in einem Dokument vorkommt. Die Berechnung der *Term Frequency* kann auf verschiedene Weisen erfolgen, wobei die gängigste Methode die Zählung der Anzahl des Begriffs im Text ist. Die Formel für die Berechnung lautet [3]:

$$TF_n(t, d) = \frac{TF_a(t, d)}{\max_{t_i \in d}(TF_a(t_i, d))}$$

mit:  
 $n$  = relative Häufigkeit  
 $a$  = absolute Häufigkeit  
 $t$  = Term  
 $d$  = Dokument

Der Zähler charakterisiert die Anzahl des Terms  $t$  in Dokument  $d$  und der Nenner beschreibt die maximale Häufigkeit eines Terms in  $d$ . Somit wird eine Verzerrung in großen Dokumenten verhindert, indem die absolute Häufigkeit normalisiert wird. *TF* liefert einen Wertebereich von 0 bis 1, wobei ein Wert nahe bei 1 auf eine hohe Häufigkeit des Begriffs im Dokument hinweist. [41]

Die *Document Frequency* dahingegen berechnet die Häufigkeit eines Begriffs in einem Korpus von Textdokumenten. *DF* misst, in wie vielen Dokumenten ein bestimmter Begriff mindestens einmal vorkommt und ermöglicht es, die Relevanz eines Begriffs in der gesamten Dokumentensammlung zu bewerten. Beispielsweise wird dies angewendet, um Stopwörter zu identifizieren, die in vielen Dokumenten vorkommen und daher wenig Aussagekraft für die Identifizierung von Stopwörtern haben. Auch der Wert von *DF* liegt zwischen 0 und 1. [41]

Die Kombination von *TF* und *DF* führt zur *TF-IDF*-Methode (*Term Frequency-Inverse Document Frequency*), welche durch die folgenden Formeln errechnet wird [3]:

$$IDF(t) = \log \frac{N}{|d : t \in d| + 1}$$

$$TF\_IDF(t, d) = TF_n(t, d) \times IDF(t)$$

mit:  
 $N$  = Gesamtanzahl der Dokumente  
 $t$  = Term  
 $d$  = Dokument  
 $n$  = relative Häufigkeit

Im Nenner von  $IDF(t)$  steht die Anzahl der Dokumente, die der Term  $t$  beinhaltet. Somit wird die TF-IDF-Methode verwendet, um die Relevanz eines Begriffes in einem Dokument innerhalb eines Korpus zu bewerten. Die TF-IDF-Methode hebt wichtige Schlüsselbegriffe hervor, die in einem bestimmten Dokument häufig vorkommen (hoher TF-Wert) und gleichzeitig selten im gesamten Korpus auftreten (hoher IDF-Wert). Diese Techniken tragen dazu bei, auf die wichtigsten Informationen aus unstrukturierten Texten zu fokussieren und den Extraktionsprozess zu verbessern. [32, 41]

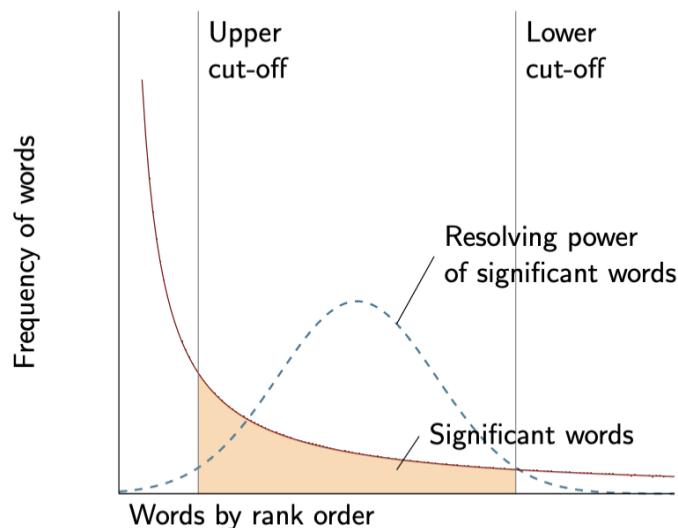


Abbildung 5.3.: Beziehung zwischen Häufigkeit und Rangordnung von Wörtern [3]

Die Abbildung 5.3 verdeutlicht das Zipf'sche Gesetz. Das Gesetz besagt, dass die Häufigkeit eines Wortes umgekehrt proportional zu seiner Rangposition ist. Mit anderen Worten, je häufiger ein Wort ist, desto niedriger ist seine Rangordnung und umgekehrt. Ähnliches besagt die Idee von Luhn. Hierbei werden in der Verteilung von wichtigen Wörtern in Dokumenten zwei Grenzen definiert, die den Bereich der Signifikanz einschränken. Außerhalb dieser Grenzen befinden sich entweder Stoppwörter (*Upper cut-off*) oder seltene Eigennamen (*Lower cut-off*). [3]

### 5.3. Textanalyse

In der Hauptphase der Textanalyse werden Techniken eingesetzt, die sich explizit auf die Extraktion von Merkmalen des nun strukturierten Textes fokussieren.

### 5.3.1. Tokenisierung und Part-of-Speech-Tagging

Die Tokenisierung ist der erste Schritt bei der Textanalyse und beinhaltet die Aufteilung eines Textes in einzelne Wörter oder Token. Die Tokenisierung ist ein wichtiger Schritt, da sie die Grundlage für die weitere Analyse bildet. Die Tokenisierung kann in verschiedenen Sprachen und für verschiedene Textformate durchgeführt werden. Durch die Tokenisierung ergeben sich verschiedene Möglichkeiten, die Worttokens hinsichtlich ihrer Flexion anzupassen. Zum Beispiel, wird durch eine Lemmatisierung die Grundform des Wortes wiederhergestellt. [28]

Nach der Tokenisierung folgt in den meisten NLP-Projekten das POS-Tagging, um jedem Token eine grammatischen Kategorie zuzuweisen, wie z. B. Substantiv, Verb, Adjektiv usw. Das POS hilft dabei, die grammatische Struktur des Textes zu erfassen. Das folgende Beispiel demonstriert die beiden beschriebenen Analyseschritte. [28]

Beispielsatz:

„Ein Unternehmen hat mehrere Abteilungen.“

Tokenisiert:

[„Ein“, „Unternehmen“, „hat“, „mehrere“, „Abteilungen“, „.“]

POS:

[„Artikel“, „Substantiv“, „Verb“, „Adjektiv“, „Substantiv“, „Satzzeichen“]

### 5.3.2. Named Entity Recognition

NER ist eine NLP-Technik, die darauf abzielt, benannte Entitäten in einem Text zu identifizieren und zu klassifizieren. Benannte Entitäten sind spezifische Bezeichnungen von Personen, Orten, Organisationen, Zeitangaben und anderen wichtigen Entitäten. [42]

Für die Erkennung gibt es sowohl regel- als auch modellbasierte Umsetzungen. Allerdings mangelt es bei den regelbasierten Systemen an Übertragbarkeit und Robustheit. Diese Art von Ansätzen sind oft domänen- und sprachspezifisch und lassen sich nur mit erhöhtem Aufwand an neue Domänen und Sprachen anpassen. NER-Systeme, die auf maschinellem Lernen basieren, lösen die Identifikationsprobleme, indem es die Probleme in Klassifikationsaufgaben umwandelt. Es werden unterschiedliche Klassifikationsmodelle genutzt, um bestimmte Zusammenhänge

im Text zu erkennen. Weiterhin können modellbasierte NER-Ansätze in *supervised* (dt. überwacht) und *unsupervised* (dt. unüberwacht) learning unterteilt werden. [43]

Beim überwachten Lernen werden dem Modell gelabelte Daten injiziert, sodass das Modell mithilfe einer ständigen Rückkopplung in eine vorgegebene Richtung hintrainiert wird. Unüberwachtes Lernen stellt eine andere Methode des maschinellen Lernens dar, bei der ein Modell ohne vorgegebene Rückmeldungen lernt. Das Hauptziel besteht darin, Repräsentationen aus den vorliegenden Daten zu generieren. Diese Repräsentationen können für Datenkomprimierung, Klassifizierung oder Entscheidungsfindung verwendet werden. Im Bereich der NER ist reines unüberwachtes Lernen nicht besonders weit verbreitet. Stattdessen wird in der Veröffentlichung von Baohang et al. [44] eine neue multimodale NER vorgestellt, welches das Ziel verfolgt, benannte Entitäten in freiem Text und Bilder zu extrahieren, um sie in benutzerdefinierte Kategorien zu klassifizieren. Hierbei werden Kombinationen aus Variational Autoencoder (VAE) und Long short-term memory (LSTM) verwendet, sodass das Gesamtmodell die Text- und Bildinformationen vereinigt und die Fehlanpassung zwischen dem veröffentlichten Text und dem Bild mitberücksichtigt (siehe Abbildung 5.4). Daher spricht man hierbei von *halb überwachtem Lernen*. Das verwendete ResNet-Modell wirkt dem im Trainingsprozess entstehenden „Vanishing Gradient“-Problem entgegen.

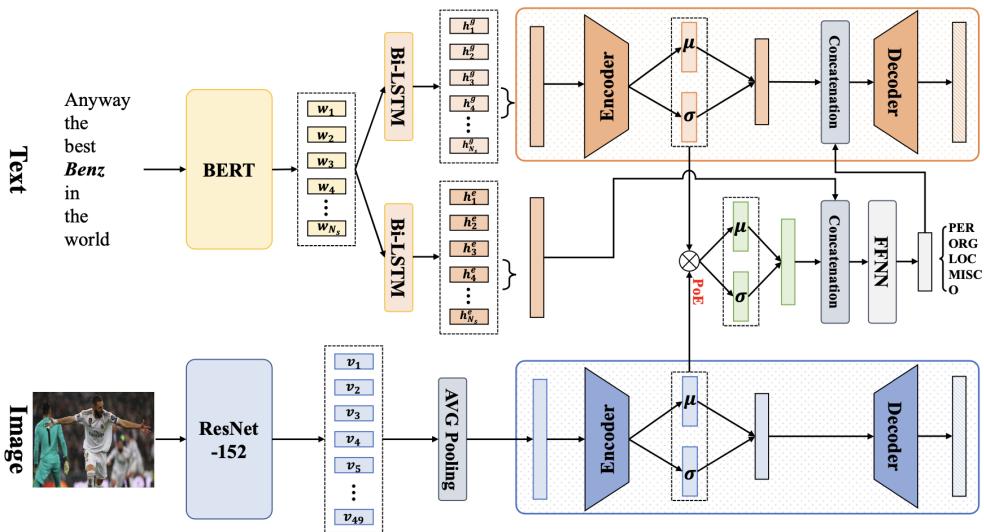


Abbildung 5.4.: Die Gesamtarchitektur des multimodalen variationalen Autoencoders für semi-supervised MNER [44]

VAE werden häufig in der Bildverarbeitung zur Anomalieerkennung oder zum Entrauschen von Bildern genutzt. LSTM eignen sich für Daten, deren Einhaltung einer Reihenfolge, besonders wichtig ist. Typische Eingabedaten für LSTM sind

sequentielle (zeitabhängige oder textuelle) Daten, welche für Klassifikation oder Vorhersagen eingesetzt werden können. [44]

LSTM ist eine spezielle Art von Recurrent Neural Network (RNN). RNN sind in der Lage, Informationen über die Zeit zu speichern. Hierfür wird eine Schleife in der Netzwerkstruktur erzeugt, sodass Informationen aus vorherigen Zeitschritten berücksichtigt werden (siehe Abbildung 5.5). Somit können RNN Abhängigkeiten und Beziehungen zwischen den Wörtern im Text besser erfassen. Das EingabefORMAT für diese neuronale Netzarchitektur sind maschinenlesbare Vektoren.

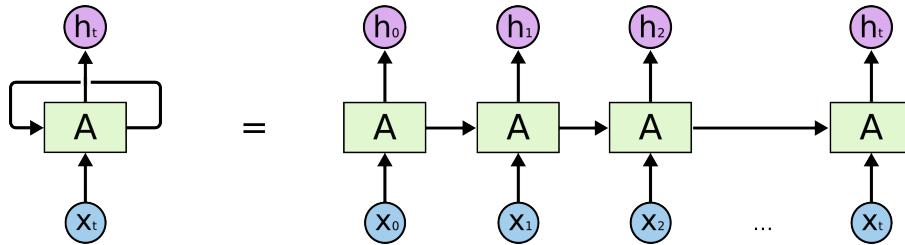


Abbildung 5.5.: Recurrent Neural Network [45]

Allerdings haben klassische RNN auch einige Herausforderungen. Eines der Hauptprobleme ist das bereits erwähnte „Vanishing Gradient“-Problem während dem Trainingsprozess. Dies führt dazu, dass Informationen aus fernen Zeitschritten kaum berücksichtigt werden können. In ResNet-Modellen wird dieses Problem durch *Skip-Connections* zwischen den Schichten im neuronalen Netz gelöst. LSTM dagegen, verfügen über sogenannte „Gates“, die es dem Modell ermöglichen, zu entscheiden, welche Informationen verworfen oder gespeichert werden sollen. Diese Gating-Mechanismen ermöglichen es dem LSTM, relevante Informationen über längere Zeittabstände hinweg zu speichern, ohne dass der Gradient während dem Training zu null wird. [46, 47]

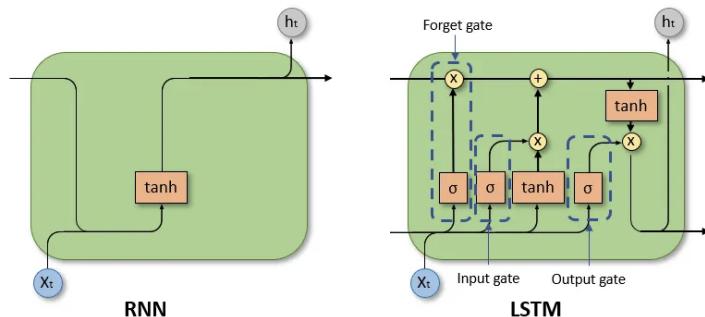


Abbildung 5.6.: Vergleich der Modellarchitektur von RNN und LSTM [46]

Das *Forget Gate* entscheidet, welche Informationen aus dem vorherigen Zustand

verworfen werden sollen. Es nimmt den vorherigen Zustand und die aktuelle Eingabe und gibt eine Zahl zwischen 0 und 1 für jede Zelle im Zustand zurück. Eine 0 bedeutet, dass die Information vollständig verworfen wird, während eine 1 bedeutet, dass die Information vollständig beibehalten wird. Dieser Mechanismus ermöglicht es dem LSTM, zu vergessen, was nicht länger relevant ist. [47]

Das *Input Gate* verwendet die aktuelle Eingabe und den vorherigen Zustand, um zu berechnen, welche Werte aktualisiert werden sollen. Ähnlich wie das *Forget Gate* gibt das *Input Gate* für jede Zelle im Zustand einen Wert zwischen 0 und 1 zurück. Somit kann geregelt werden, welche Informationen teilweise bzw. vollständig hinzugefügt werden. [47]

Im *Output Gate* wird bestimmt, welche Informationen aus dem aktualisierten Zellzustand zur Ausgabe übergeben werden sollen. Durch die Verrechnung der aktuellen Eingabe und den aktualisierten Zellzustand wird ermittelt, welche Werte in den Ausgabevektor übergeben werden. Das *Output Gate* ähnelt den anderen Gates und gibt Werte zwischen 0 und 1 zurück. [47]

Die Modellarchitektur von RNN ist im Vergleich zum LSTM deutlich einfacher aufgebaut. Während das RNN nur aus einem neuronalen Netz mit der *tanh* Aktivierungsfunktion besteht, beinhaltet ein LSTM vier neuronale Netze mit unterschiedlichen Aktivierungsfunktionen. Somit ist die Anzahl an trainierbaren Parametern im LSTM viermal so hoch.

