

Masterarbeit

Informationsextraktion unter Anwendung maschinellen Lernens zur Unterstützung eines Angebotsprozesses

ausgeführt am



FACHHOCHSCHULE DER WIRTSCHAFT

Fachhochschul-Masterstudiengang
Automatisierungstechnik-Wirtschaft

von

Christopher Franz Sperl, BSc.
1810322007

betreut und begutachtet von
FH-Prof. DI Dr. techn. Udo Traussnigg

Graz, im November 2019



Unterschrift

Ehrenwörtliche Erklärung

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die benutzten Quellen wörtlich zitiert sowie inhaltlich entnommene Stellen als solche kenntlich gemacht habe.



Unterschrift

Danksagung

An dieser Stelle möchte ich mich herzlichst bei meiner Familie für die Unterstützung während des gesamten Studiums bedanken. Ebenfalls ist es mir ein besonderes Anliegen Herrn Dr. Udo Traußnigg für sein unerschütterliches Engagement, den wertschätzenden Umgang mit Studierenden in allen Situationen und die Betreuung dieser Arbeit meinen aufrichtigen Dank auszusprechen.

Seitens unseres Unternehmens in Weiz möchte ich Herrn Dipl.-Ing. Gerhard Karner, Ing. Michael Witaltschil und Dipl.-Ing. Helmut Pregartner Danke sagen. Es waren diese drei Personen, welche mich sowohl bei meinen beiden Bachelorarbeiten als auch bei dieser Masterarbeit unterstützen und diese überhaupt erst möglich machten.

Ein herzliches Dankeschön möchte ich auch den Herren Dr. Roman Kern und Dipl.-Ing. (FH) René Kaiser ausrichten. Diese unterstützten mich beide bereitwillig beim Erstellen dieser Arbeit und halfen mir einen Einstieg in das für mich anfangs unbekannte Thema zu finden.

Kurzfassung

Die vorliegende Arbeit gibt einen Einblick in die Anwendung maschinellen Lernens zur Informationsextraktion aus natürlichsprachlichem Text. Das Ziel besteht darin, TechnikerInnen bei der Angebotslegung für Transformatoren zu unterstützen, indem Algorithmen Routineaufgaben übernehmen, und damit mehr Zeit für die Bearbeitung neuer oder besonderer Anforderungen zu schaffen. Hierzu werden einführend Fachbegriffe erklärt, um einen Überblick über das Fachgebiet zu geben. Im folgenden Theorieteil werden die Grundlagen beginnend bei relativer Wahrscheinlichkeit über endliche Automaten bis hin zu neuronalen Netzen anhand von Beispielen bearbeitet. Danach wurde im praktischen Teil der Arbeit anhand prototypischer Umsetzungen die Eignung einzelner Methoden des maschinellen Lernens getestet. Um den Entwicklungen, die im Jahr 2019 im Bereich Sprachverständnis stattfanden, Rechnung zu tragen, wurden im praktischen Teil ebenfalls Methoden des Transfer Learnings eingeführt und am Beispiel ausprobiert. Neben diesen wurden zudem softwaretechnische Aspekte, wie die Visualisierung der Daten in einer Webapplikation, prototypisch umgesetzt. Abschließend wird ein Weg aufgezeigt, der auf Basis der aus den prototypischen Umsetzungen gewonnenen Erfahrung empfehlenswert erscheint.

Abstract

This master's thesis provides insight into the application of machine learning to extract information from natural language text. The desired application will support technicians in the quotation process for transformers by using algorithms to complete routine tasks and thus create more time for the processing of new or special requirements. For this purpose, introductory technical terms are explained to afford an overview of the field. The following theoretical section addresses the basics, including relative probability, finite automata and neural networks based on examples. The practical part of the work tests the suitability of individual methods of machine learning through prototypical implementations. To reflect the developments that have occurred in the field of language comprehension in 2019, methods of transfer learning are also introduced in the practical part for testing, using an example. Furthermore, technical aspects of software, such as the visualisation of data in a web application, are also implemented as prototypes. Finally, recommendations are offered based on the experience gained from the prototypical implementations.

INHALTSVERZEICHNIS

1	Einleitung	1
1.1	Vorstellung des Unternehmens	1
1.2	Motivation und Fragestellung	2
1.2.1	Ausgangssituation	2
1.2.2	Aufgabenstellung und Prozessumfeld	2
1.2.3	Zielsetzung der Arbeit	2
1.2.4	Zielsetzung des theoretischen Teils	2
1.2.5	Zielsetzung des praktischen Teils	3
2	Vorspann	4
2.1	Text als Informationsspeicher	4
2.2	Knowledge Discovery und Data Mining	4
2.2.1	Statistik	5
2.2.2	Maschinelles Lernen	6
2.2.3	Datenbanken	6
2.3	Text-Mining	6
2.3.1	Natural Language Processing	7
2.3.2	Lerntypen der Algorithmen	7
2.3.3	Information Retrieval	8
2.3.4	Information Extraction	8
2.3.5	Text Summarization	8
2.3.6	Sentiment Analysis	9
2.3.7	Social Media Mining	9
2.3.8	Biomedical Mining	9
3	Theoretische Grundlagen	11
3.1	Bereitstellung der Daten für die maschinelle Verarbeitung	11
3.2	Maschinelle Vorverarbeitung der Daten	12
3.2.1	Aufteilen in Token	12
3.2.2	Filtern von Wörtern	14
3.2.3	Ableiten der Stammform	14
3.2.4	Lexikografische Reduktion auf Grundformen	15
3.3	Bewertungsgrundlagen	15
3.4	Erkennen von Eigennamen	17
3.5	Stochastische Modellgraphen	18
3.5.1	Bayes Klassifikator	18
3.5.2	Markov Modelle	19
3.5.2.1	Markov-Kette	19
3.5.2.2	Hidden-Markov-Modell	20
3.5.2.3	Klassische Problemstellungen für HMMs	22
3.5.2.4	Maximum-Entropy-Markov-Modell	22
3.5.3	Logistische Regression	23
3.5.4	Conditional Random Fields	25
3.6	Künstliche neuronale Netze	28
3.6.1	Einlagiges Perzeptron	28
3.6.2	Feed Forward Netzwerke	30

3.6.3	Recurrent Neural Networks	31
3.6.4	Long Short Term Memory	33
3.7	Vektorsemantik	37
3.7.1	Term Frequency Inverse Document Frequency	38
3.7.2	Skip Gram with Negativ Sampling	41
4	Stand der Forschung	44
5	Praktische Umsetzung	46
5.1	Methodik	47
5.2	Software Architektur	47
5.3	Daten	50
5.4	Dokumentähnlichkeit	52
5.5	Regelbasierte Informationsextraktion	55
5.5.1	Reguläre Ausdrücke zur Informationsextraktion	55
5.5.2	Programmbibliotheken zum Formulieren der Regeln	57
5.6	Informationsextraktion mit hybridem System	59
5.7	Open Information Extraction	63
5.8	Transfer Learning	65
5.8.1	Transformer-Modell	67
5.8.2	Scaled Dot-Product Attention	69
5.8.3	FFN im Transformer	71
5.8.4	Trainieren und Testen des Modells	71
5.9	Bidirectional Encoder Representations from Transformers	74
5.10	Machine Reading Comprehension	75
5.11	Visualisierung des Dokuments	80
6	Conclusio/Ergebnis	83
7	Ausblick	85
	Literaturverzeichnis	86
	Abbildungsverzeichnis	93
	Tabellenverzeichnis	97
	Abkürzungsverzeichnis	98

1. EINLEITUNG

1.1 Vorstellung des Unternehmens

Die Geschichte der Siemens Transformers Weiz AG reicht zurück bis zum Jahr 1892. Die damals gegründeten PICHLERwerke legten den Grundstein für die Elektroindustrie am Standort Weiz. 1908 verkaufte Ing. Franz Pichler seine Anteile an der Sparte Elektromaschinenbau an die ELIN AG. Im Jahr 1959 erfolgte die Gründung der ELIN Union AG durch einen Zusammenschluss mit der österreichischen AEG. Im Jahr 2005 übernahm Siemens den VA-TECH-Konzern und zwei Jahre später wurde die Siemens Transformers Austria GmbH & Co KG durch den Zusammenschluss der Werke Weiz und Linz gegründet, die seit 2012 nunmehr Teil der Siemens AG Österreich ist. Im Jahr 2015 fand die Feier zur Auslieferung des 4.000. Leistungstransformators am Standort Weiz statt. Im Jahr 2017 feierte der Standort sein 125-jähriges Jubiläum.



Abbildung 1.1: Luftaufnahme, auf der der Standort Weiz der Siemens AG zu sehen ist. Quelle: Siemens-Pressfoto.

Derzeit arbeiten rund 1.200 MitarbeiterInnen am Standort in Weiz, der einen Umsatz von 400 Mio. € erwirtschaftet. Von den 1.500 Lieferanten, die das Werk beliefern, stammen rund 900 aus Österreich. Das Werk fertigt ca. 150 Leistungstransformatoren, Drosseln und Phasenschiebertransformatoren sowie um die 4.000 Verteiltransformatoren pro Jahr.

Bei den in Weiz gefertigten Transformatoren handelt es sich um maßgeschneiderte Lösungen, die Leistungswerte von 20 MVA bis hin zu über 1.000 MVA und Spannungsebenen von 20 kV bis hin zu 1.100 kV unterstützen.

1.2 Motivation und Fragestellung

1.2.1 Ausgangssituation

Jedes Jahr erreichen die Angebotsabteilung der Siemens Transformers AG mehrere Tausend Anfragen für Angebote über Transformatorfertigung. Diese müssen analysiert, bewertet und beantwortet werden. Je nach Kunde ist der Inhalt dieser Anfragen unterschiedlich strukturiert, besteht jedoch immer aus einer Sammlung von Anforderungen, die mit Referenzen auf Normen in einem Nebendokument hinterlegt sind. Bei diesen Nebendokumenten kann es sich um Text-Dateien, PDF-Dateien oder Office-Formate handeln. Es erfolgt eine umfangreiche manuelle Analyse der Dokumente, bei der eine Bewertung anhand von Kenngrößen und anderen Charakteristika erarbeitet und in einem Formular namens Key-Data-Sheet festgehalten wird. Anhand dessen werden danach verschiedene Risikofaktoren berechnet, die der Abschätzung des Umsetzungsaufwandes für den spezifizierten Transformator dienen. Dieser Prozess ist in seiner derzeitigen Form arbeitsintensiv und beinhaltet viele Routineaufgaben wie zum Beispiel das Herauslesen der eigentlichen Anforderung aus dem Fließtext, das Nachschlagen von Normtexten zu referenzierten Normen oder das Vergleichen der Anforderungen mit ähnlichen, bereits bearbeiteten Projekten. Diese Routineaufgaben sind fehleranfällig und verbrauchen wertvolle Zeit der qualifizierten BearbeiterInnen. Diese Zeit wäre, so die Ansicht des Unternehmens, besser in das Studium neuer oder besonderer Anforderungen investiert.

1.2.2 Aufgabenstellung und Prozessumfeld

In Zusammenarbeit mit dem Unternehmen Know-Center GmbH soll ein System entwickelt werden, das den BearbeiterInnen der Firma Siemens Transformers AG Routineaufgaben abnimmt, damit sich diese auf das Ausarbeiten schwierigerer Detaillösungen konzentrieren können und einen schnelleren Überblick über die geforderten Spezifika des anzubietenden Transformators bekommen. Es soll jede Seite der Spezifikation auf Normen und Anforderungen geprüft und relevante Informationen für die BearbeiterInnen visuell hervorgehoben werden, nachdem das Tool diese verarbeitet hat. Die Aufgabenstellung, die dieser Arbeit zugrunde liegt, besteht darin, diesen Prozess im ersten von drei Jahren zu begleiten und Konzepte für eine Implementierung zu erarbeiten. Es gilt, herauszufinden, welche Methoden des maschinellen Lernens oder auch der Computertechnologie im Allgemeinen für ein solches Werkzeug geeignet sind und auf Basis welcher Technologien diese umsetzbar sind.

1.2.3 Zielsetzung der Arbeit

Der Horizont dieser Arbeit erstreckt sich über das erste Jahr eines dreijährigen Entwicklungsprojektes. In diesem frühen Stadium des Projektes soll die Grundlage für ein nachvollziehbares und deterministisch handelndes Werkzeug im Unternehmen gelegt werden, das in weiterer Folge die AngebotstechnikerInnen beim Erstellen von technischen Angeboten unterstützt. In der Arbeit selbst sollen mögliche Algorithmen und Architekturen beschrieben und das Ergebnis der Entwicklung dokumentiert werden.