Mithilfe der eben beschriebenen Modellarchitekturen kann man gezielt eigene Modelle erstellen und ihre Hyperparameter entsprechend anpassen, sodass man eigene Entitätenklassen definieren und damit das Modell trainieren kann. Dies wird auch als *Custom Named Entity Recognition* bezeichnet. Hinsichtlich der Extraktion von ER-Komponenten würde sich für den Beispielsatz folgende Klassen eignen:

„Ein (Kardinalität) Unternehmen (Entität) wird von mehreren (Kardinalität) Mitarbeitern (Entität) geleitet (Beziehung).“

### 5.3.3. Dependency Parsing

Das *Dependency Parsing* ist eine NLP-Technik, die verwendet wird, um die syntaktischen Abhängigkeiten zwischen den Wörtern in einem Satz zu analysieren und zu erfassen. Es zeigt, wie die Wörter im Satz grammatisch miteinander verbunden sind und welche Wörter das Subjekt, das Prädikat, das Objekt usw. bilden. [34]

Das *Dependency Parsing* ist wichtig für das Verständnis der Beziehungen zwischen den Entitäten im Text und trägt dazu bei, die semantische Struktur eines Satzes zu erfassen. Die folgende Tabelle 5.1 zeigt diese Anwendung anhand eines Beispielsatzes mithilfe der Python-Bibliothek *SpaCy*, welches für zahlreiche NLP-Aufgaben

spezialisiert ist:

Token	Dependency	Bedeutung
Ein	det	Determinator (begleitet „Unternehmen“)
Unternehmen	nsubj	Nominativsubjekt von „hat“
hat	ROOT	Hauptverb
mehrere	amod	für „Abteilungen“
Abteilungen	obj	direkte Objekt des Verbs „hat“
.	punct	Satzzeichen

Tabelle 5.1.: Tabelle mit Dependency Auflösung gemäß *SpaCy*.

Das *Dependency Parsing* wird nur innerhalb eines Satzes angewendet. Um satzübergreifende Beziehungen zu identifizieren, kann die *Coreference Resolution* angewendet werden. Diese Technik beschäftigt sich mit der Identifizierung und Auflösung von koreferenten Ausdrücken in einem Text. Koreferente Ausdrücke sind Wörter, die sich auf dieselbe Entität beziehen, jedoch unterschiedliche Formen haben. Das Ziel der *Coreference Resolution* besteht darin, diese Referenzen aufzulösen. Anhand des folgenden Satzes soll die *Coreference Resolution* demonstriert werden. [29]

„Ein Unternehmen wird von mehreren Mitarbeitern geleitet. Sie übernehmen spezifische Aufgaben und Verantwortlichkeiten in Bezug auf die Unternehmensziele.“

Nach der *Coreference Resolution*:

„Ein Unternehmen wird von mehreren Mitarbeitern geleitet. **Die Mitarbeiter** übernehmen spezifische Aufgaben und Verantwortlichkeiten in Bezug auf die Unternehmensziele.“

## 5.4. Extraktion des Entity-Relationship-Modells

In diesem Kapitel soll auf näher auf die regelbasierte Vorgehensweise zur Extraktion von ER-Komponenten eingegangen werden. Es soll untersucht werden, welche Regeln definiert werden können, um bestimmte Komponenten des ER-Modells zu filtern.

#### 5.4.1. Identifikation von Entitäten

Bei der Identifikation von Entitäten können verschiedene Regeln verwendet werden, um diese zu erkennen. Im Folgenden sind einige häufig verwendete Regeln aufgelistet [34]:

- Ein allgemeines Substantiv kann einen Entitätentyp bezeichnen.
- Ein Eigenname kann eine Entität bezeichnen.
- Die Struktur eines Spezialisierungssatzes „A ist ein B“ kann zwei Entitäten zueinander in Beziehung setzen. Somit können vererbte Entitäten erkannt werden.
- Ein Gerundium kann einen Entitätentyp bezeichnen.
- Ein Substantiv wie „Datenbank“, „Datensatz“, „System“, „Information“ und „Organisation“ kann nicht als Kandidat für einen Entitätentyp infrage kommen.

Es ist jedoch wichtig zu beachten, dass die Identifikation von schwachen Entitäten, die möglicherweise nicht leicht durch einfache Regeln erfasst werden können, eine Herausforderung darstellen. Die Satzstruktur, in denen schwache Entitäten definiert sind, können von Text zu Text abweichen. In solchen Fällen sind fortgeschrittenere Techniken wie maschinelles Lernen erforderlich, um eine präzise Identifikation zu erreichen. Ähnliches gilt auch für Spezialisierungssätze, in denen die ISA-Vererbung von Entitäten beschrieben wird.

#### 5.4.2. Identifikation von Beziehungen

Um die Abhängigkeiten zwischen verschiedenen Entitäten in einem Text zu verstehen, müssen die Verben zwischen den Substantiven im Satz analysiert werden. Die folgenden Regeln und Muster können für die Identifikation von Beziehungen verwendet werden [34]:

- Ein transitives Verb ist ein Verb, das ein direktes Objekt benötigt, um die Handlung zu vervollständigen. Somit kann es auf einen Beziehungstyp hinweisen.
- Ein Verb gefolgt von einer Präposition wie „von/durch“, „zu/nach“, „auf/an“ und „in“ kann auf eine Beziehung hinweisen.
- Bestimmte Verben, wie z.B. „einschließen“, „einbeziehen“, „besteht aus“, „enthält“, „umfasst“, „unterteilt in“ oder „beinhaltet“ können auf eine vererbende

Beziehung hindeuten. Solche Verben zeigen an, dass eine Entität aus anderen Elementen besteht oder sie umfasst.

- Ein Adverb kann eine Beziehung spezifizieren.

Für die Extraktion von Kardinalitäten aus Sätzen ist die Interpretation von Determinatoren, entscheidend, um die Beziehung zwischen Entitäten zu quantifizieren. Bestimmte Determinatoren wie beispielsweise „ein“, „zwei“, „drei“ oder „keine“ können auf eine spezifische Kardinalität hinweisen. Dahingegen weisen Determinatoren wie z.B. „jeder“, „alle“, „viele“ oder „mehrere“ auf allgemeine Kardinalitäten hin. Falls kein Determinativ oder Artikel vor einem Substantiv vorkommt, wird davon ausgegangen, dass auf eine n-fache bzw. allgemeine Kardinalität abgezielt wird. In diesem Fall treten die Entitäten in der Pluralform auf.

Die folgende Tabelle 5.2 illustriert die Übersetzungen von verschiedenen Determinatoren in die jeweiligen min/max-Kardinalitäten.

Determinatoren	min/max-Kardinalität
ein	(1,1)
viele	(1,n)
jede	(1,1)
genau	(1,1)
kein Determinativ	(n,m)
mindestens ein	(1,n)
höchstens ein	(0,1)

Tabelle 5.2.: Determinatoren und ihre zugehörigen min/max-Kardinalitäten

Die Identifikation von Beziehungen und die Extraktion von Kardinalitäten aus Texten können anspruchsvolle Aufgaben sein, da sie Kontext und Semantik erfordern.

#### 5.4.3. Identifikation von Attributen

Um wichtige Eigenschaften von Entitäten zu erkennen, können die folgenden Festlegungen angewendet werden [34]:

- Ein Substantivsatz im Genitiv zeigt eine Besitzbeziehung an und kann auf ein Attribut der besitzenden Entität verweisen.
- Wenn ein Substantiv von einem anderen Substantiv gefolgt wird und das zweite Substantiv zu einer bestimmten Menge S gehört, wobei S = [Nummer, Code, Datum, Typ, Volumen, Geburt, ID, Adresse, Name], kann dies darauf hindeuten, dass beide Substantive Attribute sind. Andernfalls könnte es sich

um eine Entität handeln.

- Bestimmte Substantive wie „Fahrzeugnummer“ oder „Personen-ID“ beziehen sich auf Attribute.
- Der Possessivfall zeigt normalerweise den Besitz an und kann auf ein Attribut hinweisen.
- Ein Substantivsatz mit dem Verb "hat/haben" kann auf Attribute hinweisen.

Diese Regeln sind nicht in jedem Fall zutreffend und müssen daher allgemeingültiger formuliert werden. In Kapitel 6.3 werden zum Teil eigene Gesetzmäßigkeiten entwickelt, um bestimmte ER-Komponenten zu finden.

Die Identifikation von Primärschlüsseln in einem Text kann eine weitere Herausforderung darstellen, da sie eine spezifische Art von Attributen sind. Ein Adverb wie beispielsweise „eindeutig“ kann auf einen Primärschlüssel einer Entität im Satz hinweisen [34]. Darüber hinaus kennzeichnet das Vorhandensein der Sequenz „-ID“ oder „-Nr“ in einem Wort auch auf ein Attribut.

Die Identifikation von Attributen und Primärschlüsseln erfordert eine genaue Zuordnung dieser Attribute zu den korrekten Entitäten. In komplexen Texten kann es schwierig sein, diese Zuordnung vorzunehmen. Auch eine Differenzierung der Attribute von normalen Entitäten kann eine große Herausforderung sein.

# 6. Implementierung

In diesem Kapitel wird die konkrete Implementierung des entwickelten Lösungskonzepts beschrieben. Bei der Implementierung werden sowohl regelbasierte und modellbasierte Ansätze verwendet. Aufgrund der strukturellen Abhängigkeit der ERM-Komponenten in den Texten eignen sich je nach Komponente unterschiedliche Methoden. In den folgenden Kapiteln wird daher die Auswahl der komponentenabhängigen Methoden einzeln erläutert und mithilfe der in Kapitel 4 und 5 gefundenen Recherchen begründet.

## 6.1. Beschreibung der verwendeten Tools und Bibliotheken

Bevor auf die Architektur eingegangen wird, sollen die verschiedenen Tools und Bibliotheken beschrieben werden, die im Rahmen der Extraktion des ER-Modells aus unstrukturiertem Text verwendet wurden. Diese Werkzeuge spielen eine entscheidende Rolle bei der Textanalyse und -verarbeitung sowie bei der Implementierung der verschiedenen Ansätze zur ER-Modellextraktion.

**.ipynb** *Jupyter Notebooks* wurden als primäre Entwicklungsumgebung für dieses Projekt gewählt. Diese bieten eine interaktive Umgebung für die Entwicklung, das Testen und die Visualisierung des Codes. *.ipynb*-Dateien ermöglichen eine schrittweise Ausführung des *Python*-Codes, was die Analyse und den Debugging-Prozess erleichtert. Die Flexibilität und die Möglichkeit, Text und Code miteinander zu kombinieren, erleichtert die Dokumentation und Präsentation der Ergebnisse.

**.json** Zur Zwischenspeicherung der Textdaten während der Verarbeitung wurde das JSON-Dateiformat verwendet. *.json*-Dateien haben ein strukturiertes Datenformat, das einfach zu lesen und schreiben ist. Sie erlauben eine effiziente Handhabung der Textdaten, da sie eine hierarchische Darstellung der Daten ermöglichen. Somit können die Textdaten in einer klaren und gut strukturierten Weise gespeichert werden. Darüber hinaus werden hierdurch die Anforderungen des ER-Modeling-Tools von Ruttman [1] erfüllt, da dieses als Eingabe eine JSON-Datei verlangt.

**SpaCy** Die Open-Source-SpaCy-Bibliothek ist eine leistungsstarke NLP-Bibliothek, die für die Textanalyse und -verarbeitung spezialisiert ist. Für die Textanalyse im ganzen regelbasierten Teil des Projektes wird das kleine deutsche *SpaCy*-Modell (*de\_core\_news\_sm*) eingesetzt. *SpaCy* bietet Funktionen wie Tokenisierung, POS, NER und Dependency Parsing, die für die Identifizierung von Entitäten, Beziehungen und Attribute im Text von entscheidender Bedeutung sind. Zusätzlich zum vorgefertigten *SpaCy*-Modell wird für den modellbasierten Teil ein benutzerdefiniertes NER-Modell erstellt und mit der *SpaCy*-Bibliothek trainiert. Dieses Modell ermöglicht eine auf die spezifischen Anforderungen des Projekts zugeschnittene NER [42].

Für die Textanalyse hätte man auch die *NLTK*-Bibliothek wählen können. Im Vergleich zu *NLTK*, das häufig in der Lehre und Forschung verwendet wird, legt *SpaCy* den Schwerpunkt auf die Bereitstellung von Software für den praktischen Einsatz. Während *NLTK* Zugriff auf eine Vielzahl von Algorithmen bietet, um Aufgaben zu erfüllen, ermöglicht *SpaCy* die effizienteste und genaueste syntaktische Analyse unter allen bisher veröffentlichten NLP-Bibliotheken. Darüber hinaus bietet es Zugang zu umfangreicheren Wortvektoren, die sich leichter anpassen lassen. [48]

**Scikit-learn & Tensorflow Keras** Für die Umsetzung des modellbasierten sowie für die *tf-idf*-Analyse des regelbasierten Extraktionsansatzes wurden die Machine-Learning-Bibliotheken *Scikit-learn* und *TensorFlow Keras* verwendet. *Scikit-learn* bietet eine Vielzahl von Machine-Learning-Algorithmen, die für die Klassifikation und Mustererkennung eingesetzt werden können. *TensorFlow Keras* ist ein leistungsfähiges Framework für die Erstellung und das Training von neuronalen Netzen. Die Kombination dieser Bibliotheken ermöglicht es, für die ER-Extraktion angepasste Modelle zu erstellen und zu trainieren.