1.2.4 Zielsetzung des theoretischen Teils

Natural Language Processing (NLP) ist ein Zweig der künstlichen Intelligenz, der Computern hilft, die menschliche Sprache zu verstehen, zu interpretieren und diese in geordnete Strukturen zu bringen. NLP stützt sich auf viele Disziplinen, darunter Informatik und Linguistik, um die Lücke zwischen menschlicher Kommunikation und Computerverständnis zu schließen. Im theoretischen Teil der Arbeit sollen die Grundlagen der verwendeten Methoden und Werkzeuge beschrieben werden, die im zu entwickelnden Werkzeug zur Anwendung kommen könnten. Es sollen die verwendeten NLP-Methoden beschrieben werden, ein Grundstock an Wissen zum Thema geschaffen werden und es soll auf die Aufbereitung der Daten eingegangen werden. Dieser theoretische Teil der Arbeit soll das Fundament schaffen, um die im praktischen

Teil der Arbeit durchgeführten Experimente zu verstehen und eine ausreichend genaue Abschätzung ihrer Eignung für das zu entwickelnde Werkzeug abgeben zu können.

1.2.5 Zielsetzung des praktischen Teils

Im praktischen Teil der Arbeit sollen die Ergebnisse der prototypischen Umsetzung dokumentiert werden. Das Dargelegte soll als Unterstützung bei der Entscheidung, welche maschinellen Lernverfahren eingesetzt werden, dienen. Es soll gezeigt werden, auf welche Weise das Werkzeug zum gegebenen Entwicklungsstand einsetzbar ist, und ein Ausblick auf zukünftige Entwicklungsziele gegeben werden.

2. VORSPANN

2.1 Text als Informationsspeicher

Im Informationszeitalter des 21. Jahrhunderts werden Tag für Tag enorme Mengen an Daten produziert und gespeichert. Der Ursprung dieser Daten ist mannigfaltig: Soziale Netzwerke, E-Mails, Aufzeichnungen von Patienten-Daten, Artikel, Blog-Posts oder Rezensionen auf Handelsplattformen sind nur einige wenige Beispiele für die großen Datenmengen, die das Internet und die Intranets der Unternehmen passieren und bearbeitet werden. In einem Bericht gesponsert von EMC wurde vorausgesagt, dass sich die Menge an Daten, die 2005 im weltweiten Internet vorhanden waren, bis zum Jahr 2020 verdreihundertfachen wird.¹

Die vorhandenen Daten sind in verschiedenster Form gespeichert; ein Beispiel für strukturierte Daten sind Tabellen oder relationale Datenbanken. Ein Beispiel für unstrukturierte Daten ist natürlichsprachlicher Text. Dieser ist eine sehr häufige Form, in dem Daten vorliegen, und somit von hohem Wert für die Informationsgewinnung im Allgemeinen. Durch die enormen Mengen, in denen natürlichsprachlicher Text vorhanden ist, stellt er für die maschinelle Weiterverarbeitung eine wertvolle Datenquelle dar. Der Nachteil dabei ist, dass, während es für einen Menschen einfach ist, die darin enthaltenen Informationen zu extrahieren, diese für Computer nur sehr schwer zu verarbeiten sind. Aus diesem Grund ist es von größtem Interesse, Algorithmen und Methoden zu entwickeln, die diese Daten zugänglich machen, in strukturierte Form bringen und damit verwertbar machen. Ist der Text in ein verwertbares Format gebracht, kann er entweder weiteren Algorithmen zur Weiterverarbeitung dienen oder je nach Verwendungszweck für menschliche Bearbeiter aufbereitet werden. Das Erschließen dieser Informationen wird mit Methoden des Text Minings bewerkstelligt.

2.2 Knowledge Discovery und Data Mining

Text Mining und Data Mining sind Begrifflichkeiten aus dem Gebiet der Knowledge Discovery. Es gibt mehrere Definitionen, die diese Begrifflichkeiten beschreiben; für diese Arbeit werden sie wie folgt gedeutet:² Unter Knowledge Discovery wird die Extraktion implizit gültiger, neuer und potenziell nützlicher Muster aus einer Datenbank verstanden. Die einzelnen Teilschritte hin zum extrahierten Muster sind in Abbildung 2.1 dargestellt. Darin wird der Extraktionsprozess aufgeschlüsselt und dargestellt. Dieser besteht aus mehreren Schritten, die das Vorbereiten der Daten, die Suche nach Mustern, die Bewertung dieser und das Verfeinern der Ergebnisse beinhalten. Die gewonnenen Erkenntnisse sollen schließlich mit einem gewissen Grad an Sicherheit für neue Daten gültig sein. Darüber hinaus sollen die Muster, nach denen die Daten extrahiert worden sind, zumindest indirekt, nachvollziehbar und verständlich sein.^{3,4}

Historisch wurden dem Prozess, nützliche Muster in Daten zu finden, verschiedene Namen gegeben, darunter Data Mining, Wissensextraktion, Datenarchäologie und Datenmusterverarbeitung. Der Begriff Data Mining wurde hauptsächlich von Datenanalysten, Statistikern und den Management-Information-Systems-Gemeinschaften verwendet. Hier soll der Begriff Data Mining als die Anwendung verschiedener Algorithmen zur Extraktion von Mustern aus gegebenen Daten verstanden werden. Damit ist der Data-Mining-Prozess an sich ein Teil des Knowledge-Discovery-Prozesses.⁵

Auch hierbei sollen nicht blind Algorithmen aus Mathematik, Statistik und dem maschinellen Lernen angewendet werden, sondern es soll auf Nützlichkeit, Nachvollziehbarkeit und Sinnhaftigkeit des Prozesses

¹Vgl. Gantz/Reinsel (2012), S. 1-16.

²Vgl. Allahyari et al. (2017), S. 1 f.

³Vgl. U. M. Fayyad/Piatetsky-Shapiro/Smyth et al. (1996), S. 82.

⁴Vgl. Frawley/Piatetsky-Shapiro/Matheus (1992), S. 57.

⁵Vgl. U. Fayyad/Piatetsky-Shapiro/Smyth (1996), S. 39.

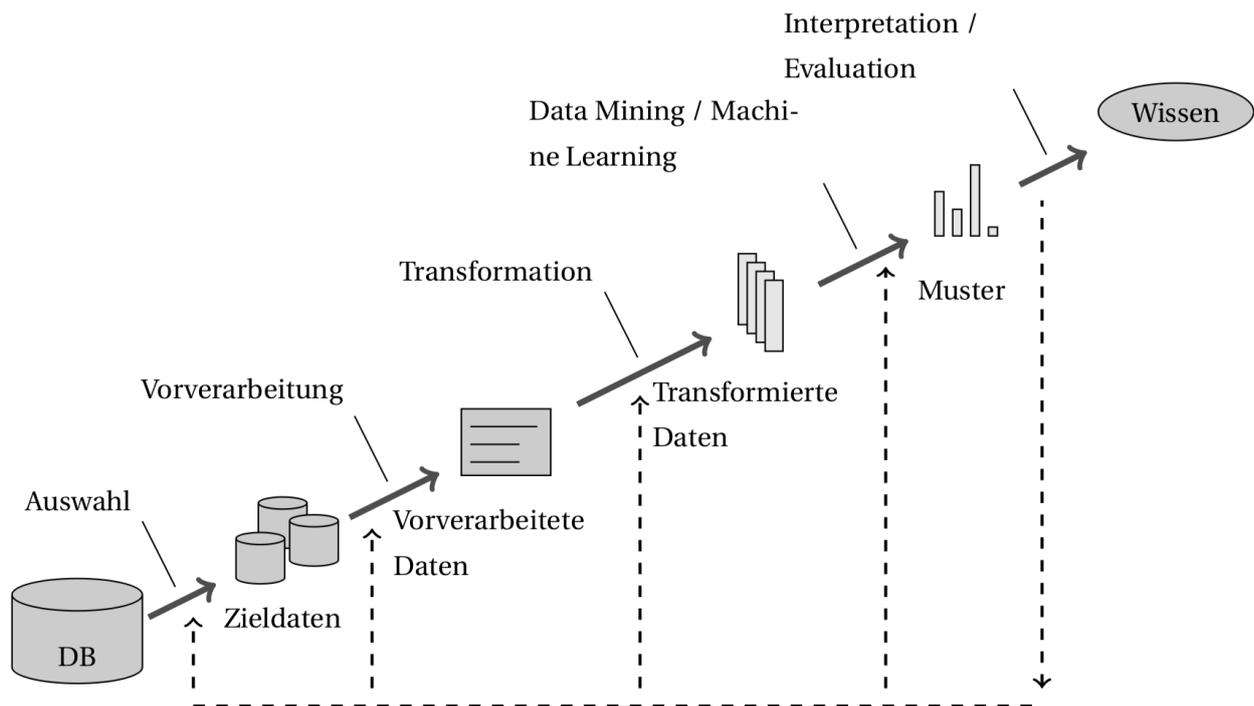


Abbildung 2.1: Darstellung eines Knowledge Discovery Prozesses. Quelle:Frochte (2018), S. 16.

geachtet werden. Die Forschung in den Bereichen Knowledge Discovery und Data Mining hat in den letzten Jahren aufgrund der gesteigerten Rechenleistung von Computern und ständiger Weiterentwicklungen der Software große Fortschritte gemacht. Data Mining entwickelt sich weiterhin aus den Schnittstellen der Bereiche maschinelles Lernen, Datenbank-Technologie, Statistik und künstliche Intelligenz, was die interdisziplinäre Natur dieses Feldes veranschaulicht.⁶

Aufgrund der hohen Relevanz dieser Bereiche sollen diese in den folgenden Unterkapiteln kurz erläutert werden.

2.2.1 Statistik

Die Statistik ist eine Disziplin der Mathematik beruhend auf Beobachtung. Sie ist ein unerlässliches Hilfsmittel zur Analyse von Daten und dient der Verdeutlichung zugrundeliegender Konzepte und Muster. Das maschinelle Berechnen von Daten ist ein essenzieller Bestandteil der Statistik.⁷ Dabei ermöglichen statistische Methoden dem Anwender die Beurteilung von Zähl- und Messergebnissen. In diesem Fall bedient sich die Statistik bei der Wahrscheinlichkeitsrechnung, mit der Aussagen über sogenannte Zufallsvorgänge getroffen werden.⁸

Die relative Häufigkeit trifft eine Aussage der Form

$$P = \frac{\text{Günstige}}{\text{Mögliche}} \quad (2.1)$$

und wird meist mit dem Buchstaben P für Probability dargestellt, so auch in dieser Arbeit. Mag diese Aussage an dieser Stelle noch trivial wirken, so wird im späteren Verlauf dieser Arbeit möglicherweise deutlich, wie wichtig sie für die Erschließung von Wissen auf Grundlage von Text ist. Maschinelle Lernverfahren beruhen teils auf Statistik, aber nicht ausschließlich.

⁶Vgl. Allahyari et al. (2017), S. 2.

⁷Vgl. Rice (2006), S. xv.

⁸Vgl. Timischl (1995), S. 39.

2.2.2 Maschinelles Lernen

Maschinelles Lernen wird als Teilgebiet der künstlichen Intelligenz angesehen und hat wiederum das sogenannte Deep Learning als Unterdisziplin. Von einigen wird maschinelles Lernen als Hilfsmittel für Disziplinen wie Information Retrieval und Data Mining gesehen. Auch wenn dies eine Frage der Sichtweise ist, kann man maschinelles Lernen auch als Schnittmenge von Informatik, Mathematik und seiner Anwendungen begriffen werden.⁹

In jedem Fall ist es eine Disziplin, die in der maschinellen Textverarbeitung reichlich Einsatz findet. Algorithmen wie k-Nearest-Neighbors beim Erkennen von Mustern oder Conditional Random Fields bei der Klassifizierung sequenzieller Daten sind Dreh- und Angelpunkt dieser. In vielen Fällen erscheint maschinelles Lernen unverzichtbar, weil die schiere Anzahl der zu verarbeitenden Dokumente und die kurze, benötigte Reaktionszeit der Anwendung die manuelle Alternative impraktikabel machen. Die Ergebnisse von maschinellem Lernen bei der maschinellen Textverarbeitung haben mittlerweile eine Akkuranz erreicht, die mit jener von ausgebildeten Fachkräften vergleichbar ist.¹⁰ Maschinelle Lernverfahren arbeiten zu großen Teilen am effektivsten, wenn ein ausreichend großer Datenvorrat vorhanden ist. Dieser soll in entsprechender Qualität, Struktur und mit hinreichend schneller Datenanbindung zur Verfügung gestellt werden.

2.2.3 Datenbanken

Durch das Erkennen von Wissen in großen Datenbanken können aus den vorhandenen Datensätzen interessante Informationen, wie zum Beispiel Regelmäßigkeiten oder Metainformationen, extrahiert und aus verschiedenen Blickwinkeln untersucht werden. Datenbanken dienen dabei als schnelle und verlässliche Quellen für die Daten zur Wissensgenerierung.¹¹

Während Struktur und Volumen der vorhandenen Daten augenscheinlich wichtige Kenngrößen für die Güte einer Datenbank darstellen, sind vor allem auch die Performance der Datenbank an sich und die Eignung der Datenbank für Data Analysis eine wichtige Kenngröße. In-Memory-Database-Systeme (IMDB) wie Apache Ignite¹² erhöhen die Verarbeitungsgeschwindigkeit auf großen Datenbeständen wesentlich. Ausgeklügelte Systeme, die Daten zur Verarbeitung in schnellen Arbeitsspeichern zur Verfügung stellen, oder Systeme, die Daten auf Rechenverbänden verteilt verarbeiten, ermöglichen die Verarbeitung von Datenmengen, die vor wenigen Jahren kaum denkbar gewesen wären. Bei Rechen- und Daten-Zentren, die in der maschinellen Wissensgewinnung zum Einsatz kommen, wird mittlerweile von sogenannten E-Science-Infrastrukturen¹³ gesprochen. Auch wenn sich dieser Begriff auf jeden modernen Rechner anwenden lässt, so meint er im Allgemeinen große Datenzentren, die über den Heimrechner bei Weitem hinausgehen. Ein weiterer Trend ist der Wechsel von lokalen Datenbanken hin in die Cloud, in der Speicherplatz und verschiedene Rechendienstleistungen remote zur Verfügung gestellt werden. Im Folgenden soll vom Rahmen der Wissensgewinnung aus natürlichsprachlichem Text hin zur tatsächlichen Anwendung gegangen werden.

2.3 Text-Mining

Der Begriff Text Mining wurde erstmals eingeführt von Feldman und Dagan¹⁴. Dabei wird Text Mining oder Knowledge Discovery from Text als Extraktion hochqualitativer Informationen aus folgenden Datenformen verstanden:

- Strukturierte Textdaten: Ein Beispiel hierfür sind relationale Datenbanken.¹⁵

⁹Vgl. Frochte (2018), S. 9.

¹⁰Vgl. Sebastiani (2002), S. 47.

¹¹Vgl. Chen/Han/Yu (1996), S. 866 f.

¹²Vgl. Zheludkov/Isachenko et al. (2017), S. 6 ff.

¹³Vgl. Yahyapour (2018), S. 369 ff.

¹⁴Vgl. Feldman/Dagan (1995), S. 112-117.

¹⁵Vgl. Džeroski (2009), S. 887-911.

- Semistrukturierte Textdaten: Beispiele sind JSON, XML oder Sourcecode.
- Unstrukturierte Textdaten: Beispiele sind gesprochene oder geschriebene natürliche Sprache.

Text Mining deckt dabei eine breite Palette verwandter Themengebiete und Algorithmen für die Analyse von Text ab, die verschiedene Disziplinen wie: Information Extraction, Natural Language Processing, maschinelles Lernen, Web- und biomedizinische Wissenschaften umfasst.¹⁶

Diese Themengebiete sind nicht scharf voneinander zu trennen, sondern bilden ein in sich verschmelzendes Netz aus Teilbereichen, die sich mit der Wissenserschließung aus Text befassen. Im Folgenden soll ein Überblick über einige dieser Teilbereiche gegeben und ihre Bedeutung erörtert werden.

2.3.1 Natural Language Processing

Während Text Mining das Verarbeiten aller Datenformen umfasst, ist der Teilbereich des Natural Language Processings jener Teilbereich des Text Minings, der sich mit der Verarbeitung von natürlicher Sprache befasst. Dies umfasst sowohl gesprochene als auch geschriebene Sprache, wenn sich auch die initialen Methoden, die bei der Verarbeitung der beiden Informationstypen Anwendung finden, stark voneinander unterscheiden. Während im Zeitalter der digitalen Pioniere, der Lochkarten und der Batchverarbeitung die Verarbeitung eines einzelnen Satzes bis zu sieben Minuten beanspruchte, werden in Zeiten von Google, High Performance Computing und Steam Processing teils ganze Bücher in unter einer einzigen Sekunde verarbeitet.¹⁷ Während in der Vergangenheit Modelle wie logistische Regression oder Algorithmen wie Support Vector Machines das Natural Language Processing dominierten, erreichen heutzutage Deep-Learning-Modelle bereits vergleichbare und teils sogar bessere Ergebnisse. Die Forschung geht derzeit immer weiter in Richtung Convolutional-, Recurrent- und/oder Recursiv-Neural Networks. Diese Algorithmen eignen sich für Deep-Learning-Ansätze und werden untereinander kombiniert. Sie verwenden dicht besetzte Matrizen, wohingegen die klassischen Modelle üblicherweise hochdimensionale, schwach besetzte Matrizen verwendeten, um deren Dateninhalt abzubilden. Moderne Verfahren kombinieren die einzelnen Modelle miteinander und erreichten damit bereits in den Disziplinen der Computer Vision und Mustererkennung große Erfolge. Der Einzug dieser Modelle in die Sprachverarbeitung ist der nächste logische Schritt.¹⁸

2.3.2 Lerntypen der Algorithmen

Im Bereich des maschinellen Lernens gibt es zwei verschiedene Arten des Lernens. Die eine ist das unüberwachte Lernen, das im Englischen als Unsupervised Learning bezeichnet wird. Hierbei kann das Modell nicht anhand der Daten trainiert werden, weil diese nicht mit Labels versehen sind. Die Lernalgorithmen versuchen stattdessen, anhand inhärenter Merkmale der Daten selbst Muster und Zusammenhänge abzuleiten. Ein Beispiel für ein solches Lernverfahren ist der k-Means-Clustering-Algorithmus. Bei diesem handelt es sich um sogenanntes Lazy Learning,¹⁹ weil der Algorithmus nicht anhand der Daten im Vorhinein trainiert werden muss, sondern zur Zeit der Ausführung anhand der Daten direkt lernt. Im Gegensatz zu den unüberwachten Lernmethoden gibt es das überwachte Lernen, im Englischen als Supervised Learning bezeichnet. Hierbei werden die Modelle, mithilfe von Algorithmen wie neuronalen Netzen oder Conditional Random Fields, an gelabelten Daten trainiert.²⁰

Beim überwachten Lernen wird üblicherweise eine inhärente Grundwahrheit der gelabelten Daten angenommen. Aus dieser Annahme folgt - der Natur der Dinge entsprechend - dass die Qualität der Trainingsergebnisse direkt mit der Qualität der Labels der Daten korreliert. Die Daten werden, im allgemeinen Fall, an

¹⁶Vgl. Allahyari et al. (2017), S. 2.

¹⁷Vgl. Cambria/White (2014), S. 48 ff.

¹⁸Vgl. Young et al. (2018), S. 56-73.

¹⁹Vgl. Dinov (2018), S. 267-287.

²⁰Vgl. Allahyari et al. (2017), S. 2 f.

anderen Daten trainiert als jenen, anhand derer sie bewertet werden. Würden die Modelle anhand derselben Daten wie jene, an denen sie trainiert wurden, evaluiert werden, würde sich ein verfälschtes Ergebnis zeigen. Der Algorithmus würde unter Umständen nur das auswendig Gelernte wiedergeben, nicht jedoch die erlernten Merkmale anwenden. Beim überwachten Lernen wird deshalb der Datenbestand meist in Daten zum Training und Daten zur Evaluierung aufgeteilt. Ein weiteres Szenario, das beim Trainieren von Modellen auftreten kann, ist, dass anhand der Daten nicht im Vorhinein bekannt ist, wie die optimale Strategie aussieht. Es ist jedoch bekannt, wie das gewünschte Ergebnis aussieht, das erreicht werden soll. Bei solchen Aufgaben kommt bestärkendes Lernen zum Einsatz; im Englischen wird von Reinforcement Learning gesprochen. Hierbei generiert der Algorithmus inkrementell verschiedene Vorgehensweisen und evaluiert diese. Immer dann, wenn das erhaltene Ergebnis besser ist als das bisher beste Ergebnis, wird dem Algorithmus positives Feedback gegeben. Sollte das erhaltene Ergebnis schlechter sein als das bisher beste Ergebnis, wird negatives Feedback gegeben. Anhand des Feedbacks versucht der Algorithmus ein optimales Ergebnis zu erzielen.²¹

Ein Ansatz, der hierbei gegebenenfalls zur Anwendung kommen kann, ist der sogenannte ‚delayed reward‘, bei dem dem Algorithmus erst nach einigen Iterationen Feedback zu den gewählten Vorgehensweisen gegeben wird.

2.3.3 Information Retrieval

Beim Information Retrieval geht es um das Auffinden relevanter Informationsressourcen.²² Als gut verständliches Beispiel hierfür soll die Suchmaschine BASE²³ (Bielefeld Academic Search Engine) genannt sein. Dies ist eine Suchmaschine für akademische Quellen, die für wissenschaftliche Arbeiten oder zum Lernen verwendet werden. Dabei geht es weniger darum, zu erkennen, was genau der Inhalt des Dokuments ist, als darum, anhand der Metadaten des Dokumentes seine Relevanz für den vorgegebenen Suchbegriff zu bewerten.

2.3.4 Information Extraction

Informationsextraktion ist die Aufgabe, Informationen oder Fakten aus unstrukturierten oder semistrukturierten Dokumenten automatisch zu extrahieren. Dieser Schritt dient oft als Basis für die weitere Verarbeitung dieser Informationen. Als Beispiele sollen das Erkennen von Entitäten (Named Entity Recognition) oder das Erkennen der Wortarten (Part of Speech Tagging) in Texten genannt sein. Um die Leistung eines solchen Systems zu messen, sollte es idealerweise an einem repräsentativen Dokument, dessen Informationen mit Sicherheit bekannt sind, gemessen werden.²⁴

Daten, die solchen Bewertungen dienen, dienen auch zum Anlernen - im Englischen spricht man vom Teaching - solcher Algorithmen. Diese Daten sind mit Annotationen versehen und werden als gelabelte Daten bezeichnet. Sind diese gelabelten Daten nicht in ausreichendem Maße vorhanden oder ist der Datenbestand so groß, dass es händisch nicht möglich ist, alle nötigen Relationen abzubilden, wird unter anderem zu unüberwachten Lernmethoden gegriffen (siehe Abschnitt 2.3.2).

2.3.5 Text Summarization

Die Textzusammenfassung, im Englischen Text Summarization oder Text Compression genannt, hat zur Aufgabe, aus einem gegebenen Ursprungstext eine kurze und prägnante Zusammenfassung zu erstellen, die die wesentlichen Ideen des Quelltextes erfasst und verkürzt wiedergibt.²⁵ Welche Informationen in ei-

²¹Vgl. Frochte (2018), S. 23 f.

²²Vgl. Allahyari et al. (2017), S. 2.

²³Vgl. Lösch (2011), S. 11 ff.

²⁴Vgl. King/Lowe (2003), S. 625.

²⁵Vgl. L. Liu et al. (2018), S. 8109.

ner Zusammenfassung stehen sollen, hängt von den Bedürfnissen des Benutzers ab. Themenorientierte Zusammenfassungen konzentrieren sich auf das Thema, das einen Benutzer interessiert, und extrahieren die Informationen im Text, die sich auf das angegebene Thema beziehen. Auf der anderen Seite versuchen generische Zusammenfassungen möglichst viel des Informationsinhalts abzudecken, wobei die allgemeine thematische Organisation des ursprünglichen Textes erhalten bleiben soll.²⁶

2.3.6 Sentiment Analysis

Zu wissen, wie die Wahrnehmung eines Textes sowie das Gefühl der Person hinter der Nachricht beim Verfassen dieser ist, kann von großem Interesse sein. Die Sentiment-Analyse versucht genau über diese Eigenschaft eine qualifizierte, quantitative Aussage zu treffen. Eine solche Aussage kann von großem Wert sein; als Beispiel soll die Tourismusbranche genannt sein, in der damit große Mengen an Feedback zu einem Urlaubsaufenthalt automatisiert ausgewertet und bewertet werden können.²⁷

Anhand einer Übersicht über die Bewertungen, aufgeteilt auf Regionen, könnten beispielsweise Entscheidungen getroffen werden, in welchen Teilregionen welche Investitionen ein effektives Mehr an Kundenzufriedenheit verursachen könnten. Aufgrund der großen wirtschaftlichen Bedeutung solcher Aussagen ist die Sentiment-Analyse eines der Hauptanwendungsgebiete des Natural Language Processings. Es wird hierbei auch von Opinion-Mining gesprochen.²⁸

2.3.7 Social Media Mining

Unter Social Media Mining wird der Prozess des Extrahierens, Analysierens und Repräsentierens von umsetzbaren Mustern aus Social Media-Daten verstanden. Social Media Mining führt dabei grundlegende Konzepte und Algorithmen ein, die für die Untersuchung der enormen, in Social Media auftretenden Datenmengen geeignet sind. Es werden dabei Theorien und Methoden aus verschiedenen Disziplinen wie Informatik, maschinellem Lernen, Data-Mining, Netzwerkwissenschaft, Soziologie, Ethnografie, Statistik, Optimierung und Mathematik zum Einsatz gebracht.²⁹

Auf Social Media-Seiten generierte Daten unterscheiden sich von herkömmlichen Daten für klassisches Text-Mining. Social Media-Daten sind weitgehend von Benutzern erzeugte Inhalte. Diese sind wortreich, verrauscht, verteilt, unstrukturiert und dynamisch.³⁰

Es sind zwei Eigenheiten von Social Media, die diese Disziplin prägen. Die Erste ist, dass der auf Social-Media-Seiten verwendete Slang, beispielsweise auf Seiten wie Twitter, gegebenenfalls nicht den Regeln der Grammatik entspricht und durch Jugendsprache und Abkürzungen geprägt ist. Das Zuordnen dieser Wortspielereien ist die erste Eigenheit, die eine Herausforderung an Social Media-Mining darstellt. Die Zweite ist die große Menge an Daten, die hochdynamisch produziert und verändert wird.