Zusätzlich zu den oben genannten Hauptbibliotheken wurden auch andere kleinere Bibliotheken verwendet, die im GitHub-Repository in der README.md aufgelistet sind. Diese unterstützenden Bibliotheken bieten zusätzliche Funktionen und Werkzeuge, die für die Implementierung des Projekts von Bedeutung sind.

## 6.2. Architektur der implementierten Lösung

In Abbildung 6.1 ist eine Gesamtübersicht der einzelnen Prozesse aus der Implementierung dargestellt. Das Gesamtprojekt wird untergliedert in einen regelbasierten und modellbasierten Teil. Das modellbasierte *Jupyter Notebook* kann entweder zum Trainieren neuer bzw. vorhandener Modelle oder zur Extraktion der ER-Komponenten aus einem Text genutzt werden. Allerdings werden diese Ergebnisse nicht in eine für das ER-Modeling Tool angemessene *.json*-Datei eingefügt,

sondern gelten lediglich nur zum Vergleich der regelbasierten Ergebnisse. Somit kann überprüft werden, ob gegebenenfalls das Endergebnis aus dem regelbasierten Prozess noch händisch modifiziert werden muss.

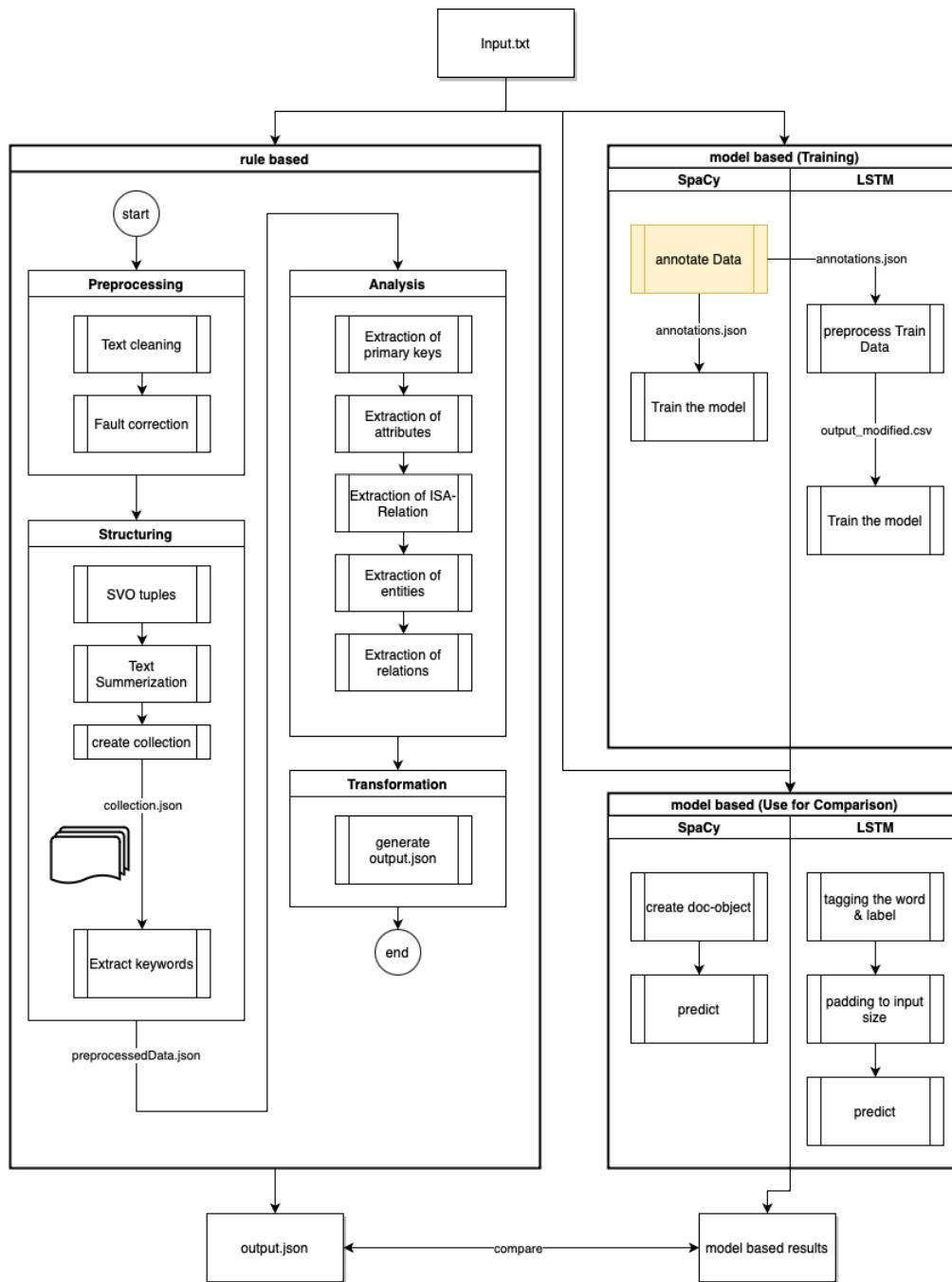


Abbildung 6.1.: Gesamtablaufdiagramm der Implementierung

Der Input für alle Ansätze ist ein Text, welcher in einer `.txt`-Datei gespeichert ist. Die `output.json`-Datei beinhaltet alle gefundenen ER-Komponenten und Beziehungen in einer für das ER-Modeling-Tool lesbaren Struktur. Die Artefakte (im Bild: gewöhnliche Rechtecke oder Pfeilbeschriftungen) repräsentieren die erstellten Dateien bzw. Ergebnisse. Diese Dateien werden für die nachfolgenden Prozesse benötigt.

Der regelbasierte Ansatz ist entsprechend dem NLP-Workflow aus Kapitel 5 aus den Teilschritten Vorverarbeitung, Strukturierung, Analyse und Transformation aufgebaut. In der Vorverarbeitung werden die Textdaten bereinigt und hinsichtlich Rechtschreibfehler korrigiert. Die Strukturierungsphase enthält Prozesse, welche den Text in aussagekräftigere Teile zerlegen und zwischenspeichern. Im Hauptblock der Analyse werden die ER-Komponenten nacheinander extrahiert.

Im modellbasierten Teil wird je nach ausgewähltem Modelltyp (*SpaCy*-Transformer oder LSTM) ein anderer Prozessweg durchgeführt. Aufgrund der individuellen Eingabevoraussetzungen der Modelle müssen sowohl für den Trainingsfall als auch für den Anwendungsfall, die Textdaten in die zugelassene Form gebracht werden. Der gelb hervorgehobene Prozessblock deutet auf einen Annotierungsprozess hin, welcher mithilfe eines externen Tools durchgeführt wird. Für das LSTM Modell muss die Ausgabe des Annotierungstools nochmals in eine `.csv`-Datei vorverarbeitet werden, sodass es für das Training genutzt werden kann. Nach den Trainingsprozessen kann das Modell auf neue Textdaten angewendet werden. Für das *SpaCy*-Modell genügt es, den Text in ein *doc*-Objekt umzuwandeln, sodass die Analyse gestartet werden kann. Der Text für das LSTM-Modell benötigt dagegen (analog zum Trainingsprozess), eine zusätzliche Bearbeitung, welche die Wörter und Label in Zahlen decodiert und den Eingabewortvektor auf eine festgelegte Größe skaliert (*sog. Padding*).

Eine genauere Beschreibung der einzelnen Prozesse für den regelbasierten und modellbasierten Ansätze sind in Kapitel 6.3 enthalten. Weiterhin sei angemerkt, dass die beiden Methoden in zwei, voneinander getrennten *Jupyter Notebooks* codiert sind. Bis auf die *Input*-Datei teilen sich beide Teilsysteme keine Dateien untereinander. Die Versionen der gespeicherten Modelle (*SpaCy* und LSTM) werden separat verwaltet und gespeichert.

### 6.3. Umsetzung der einzelnen Schritte in der Architektur

Nachdem in Kapitel 6.2 auf die Gesamtarchitektur der Implementierung eingegangen wurde, sollen nun die einzelnen Teilschritte näher erläutert werden. Hierbei werden teilweise die Themen aus der Methodik wieder aufgegriffen und für die eigene Anwendung angepasst.

Beginnend mit der regelbasierten Variante kann festgehalten werden, dass es wich-

tig ist, nicht zu viele spezielle Regeln zu definieren. Ansonsten wird die Implementierung zu sehr anwendungsspezifisch und es entstehen teilweise Fehler für andere Textstilarten. Der erste Prozess in der Datenvorverarbeitung ist die Textbereinigung. Dieser Prozess ist in drei Schritten aufgeteilt. Jedes Ergebnis der einzelnen Vorverarbeitungs- und Strukturierungsschritte wird in die *preprocessedData.json* gespeichert. Zu Beginn wird der unstrukturierte Text aus der *.txt*-Datei gefiltert. Da diese Textdatei in folgender Form vorliegen muss, wird nur der Text nach dem Schlüsselbegriff „Text:“ extrahiert. Der Schlüsselbegriff „Sprache:“ definiert die Verarbeitungssprache für den Text:

---

Input.txt

---

Text: Ein Geschäft hat viele Filialen.  
Jede Filiale darf von höchstens einem Filialleiter  
geführt werden. Ein Filialleiter darf höchstens zwei Filialen leiten.  
Ohne eine Filiale wird kein Filialleiter bestimmt.  
Die Filiale bietet viele Produkte an.  
Das Produkt wird von vielen Filialen angeboten.  
Die Filiale beschäftigt viele Mitarbeiter.  
Ein Mitarbeiter darf höchstens einen Verkauf bearbeiten.  
Im Verkauf können viele Produkte involviert sein.  
Jedes Produkt enthält Preis, Produktnname und eindeutige Produkt-ID.  
Es gibt zwei spezielle Produkttypen: Neuwaren, bei denen  
die Verpackungsnummer registriert ist.  
Bei Gebrauchtwaren wird die Anzahl der Vorbereiter mitgeführt.  
Eine Filiale wird durch die Filial-ID eindeutig identifiziert.  
Die Filiale hat einen Filialnamen, eine Filialadresse und eine Telefonnummer.  
Verkauf enthält Datum, Uhrzeit und Betrag.  
Jeder Mitarbeiter hat einen Mitarbeiternamen  
bestehend aus Vorname und Nachname, eine oder  
mehrere Mitarbeiteradressen und eine Telefonnummer.  
Die Mitarbeiter und der Filialleiter werden durch eine ID identifiziert.  
Sprache: de

---

In diesem ersten Schritt werden auch bestehende *Whitespaces* oder Leerzeilen mitberücksichtigt. Im zweiten Schritt der Textbereinigung werden mithilfe des *SpaCy*-Modells (*de\_core\_news\_sm*), die einzelnen Sätze aus dem Text gefunden und gespeichert. Im letzten Schritt der Textbereinigung werden unnötige Sätze entfernt, die keine notwendigen Informationen für das ER-Modell-Design liefern. Hierbei werden Sätze mithilfe von *RegEx*-Übereinstimmungen bestimmter Wörter, wie z.B. „Datenbank“ oder „Modellierung“, aussortiert.

Der *Python*-Code für den zweiten Schritt ist im Folgenden dargestellt. Das Wesentliche in diesem Quellcodeausschnitt ist die für *SpaCy* typische Erstellung eines *doc*-Objektes, mithilfe dessen man mächtige Methoden aufrufen kann. Die Implementierung basiert auf dieser grundlegenden Vorgehensweise.

---

```

1 with open(output, 'r', encoding='UTF-8') as f:
2     data = json.load(f)
3
4 nlp = spacy.load('de_core_news_sm')
5
6 text = data.get('Text', [])
7 doc = nlp(text)
8 sentences = [sent.text.strip() for sent in doc.sents]
9 data["sentences"] = sentences
10
11 with open(output, "w") as file:
12     json.dump(data, file, indent=2, ensure_ascii=False)

```

---

Quelltext 6.1: Aufteilung des Textes in einzelne Sätze

Der nächste Prozess in der Vorverarbeitung ist die Fehlerkorrektur von Wörtern. Hierbei kommt die in Kapitel 5.1 erläuterte *Levenshtein-Ähnlichkeit* zum Einsatz. Durch die Python-Bibliothek *pyspellchecker* wird für jedes Wort in einem Satz überprüft, ob ein Fehler vorliegt. Ist das der Fall, so wird das Wort mit einem anderen Wort aus einer vorgeschlagenen Liste, welches die gleiche *Levenshtein-Distanz* aufweist, ausgetauscht. Im folgenden Beispiel sind einige Ersetzungen visualisiert.

---

de\_rule\_based.ipynb-Output

---

```

id ---ersetzt durch---> in ---Liste---> {'in', 'di', 'd', 'kid', 'nd', 'ad',
'im', 'did', 'iß', 'it', 'if', 'ie', 'cd', 'is', 'ed', 'i', 'il'}
datum ---ersetzt durch---> darum ---Liste---> {'darum'}
betrag ---ersetzt durch---> betrat ---Liste---> {'betrat', 'betrat',
'betrug', 'betrog'}

```

---

Nach mehrmaligem Testen dieser Bibliothek ist anzumerken, dass auch korrekte Wörter in ein anderes Wort, welches nicht zum Kontext passt, ersetzt wird. Daher wird für den weiteren Workflow, die Ergebnisse aus dieser Fehlerkorrektur nur in die Datei *preprocessedData.json* gespeichert, jedoch nicht für die weitere Analyse verwendet.

Im Aufgabenblock der Strukturierung liegt der Hauptfokus auf die Erfassung der Subjekt-Verb-Objekt-Satzstruktur und einiger wichtigen *Term-Frequency*-Analysen mithilfe man den Text in kleinere informationsreiche Teile zerlegen kann. Während für die Textzusammenfassung ausschließlich die normalisierte Worthäufigkeit (TF) angewendet wird, verhilft die invertierte Dokumentenhäufigkeit (TF-IDF), die wichtigsten Schlüsselwörter aus dem Text zu erfassen. Im Gegensatz zu der SVO-Generierung dienen diese zwei genannten Strukturierungsschritte nur zur

optionalen Anwendung. Diese Ergebnisse werden nicht aktiv in den NLP-Workflow eingehen, können jedoch sehr nützlich sein, wenn der Text sehr lang ist oder man nach den wichtigsten Schlüsselwörtern im Text sucht, um diese mit den gefundenen Entitäten und Attribute zu vergleichen. Des Weiteren ist zu beachten, dass für die Berechnung der TF-IDF für die Schlüsselwortextraktion, ein Dokumentenkorpus (*collection.json*) erstellt werden muss, in denen alle vorhandenen ER-Modell-Beispieltexte gespeichert sind. Die Berechnung der TF-IDF übernimmt die von der Bibliothek *Scikit-learn* bereitgestellte *TfidfVectorizer()*-Funktion. In den Schlüsselwörtern sind keine Zahlen inbegriffen.