2.3.8 Biomedical Mining

Das Volumen der veröffentlichten biomedizinischen Forschung und damit die zugrunde liegende Wissensbasis wachsen mit zunehmender Geschwindigkeit. Zu jenen Werkzeugen, die Forscher bei der Bewältigung dieser Informationsflut unterstützen können, gehört das Text-Mining.³¹

Durchschnittlich werden täglich mehr als 3.000 neue Artikel in von ExpertInnen begutachteten Zeitschriften veröffentlicht.³²

²⁶Vgl. Erkan/Radev (2004), S. 458.

²⁷Vgl. Alaei/Becken/Stantic (2019), S. 175 ff.

²⁸Vgl. Pang/L. Lee et al. (2008), S. 1 ff.

²⁹Vgl. Zafarani/Abbasi/H. Liu (2014), S. 16.

³⁰Vgl. Gundecha/H. Liu (2012), S. 4.

³¹Vgl. Cohen/Hersh (2005), S. 57-71.

³²Vgl. J. Lee et al. (2019), S. 1.

Damit wird deutlich, dass zur Hebung der in diesen Ressourcen verarbeitenden Wissensschätze maschinelle Methoden zum Einsatz kommen müssen. Andererseits kommen viele der Erkenntnisse in den Bereichen der maschinellen Textverarbeitung aus der biomedizinischen Gemeinschaft, weshalb dieser Teilbereich zur Vollständigkeit dieser Einleitung Erwähnung gefunden haben muss.

Der folgende Abschnitt wird sich mit der Theorie hinter Umsetzungen sowie mit bereits geleisteten Arbeiten im Bereich der maschinellen Wissensgewinnung aus Text befassen, um danach in den prototypischen Umsetzungen Anwendung zu finden.

3. THEORETISCHE GRUNDLAGEN

Die enorme Menge an Informationen, die in natürlichsprachlichen Texten gespeichert sind, können nicht ohne Vorbereitung zur Weiterverarbeitung durch Computer verwendet werden. Für den Computer ist der Text eine einfache Folge von Zeichenketten ohne spezifische Bedeutung. Daher sind spezifische Vorverarbeitungsschritte und Algorithmen erforderlich, um nützliche Muster zu extrahieren. Text-Mining bezieht sich im Allgemeinen auf das Extrahieren von interessanten Informationen und Wissen aus unstrukturiertem Text.³³

In diesem Kapitel sollen die theoretischen Grundlagen beginnend bei eben dieser Vorverarbeitung des Textes dargelegt werden. Diese Grundlagendarbietung soll mit der Bereitstellung der Daten beginnen und dann zu jenen Modellen führen, die im praktischen Teil der Arbeit Verwendung finden und die gewünschte Information aus dem Text extrahieren. Der Zweck dieser Arbeit ist es, Technologien zu erforschen welche aus technischen Spezifikationen Eckdaten über das darin beschriebene Produkt extrahieren. Diese Daten werden danach in eine strukturierte Form gebracht, um der weiteren technischen Abklärung dienlich zu sein. Die erste sich damit ergebende Herausforderung ist das Speichern der Daten und die Vorverarbeitung eben dieser.

3.1 Bereitstellung der Daten für die maschinelle Verarbeitung

Gängige Formate, in denen natürlichsprachliche Daten zur Verfügung stehen, sind zumeist PDF-, MS-Word- oder ODF-Dateien. Ein noch komplexeres Format, um Textinformation zu übermitteln, wäre eine Bilddatei, zum Beispiel in Form einer JPEG-Datei. Zum Verarbeiten eines Bildes zur Texterkennung wird eine optische Zeichenerkennung benötigt; im Englischen ist hierfür der Ausdruck Optical Character Recognition (OCR) gebräuchlich. Moderne OCR-Prozesse erreichen dabei Werte von über 99 % an richtig wiedergegebenen Zeichen in Dokumenten mit gut erkennbarer Schrift.³⁴

Auch wenn diese Erkennungsrate hoch erscheint, wird von einem sehr gut leserlichen Text ausgegangen. Bei Texten mit mehreren Tausend Zeichen sind ebenfalls mehrere Hundert Fehler zu erwarten. Die optische Zeichenerkennung mittels OCR liegt außerhalb des Fokus dieser Arbeit, dennoch soll als Open Source Tool für OCR die Tesseract OCR Engine genannt sein. Die Entwicklung dieser Engine begann im Jahre 1984 in den HP-Laboren in Bristol als PhD-Forschungsprojekt und ist seit dem Jahr 2005 als Open-Source-Software verfügbar.³⁵ Für die Umwandlung von PDF-, MS-Word- oder ODF-Dateien soll das Open Source Tool Apache Tika³⁶ genannt werden. Dieses wurde ursprünglich in Java entwickelt, ist über sein Command Line Interface (CLI) aber auch in anderssprachigen Softwareprojekten einbindbar. Dieser Vorgang der initialen Informationsbereitstellung in maschinenlesbarer Form kann, je nach Qualität der eingehenden Daten, eine große erste Hürde für ein Projekt dieser Art darstellen. Des Öfteren ist es nicht ausreichend, den Text in Unicode-Zeichen zu konvertieren und diese danach über eine Datenbank zur Weiterverarbeitung bereitzustellen. Auch die Metadaten der Dokumente können für die Weiterverarbeitung von Interesse sein. So ist es für die weiteren Verarbeitungsschritte durchaus von Vorteil oder sogar notwendig, bereits beim Einspielen der Dokumente zu wissen, in welcher Sprache der darin geschriebene Text verfasst wurde. Ein weiteres Beispiel für die Nützlichkeit von Metadaten wäre zu wissen, wie groß der Anteil an enthaltener Bildinformation in den einzelnen Dateien ist. Weil diese unter Umständen nicht maschinell verarbeitet werden können, kann es von Vorteil sein, deren Häufigkeit als statistische Unsicherheit in die Betrachtungen der Dokumente einfließen zu lassen.

³³Vgl. Hotho/Nürnberger/Paaß (2005), S. 19 f.

³⁴Vgl. Singh (2013), S. 545.

³⁵Vgl. Smith (2007), S. 629 ff.

³⁶Vgl. Mattmann/Zitting (2011), S. 1-256.

Wurde diese erste Hürde überwunden, stellt sich die Frage nach der Repräsentation des Textes für die weitere Verarbeitung. Die am häufigsten verwendete Repräsentation für Texte ist die sogenannte Bag-of-Words (BOW)-Repräsentation.³⁷ Diese berücksichtigt die Anzahl der Vorkommen jedes Wortes, ignoriert aber deren Reihenfolge. Diese Darstellung führt zu einer Vektordarstellung, die mit Dimensionsreduktionsalgorithmen bearbeitet werden kann.³⁸ Als Beispiel für einen solchen reduzierenden Algorithmus ist das Latent Semantic Indexing (LSI) zu nennen, bei dem werden Wörter mit ähnlichen Bedeutungen in einem niedrigdimensionalen, latenten Raum nahe zueinander abgebildet. Je nachdem, wie oft gewisse Wörter im BOW auftreten, können Rückschlüsse auf das Dokument selbst gezogen werden. Wenn in einem Dokument zum Beispiel viele verschiedene Hunderassen genannt werden, ist es wahrscheinlich, dass das Dokument von Hunden handelt. Ein trainiertes LSI-Modell könnte hierbei genutzt werden, um die einzelnen Hunderassen dem Wort Hund zuzuordnen, und damit die Dimensionalität des BOW zu reduzieren. Aufgrund der Tatsache, dass Suchmaschinen zu großen Teilen mit einem BOW sowie auch mit LSI arbeiten, wird dieser Mechanismus auch bei der Suchmaschinenoptimierung von Webseiten über Keywordstacking ausgenutzt, um besser gefunden zu werden. Eine weitere Möglichkeit, Wortsequenzen darzustellen, wäre es, die gesamte Sequenz abzuspeichern und diese zu analysieren. Der Vorteil einer solchen Vorgehensweise liegt darin, dass die Reihenfolge der Wörter eine Rolle spielt und somit in die Voraussagen miteinbezogen werden kann. Ein Nachteil ist, dass der Rechenaufwand und die vorhandenen Varianten eine entsprechende Analyse schwierig gestalten. Werden natürlichsprachliche Texte näher betrachtet, kann es logisch erscheinen, dass der Informationsgehalt eines Satzes an sich meist auf nur wenigen Wörtern beruht. Diese Annahme wird bei den N-Gramm-Modellen verwendet. Bei diesen wird von Wortverkettungen mit N typischerweise aufeinander-folgenden Wörtern gesprochen. Ein Bigramm beispielsweise besteht aus zwei Wörtern, ein Trigramm aus drei und so weiter. Dem Modell kann dabei die Wahrscheinlichkeit des Auftretens der Wortverkettung zugeordnet werden.³⁹

Bevor man sich allerdings ernsthafte Gedanken über die Berechnung dieser Wahrscheinlichkeiten und deren Zuordnungen machen kann, sollte die Aufbereitung des zu verarbeitenden Textes für die darauf folgende maschinelle Verarbeitung, genauer betrachtet werden.

3.2 Maschinelle Vorverarbeitung der Daten

Die maschinelle Vorverarbeitung des Textes ist eine der Schlüsselkomponenten in Text-Mining-Algorithmen. Beispielsweise umfasst eine Textkategorisierungspipeline: Vorverarbeitung, Feature-Extraktion, Feature-Auswahl und Klassifizierung anhand der ausgewählten Features. Obwohl die Annahme gilt, dass die Feature-Extraktion, die Feature-Auswahl und der Algorithmus einen erheblichen Einfluss auf die Klassifizierung haben, kann die Vorverarbeitung einen signifikanten Einfluss auf diesen Erfolg haben.⁴⁰ Die Vorverarbeitungsschritte umfassen normalerweise Aufgaben wie Tokenisierung, Filterung, Stemming und Lemmatisierung.⁴¹ Im Folgenden sollen diese beschrieben werden. Es ist im Allgemeinen allerdings zu beachten, dass die tatsächlich angewendeten Vorverarbeitungsschritte variieren können. Je nachdem, welche Methoden im späteren Analyseverlauf zur Anwendung kommen, können unterschiedlich vorbereitete Daten von Vorteil sein.

3.2.1 Aufteilen in Token

Die erste Aufgabe ist es, den Text in Token aufzuteilen. Dieser Vorgang wird als Tokenisierung bezeichnet. Die einzelnen Wörter und Satzzeichen aufzuteilen, ist im Deutschen und im Englischen zu gewissen Teilen

³⁷Vgl. Jungermann (2007), S. 16.

³⁸Vgl. Allahyari et al. (2017), S. 3.

³⁹Vgl. Jurafsky/Martin (2018), S. 38 ff.

⁴⁰Vgl. Allahyari et al. (2017), S. 3 f.

⁴¹Vgl. Uysal/Gunal (2014), S. 104-112.

anhand der Leerzeichen möglich. In vielen anderen Sprachen, beispielweise Mandarin, ist dies schwieriger.⁴²

Beispiel

Eingabe: Franz geht in den Baumarkt.

Ausgabe: [Franz][geht][in][den][Baumarkt][.]

Für dieses einfache Exempel würde das Suchen nach Leerzeichen und Satzzeichen mit Regular Expressions bereits zur Worttrennung ausreichen. Nähere Informationen zu Regular Expressions finden sich in Jurafsky und Martin (2018)⁴³. Was aber, wenn das Beispiel ein Datum oder Abkürzungen enthält?

Eingabe: Am 18.11.2018 ging Franz in den örtl. Baumarkt.

Ausgabe: [Am][18][.][11][.][2018][ging][Franz][in][den][örtl.][Baumarkt][.]

Diese Ausgabe entspricht dem, was ein Computer identifizieren würde, falls eine einfache Regel, die Satzzeichen und Leerzeichen berücksichtigt, angewendet wird. Das gewünschte Ergebnis könnte hingegen sein:

Eingabe: Am 18.11.2018 ging Franz in den örtl. Baumarkt.

Ausgabe: [Am][18.11.2018][ging][Franz][in][den][örtl.][Baumarkt][.]

Bereits an einem so simplen Beispiel ist zu erkennen, dass auch dieser einfach erscheinende Arbeitsschritt zu einer anspruchsvollen Aufgabe werden kann. Zur Verdeutlichung Mandarin:⁴⁴

Deutsch: Franz geht in den Baumarkt.