Die Textzusammenfassung wird gemäß den folgenden Stichpunkten chronologisch umgesetzt:

1. Zählen der Anzahl pro Wort im Text (Stoppwörter ausgeschlossen)
2. Berechne das normalisierte Gewicht pro verwendetem Wort durch Division der jeweiligen Wortanzahl mit der maximal vorkommenden Wortfrequenz
3. Berechne das Satzgewicht durch Addieren des Gewichts pro Wort
4. Suche nach Sätzen mit der höchsten Gewichtung

Hierbei werden die einzelnen Sätze, durch die Addition der normalisierten TF für jedes Wort im Satz, bewertet. Je nach ursprünglicher Textlänge wird eine bestimmte Anzahl an Sätzen in abfallender Reihenfolge der Bewertungszahl ausgewählt. Hierzu wird zu Beginn ein entsprechender Faktor bestimmt, mit dem die Anzahl der Sätze berechnet wird. Gibt es weniger als drei Sätze im Text, so wird keine Textzusammenfassung durchgeführt. Die entsprechenden Ergebnisse des Jupyter Notebooks sehen wie folgt aus:

---

de\_rule\_based-Output

---

```
{'Geschäft': 1, 'Filialen': 3, 'Filiale': 6, 'höchstens': 3,
 'Filialleiter': 4, ...}
{'Geschäft': 0.1666666666666666, 'Filialen': 0.5, 'Filiale': 1.0,
 'höchstens': 0.5, 'Filialleiter': 0.6666666666666666, ...}
{Jede Filiale darf von höchstens einem Filialleiter geführt werden.:
 0.6666666666666666, ...}
Sätze: 5 = 17 mal 0.3

[Jede Filiale darf von höchstens einem Filialleiter geführt werden., Ein
 Filialleiter darf höchstens zwei Filialen leiten., Ein Mitarbeiter darf
 höchstens einen Verkauf bearbeiten., Jedes Produkt enthält Preis,
 Produktname und eindeutige Produkt-ID., Eine Filiale wird durch die
 Filial-ID eindeutig identifiziert.]
```

---

Für die Generierung von SVO-Tupels muss der Satz mithilfe von dem in *SpaCy* bereitgestellten *Dependency Parsing* und POS-Tagging analysiert werden. Durch bestimmte Befehle kann die grammatischen Struktur von Sätzen analysiert werden, sodass die in Abbildung 6.2 dargestellte Visualisierung angezeigt wird.

Ähnlich wie in der Baumstruktur werden die Nachfolger bzw. Vorgänger eines Wortes mit „children“ oder „parent“ adressiert.

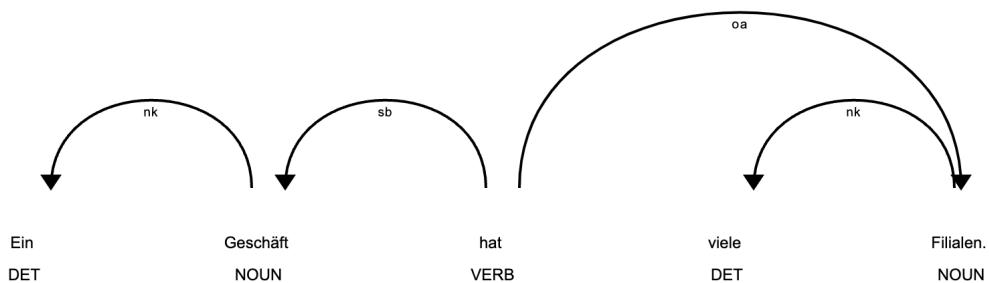


Abbildung 6.2.: Visualisierung der grammatischen Satzstruktur

Anhand dieser Struktur kann für die Extraktion von Subjekt, Verb und Objekt eine allgemeingültige Logik entwickelt werden, die im Struktogramm in Abbildung 6.3 dargestellt ist.

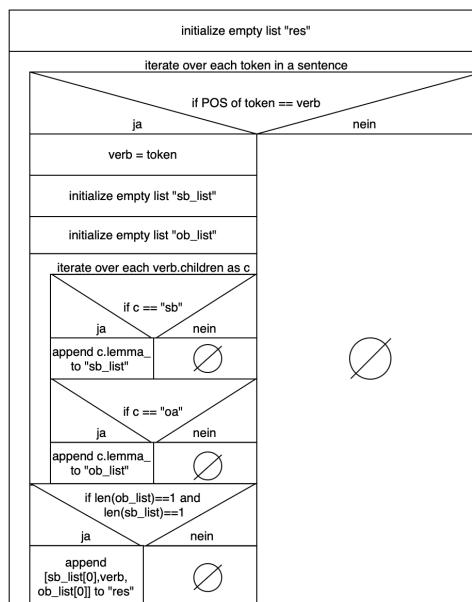


Abbildung 6.3.: Struktogramm für SVO-Extraktion aus einem Satz

Es werden jeweils die Sätze aus dem Text einzeln in die Funktion eingegeben. In jedem Satz darf es mehrere Verben geben, jedoch muss jedes Verb genau zu einem Subjekt und Objekt gehören. Für die Ausgabe von erfassten Beziehungen in Texten kann die sogenannte Lemmatisierung eingesetzt werden. Hierbei wird das Verb von der flektierten Form wieder in die Grundform gebracht. Beispielsweise lautet das Wort „angeboten“ nach der Lemmatisierung „anbieten“.

Nach den Schritten des Vorverarbeitungs- und Strukturierungsprozesses enthält die *preprocessedData.json*, die nachfolgenden Informationen.

---

[ preprocessed.json ]

---

```
{  
    "Text": "Ein Geschäft hat viele Filialen. Jede Filiale darf von  
höchstens einem Filialleiter geführt werden.  
...","  
    "Sprache": "de",  
    "sentences": [  
        "Ein Geschäft hat viele Filialen.",  
        "Jede Filiale darf von höchstens einem Filialleiter geführt werden.",  
        "Ein Filialleiter darf höchstens zwei Filialen leiten.",  
        ...  
    ],  
    "sentences_cleaned": [  
        "Ein Geschäft hat viele Filialen.",  
        "Jede Filiale darf von höchstens einem Filialleiter geführt werden.",  
        "Ein Filialleiter darf höchstens zwei Filialen leiten.",  
        ...  
    ],  
    "sentences_corrected": [  
        "Ein Geschäft hat viele Filialen.",  
        "Jede Filiale darf von höchstens einem Filialleiter geführt werden.",  
        "Ein Filialleiter darf höchstens zwei Filialen leiten.",  
        ...  
    ],  
    "text_summerized": [  
        "Jede Filiale darf von höchstens einem Filialleiter geführt werden.",  
        "Ein Filialleiter darf höchstens zwei Filialen leiten.",  
        ...  
    ],  
    "text_spo": [  
        "Geschäft hat Filiale",  
        "Filiale bietet Produkt",  
        "Filiale beschäftigt Mitarbeiter",  
        "Produkt enthält Preis",  
        "Filiale hat Filialnamen",  
        "Verkauf enthält Datum"  
    ]  
}
```

---

Der Hauptteil des Analyseblockes konzentriert sich auf die regelbasierte Erfassung der einzelnen ER-Komponenten aus dem Text. Auf der Basis von den in Kapitel 5.4 genannten Gesetzmäßigkeiten, werden eigene Regeln entworfen, welche sich für eine präzise und zugleich weitestgehend allgemeingültige ER-Modellgenerierung eignen.

**Primärschlüssel** Die Primärschlüssel können durch einen *RegEx*-Vergleich gefunden werden. Betrachtet man den folgenden Satz (siehe Abbildung 6.4), so weisen die Wörter „-id“ oder „-nummer“ auf einen Schlüsselkandidaten hin.

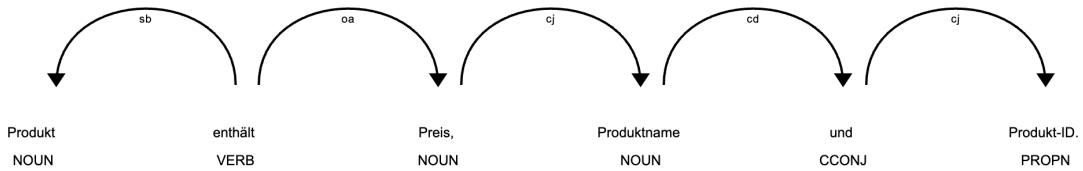


Abbildung 6.4.: Beispielsatz für die Extraktion von Primärschlüssel

Zuvor werden die Wörter in die Kleinschreibweise überführt, sodass ein gewisser Spielraum im Vergleich besteht. Jedoch besteht die Limitierung auf den Bindestrich, welcher in einem Primärschlüsselwort enthalten sein muss. Die folgenden Regeln sind als *Python*-Code implementiert:

- Alle Substantive ermitteln und anschließend Primärschlüssel über *RegEx* filtern.
- Falls der Satz nur das Wort „eindeutig“ enthält, dann sollte das nächstgelegene Substantiv, der Primärschlüssel mit „Substantiv-ID“ sein.
- Wenn nur die Wörter „ID“ oder „id“ vorkommen, dann sollte das nächstgelegene Substantiv, der Primärschlüssel mit „Substantiv-ID“ sein.

**Attribut** Attribute werden identifiziert, sobald ein Satz eine Auflistung von mehr als zwei Nomen beinhaltet (vgl. Abbildung 6.4). Das erste Nomen, welcher im Satz vorkommt, ist die Entität, zu welchem die restlichen Nomen bzw. Attribute dazugehören. Durch diese einfache Regel wird ein Großteil der Attribute gefunden.

**ISA-Vererbung** Um die Spezialisierung bzw. Generalisierung von Entitäten zu detektieren, werden die folgenden Regeln befolgt:

- Wenn ein Satz folgende Verben: [„einbeziehen“, „bestehen“, „umfassen“, „teilen“, „beinhalten“] enthält, dann beschreibt der Satz eine Verallgemeinerung.
- Falls der Satz „Typ“ als Wort (Substantiv) enthält, so ist das erste Substantiv bzw. Entität die Generalisierung der nachfolgenden Substantive (Entitäten).

Es ist anzumerken, dass die Erkennung von ISA-Beziehungen mithilfe regelbasierter Ansätzen eingeschränkt ist. Beispielsweise kann die Reihenfolge der Zugehörigkeit für die Entitäten unterschiedlich sein oder die ISA-Beschreibung kann sich über mehrere Sätze ausdehnen.

**Entität** In der *preprocessed.json* unter dem *Item*, in dem die SVO-Tupels gespeichert sind, kann in jedem Satz das Subjekt und Objekt, entweder eine Entität oder ein Attribut sein. Daher werden diese Tupelwörter nochmals mit den bisher gefundenen Attributen und Primärschlüssel verglichen. Ist das gleiche Tupelwort ebenfalls in der Liste für Attribute oder Primärschlüssel enthalten, so wird dieses verworfen. Als Endergebnis erhält man somit eine Liste, welche nur noch die finalen Entitäten beinhaltet.

**Beziehung** Für die Beziehungen können wieder die SVO-Tupels herangezogen werden. Wenn sowohl das Subjekt als auch das Objekt in der Liste mit den endgültigen Entitäten enthalten ist, so ist das Verb eine Beziehung zwischen zwei Entitäten. Der folgende Quellcode verdeutlicht diese Umsetzung.

---

```

1 relations = []
2 for sentence in spo:
3     doc = nlp(sentence)
4     if doc[0].text.lower() in final_ents and doc[-1].text.lower()
5         in final_ents:
6             relations.append([doc[0].text.lower(), doc[1].text,
7                               doc[-1].text.lower()])
8
9 print(relations)

```

---

Quelltext 6.2: Erfassung von Beziehungen

Es gibt Sätze, die nicht in einem gewöhnlichen SVO-Stil formuliert sind, die jedoch weitere Beziehungen zwischen Entitäten definieren. In Abbildung 6.5 ist ein solcher Beispielsatz dargestellt.

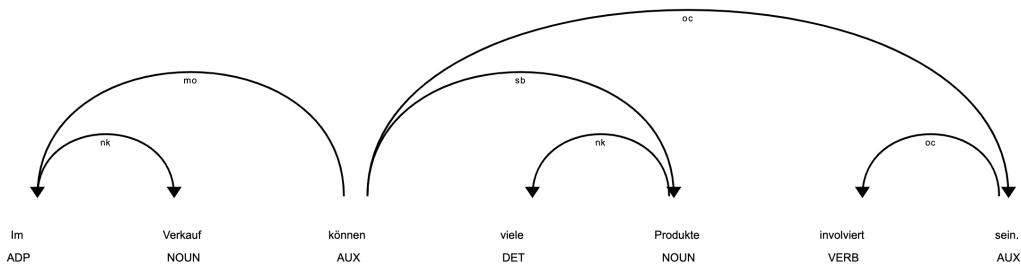


Abbildung 6.5.: Beispielsatz für die Extraktion von Beziehungen

Um solche Sätze ebenfalls zu berücksichtigen, wird überprüft, ob in einem Satz neben einem Verb, zwei Substantive vorkommen, welche nicht in der Attributs-, ISA- und Primärschlüsselliste enthalten sind. Ist das der Fall, so kann aus diesem Satz wieder ein Beziehungstupel extrahiert werden, falls es nicht bereits aus dem SVO-Tupel entnommen wurde.

**Kardinalität** Für jede Beziehung zwischen zwei Entitäten sind jeweils zwei min/max-Kardinalitäten zu bestimmen. Aus den Sätzen in Abbildung 6.2 und Abbildung 6.5 geht hervor, dass die Kardinalitäten jeweils aus den Determinatoren der Substantive entnommen werden können. Nachdem die SVO-Tupels mithilfe einer Indizierung wieder zu den vollständigen Sätzen zugeordnet sind, können die entsprechenden Determinatoren von jeder Entität bestimmt werden. Diese werden dann anhand eines Vergleiches in den entsprechenden min/max-Wert übersetzt (siehe Anhang A). Für die korrekte min/max-Notation müssen die Kardinalitäten der Entitäten in einem SVO-Tupel vertauscht werden.