Chinesisch(vereinfacht): 弗兰兹去了五金店。

Für Pakete, die einen Tokenizer zur Verfügung stellen, sind beispielsweise NLTK⁴⁵ das unter der Programmiersprache Python Verwendung findet, und für Programmiersprachen wie Java oder C# das Bündel Stanford CoreNLP⁴⁶ zu nennen.

Die Programmiersprache Python zeichnet sich dadurch aus, dass sie eine sehr kompakte Schreibweise zulässt. Aus diesem Grund werden Codebeispiele vorrangig mit Python 3.7 gezeigt. Das Aufteilen des Beispielsatzes in seine Bestandteile würde in Python 3.7 wie in Abbildung 3.1 verdeutlicht aussehen.

Python 3.7

```
1 import nltk
2 from nltk.tokenize import word_tokenize
3 satz = 'Franz geht in den Baumarkt.'
4 print(nltk.word_tokenize(satz))

out: ['Franz', 'geht', 'in', 'den', 'Baumarkt', '.']
```

Abbildung 3.1: Das Aufteilen eines Satzes in einzelne Wörter mit Python und NLTK. Quelle: Eigene Darstellung.

Es ist Aufgabe der Algorithmen, zu erkennen, welche Wörter in einem Token zusammengehören und welche nicht. Hierbei kann sich die Zusammengehörigkeit zweier Wörter je nach Anwendungsbereich und dem

⁴²Vgl. Carstensen et al. (2010), S. 264-271.

⁴³Vgl. Jurafsky/Martin (2018), S. 10-19.

⁴⁴Der Text wurde mit dem Google Übersetzer, von Deutsch nach Chinesisch(vereinfacht), übersetzt.

⁴⁵Loper/Bird, 2002

⁴⁶Manning et al. (2014), S. 55-60.

sie umgebenden Kontext unterscheiden. Vor allem in Internettextran wie in Social Media – und auch in Sprachen, die Wörter nicht durch Leerzeichen trennen – ist es schwierig, zu erkennen, welche Aufteilung den Sinn des Satzes am besten wiedergibt. Die wesentliche zu bewältigende Schwierigkeit ist das Auflösen von Mehrdeutigkeiten in jenen Texten, die in Token aufgeteilt werden sollen. Zwei unterschiedliche Ansätze zur Disambiguierung (Vereindeutigung) sind erstens die Verwendung von statistischen Methoden und zweitens die Verwendung einer Wissensdatenbank, in der Spezialfälle abgebildet sind.⁴⁷

Je nach Domäne sind die auftretenden Spezialfälle unterschiedlich. In technischen Spezifikationen könnte hierbei bereits erstmals an die praktische Anwendung des Tokenisierens einer Norm gedacht werden. Am Beispiel „EN ISO 9001:2018“ ist schnell zu erkennen, dass das alleinige Verwenden der Leerzeichen zum Aufteilen der Information nicht zwingend die vorteilhafteste Variante darstellt.

3.2.2 Filtern von Wörtern

Je nach Weiterverarbeitung kann es sinnvoll sein, sogenannte Stoppwörter aus dem Text maschinell zu entfernen. Das Entfernen von Stoppwörtern verringert die Dimensionalität des Begriffsraums. Die gebräuchlichsten Wörter in Textdokumenten sind Artikel, Präpositionen und Pronomen, die wenig zur Bedeutung der Dokumente beitragen. Es gibt vier gebräuchliche Methoden, um die zu filternden Wörter zu deklarieren. Als Erstes ist es naheliegend, eine Liste mit Wörtern zu erstellen, die entfernt werden sollen. Einige Stoppwörter aus dem Englischen sind in Abbildung 3.2 zu sehen. Im Falle von Artikeln ist dies im Deutschen schnell ersichtlich. Das Verwenden einer solchen Wortliste ist die klassische Methode, die in solch einem Fall zur Anwendung kommt. Zusätzlich zu dieser Methode kann die sogenannte Z-Methode zum Einsatz kommen. Diese auf dem Zipfschen Gesetz basierende Methode besagt, dass die am häufigsten vorkommenden Wörter entfernt werden können; Gleiches gilt für Wörter, die nur einmal vorkommen. Ebenfalls entfernt werden können Wörter, die über viele Dokumente hinweg betrachtet häufig vorkommen, also solche, die eine niedrige Inverse Document Frequency⁴⁸ aufweisen. Die dritte Methode ist die Mutual-Information-Methode. Hierbei handelt es sich um eine überwachte Lernmethode, die mittels Algorithmus den Informationsgehalt eines Wortes für das Dokument an sich ermittelt. Die vierte Methode ist das Term Based Random Sampling (TBRS), bei dem es sich um ein manuelles Verfahren handelt, das die relative Entropie, auch Kullback-Leibler-Divergenz⁴⁹ genannt, verwendet. Hierbei werden zufällige Zeichenfolgen aus Dokumenten hinsichtlich ihres Informationsgehalts bewertet und die am wenigsten informativen Wörter der Stoppwortliste hinzugefügt.⁵⁰

3.2.3 Ableiten der Stammform

Die Stammformreduktion, im Englischen Stemming genannt, ist eine Methode zur Vereinfachung von Wörtern hin zu deren Wortstamm.⁵¹ Bei der Stammformreduktion werden Wörter um Pre- und Suffixe reduziert. Der hierbei am häufigsten verwendete Stemming-Algorithmus ist der sogenannte Porter-Stemmer, bei dem es sich um einen populären Suffixstemmer für die englische Sprache handelt.⁵² Ein simples Beispiel für Stammformreduktion im Deutschen könnte wie folgt aussehen:

Eingabe: gewandert, schnellen, blaues

Ausgabe: wandert, schnell, blau

Beispiel

⁴⁷Vgl. Webster/Kit (1992), S. 1108 ff.

⁴⁸Vgl. Jivani et al. (2011), S. 1933.

⁴⁹Vgl. Do/Vetterli (2002), S. 146-158.

⁵⁰Vgl. Vijayarani/Illamathi/Nithya (2015), S. 10.

⁵¹Vgl. Lovins (1968), S. 22.

⁵²Vgl. Willett (2006), S. 219-223.

```

1 import nltk
2 from nltk.corpus import stopwords
3 print(stopwords.words('english'))

```

```

out: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your',
'your', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her',
'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs',
'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those',
'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if',
'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how',
'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no',
'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can',
'will', 'just', 'don', 'should', 'now']

```

Abbildung 3.2: Beispiele für Stoppwörter mit Python und NLTK. Quelle: Eigene Darstellung.

Im Englischen wird die mit diesem Verfahren erreichbare Reduktion in Richtung Stammform noch deutlicher ersichtlich. Suffixe wie -ing oder -s sind dort sehr häufig. Nicht durch Stemming entfernt werden dürfen hingegen Wortendungen wie das -n't bei don't, da dies die Bedeutung beeinflussen würde. Statistische Methoden zur Bewertung der Häufigkeit von Affixen in verschiedenen Sprachen würden diese ebenfalls für das Stemming suggerieren. Daraus folgt die Notwendigkeit für das Erstellen eines Stemming-Algorithmus, die Sprache an sich zu verstehen.

3.2.4 Lexikografische Reduktion auf Grundformen

Die lexikografische Reduktion von Wörtern auf eine Grundform wird als Lemmatisierung bezeichnet. Während beim Stemming mit Affixen der Wörter gearbeitet wird, wird bei der Lemmatisierung typischerweise ein Wörterbuch verwendet. Für die maschinelle Verarbeitung von Wörtern ist diese Vorverarbeitung von Vorteil.⁵³

Sowohl Lemmatisierung als auch Stemming dienen der Reduktion der verschiedenen Wörter gleicher Bedeutung in Texten, um diese im Allgemeinen einfacher weiterverarbeiten zu können. Gegebenenfalls kann sich dieses Vorgehen aber negativ auf die weitere Verarbeitung auswirken; dies kann beispielsweise dann der Fall sein, wenn sich Wörter mit unterschiedlicher Semantik auf dieselbe Grundform reduzieren. In Abbildung 3.3 soll der Unterschied der beiden Strategien verdeutlicht werden.

3.3 Bewertungsgrundlagen

Die empirische Bewertung spielt eine zentrale Rolle bei der Einschätzung der Leistung von Natural-Language-Processing (NLP)-, Information-Retrieval (IR)- oder Information-Extraction (IE)-Systemen. Die erreichte Performance wird typischerweise auf Basis synthetischer, eindimensionaler Indikatoren geschätzt. Die hierbei verwendeten Indikatoren sind die erreichte Accuracy, Precision, Recall und F-Score.⁵⁴

⁵³Vgl. Dietrich (2010), S. 6 ff.

⁵⁴Vgl. Goutte/Gaussier (2005), S. 345 ff.

Python 3.7

```

1 from nltk.stem import PorterStemmer
2 from nltk.tokenize import sent_tokenize, word_tokenize
3 from nltk.stem import WordNetLemmatizer
4
5 lt = WordNetLemmatizer()
6 ps = PorterStemmer()
7
8 print(ps.stem('having') + ', ' + lt.lemmatize('having'))

out: [hav, have]
```

Abbildung 3.3: Hier ist der Unterschied zwischen Affix- und Wörterbuchbasierter Grundformbildung zu sehen. Quelle: Eigene Darstellung.

		Label	
		<i>positiv</i>	<i>negativ</i>
Beobachtung	<i>positiv</i>	TP	FN
	<i>negativ</i>	FP	TN

Tabelle 3.1: Konfusionsmatrix über Label der Daten und der tatsächlichen Beobachtung. Quelle: Eigene Darstellung.

Um diese Bewertungsmaße zu verstehen, wird nachfolgend mithilfe von Tabelle 3.1 ein Beispiel betrachtet. Angenommen ein Computer versieht jedes einzelne Wort in einem kleingeschriebenen Text mit einer Annotation (Label), die aussagt, ob das Wort großgeschrieben werden muss. Zu jedem Wort, das seiner Einschätzung nach großgeschrieben gehört, schreibt er ein kleines p für positiv und zu jedem Wort, bei dem er glaubt, es gehöre kleingeschrieben, ein kleines n für negativ. Um die verrichtete Arbeit später zu bewerten, kommt typischerweise ein Mensch zum Einsatz. Dieser notiert für jedes richtig gelabelte positiv ein TP für True-Positiv und für jedes zu Unrecht als positiv gelabelte Wort ein FP für False-Positiv. Für die negativen Label macht er dasselbe mit den Notationen TN True-Negativ und FN False-Negativ. Nachdem dies geschehen ist, kann die Frage nach der Accuracy gestellt werden. Die Antwort auf diese Frage ist

$$a = \frac{TP + TN}{TP + TN + FP + FN} \cdot 100\%, \tag{3.1}$$

hierbei ist $a\%$ die Accuracy, mit der der Computer vorgegangen ist. Wird die Frage gestellt, mit welcher Wahrscheinlichkeit der Computer ein großgeschriebenes Wort erkennt, bezieht sich diese auf die Abschätzung der Wahrscheinlichkeit

$$p = \frac{TP}{TP + FP} \cdot 100\%, \tag{3.2}$$

die sich ebenfalls in % ergibt. Die Genauigkeit einer positiven Annotation ist somit die Precision. Diese sagt aber nichts darüber aus, ob der Computer viele der Wörter überhaupt gelabelt hat. Um die Frage nach der Vollständigkeit der als positiv gelabelten Wörter zu beantworten, wird die Formel für den Recall $r\%$ benötigt:

$$r = \frac{TP}{TP + FN} \cdot 100\% \tag{3.3}$$

Das mit χ gewichtete harmonische Mittel aus Precision und Recall führt schließlich zu

$$F_\chi = (1 + \chi^2) \frac{pr}{r + \chi^2 p} = \frac{(1 + \chi^2)TP}{(1 + \chi^2)TP + \chi^2 FN + FP}. \tag{3.4}$$

Hierbei handelt es sich um die gewichtete F-Score $F_\chi/\%$. Die Gewichtung χ wird oft mit 1 angenommen. Somit ergibt sich die gebräuchliche F1-Score; diese lautet:

$$F_1 = 2 \frac{pr}{r+p} = \frac{2TP}{2TP + FN + FP} \quad (3.5)$$

Die F-Score ist hierbei ausgeglichen und kann an die Bedürfnisse angepasst werden. Er bevorzugt Precision, wenn $\chi > 1$, und Recall wenn $\chi < 1$.⁵⁵

3.4 Erkennen von Eigennamen

Der Prozess der Information Extraction (IE) wandelt die in Text eingebetteten, unstrukturierten Informationen in strukturierte Daten um. Damit wird eine weitere Verarbeitung ermöglicht. Der erste Schritt zur Extraktion von Informationen ist das Erkennen von Eigennamen im Text. Im Kontext des Natural Language Processing (NLP) wird dieser Vorgang Named Entity Recognition (NER) genannt. Dabei ist es die Aufgabe der NER, den Eigennamen zu erkennen und diesem einen Typ zuzuweisen. Beispiele für Typen von Eigennamen wären Person, Ort oder Hunderasse. Eigennamen sind im Allgemeinen Substantive. Welche der Substantive als Eigennamen zu erkennen sind, ist anwendungsspezifisch. Menschen, Orte und Organisationen sind häufig.⁵⁶