Für die Interpretation der Adverbien „höchstens“ und „mindestens“ ist auch das nachfolgende Wort zu beachten, da dieses entweder die obere (max) oder untere Grenze (min) festlegt (siehe Abbildung 6.6).

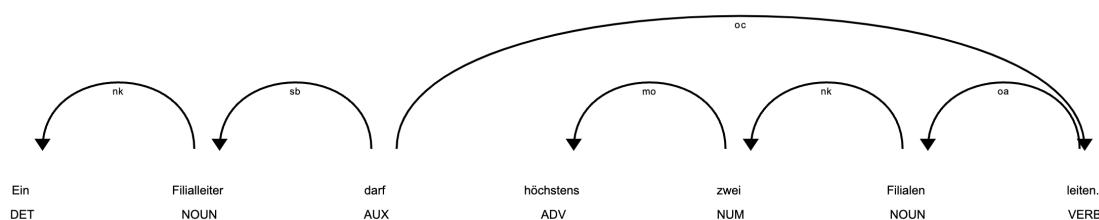


Abbildung 6.6.: Beispielsatz mit dem Adverb „höchstens“

Aufgrund der unbegrenzten Möglichkeit zur Übersetzung dieses Adverbs, eignet sich

hierfür keine regelbasierte Anwendung. Hierdurch werden die Regeln zu stark auf ein Anwendungsfall spezialisiert. Aus Gründen der Einfachheit wird nur der Fall für „höchstens ein“ und „mindestens ein“, wie es in der Tabelle 5.2 vermerkt ist, in der Implementierung mitberücksichtigt. Darüber hinaus dürfen in einem Satz maximal eine Beziehung vorkommen, sodass eine korrekte Zuordnung der Kardinalitäten stattfinden kann.

Bevor der letzte Schritt der Transformation in die *output.json* erläutert wird, ist im Folgenden eine Auflistung der finalen Listen für die ER-Komponenten aufgezeigt.

---

de\_rule\_based-Output

---

```

final_ents:
['verkauf', 'mitarbeiter', 'filiale', 'geschäft', 'produkt']
-----
final_pk:
['filialleiter-id', 'filial-id', 'mitarbeiter-id', 'produkt-id']
-----
final_attr_and_id_list_ents:
[['produkt', ['produktnname', 'preis', 'produkt-id']], ['gebrauchtwaren',
['vorbesitzer', 'anzahl']], ['filiale', ['filialnamen', 'filial-id',
'telefonnummer', 'filialadresse']], ['verkauf', ['betrag', 'datum',
'uhrzeit']], ['mitarbeiter', ['nachname', 'mitarbeiteradressen',
'mitarbeiter-id', 'mitarbeiternamen', 'filialleiter', 'vorname',
'telefonnummer']]]
-----
final_isa_list:
[]
-----
final_card_changed:
[('geschäft', 'hat', '1', 'n'), ('filiale', 'hat', '1', '1'),
('filiale', 'bietet', '1', 'n'), ('produkt', 'bietet', 'n', 'm'),
('filiale', 'beschäftigt', '1', 'n'), ('mitarbeiter', 'beschäftigt', 'n', 'm'),
('mitarbeiter', 'bearbeiten', '0', '1'), ('verkauf', 'bearbeiten', '1',
'1'), ('verkauf', 'involviert', '1', 'n'), ('produkt', 'involviert', 'n',
'm')]

Entität 'geschäft': Beziehung = 'hat', min = '1', max = 'n'
Entität 'filiale': Beziehung = 'hat', min = '1', max = '1'
Entität 'filiale': Beziehung = 'bietet', min = '1', max = 'n'
Entität 'produkt': Beziehung = 'bietet', min = 'n', max = 'm'
Entität 'filiale': Beziehung = 'beschäftigt', min = '1', max = 'n'
Entität 'mitarbeiter': Beziehung = 'beschäftigt', min = 'n', max = 'm'
Entität 'mitarbeiter': Beziehung = 'bearbeiten', min = '0', max = '1'
Entität 'verkauf': Beziehung = 'bearbeiten', min = '1', max = '1'
Entität 'verkauf': Beziehung = 'involviert', min = '1', max = 'n'
Entität 'produkt': Beziehung = 'involviert', min = 'n', max = 'm'

```

---

Mithilfe dieser Listen werden im letzten Transformationsprozess die ER-Komponenten

in die entsprechende JSON-Struktur übertragen. Jede Komponente besitzt ihre eigene eindeutige ID, welche durch eine zufällige 13-stellige Zahl generiert wird. Für Verbindungen von Attributen und Beziehungen müssen auch die IDs der angehörigen Entitäten in der Struktur vermerkt werden. Im Anhang B befindet sich ein Auszug aus der *output.json*-Datei.

Zusammengefasst kann festgehalten werden, dass die Herausforderung der regelbasierten Methode, in der Zuordnung der gefundenen Substantive in die einzelnen Komponenten liegt. Des Weiteren war die Verbindung von Attributen und Beziehungen zu bestimmten Entitäten nicht ganz trivial, sodass dies auch den Transformationsschritt in die geforderte JSON-Struktur erschwert hat.

Der modellbasierte Ansatz dient primär als Vergleichstool für die gefundenen ER-Komponenten aus dem regelbasierten Algorithmus. Die Herausforderung während dem Entwurf der zwei Modelle lag in den Aspekten: Annotieren, Datenvorverarbeitung, Trainieren und Anpassung der Modellarchitektur. Mit Ausnahme des Annotierens der Textdaten wurden für die weiteren Aspekte modellspezifische Einstellungen vorgenommen. Während die Implementierung und Hyperparametereinstellungen des *SpaCy*-Modells deutlich leichter ist, ergaben sich diesbezüglich Schwierigkeiten für das LSTM-Modell.

Das Annotieren der Textdaten wird mithilfe eines kostenlosen Annotierungstools<sup>4</sup> erledigt. Durch eine benutzerfreundliche Umgebung wird dem Nutzer ermöglicht, eigene Labels zu erstellen und diese den jeweiligen Wörtern zuzuordnen (siehe Abbildung 6.7).



Abbildung 6.7.: Benutzeroberfläche des Labelingtools

Als Ausgabe erhält man eine strukturierte JSON-Datei (siehe Anhang C). Nach der Umwandlung dieser Datei in das *.spacy*-Format kann es direkt für das Trainieren des Transformermodells von *SpaCy* verwendet werden.

Anhand des Befehls in Zeile 1 vom Quellcode 6.3 wird eine Konfigurationsdatei erstellt, in dem alle wichtigen Hyperparameter, wie z.B. die *Batch Size*, *Epochs* oder die *Learning Rate* vermerkt sind. Alternativ kann man diese Konfigurationsdatei auch über die offizielle *SpaCy*-Dokumentation [42] anpassen und herunterladen.

<sup>4</sup>Link: <https://tecoholic.github.io/ner-annotator/>

Der Befehl 2 startet das Training des Transformers mit den spezifizierten Trainingsdaten.

---

```

1 ! python -m spacy init config config.cfg --lang de --pipeline ner
    --optimize efficiency
2 ! python -m spacy train config.cfg --output ./ --paths.train
    ./training_data.spacy --paths.dev ./training_data.spacy

```

---

Quelltext 6.3: Terminal-Escape Befehle für das Trainieren des SpaCy-Modells

Aus Gründen der Einfachheit wird vorerst mit dem Trainingsdatensatz trainiert und getestet, d.h. das Modell kennt beim Testen die bereits eingegebenen Daten schon. Dies wird auch als *Reclassification* bezeichnet. Dementsprechend ergibt sich ein sehr gutes Ergebnis, sodass alle ER-Komponenten im Text gefunden werden (siehe Abbildung 6.8).



Abbildung 6.8.: Ergebnisse der *Reclassification* mit dem SpaCy-Modell.

Die trainierten Modelle werden in dem angegebenen Pfad in Zeile 2 gespeichert. Eines enthält die Gewichtsparameter, mit dem das Modell über alle Epochen hinweg die besten Performanceergebnisse erzielt hat, wohingegen das andere Modell die Gewichtsparameter nach der letzten Trainingsepoke besitzt.

Der Trainingsprozess für das LSTM-Modell ist deutlich komplexer. Bevor das LSTM mit den annotierten Textdaten trainiert werden kann, müssen diese Daten vorverarbeitet werden, sodass es als Input für das Modell geeignet ist. Die *annotation.json*-Datei wird in eine *.csv*-Datei umgewandelt. Die Form dieser Datei ist im Anhang D dargestellt.

Die mithilfe der *Tensorflow Keras*-Bibliothek und der *Sequential-Application Programming Interface* (API) entworfene Modellarchitektur ist in Abbildung 6.9 dargestellt.

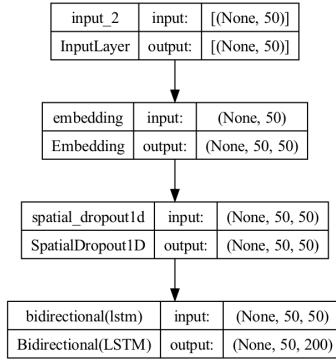


Abbildung 6.9.: Architektur des entworfenen LSTM-Modells

Die Architektur besteht aus fünf Schichten. Die *Output-Schicht* ist in der Abbildung 6.9 nicht dargestellt. Neben der *Input-Schicht* umfasst das Modell eine *Embeddingschicht*, *Dropoutschicht* und ein bidirektionales LSTM. Die *Dropout-Schicht* verhindert ein *Overfitting* des Modells beim Lernprozess, indem einige Gewichte in dieser Schicht eliminiert werden. Während der Anwendung des Modells auf Testdaten werden diese *Dropoutlayer* ignoriert. Die Verwendung der bidirekionalen Variante des LSTM ermöglicht es, den Kontext des Textes besser in den Trainingsprozess vom Modell zu übertragen. Darüber hinaus verdoppelt sich die Anzahl der Gewichtsparameter im Modell, welches dem *Bias*-Fehler (auch als *train loss* bezeichnet) entgegenwirkt. Allerdings ist zu beachten, dass das Modell nicht zu komplex gestaltet werden soll, da ansonsten wieder das *Overfitting* begünstigt wird. Der folgende Output demonstriert die einzelnen Schichten mit der jeweiligen Anzahl an trainierbaren Parameter.

---

de\_model\_based\_keras-Output

---

Layer (type)	Output Shape	Param #
<hr/>		
embedding (Embedding)	(None, 50, 50)	4400
spatial_dropout1d (Spatial Dropout1D)	(None, 50, 50)	0
bidirectional (Bidirectional)	(None, 50, 200)	120800
<hr/>		
Total params: 125200 (489.06 KB)		
Trainable params: 125200 (489.06 KB)		
Non-trainable params: 0 (0.00 Byte)		
<hr/>		

---

Der in der Ausgabe nicht dargestellte *Input-Layer* besitzt eine Größe von 88 Neuronen, welches der Anzahl an Wörtern entspricht. Multipliziert man diesen Wert mit der Eingangsgröße der *Embeddingschicht* (50 Neuronen), so erhält man die Anzahl an trainierbaren Parameter 4400 für diese Schicht.

Bei der Verarbeitung mit NLP werden Wörter und Phrasen oft als eindimensionale Vektoren dargestellt, bei denen jede Dimension einem anderen Wort des Vokabulars entspricht. Man spricht hierbei auch von *one-hot-encoding*. Diese Vektoren sind hochdimensional, was die Arbeit mit ihnen erschwert. Anstatt diese hochdimensionalen Vektoren zu verwenden, wird in der *Embeddingschicht* jedes Wort auf einen niedrigen dimensionalen Vektor abgebildet, bei dem jede Dimension ein bestimmtes Merkmal des Wortes darstellt. [49]

Die *Dropoutschicht* besitzt keine trainierbaren Parameter. Dahingegen hat das bidirektionale LSTM 120800 trainierbare Parameter. Dieser Wert entsteht durch die folgende Betrachtung:

$$p = 2 \times 4 \times (h \times (h + i) + h) = 120800$$

mit:  $p$  = Anzahl an trainierbaren Parameter  
**Faktor 2** = Bidirektionalität  
**Faktor 4** = vier neuronale Netze  
 $h$  = 100 (LSTM Zellen)  
 $i$  =  $50 + 100$  (LSTM Zellen) = 150 (Dimension des Inputs)

Die Zahl der LSTM Zellen bzw. der *Output-Unit* ist in der vorherigen Abbildung 6.9 nicht abgebildet. Diese Zahl wird festgelegt, sobald die LSTM-Schicht in die Modellstruktur hinzugefügt wird (siehe Zeile 5 im Quelltext 6.4).

---

```

1 model = keras.Sequential()
2 model.add(InputLayer((max_len)))
3 model.add(Embedding(input_dim=num_words, output_dim=max_len,
                      input_length=max_len))
4 model.add(SpatialDropout1D(0.1))
5 model.add(Bidirectional(LSTM(units=100, return_sequences=True,
                           recurrent_dropout=0.1)))
6 model.summary()
```

---

#### Quelltext 6.4: Modellaufbau über die Sequential-API

In Zeile 2 des Quelltextes 6.4 ist eine feste Eingangsdimension für die *Inputschicht* festgelegt. Anhand der Verteilung von durchschnittlichen Satzlängen (siehe Abbildung 6.10), wird für die maximale Satzlänge bzw. Eingangsdimension (*max\_len*) der Wert 50 gewählt.

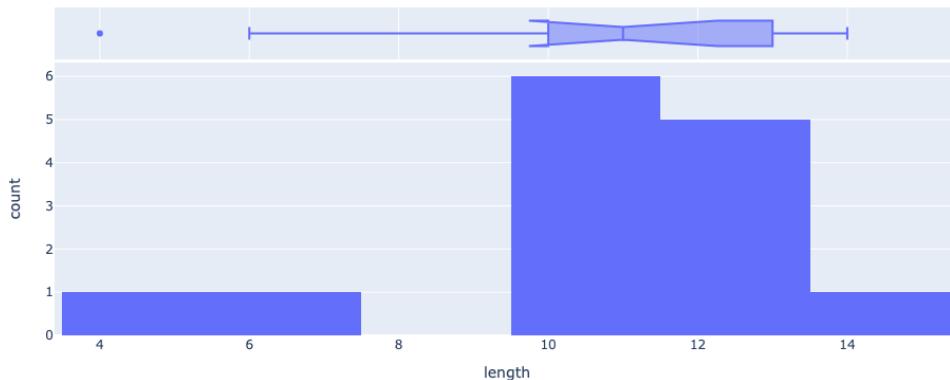


Abbildung 6.10.: Verteilung der Satzlängen im kleinen Datensatz zum Testen. Der Median liegt bei der Länge 11.