Im hier behandelten Anwendungsfeld wären Entitäten wie Normen, Herstellernamen und Komponentennamen denkbare Typen von Entitäten. Die NER ist wahrscheinlich die grundlegendste Aufgabe der Information Extraction (IE). Die darauf folgende Extraktion komplexerer Strukturen, wie Beziehungen und Ereignisse, hängt von der genauen Erkennung der Entitäten als Vorverarbeitung ab. Abseits der IE hat die NER weitere Anwendungsfelder. So enthält zum Beispiel beim Beantworten von Quizfragen die gegebene Antwort ebenfalls Entitäten, die erkannt und mit einem Soll verglichen werden.⁵⁷

Die Aufgabe der NER wurde erstmals im Jahr 1995 bei der sechsten Message Understanding Conference (MUC-6) als ein Teilschritt der IE vorgestellt. Seitdem wird der NER große Aufmerksamkeit der Wissenschaft, in mehreren Evaluationsprogrammen zuteil, wie beispielsweise dem Automatic-Content-Extraction (ACE)-Programm, den Aufgabenstellungen der Conference on Natural Language Learning (CoNLL) oder der Critical Assessment of Information Extraction Systems in Biology (BioCreAtIvE) Challenge. Frühe Ansätze der NER waren von Hand erstellte Regelsätze. Weil diese einen hohen Aufwand an manueller Arbeit und Expertise des Bearbeiters erforderten, wurde versucht, im Folgenden diesen Ansatz durch statistische Modelle zu ersetzen. Die hierfür verwendeten Ansätze umfassen beispielsweise Hidden-Markov-Modelle, Maximum-Entropie-Markov-Modelle, Support-Vector-Maschinen oder Conditional Random Fields.⁵⁸

Ebenfalls ist an dieser Stelle zu erwähnen, dass NER alleine nicht hinreichend ist, um die Eigennamen in einem Text zuzuordnen. Beispielsweise ist das Wort Autobahn je nach umgebendem Kontext entweder eine Straße, auf der Autos fahren, oder ein Lied der Band Kraftwerk. Der Vorgang, bei dem solche Ambiguitäten aufgelöst werden, nennt sich Named Entity Disambiguation (NED). Dieser Vorgang kann mithilfe lexikalisch semantischer Wissensdatenbanken (Gazetteers), wie beispielsweise DBpedia⁵⁹ oder YAGO⁶⁰, bearbeitet werden.⁶¹

⁵⁵Vgl. Sokolova/Japkowicz/Szpakowicz (2006), S. 1015-1021.

⁵⁶Vgl. Jurafsky/Martin (2018), S. 327 f.

⁵⁷Vgl. X. Li/Roth (2002), S. 1-7.

⁵⁸Vgl. Jiang (2012), S. 16 f.

⁵⁹Vgl. Auer et al. (2007), S. 722-735.

⁶⁰Vgl. Kasneci/Suchanek/Weikum (2006), S. 1-27.

⁶¹Vgl. Hoffart et al. (2011), S. 782.

3.5 Stochastische Modellgraphen

Modelle sind Abbildungen der Realität, die versuchen, diese für eine Anwendung hinreichend genau wiederzugeben. Bei der Klassifizierung von Text hat man es mit Wörtern oder Wort-Tupel zu tun, denen entsprechende Label zuzuweisen sind. Um diese Aufgabe zu erledigen, eignen sich verschiedene Modelle, die

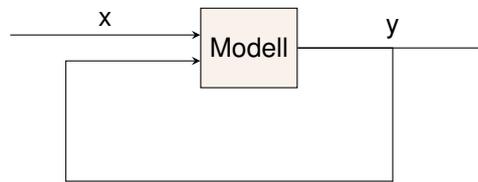


Abbildung 3.4: Modell das einen Eingangsvektor einem Ausgangsvektor zuordnet mit Rückführung. Quelle: Eigene Darstellung.

in diesem und den folgenden Kapiteln behandelt werden. Es soll auf die Modelle selbst und auf ihre Eigenheiten eingegangen werden. Darüber hinaus gibt es auch Modelle, die ihren Ausgang, wie in Abbildung 3.4 dargestellt, in die Vorhersage des folgenden Ausgangswertes mit einbeziehen; andere beziehen keinen, mehrere oder sogar alle vorhergehenden Zustände in ihre Vorhersagen mit ein. Ein Beispiel für ein Modell, das einem Eingangsvektor einen Ausgangsvektor zuordnet, ist das Hidden-Markov-Modell (HMM). Dieses Modell soll in Abschnitt 3.5.2 betrachtet werden, um weiter darauf aufzubauen. Als Erstes sollen stochastische Modellgraphen betrachtet werden. Bei diesen handelt es sich um endliche Zustandsautomaten, die einer Beobachtung einen Zustand zuordnen. Die Übergänge zwischen den Zuständen werden dabei, mit verschiedenen Methoden, Wahrscheinlichkeiten zugewiesen. Dies erfolgt teils generativ, teils diskriminative. Neben der rein mathematischen Darstellung als Formel soll hierbei die visuelle Darstellung als Graph angewendet werden, wodurch die Modelle einfacher fassbar werden.

3.5.1 Bayes Klassifikator

Grundsätzlich gibt es zwei Gruppen von Klassifizierungsalgorithmen: Generative und diskriminative Modelle. Generative Modelle berechnen die Wahrscheinlichkeit für das Auftreten eines bestimmten Merkmals anhand der Auswertung der Wahrscheinlichkeiten $P(\mathbf{x}, \mathbf{y})$. Der Vektor \mathbf{x} repräsentiert hierbei die Daten und Vektor \mathbf{y} die zugewiesenen Labels. Anhand der Bayes-Regel wird, aus den Wahrscheinlichkeiten $P(\mathbf{x}, \mathbf{y})$, eine bedingte Zuordnung der Form $P(\mathbf{y}|\mathbf{x})$ gemacht. Im Gegensatz dazu machen diskriminative Modelle diese Zuordnung $P(\mathbf{y}|\mathbf{x})$ direkt. Beide Vorgehensweisen haben sowohl Vor- als auch Nachteile und sind für bestimmte Anwendungsgebiete besser oder schlechter geeignet.⁶²

Ein Beispiel für generative Modelle sind Hidden-Markov-Modelle und ein Beispiel für diskriminative Modelle ist das logistische Regressions-Modell. Der Naive-Bayes-(NB)-Klassifikator kann verwendet werden um ein generatives Modell in ein diskriptives Modell umzurechnen, dieser lautet:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{y})P(\mathbf{y})}{P(\mathbf{x})}; \quad (3.6)$$

daraus folgend kann das wahrscheinlichste Label mittels

$$\hat{y} = \underset{\mathbf{y}}{\operatorname{argmax}} P(\mathbf{y}|\mathbf{x}) \quad (3.7)$$

den Daten zugeordnet werden. Hierbei steht \hat{y} für das Label mit der höchsten zugehörigen Wahrscheinlichkeit. Wird in Gleichung 3.6 der Nenner $P(\mathbf{x})$ betrachtet, zeigt sich, dass dieser für alle $P(\mathbf{y}|\mathbf{x})$ gleich ist. Somit kann er weggelassen werden und er ergibt sich

⁶²Vgl. Raina et al. (2004), S. 545 f.

$$P(\mathbf{y}|\mathbf{x}) \propto P(\mathbf{x}|\mathbf{y})P(\mathbf{y}) \tag{3.8}$$

als die zu optimierende Gleichung. Unter der Naive-Bayes (NB)-Annahme, dass die einzelnen Daten bedingt unabhängig voneinander sind, kann daraus das NB-Modell wie folgt gebildet werden:

$$P(\mathbf{y}|\mathbf{x}) = P(\mathbf{y}) \prod_i^M P(x_i|\mathbf{y}) \tag{3.9}$$

wobei M die Anzahl der Daten $x \in$ Vektor \mathbf{x} bezeichnet.

3.5.2 Markov Modelle

3.5.2.1 Markov-Kette

Um Markov-Modelle und im weiteren HMM zu verstehen, bietet es sich an, zuerst eine Markov-Kette zu betrachten und an ihr Markov Prozesse zu beschreiben. Eine Markov-Kette für die drei Zustände s_1 , s_2 und s_3 ist in Abbildung 3.5 dargestellt. Die einzelnen Zustände s_n sind Teil einer Menge $S = \{s_1, s_2, s_3\}$. Der Zustand s_2 entspricht dem Anfangszustand, den der Prozess einnimmt, und wird deswegen mit *Initial* gekennzeichnet.

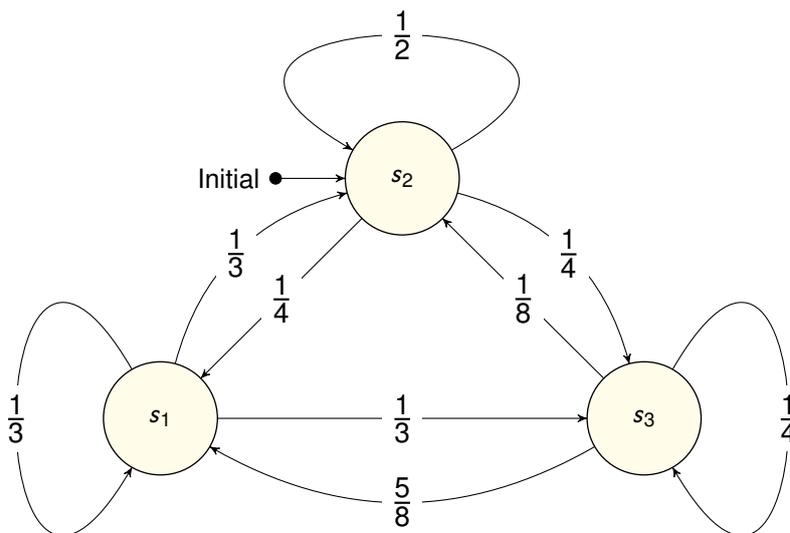


Abbildung 3.5: Markov-Kette als Prozessschaubild dargestellt. Quelle: Eigene Darstellung.

Die Menge S könnte einer Zuordnung – ähnlich einem Dictionary in der Programmiersprache Python – entsprechen und wie folgt aussehen: $\{s_1 = \text{Tor offen}, s_2 = \text{Tor geschlossen}, s_3 = \text{in Bewegung}\}$. Hierbei betrachtet man ein Tor zu einem Zeitpunkt $t(n)$ mit $n \in \{0, 1, 2, \dots, N\}$ und sieht sich seinen Zustand an, anhand des Zustands des Tors zum Zeitpunkt $t(n)$ soll nun auf den Zustand des Tors am darauf folgenden Zeitpunkt $t(n + 1)$ geschlossen werden. Ist zum Beispiel der Zustand des Tors zum Zeitpunkt $t(1) = s_1$, das Tor also offen, dann bedeutet das, dass zum Zeitpunkt $t(2)$ das Tor mit einer Wahrscheinlichkeit von $\frac{1}{3}$ noch immer offen ist, ebenfalls mit einer Wahrscheinlichkeit von $\frac{1}{3}$ geschlossen ist oder sich mit der verbleibenden Wahrscheinlichkeit von $\frac{1}{3}$ gerade schließt. Damit muss sich für die von einem Zustand ausgehenden Wahrscheinlichkeiten zu anderen Zuständen, wie in Abbildung 3.5 zu sehen ist, als Summe dieser immer 1 ergeben. Das Wechseln von einem Zustand zu einem anderen wird als Transition bezeichnet.⁶³

⁶³Vgl. Jungermann (2006), S. 17 ff.