Dies bedeutet, dass der Wortvektor eines Satzes auf diese Größe angepasst werden muss. Nach dem *Tagging*-Prozess sind die Wörter als Zahlen decodiert. Aufgrund der unterschiedlichen Größe der Sätze müssen diese durch das *Padding* auf die einheitliche Eingangsgröße von 50 erweitert werden. Diese eben beschriebenen Prozesse sind in der Abbildung 6.1 sowohl im *model based (Training)*-Block als auch im *model based (Use for Comparison)*-Block vermerkt.

Bevor das Modell mithilfe der *.fit*-Methode trainiert werden kann, muss es durch Festlegung von wichtigen Hyperparametern kompiliert werden. Hierzu werden die folgenden Parameter gewählt:

---

```

1 model.compile(optimizer="adam",
2                 loss="sparse_categorical_crossentropy",
3                 metrics=["accuracy"])

```

---

Quelltext 6.5: Modellaufbau über die Sequential-API

Der *Adam-Optimizer* repräsentiert eine spezielle Form des Gradientenverfahrens (engl. *Gradient Descent*), welches für den Trainingsprozess zur Optimierung der Gewichtsparameter verwendet wird. Zur Berechnung der neuen Gewichtswerte soll dieser *Gradient Descent*-Algorithmus die *sparse\_categorical\_crossentropy* als Verlustfunktion (engl. *Loss function*) verwenden, da dieses das *one-hot-encoding* bereits intern erledigt. Darüber hinaus eignet sich diese *Loss function* für Klassifikationsprobleme, weil sich hierbei, im Vergleich zum *Mean Squared Error*, die Gewichtsparameter deutlich schneller anpassen. Der Grund hierfür liegt in der Charakteristik von der *Categorical Cross Entropy*-Funktion. Aufgrund der Logarithmierung der Werte zwischen 0 bis 1 ergeben sich im Fehlerfall deutlich höhere

Verlustwerte, als für die gleichen Werte mit dem *Mean Squared Error* (siehe Abbildung 6.11). Dadurch erreicht man mithilfe der *Categorical Cross Entropy*-Funktion größere *Step Size*-Werte für das Gradientenverfahren, wodurch das Training beschleunigt wird. Im Anhang E sind die Formeln für verschiedene Verlustfunktionen aufgelistet.

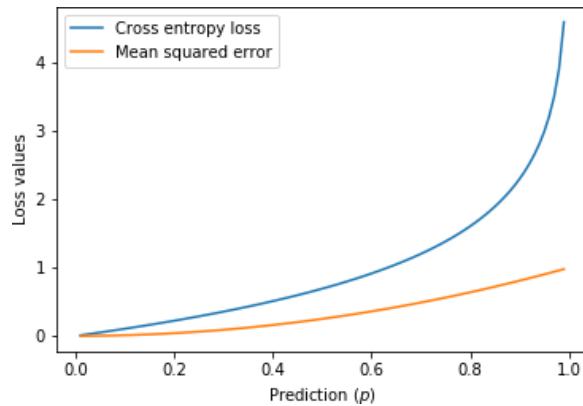


Abbildung 6.11.: Vergleich der Verlustfunktionen von *Mean Squared Error* und *Cross Entropy*. *Ground Truth* ist bei 0. [50]

Als Beurteilungsmaß wurde die Genauigkeit (engl. *accuracy*) ausgewählt. Im Falle einer unbalancierten Verteilung der Labels im Datensatz ist zu beachten, dass anstatt der Genauigkeit, *Precision* oder *Recall* für die Beurteilung aussagekräftiger ist. Der Grund hierfür ist, dass der Genauigkeitswert für unbalancierte Datensätze meist nahezu 1 beträgt und somit die *Error Rate* (1-Genauigkeit) zu 0 wird. Dies wird auch als *null information rate* bezeichnet. Alternativ kann man Strategien, wie z.B. *Data Upsampling* oder *Data Stratification* nutzen, um die Unausgewogenheit der Klassen in den Datensätzen zu umgehen. [51]

# 7. Evaluierung

Durch die Evaluierung des Projektes soll beurteilt werden, inwieweit die Ziele der Arbeit erreicht wurden. Hierzu werden die in der Tabelle 3.1 erarbeiteten Anforderungen als Vergleich zu den Arbeitsergebnissen herangezogen.

**A1** Aus den zwei Möglichkeiten der Schnittstellenauswahl wurde die Nutzung der *Upload*-Funktion durch eine entsprechende JSON-Datei umgesetzt. Der Transformationsprozess wurde für unterschiedliche Datenbankszenarien erfolgreich getestet, sodass eine korrekte Überführung der im Text gefundenen ER-Komponenten in das *ER-Modeling-Tool* möglich ist.

**A2** Durch den regelbasierten Algorithmus werden nur ER-Komponenten extrahiert, die den definierten grammatischen Gesetzmäßigkeiten entsprechen. Im Gegensatz zum modellbasierten Ansatz, wird anstatt einer quantitativen Ergebnismenge auf eine qualitativ korrekte ER-Extraktion Wert gelegt. Die Korrektheit wird zusätzlich durch den modellbasierten Vergleich erhöht.

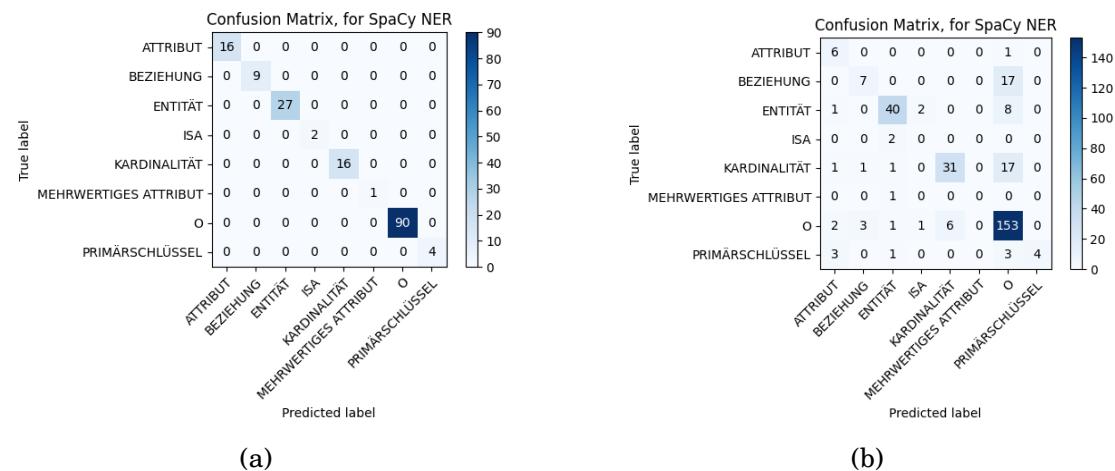


Abbildung 7.1.: Evaluationsvergleich des *SpaCy*-Modells über eine *Confusion Matrix*.  
 (a) *Reclassification*. (b) *Crossvalidation* mit größerem Trainings- und Testdatensatz.

Es eignet sich hierfür eine Evaluation über eine *Confusion Matrix* (siehe Abbildung 7.1). Die *Confusion Matrix* in Abbildung 7.1a zeigt den *Reclassification-Fall*, in dem alle zuvor gelabelten ER-Komponenten beim Testen korrekt klassifiziert werden. Jedoch gibt die *Confusion Matrix* in Abbildung 7.1b mehr Auskunft über die Generalisierungsfähigkeit des *SpaCy*-Modells, da hierbei durch das *Crossvalidation*, auch Textdaten untersucht werden, die das Modell noch nicht kennt und somit nicht Teil des Trainingsprozesses waren. Analog zum *SpaCy*-Modell wird auch das LSTM-Modell, mithilfe der in Abbildung 7.2 dargestellten Lernkurven, bewertet. Nach 10 Trainingsepochen wird eine solide Validationsgenauigkeit von 92.1% und ein Validationsfehler von 0.53 erreicht. Im Anhang F befindet sich ein Auszug aus den Ergebnissen der Evaluationsmetriken während dem Training des *SpaCy*-Modells.

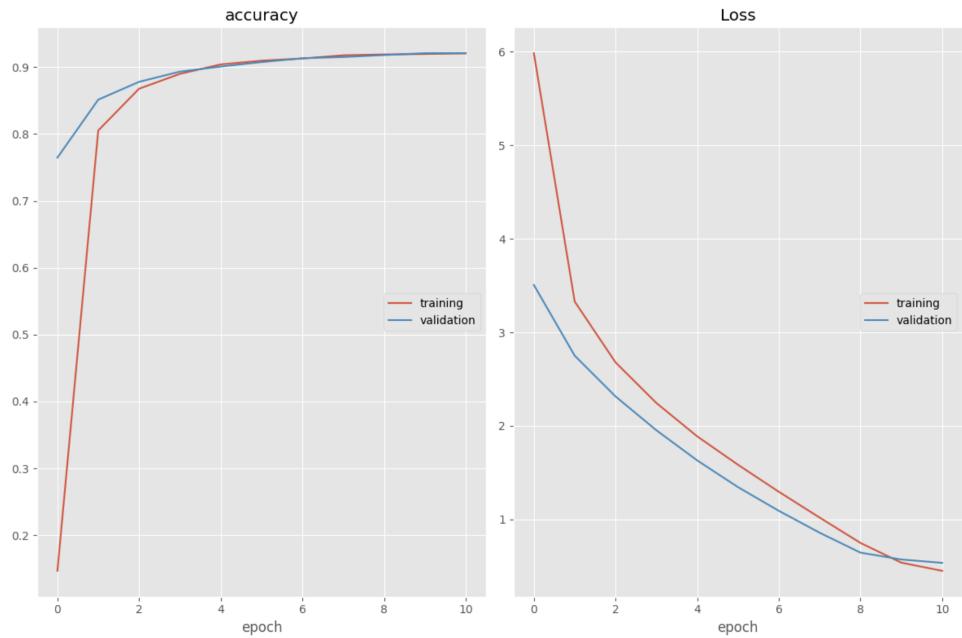


Abbildung 7.2.: Lernkurven vom Training des LSTM-Modells.

Im Vergleich zu den Auswertungen im Implementierungskapitel wurde für die Evaluation in Abbildung 7.1b und 7.2 ein deutlich größerer Datensatz aus 475 distinkten Wörtern verwendet. Für das *SpaCy*-Modell wurden 77 Sätze für das Training und 26 Sätze für das Testen reserviert. Das LSTM-Modell wurde mit 20 Sätzen (20% der Gesamtanzahl an Sätzen) validiert, wobei der Rest für das Training verwendet wurde. Die Testergebnisse entstehen auf Basis einer Evaluation von unbekannten Textdaten, wodurch die Ergebnisse automatisch etwas schlechter ausfallen, als eine Evaluation von bereits bekannten Trainingsdaten.

Anhand der Evaluationsergebnisse der Modelle (Abbildung 7.1b und 7.2) kann gefolgert werden, dass ein kleiner Optimierungsbedarf in der Generalisierungsfähigkeit

(= Anwendung des Modells auf neue bzw. unbekannte Testdaten) der Modelle besteht. Dies erkennt man an den fehlerhaften Klassifikationen (*True label ≠ Predicted label*) in der *Confusion Matrix* in Abbildung 7.1b.

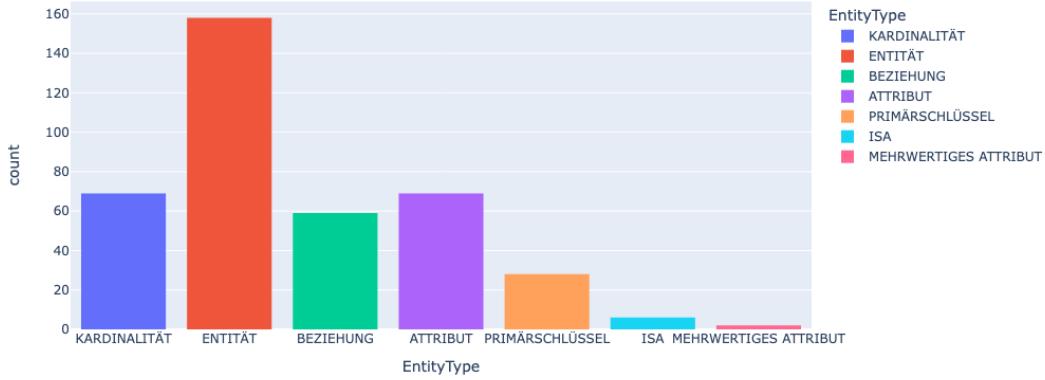


Abbildung 7.3.: Verteilung der Anzahl an Wortkategorien im Datensatz.

Darüber hinaus wird durch die Abbildung 7.3 deutlich, dass der Datensatz enorm unausgewogen ist. Durch die ungleichmäßige Verteilung der Labels im Datensatz trifft das Modell voreingenommene Entscheidungen. Hierzu müssen die Modelle sowohl mit einer noch größeren Menge an Trainingsdaten als auch unter Berücksichtigung von ausgewogenen Trainingsdaten geschult werden.

**A3** Die zeitliche Performance der Textverarbeitung im Projekt ist sehr zufriedenstellend, da die durchschnittlich benötigte Zeit für die manuelle Extraktion deutlich über der Projektlaufzeit ist. Für den in Kapitel 2.2.3 herangezogenen Beispieldaten beträgt die Gesamtlaufzeit vom regelbasierten *Jupyter Notebook* ca. 2.6 Sekunden. Auch für größere Textmengen arbeiten die Prozesse im *Jupyter Notebook* zuverlässig und enorm schnell.

**A4** Eine Validierung der regelbasierten Ergebnisse ist durch den Vergleich der modellbasierten Extraktionen (*SpaCy & LSTM*) möglich. Im Falle von Diskrepanzen zwischen den regelbasierten und modellbasierten Ergebnissen können mögliche Fehler manuell korrigiert werden.

**A5** Die Flexibilität des Gesamtsystems ist teilweise erfüllt, da der regelbasierte Ansatz für einige Prozesse ausschließlich für die deutsche Sprache verwendbar

ist. Allerdings können die Modelle problemlos mit englischen Textdaten trainiert werden. In den *.txt*-Inputdateien kann unter „Sprache:“ die Verarbeitungssprache ausgewählt werden. Momentan ist diese Definition funktionslos. Falls das Projekt in Zukunft für mehrere Sprachen zur Verfügung steht, könnte dieses Textfeld als Schalter fungieren.