Eine weitere Darstellung neben dem Prozessschaubild ist die Darstellung als Transitionsmatrix. Die in diesem Modell gezeigten Markov-Prozesse sind 1. Ordnung, weil jeder Folgezustand lediglich von einem Zustand zuvor abhängt.

$$\mathbf{A} = \begin{matrix} & s_1|t(n+1) & s_2|t(n+1) & s_3|t(n+1) \\ \begin{matrix} s_1|t(n) \\ s_2|t(n) \\ s_3|t(n) \end{matrix} & \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{5}{8} & \frac{1}{8} & \frac{1}{4} \end{pmatrix} & & \end{matrix} \quad (3.10)$$

Auch hier kann die Transitionswahrscheinlichkeit $P(s_j|s_i)/\%$ von einem Zustand zum Folgenden abgelesen werden. Die Werte in den Zeilen müssen sich zu 1 summieren. Zustände, die sich so verhalten, besitzen die Markov-Eigenschaft, die besagt, dass der Prozess selbst ohne Gedächtnis angenommen wird und das weitere Verhalten des Prozesses lediglich von seinem aktuellen Zustand und den Übergangswahrscheinlichkeiten abhängig ist. Die in dieser Darstellung fehlende Information ist, welcher Anfangswert zum ersten Zeitpunkt auftritt? Aus diesem Grund wird ebenfalls ein Startvektor benötigt, der die Wahrscheinlichkeit für den Initialstatus enthält. Dieser lautet im vorliegenden Beispiel $\pi = (0, 1, 0)$. Damit kann ebenfalls Zustand s_2 zum Zeitpunkt $t(0)$ durch die Multiplikation $\mathbf{A} \times \pi$ eindeutig zugeordnet werden. Prozesse, die ihre Zustände den Folgenden auf diese Art und Weise zuordnen, werden Markov-Prozesse genannt, und eine Markov-Kette ist ein Beispiel für ein Markov-Modell.⁶⁴

3.5.2.2 Hidden-Markov-Modell

Wie der Name vermuten lässt, ist bei einem Hidden-Markov-Modell (HMM) etwas versteckt. Als Beispiel dient an dieser Stelle das Part of Speech (POS) Tagging eines Satzes. Nachdem das erste Wort jener Wortart zugeteilt wurde, der es am wahrscheinlichsten entspricht, besitzt der Algorithmus die Information über den POS-Tag des Wortes. Beim Zuteilen, einer Wortart an das nächste Wort hat der Algorithmus bereits sowohl das zuzuordnende Wort als auch den POS-Tag des vorherigen Wortes als Information zur Verfügung; lediglich der POS-Tag des aktuellen Wortes ist für den Algorithmus unsichtbar und somit versteckt. Diese versteckten Merkmale stellen für sich genommen untereinander Markov-Ketten dar.

Diese Art der Problemstellung tritt häufig auf, unter anderem im Bereich der Bilderkennung, bei der Analyse von Bildfolgen, oder im Bereichen der Biologie bei der Analyse von Erbgut. Um festzulegen, welche versteckten Prozesse auftreten können, werden diese in einer Menge S , in Analogie zum vorhergehenden Kapitel, mit $\{ s_1 = \text{Tor offen}, s_2 = \text{Tor geschlossen}, s_3 = \text{in Bewegung} \}$ definiert. Dabei ist zu beachten, dass $N = |S| < \infty$ ist und jedes $s_n \in S$ nur einmal in S vorkommt. Damit ist jeder Zustand einzigartig und die Zustände müssen alle die Markov-Eigenschaft besitzen. Eine so festgelegte Menge wird als Zustandsalphabet bezeichnet.

In Abbildung 3.6 wird dargestellt, wie eine Abfolge der Zustände aus der Menge S einen Vektor von Zuständen $\mathbf{o}(t)$ generiert. Diese Beobachtungen $o_m \in O$ kann man sich zum Beispiel folgendermaßen vorstellen: Eine Person befindet sich in einem geschlossenen Raum, mit einem Tor, das nicht zu sehen aber zu hören ist. Diese Person hört nun folgende Tonfolge: $\{ o_1 = \text{Quietschen}, o_2 = \text{Quietschen}, o_3 = \text{Knall} \}$. Aus dieser Beobachtung könnte die Person die Annahme treffen, dass das Tor soeben geschlossen worden ist und dies mit hoher Wahrscheinlichkeit auch stimmt. Die angenommene Sequenz lautet somit: $\{ y_1 = \text{weder noch}, y_2 = \text{weder noch}, y_3 = \text{Tor geschlossen} \}$. Dabei wurden, über das Beobachtete, den unsichtbaren Zuständen eine Sequenz von wahrscheinlichen Annahmen zugeordnet. Eine Zuordnung wird als Label bezeichnet und die Menge, aus der diese Label stammen, wird Y mit seinen Elementen y genannt. Dieser Sachverhalt ist in Abbildung 3.6 dargestellt. Diese Darstellung wird aber im Allgemeinen nicht verwendet, sondern es

⁶⁴Vgl. Brémaud (2017), S. 117 ff.

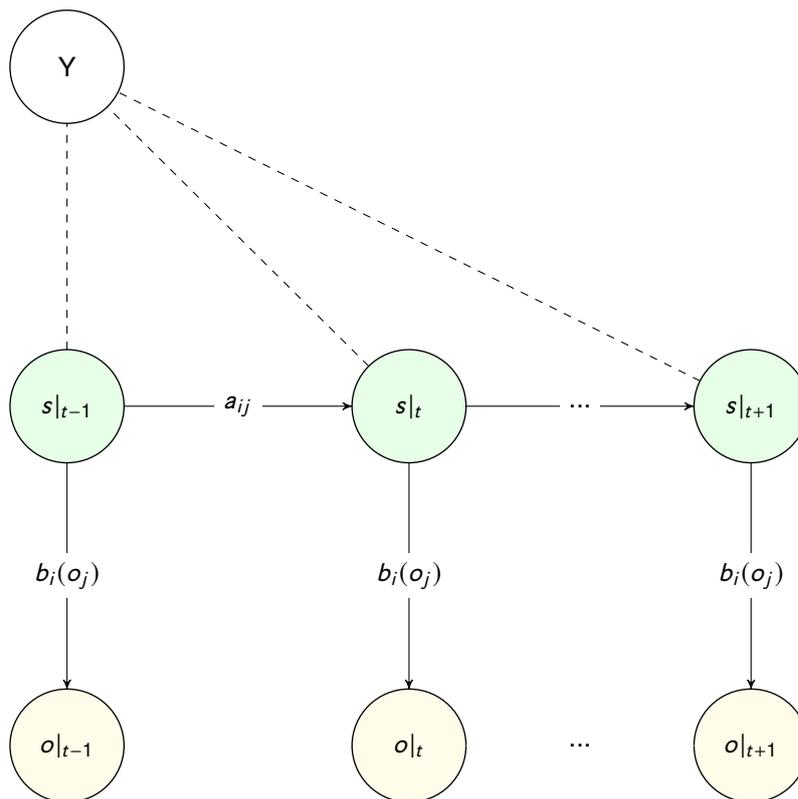


Abbildung 3.6: Darstellung eines HMM mit den unsichtbaren Zuständen mit grünem Hintergrund und den beobachtbaren Zuständen auf gelbem Hintergrund. Quelle: Eigene Darstellung.

werden nur die sichtbaren Beobachtungen und Zuordnungen gezeigt. Damit ergibt sich das Modell wie in Abbildung 3.7 gezeigt. In diesem Beispiel ist zu beachten, dass die Zustände in S alle einzigartig waren, die Beobachtungen und die zugeordneten Labels in ihren Sequenzen aber auch mehrmals oder gar nicht vorkommen können. Die Anzahl der beobachteten und jene der zugeordneten Zustände ist dieselbe, jedoch müssen diese nicht mit der Anzahl der möglichen Zustände in S übereinstimmen. Um dieses HMM mathematisch zu beschreiben, wird folgende Notation verwendet: Die aus Gleichung 3.10 bekannte Transitionsmatrix wird mit $\mathbf{A} \in \mathbb{R}^{n \times n}$ bezeichnet; die enthaltenen Elemente a_{ij} geben die Wahrscheinlichkeiten für die Änderung eines Zustandes s_i in einen Zustand s_j wieder. Die Menge dieser möglichen Zustände ist die Menge $S = \{s_1, s_2, \dots, s_n\}$. Das Beobachtungsalphabet soll $O = \{o_1, o_2, \dots, o_m\}$ sein und die Emissionsmatrix, die die Wahrscheinlichkeit beinhaltet für einen Zustand s_i die Beobachtung o_j zu machen, wird mit $\mathbf{B} \in \mathbb{R}^{n \times m}$ beschrieben. Ebenfalls aus Abschnitt 3.5.2.1 bekannt ist die Anfangsverteilung $\pi \in \mathbb{R}^n$. Sie enthält die Wahrscheinlichkeiten, dass ein s_i initial auftritt. Damit ist ein HMM als $\lambda = (O, S, \mathbf{A}, \mathbf{B}, \pi)$ beschreibbar.⁶⁵ Für eine Sequenz der Länge M ergibt sich damit

$$P_M(\mathbf{o}, \mathbf{y}) = \prod_{j=1}^M P(o_j | y_j) P(y_j | y_i); \tag{3.11}$$

mit \mathbf{o} und \mathbf{y} als Vektoren welche Elemente der Mengen S und Y in nun geordneter Reihenfolge enthalten. Dabei wird ersichtlich, dass die Wahrscheinlichkeit $P(y_j | y_i)$ der entsprechenden Übergangswahrscheinlichkeit a_{ij} in der Transitionsmatrix \mathbf{A} . $P(o_j | y_j)$ entspricht $b_{j(o)}$, dem Eintrag an dieser Stelle in der Emissionsmatrix \mathbf{B} . Damit ist die Wahrscheinlichkeit für das Auftreten von $P(\mathbf{o}, \mathbf{y})$ beschrieben. Es ist anzumerken, dass $P(\mathbf{o}, \mathbf{y})$ Einzelbeobachtungen beschreibt, die sich gegenseitig nicht beeinflussen, also Markov-Prozesse welche die Markov-Eigenschaft besitzen.

⁶⁵Vgl. Rabiner/Juang (1986), S. 4-16.

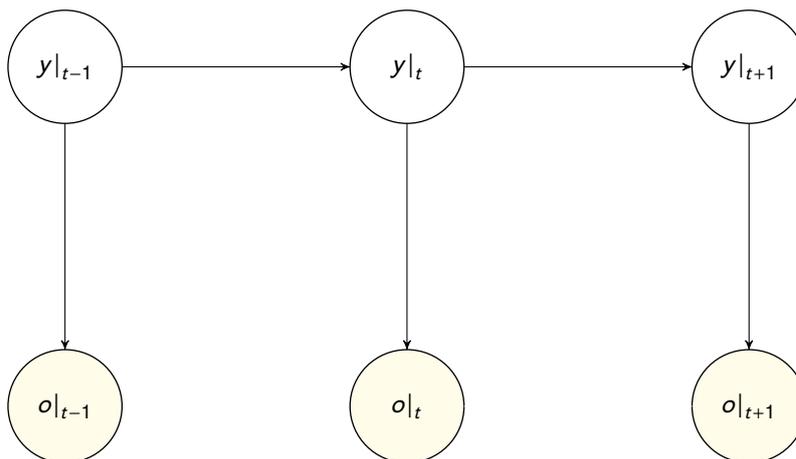


Abbildung 3.7: Modell eines HMM mit den Labeln auf weissem Hintergrund und den beobachtbaren Zuständen auf gelbem Hintergrund. Quelle: Eigene Darstellung.

3.5.2.3 Klassische Problemstellungen für HMMs

Die in Kapitel 3.5.2.2 beschriebenen Hidden-Markov-Modelle werden zum Lösen der folgenden drei Problemstellungen verwendet:⁶⁶

- Evaluation: Bei einem gegebenen HMM mit den Parametern λ ist die Wahrscheinlichkeit für das Auftreten einer bestimmten Beobachtungssequenz $\mathbf{o} = \{o_1, \dots, o_T\}$ zu berechnen.

$$P(\mathbf{o}|\lambda) = \sum_{\mathbf{s}} P(\mathbf{o}|\mathbf{s})P(\mathbf{s}), \tag{3.12}$$

wobei $\mathbf{s} = \{s_1, \dots, s_T\}$ die Zustandssequenz bezeichnet.

- Dekodierung: Bei einem gegebenen HMM mit den Parametern λ und einer gegebenen Beobachtungssequenz \mathbf{o} ist jene Zustandssequenz $\mathbf{s} = \{s_1, \dots, s_T\}$ zu finden, die am wahrscheinlichsten aufgetreten ist.

$$\operatorname{argmax}_{\mathbf{s}} P(\mathbf{o}|\mathbf{s}) \tag{3.13}$$

- Lernen: Bei gegebener Struktur (\mathbf{s}, \mathbf{o}) eines HMM mit den Parametern λ sind jene Modellparameter zu finden, die die Realität am besten beschreiben.

$$\operatorname{argmax}_{\lambda} P(\mathbf{o}|\lambda) = \sum_{\mathbf{s}} P(\mathbf{o}|\mathbf{s}, \lambda)P(\mathbf{s}) \tag{3.14}$$

Diese Probleme werden durch Algorithmen der dynamischen Programmierung bewältigt. Beispiele hierfür sind Forward-Backward-Algorithmen, Baum-Welch-Algorithmen und der Viterbi-Algorithmus. Beim maschinellen Lernen besteht eine Aufgabe darin, die Modelle so zu gestalten, dass diese danach zur Vorhersage genutzt werden können. Somit handelt es sich hierbei oft um die Aufgabenstellung, die Modellparameter maschinell zu finden. Je nachdem, ob das Modell seine Parameter mit menschlichem Zutun oder ohne dieses lernt, wird entweder von Supervised- oder Unsupervised-Learning gesprochen.