**A6** Das Projekt wurde dokumentiert und die wichtigsten Schritte zur Einbettung des Projektes in eine eigene virtuelle Umgebung sind in der README.md im *Github*-Repository vermerkt.

Im Großen und Ganzen konnten fast alle Anforderungen erfolgreich umgesetzt werden. Die Ziele der Anforderungen A1, A3, A4, A6 wurden vollständig erfüllt. Wohingegen die Anforderungen A2 und A5 hinsichtlich der Modellrobustheit und Spracherweiterung noch justiert werden können. Nichtsdestotrotz lag der Fokus eher auf die deutsche Sprache, wodurch das Ziel des Projektes dennoch erreicht wurde.

## 8. Projektplanung und -verlauf

Dieses Kapitel beschreibt die Planung des Projekts sowie den tatsächlichen Verlauf während der Implementierung. Die Projektplanung ist entscheidend, um alle in Kapitel 3 beschriebenen Anforderungen erfolgreich in das System zu integrieren.

Zur Veranschaulichung des zeitlichen Ablaufs wird ein Gantt-Diagramm verwendet, das die geplanten Aufgaben und deren Fortschritt über die Projektzeit hinweg visualisiert (siehe Abbildung 8.1). Es zeigt die zeitliche Abfolge der einzelnen Aktivitäten sowie Verzögerungen oder beschleunigte Entwicklungen.

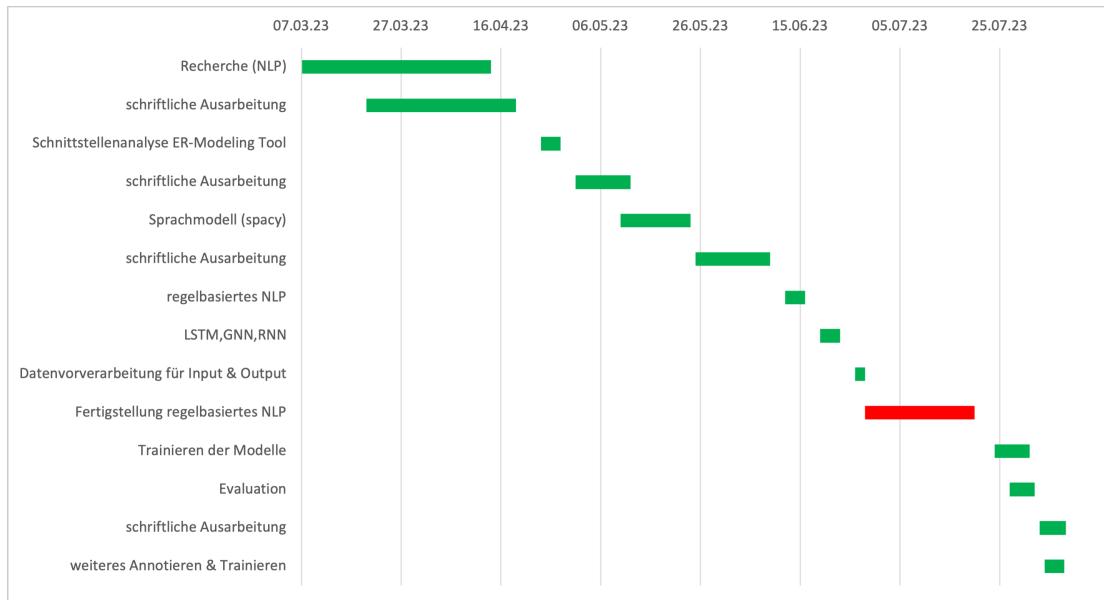


Abbildung 8.1.: vereinfachtes Gantt-Diagramm

Der Projektverlauf zeigt, dass zu Beginn die Literaturrecherche über bestehende Möglichkeiten zur NLP-basierten ER-Modellierung den Grundstein für einen fundierten Ansatz legten. Hierzu zählt auch die Untersuchung der vorhandenen Werkzeuge, die für das Projekt infrage kommen würden. Die sorgfältige Aussortierung der Möglichkeiten gehört auch zur Aktivität „Recherche(NLP)“. Beispielsweise ist die in Kapitel 4.1 beschriebene Vorgehensweise aus der Arbeit von Kashmiri et al. [30] bezüglich der Attributszuordnung mithilfe von *Ontologie* und *Web*

*Mining* zu umständlich und benötigt zusätzliche Ressourcen (API, Datenbanken etc.). Somit wurde diese Variante aus der möglichen Umsetzung aussortiert. Als Basis dieser Recherchen konnten präzisere Anforderungen an den automatisierten ER-Modellierungsansatz definiert werden.

Im Verlauf des Projekts wurden die geplanten Aktivitäten gemäß dem Gantt-Diagramm durchgeführt. Zwischen den Implementierungsphasen wurde regelmäßig die schriftliche Ausarbeitung ergänzt. Die Aufgabe „Schnittstellenanalyse ER-Modeling Tool“ umfasst die Untersuchung der Integrationsmöglichkeiten der unstrukturierten Textdaten oder der Endergebnisse. Da das Nutzen der vorhandenen *Upload*-Funktion deutlich einfacher ist, als die Integration der *Python*-Dateien in die *R*-Umgebung des Tools, hat man sich für die Formatierung der Endergebnisse in einer vom Tool akzeptierten JSON-Datei entschieden. Die Erkenntnisse der Analyse konnten schnell erzielt werden, was sich aus dem Gantt-Diagramm ablesen lässt. Das Einarbeiten und Testen des *SpaCy*-Modells (Aktivität: „Sprachmodell (spacy)“) benötigte ca. 2 Wochen. In dieser Arbeitsphase erlernte man die Funktionsweise des Transformermodells, die Methodenaufrufe für unterschiedliche Objekte und die Anwendung des Modells auf eigene Textdaten. Ab dem 12.06.2023 begann die Phase „regelbasiertes NLP“, in der die angepassten Regeln aus der zu Beginn durchgeführten Recherche, in Python-Code umgesetzt wurden.

Das Fertigstellen der regelbasierten Variante war ursprünglich zwischen dem Zeitraum vom 28.06.2023 bis zum 30.06.2023 angesetzt. Aufgrund der Prüfungsphase hat sich dies bis auf den 20.07.2023 verzögert. Während der Prüfungsvorbereitung wurden die Implementierungs- und Dokumentationstätigkeiten pausiert. Die Entwicklung und Implementierung der regelbasierten NLP-Algorithmen stellten die anspruchsvollste Phase dar, die jedoch nach der Prüfungsphase erfolgreich abgeschlossen wurde.

Vom 19.06.2023 bis zum Beginn der Phase „Fertigstellung regelbasiertes NLP“ widmete man sich, an die Einarbeitung und Implementierung der Alternative zum *SpaCy*-Modell. In diesen zwei Aktivitäten „LSTM,GNN,RNN“ und „Datenvorverarbeitung für Input & Output“ wurde das LSTM aus den verschiedenen Arten von RNN ausgewählt und entworfen. Die entsprechende Datenvorverarbeitung für das LSTM stellte die Grundlage für das Trainieren und Testen des Modells mit den annotierten Textdaten dar.

In den letzten drei Wochen vor der Abgabe der Projektarbeit widmete man, in ähnlichem Zeitverhältnis, den Fokus auf die Evaluation, die schriftliche Ausarbeitung und das Annotieren bzw. Trainieren der Modelle.

# 9. Zusammenfassung und Ausblick

## 9.1. Erreichte Ergebnisse

Trotz dieser anspruchsvollen NLP-Aufgabe wurden alle notwendigen Umsetzungen hinsichtlich der Extraktion von ER-Komponenten aus Texten durchgeführt. Es wurden zwei Teilsysteme entwickelt, welche beide, ER-Komponenten aus Texten in .txt-Dateien extrahieren können. Aufgrund der höheren Zuverlässigkeit der regelbasierten Ergebnisse werden diese für die Darstellung des finalen ER-Modells im ER-Modeling-Tool von Simon Ruttman [1] verwendet. Um die regelbasierten Ergebnisse zu validieren, können die entwickelten *SpaCy*- und LSTM-Modelle genutzt werden. Im Falle von Abweichungen kann das im Tool über einen JSON-Upload hochgeladene ER-Modell bearbeitet werden.

Für die regelbasierte Extraktion von Entitäten, Attribute und Beziehungen werden spezielle NLP-Methoden verwendet. Hierbei wird die Grammatik der Sätze untersucht, sodass zutreffende Schlussfolgerungen hinsichtlich der Zugehörigkeit einer ER-Komponente gezogen werden kann. Beispielsweise werden die Sätze in SVO-Tupels komprimiert, um Entitäten und Beziehungen zu filtern. Die Primärschlüssel und ISA-Beziehungen werden durch bestimmte *RegEx*-Vergleiche identifiziert. Zu den Entitäten gehörende Attribute werden anhand der aufzählenden Satzstrukturweise zugeordnet. Fehlerhaft eingeteilte Attribute werden aus der finalen Entitätsliste entfernt, sobald diese Attribute in einer weiteren finalen Attributliste gefunden wurden. Die Kardinalitäten einer Beziehung werden über einen Tabellenvergleich gefunden. Hierbei werden die Determinatoren oder Adverbien der Entitäten im Satz in die passende Kardinalität übersetzt. Aufgrund der einfachen und vielfältigen Lösung für grundlegende NLP-Aufgaben besteht der regelbasierte Ansatz zum Großteil aus Methoden der *SpaCy*-Bibliothek.

Die Datenvorverarbeitung des regelbasierten Algorithmus filtert unwichtige Sätze aus dem Text und entfernt überflüssige Leerräume in Sätzen. Die Fehlerkorrektur durch die *Levenshtein-Distanz* kann optional genutzt werden, allerdings werden hierbei auch korrekte Wörter verändert. Aus diesem Grund wird es für den NLP-Workflow übersprungen. In der Strukturierungsphase wird neben der SVO-Tupel-Generierung auch eine Testzusammenfassung und eine Schlüsselwortextraktion angeboten. Diese Informationen werden nur zusätzlich zwischengespeichert, sodass diese bei Bedarf genutzt werden können.

Die wesentliche Errungenschaft bzw. Herausforderung in der modellbasierten Extraktion waren die Datenvorverarbeitungen, wodurch das Training für zwei neue Modelle ermöglicht wurde. Hierzu zählt das Annotieren und die Anpassung der annotierten Daten für das Training des LSTM-Modells. Das Ergebnis sind funktionsfähige neuronale Netze, die zufriedenstellende Resultate hinsichtlich einer korrekten Informationsextraktion aus Texten liefern.

In Summe ist durch das Projekt eine solide Basis zur automatisierten Extraktion von ER-Komponenten aus Texten geschaffen worden, welches in das ER-Modeling Tool von Simon Ruttman [1] überführt werden kann.

## 9.2. Ausblick

In den folgenden Abschnitten werden mögliche Weiterentwicklungen sowie potentielle Anwendungsbereiche beleuchtet, die sich aus den gewonnenen Erkenntnissen und der angewandten Methodik ergeben könnten.

Die im Kapitel 5.3.3 erwähnte *Coreference Resolution* eignet sich bisher nicht für die deutsche Sprache. Verschiedene Recherchen und Testergebnisse zeigten, dass die vortrainierten *Coref*-Modelle für die englische Sprache deutlich besser funktionieren. Falls das Projekt für die englische Sprache erweitert wird, könnte man an die *SpaCy*-Pipeline, neben dem *Custom Named Entity Recognition* und dem *sentencizer*, noch ein *Coref*-Modell anhängen.

Im regelbasierten Algorithmus wurden folgende Aspekte nicht mitberücksichtigt:

- Attribute für Beziehungen.
- spezielle Kardinalitäten („zwei“, „drei“ etc.).
- schwache Entitäten, Beziehungen, Attribute etc.
- beschriebene ISA-Vererbung über mehrere Sätze hinweg.
- mehrwertige oder komplexe Attribute.

Der modellbasierte Ansatz ist bereits in der Lage, einige Auflistungspunkte, wie z.B. mehrwertige Attribute, als ER-Modellkomponenten zu detektieren. Jedoch werden diese Ergebnisse nicht in die finale *.json*-Datei integriert. Man könnte in Zukunft eine Erweiterung vornehmen, sodass auch aus den Modellen einige ER-Komponenten in die Ausgabe mit aufgenommen werden. Somit werden die Modelle nicht nur als Vergleich genutzt. Für die restlichen Aspekte würde eine regelbasierte Integration wenig Sinn ergeben, da eine allgemeingültige Formulierung der Regeln sehr schwierig ist. Die Umsetzung müsste stark auf den Anwendungsfall zugeschnitten sein,

was zu vielen Abhängigkeiten führt.

Eine weitere grundlegende Verbesserung ist das Trainieren der Modelle mit noch mehr Textdaten, sodass auch selten vorkommende ER-Komponenten, wie beispielsweise mehrwertige Attribute, besser erlernt werden können. Die vortrainierten Modelle können gemäß dem Prinzip vom *Transfer Learning* auf weitere Domänen hintrainiert werden, um die Modelle für weitere Zwecke, wie z.B. zum Erstellen von *Knowledge Graphen*, nutzen zu können.

Des Weiteren ist eine Integration der bisher externen Projektdateien in das ER-Modeling-Tool von Simon Ruttman [1] denkbar. Somit ergänzen sich beide Projekte zu einem einheitlichen Gesamttool, das einen Text über eine entsprechende Schnittstelle (z.B. Eingabefeld) akzeptiert und daraus direkt ein ER-Modell generiert.

# Literatur

- [1] Simon Ruttman. *Erzeugung einer Software zur Modellierung von relationalen Datenbanken*. 2022.
- [2] Monisha Kanakaraj und Ram Mohana Reddy Guddeti. *Performance analysis of Ensemble methods on Twitter sentiment analysis using NLP techniques*. IEEE, 2015, S. 169–170.
- [3] Winfried Bantel. *Vorlesungsfolien NLP: Einführung*. 2022.
- [4] Adil Rajput. *Natural language processing, sentiment analysis, and clinical analytics*. Elsevier, 2020, S. 79–97.
- [5] R Suganya u. a. *Product Review Analysis by Web Scraping Using NLP*. Springer, 2022, S. 427–436.
- [6] Vo Thi Ngoc Chau und Suphamit Chittayasothorn. *A Bitemporal SQL Database Design Method from the Enhanced Entity-Relationship Model*. IEEE, 2021, S. 85–90.
- [7] Huajing Li u. a. *Scalable community discovery on textual data with relations*. ACM, 2008, S. 1203–1212.
- [8] Thanveer Shaik u. a. *A review of the trends and challenges in adopting natural language processing methods for education feedback analysis*. Bd. 10. IEEE, 2022, S. 56720–56739.
- [9] Workneh Y Ayele. *Adapting CRISP-DM for Idea Mining: A Data Mining Process for Generating Ideas Using a Textual Dataset*. 2021.
- [10] Steffen Huber u. a. *DMME: Data mining methodology for engineering applications - a holistic extension to the CRISP-DM model*. Bd. 79. Elsevier, 2019, S. 403–408.
- [11] M Uma u. a. *Formation of SQL from natural language query using NLP*. 2019, S. 1–5.
- [12] Hemlata Shelar u. a. *Named entity recognition approaches and their comparison for custom ner model*. Bd. 39. 3. Taylor & Francis, 2020, S. 324–337.
- [13] Sidharth Mehra und Mohammed Hasanuzzaman. *Detection of Offensive Language in Social Media Posts*. 2020.
- [14] Vineet Raina u. a. *Natural language processing*. Springer, 2022, S. 63–73.
- [15] Viincenza Carchiolo u. a. *Medical prescription classification: a NLP-based approach*. IEEE, 2019, S. 605–609.

- [16] Savaş Yıldırım u. a. *Classification of "hot news" for financial forecast using NLP techniques*. IEEE, 2018, S. 4719–4722.
- [17] Soufyane Ayanouz, Boudhir Anouar Abdelhakim und Mohammed Benhmed. *A smart chatbot architecture based NLP and machine learning for health care assistance*. 2020, S. 1–6.
- [18] o.V. *Dokumentenretrieval*. Page Version ID: 219215279. Jan. 2022. URL: <https://de.wikipedia.org/w/index.php?title=Dokumentenretrieval&oldid=219215279> (besucht am 08. 06. 2023).
- [19] Danny Jácome u. a. *Contextual Analysis of Comments in B2C Facebook Fan Pages Based on the Levenshtein Algorithm*. Springer, 2019, S. 528–538.
- [20] Fei Xia und Meliha Yetisgen-Yildiz. *Clinical corpus annotation: challenges and strategies*. 2012, S. 21–27.
- [21] Ari Schlesinger, Kenton P O'Hara und Alex S Taylor. *Let's talk about race: Identity, chatbots, and AI*. 2018, S. 1–14.
- [22] o.V. *Homonym*. Page Version ID: 234267412. Juni 2023. URL: <https://de.wikipedia.org/w/index.php?title=Homonym&oldid=234267412> (besucht am 08. 06. 2023).
- [23] Gregor Grambow. *Vorlesungsfolien: Datenbanksysteme-Das Entity-Relationship-Modell*. 2022.
- [24] Heidi Gregersen und Christian S Jensen. *Temporal entity-relationship models-a survey*. Bd. 11. 3. IEEE, 1999, S. 464–497.
- [25] o.V. *Entity-Relationship-Modell*. Page Version ID: 233659565. Mai 2023. URL: <https://de.wikipedia.org/w/index.php?title=Entity-Relationship-Modell&oldid=233659565> (besucht am 08. 06. 2023).
- [26] Ramez Elmasri, Shamkant B Navathe und Angelika Shafir. *Grundlagen von Datenbanksystemen*. Bd. 3. Pearson Studium München, 2002.
- [27] Lilac AE Al-Safadi. *Natural language processing for conceptual modeling*. Bd. 3. Citeseer, 2009, S. 47–59.
- [28] Sutirtha Ghosh u. a. *Automated generation of er diagram from a given text in natural language*. IEEE, 2018, S. 91–96.
- [29] Volha Bryl u. a. *Supporting natural language processing with background knowledge: Coreference resolution case*. Springer, 2010, S. 80–95.
- [30] PGTH Kashmira und Sagara Sumathipala. *Generating entity relationship diagram from requirement specification based on nlp*. IEEE, 2018, S. 1–4.
- [31] Muhammad Shahbaz u. a. *Automatic generation of extended er diagram using natural language processing*. Bd. 7. 8. 2011, S. 1–10.
- [32] Cyril Goutte und Eric Gaussier. *A probabilistic interpretation of precision, recall and F-score, with implication for evaluation*. Springer, 2005, S. 345–359.

- [33] Adam Shafi. *How to Learn the Definitions of Precision and Recall (For Good)*. Apr. 2022. URL: <https://towardsdatascience.com/precision-and-recall-88a3776c8007> (besucht am 08.06.2023).
- [34] Mohammad Kasra Habib. *On the automated entity-relationship and schema design by natural language processing*. Bd. 8. 11. 2019, S. 42–48.
- [35] Elisa Margareth Sibarani u. a. *A Study of Parsing Process on Natural Language Processing in Bahasa Indonesia*. IEEE, 2013, S. 309–316.
- [36] Nazlia Omar, JRP Hanna und Paul McKeivitt. *Heuristic-based entity-relationship modelling through natural language processing*. Artificial Intelligence Association of Ireland, 2004, S. 302–313.
- [37] M Omar und A Abdulla. *The entities extraction for entity relationship models from natural language text via machine learning algorithms*. 2020.
- [38] Eman S Btoush und Mustafa M Hammad. *Generating ER diagrams from requirement specifications based on natural language processing*. Bd. 8. 2. 2015, S. 61–70.
- [39] Robin Richard Schulz. *Verknüpfung von Text-und Modellentitäten von Softwarearchitektur-Modellen mithilfe von Wortvektoren*. 2020.
- [40] GalaxyInferno. *What is validation data used for? - Machine Learning Basics*. Juni 2022. URL: <https://galaxyinferno.com/what-is-validation-data-used-for-machine-learning-basics/> (besucht am 09.06.2023).
- [41] o.V. *Tf-idf-Maß*. URL: <https://de.wikipedia.org/wiki/Tf-idf-Ma% C3%9F> (besucht am 10.08.2023).
- [42] o.V. *Linguistic Features*. URL: <https://spacy.io/usage/linguistic-features#named-entities> (besucht am 10.08.2023).
- [43] Alireza Mansouri, Lilly Suriani Affendey und Ali Mamat. *Named entity recognition approaches*. Bd. 8. 2. Citeseer, 2008, S. 339–344.
- [44] Baohang Zhou u. a. *A Span-based Multimodal Variational Autoencoder for Semi-supervised Multimodal Named Entity Recognition*. 2022, S. 6293–6302.
- [45] Christopher Olah. *Understanding LSTM Networks*. Aug. 2015. URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (besucht am 02.08.2023).
- [46] Jonte Dancker. *A Brief Introduction to Recurrent Neural Networks*. Dez. 2022. URL: <https://towardsdatascience.com/a-brief-introduction-to-recurrent-neural-networks-638f64a61ff4> (besucht am 02.08.2023).
- [47] Zhou Beiqi. *Named Entity Recognition (NER) using Keras LSTM Spacy*. Juni 2021. URL: <https://zhoubeiqi.medium.com/named-entity-recognition-ner-using-keras-lstm-spacy-da3ea63d24c5> (besucht am 02.08.2023).

- [48] Nishanth N. *Training Custom Named Entity Recognition*. Juli 2022. URL: <https://towardsdatascience.com/train-ner-with-custom-training-data-using-spacy-525ce748fab7> (besucht am 04.08.2023).
- [49] Enes Zvornicanin. *What Are Embedding Layers in Neural Networks?* Mai 2023. URL: <https://www.baeldung.com/cs/neural-nets-embedding-layers> (besucht am 10.08.2023).
- [50] Seth Weidman. *Deep Learning from Scratch*. URL: <https://www.oreilly.com/library/view/deep-learning-from/9781492041405/ch04.html> (besucht am 10.08.2023).
- [51] Manuele Bicego und Antonella Mensi. *Null/No Information Rate (NIR): a statistical test to assess if a classification accuracy is significant for a given problem*. 2023.
- [52] Wei-Hung Weng. *Machine learning for clinical predictive analytics*. Springer International Publishing, 2020, S. 199–217.

# A. Anhang A

---

```
1 def get_determiners_for_noun(index):
2     doc = nlp(sentences[index])
3     noun_det_pairs = []
4     for i, token in enumerate(doc):
5         if token.dep_ == "nk":
6             noun = token.head.text
7             det = token.text
8             prev_word = doc[i-1].text if i > 0 else None
9             min_val, max_val = convert_det_to_min_max(det,
10                 prev_word)
11             noun_det_pairs.append((index, noun, prev_word, det,
12                 min_val, max_val))
13     return noun_det_pairs
14
15 def convert_det_to_min_max(det, prev_word):
16     if "mindestens" == prev_word:
17         return 1, "n"
18     elif "höchstens" == prev_word:
19         return 0, 1
20     elif "ein" in det.lower():
21         return 1, 1
22     elif "viele" in det.lower():
23         return 1, "n"
24     elif "jede" in det.lower():
25         return 1, 1
26     elif "genau" in det.lower():
27         return 1, 1
28     else:
29         return "n", "m"
30
31 result_cardinality = []
32 for item in index_list:
33     index = item[1]
34     result = get_determiners_for_noun(index)
35     result_cardinality.append(result)
```

---

Quelltext A.1: Erfassung und Übersetzung der Kardinalität

## B. Anhang B

---

```
1  {
2      "projectVersion": 1,
3      " projectName": "notNamed",
4      "erContent": {
5          "drawBoardContent": {
6              "drawBoardElements": [
7                  {
8                      "id": "StrongEntity--8293697183674",
9                      "displayName": "verkauf",
10                     "isHighlighted": false,
11                     "isSelected": false,
12                     "x": 0,
13                     "y": 0,
14                     "width": 151.015625,
15                     "height": 67,
16                     "objectType": "DrawBoardElement",
17                     "erType": "StrongEntity",
18                     "owningSide": null
19                 },
20                 ...
21             "connections": [
22                 {
23                     "id": "NormalAttribute--4409807264336 -->
24                         StrongEntity--1512229580642 -
25                         Connection--1942537747555",
26                     "start": "NormalAttribute--4409807264336",
27                     "end": "StrongEntity--1512229580642",
28                     "min": "1",
29                     "max": "1",
30                     "objectType": "Connection",
31                     "isSelected": false,
32                     "withArrow": false,
33                     "withLabel": false,
34                     "connectionType": "AttributeConnector",
35                     "isHighlighted": false
36                 },
37                 ...
38             ]
39         }
40     }
41 }
```

---

Quelltext B.1: Auszug aus output.json

## C. Anhang C

---

de\_model\_based\_spacy-Output

---

```
[['Text: Ein Geschäft hat viele Filialen. Jede Filiale darf von  
höchstens einem Filialleiter',  
 {'entities': [[6, 9, 'KARDINALITÄT'],  
 [10, 18, 'ENTITÄT'],  
 [19, 22, 'BEZIEHUNG'],  
 [23, 28, 'KARDINALITÄT'],  
 [29, 38, 'ENTITÄT'],  
 ...
```

---

## D. Anhang D

Index	Word	EntityType
0	1	Text: o
1	1	Ein KARDINALITÄT
2	1	Geschäft ENTITÄT
3	1	hat BEZIEHUNG
4	1	viele KARDINALITÄT
5	1	Filialen ENTITÄT
6	1	.
7	1	Jede KARDINALITÄT
8	1	Filiale ENTITÄT
9	1	darf o
10	1	von o
11	1	höchstens einem KARDINALITÄT
12	1	Filialleiter ENTITÄT
13	2	geführt o
14	2	werden o
15	2	.
16	2	Ein KARDINALITÄT
17	2	Filialleiter ENTITÄT
18	2	darf o
19	2	höchstens zwei KARDINALITÄT

Abbildung D.1.: Struktur der umgewandelten *csv*-Datei für das LSTM-Modell

## E. Anhang E

Task	Error type	Loss function	Note
Regression	Mean-squared error	$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	Easy to learn but sensitive to outliers (MSE, L2 loss)
	Mean absolute error	$\frac{1}{n} \sum_{i=1}^n  y_i - \hat{y}_i $	Robust to outliers but not differentiable (MAE, L1 loss)
Classification	Cross entropy = Log loss	$\begin{aligned} -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] = \\ -\frac{1}{n} \sum_{i=1}^n p_i \log q_i \end{aligned}$	Quantify the difference between two probability distributions
	Hinge loss	$\frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i \hat{y}_i)$	For support vector machine
	KL divergence	$D_{KL}(p  q) = \sum_i p_i (\log \frac{p_i}{q_i})$	Quantify the difference between two probability distributions

Abbildung E.1.: Auflistung verschiedener Verlustfunktionen. [52]

## F. Anhang F

```
===== Initializing pipeline =====
[2023-08-13 19:53:39,710] [INFO] Set up nlp object from config
[2023-08-13 19:53:39,716] [INFO] Pipeline: ['tok2vec', 'ner']
[2023-08-13 19:53:39,718] [INFO] Created vocabulary
[2023-08-13 19:53:39,718] [INFO] Finished initializing nlp object
[2023-08-13 19:53:39,836] [INFO] Initialized pipeline components: ['tok2vec', 'ner']
✓ Initialized pipeline

===== Training pipeline =====
ℹ Pipeline: ['tok2vec', 'ner']
ℹ Initial learn rate: 0.001
E   #      LOSS TOK2VEC LOSS NER ENTS_F ENTS_P ENTS_R SCORE
---  ---  -----  -----  -----  -----  -----  -----
  0    0      0.00    69.20   8.95    7.49   11.11   0.09
  21   200    182.58  2663.95  56.36   65.96   49.21   0.56
  47   400     71.33   58.70   57.78   65.66   51.59   0.58
  79   600     95.29   29.06   58.15   65.35   52.38   0.58
 118   800    136.31   58.73   54.13   64.13   46.83   0.54
 166  1000     84.46   23.73   56.39   63.37   50.79   0.56
 223  1200     81.04   17.08   55.65   61.54   50.79   0.56
 290  1400    191.15   29.84   57.14   65.31   50.79   0.57
 378  1600    190.20   34.97   57.87   62.39   53.97   0.58
...
 3143 4600    370.91   32.43   55.75   63.00   50.00   0.56
 3343 4800    239.31   13.01   55.11   62.63   49.21   0.55
✓ Saved pipeline to output directory
model-last
```

Abbildung F.1.: Visualisierung von den Ergebnissen der Evaluationsmetriken während dem Training des *SpaCy*-Modells.