3.5.2.4 Maximum-Entropy-Markov-Modell

Bereits mit den in Kapitel 3.5.2.2 vorgestellten Hidden-Markov-Modellen lassen sich viele Klassifizierungsprobleme, wie POS Tagging oder IE erfolgreich lösen. Aber in einigen Anwendungsfällen wäre es vorteilhaft,

⁶⁶Vgl. McCallum/Freitag/Pereira (2000), S. 591-598.

ein Label y einer Beobachtung o nicht nur aufgrund einer einzelnen Beobachtung, sondern basierend auf einer Reihe an Beobachtungen zuzuweisen. Ein weiterer Nachteil von HMM ist, dass beim Erstellen und Trainieren des Modells die Beobachtungen aus den Zuständen generiert werden. Dies ist auch in Abbildung 3.7 zu sehen, in der ein Maximum-Entropy-Markov-Modell (MEMM) dargestellt wird; der Unterschied zum HMM besteht in der Orientierung der Pfeilspitzen. Meist aber ist es die Aufgabe, aufgrund einer Beobachtung Rückschlüsse auf einen Zustand zu ziehen. Werden die Pfeilrichtungen in Abbildung 3.8 betrachtet, ist dieser Unterschied erkennbar. Dieselbe Aussage kann auch aus einem HMM generiert werden, allerdings nur indirekt. Diese beiden beschriebenen Unzulänglichkeiten, die bei Hidden-Markov-Modellen auftreten können sollen mit dem MEMM beseitigt werden. Hierfür soll der in Kapitel 3.5.3 gezeigte Maximum-Entropy-Ansatz verwendet werden.⁶⁷

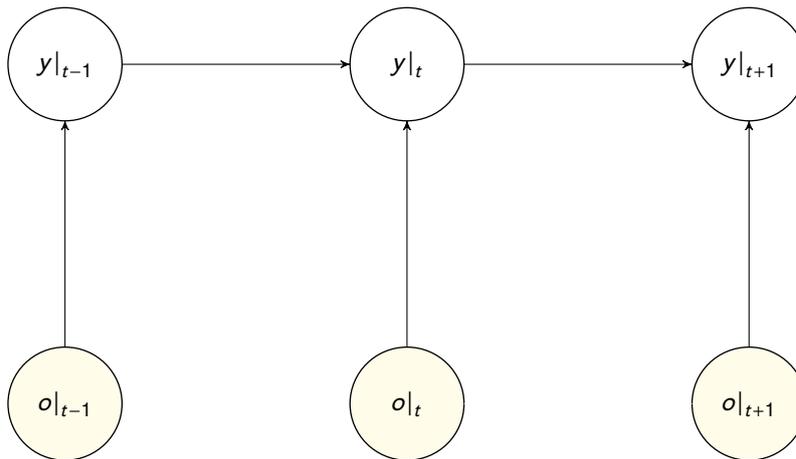


Abbildung 3.8: Modell eines MEMM mit den Labels auf weißem Hintergrund und den beobachtbaren Zuständen auf gelbem Hintergrund. Quelle: Eigene Darstellung.

Die zu modellierende Wahrscheinlichkeit kann für ein MEMM geschrieben werden als:

$$P_M(\mathbf{y}|\mathbf{o}) = \prod_{t=1}^T P(y_t|y_1, \dots, y_{t-1}, \mathbf{o}). \quad (3.15)$$

Dabei folgt die Notation jener in Gleichung 3.11, mit $|\mathbf{O}| \wedge |\mathbf{Y}| = T$. Eine Schwierigkeit, die bei MEMM bei der Vorhersage auftreten kann, ist das Label-Bias-Problem. Dieses kann auftreten, weil sich die Übergangswahrscheinlichkeiten für jeden Zustand auf Eins summieren. Zustände mit wenigen Folgezuständen können das Ergebnis verfälschen, Zustände mit nur einem einzigen Ausgang ignorieren ihre Beobachtungen vollständig.⁶⁸

3.5.3 Logistische Regression

Die in Kapitel 3.5.2.2 vorgestellten Hidden-Markov-Modelle und der in Kapitel 3.5.1 besprochene Bayes-Klassifikator betrachten die auftretenden Merkmale als strikt voneinander unabhängig. Es wird jeweils die Wahrscheinlichkeit für das Auftreten einer Kombination aus x und y den Berechnungen zugrunde gelegt. Dabei wird angenommen, dass das Auftreten eines Merkmales keinen Einfluss auf das Auftreten weiterer Merkmale nimmt und somit für sich steht. Wenn aber als Beispiel ein deutscher Satz betrachtet wird, so ist die Wahrscheinlichkeit, dass es sich bei einem Wort um ein Nomen handelt, offensichtlich stark abhängig davon, ob es mit einem Großbuchstaben beginnt oder nicht. Ein weiteres Merkmal könnte sein, dass vor

⁶⁷Vgl. McCallum/Freitag/Pereira (2000), S. 591 ff.

⁶⁸Vgl. Lafferty/McCallum/Pereira (2001), S. 2 f.

dem aktuellen Wort ein Wort als Artikel klassifiziert wurde. Diesen Abhängigkeiten der Merkmale untereinander kann durch den Einsatz von zum Beispiel Conditional Random Fields (CRF) Rechnung getragen werden. Eine Grundlage für das Verstehen von CRF ist die logistische Regression, weshalb diese hier erklärt werden soll.

Hierzu ein Beispiel, das ausführlich in Krafft (1996)⁶⁹ gezeigt wurde und hier in verkürzter Form, leicht abgeändert, wiedergegeben werden soll:

Angenommen die zu erklärende Variable $y \in Y$ nimmt nur die Zustände WAHR oder FALSCH an. Eine Fragestellung hierzu würde beispielsweise lauten: ‚Wurde ein Signalausschlag am Oszilloskop zu einem gegebenen Zeitpunkt bemerkt?‘. Daraufhin nimmt die Variable folgende Werte an:

$$y = \begin{cases} 1 & \text{ein Signal wurde bemerkt} \\ 0 & \text{sonst.} \end{cases} \quad (3.16)$$

Wenn eine solche Variable analysiert werden soll, kann als einfachstes Verfahren das Kleinste-Quadrate-Schätzverfahren verwendet werden, das an dieser Stelle aber nicht näher betrachtet werden soll. In diesem Beispiel wird angenommen, dass eine latente Variable y^* existiert, für die gilt:

$$y = \begin{cases} 1 & \text{wenn } (y^* > 0) \\ 0 & \text{sonst.} \end{cases} \quad (3.17)$$

Von dieser Annahme ausgehend wird angenommen, dass ein Regressionsmodell

$$y_i^* = \theta_0 + \sum_{k=1}^K \theta_k x_{ik} + u_i, \quad (3.18)$$

gegeben ist, in dem $y_i^*(i \in I)$ die latente (nicht beobachtete) Variable darstellt. θ_0 ist ein konstanter Wert und θ_k ist der Koeffizient zur unabhängigen Variablen $x_{ik}(i \in I)(k \in K)$. u_i bezeichnet den Störterm. I ist die Menge aller Objekte und K ist die Menge der latenten Variablen. Anders als im Verfahren der Kleinsten-Quadrate wird somit davon ausgegangen, dass nicht sichtbare Variable $y^* \exists$ für die die Variablen y_i beobachtet werden. Dabei wird im Beispiel mit y_i die Beobachtung eines Signales und mit y^* die Wahrscheinlichkeit der Beobachtung eines Signales ausgedrückt. Aus Gleichung 3.17 und Gleichung 3.18 kann damit abgeleitet werden:

$$P_i = P(y_i = 1) = P[u_i > -(\theta_0 + \sum_{k=1}^K \theta_k x_{ik})] = 1 - F[(\theta_0 + \sum_{k=1}^K \theta_k x_{ik})] \quad (3.19)$$

Hierbei steht F für die kumulierte Verteilung von u . Ist diese symmetrisch, kann vereinfacht werden zu:

$$P_i = F(\theta_0 + \sum_{k=1}^K \theta_k x_{ik}) \quad (3.20)$$

Weil die Werte y_i einen Prozess darstellen, der die zwei Werte, die in Abhängigkeit der Beobachtungen x_{ik} variieren, annehmen kann, kann folgende Likelihood-Funktion L aufgestellt werden:

$$L = \prod_{y_i=1} P_i \prod_{y_i=0} 1 - P_i \quad (3.21)$$

Beim Optimieren von L wird bei der Maximum-Likelihood-Methode eine schrittweise Variation der Parameter vorgenommen. Dies geschieht so, dass L maximal wird. Je nachdem wie u angenommen wird, ergeben sich

⁶⁹Vgl. Krafft (1996), S. 2ff.

unterschiedliche Verläufe von F . Wird von einer logistischen Verteilung ausgegangen (eine Alternative wäre zum Beispiel eine Normalverteilung), dann ergibt sich das logistische Modell (Logit-Modell) als

$$F(Z_i) = \frac{\exp(Z_i)}{1 + \exp(Z_i)} \quad (3.22)$$

das sich vereinfachen lässt zu

$$F(Z_i) = \frac{1}{1 + \exp(-Z_i)} \quad (3.23)$$

Der Ausdruck Z_i steht dabei für den linearen Prädiktor des logistischen Modells für das i -te Objekt, also:

$$Z_i = \theta_0 + \sum_{k=1}^K \theta_k x_{ik} + u_i = \log \frac{F(Z_i)}{1 - F(Z_i)} \quad (3.24)$$

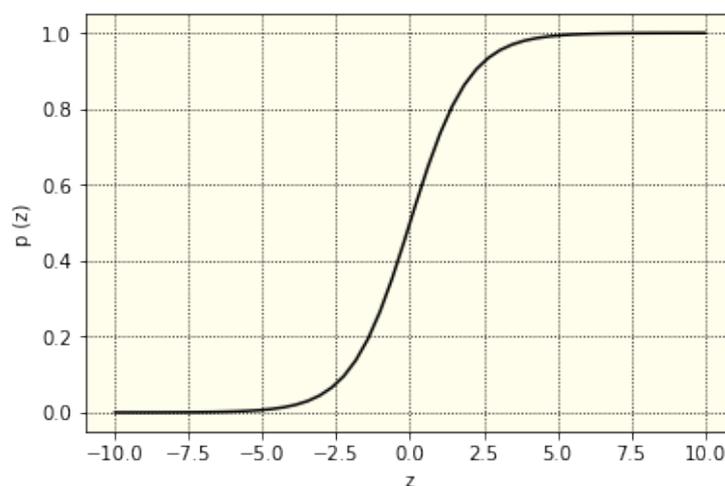


Abbildung 3.9: Qualitativer Plot einer Sigmoid-Funktion p über die Werte z , im Deutschen auch mit Schwanenhalsfunktion oder S-Kurve benannt. Quelle: Eigene Darstellung.

Daraus ergibt sich ein S-Förmiger Verlauf der abhängigen Variablen. Diese Form wird, wie sie in Abbildung 3.9 dargestellt ist, auch als Sigmoid bezeichnet. Sie bietet den Vorteil, im Gegensatz zur Heaviside-Funktion an jeder Stelle differenzierbar zu sein. Eine weitere bekannte Funktion, die als linearer Prädiktor eingesetzt wird, ist der Tangens Hyperbolicus (Tanh).

Ein Vorteil eines solchen Funktionsverlaufes ist, dass jeder Wert von Z_i einen Wert zwischen 0 und 1 einnimmt. Es wird ein Wert von 0,5 eingenommen, sollte Z_i den Wert 0 annehmen. Die Funktion ist symmetrisch zum Wendepunkt bei $P = 0,5$. Als Faustregel für die Anwendbarkeit der logistischen Regression sollte eine Datenmenge von mehr als 50 Beobachtungen vorhanden sein.⁷⁰

Damit kann für mehrere Ereignisse eine Wahrscheinlichkeit für deren Auftreten zwischen 0 und 1 normiert berechnet und dies auf eine differenzierbare Funktion, die Optimierungsalgorithmen wie den Stochastic Gradient Descent (SGD) zulässt, abgebildet werden.

3.5.4 Conditional Random Fields

Ein CRF ist ein deskriptives, ungerichtetes grafisches Modell. Das Label-Bias-Problem des in Kapitel 3.5.2.4 gezeigten MEMM teilt es, aufgrund einer dem MEMM verschiedenen Partitionsfunktion Z , nicht. Ein Graph G über X und Y , wie in Abbildung 3.10 dargestellt, ist dann ein CRF, wenn jeder Wert $x \in X$ auf G zur Wahrscheinlichkeit $P(y|x)$ faktorisiert.⁷¹

⁷⁰Vgl. Krafft (1996), S. 4.

⁷¹Vgl. Sutton/McCallum (2012), S. 291.

Ein CRF kann sowohl als sequenzielles Gegenstück zur logistischen Regression als auch als Hidden-Markov-Modell mit bedingter Wahrscheinlichkeit betrachtet werden. Somit kann jedes HMM auch als eine Sonderform eines CRF dargestellt werden aber nicht vice versa. Ein CRF wird im Allgemeinen beschrieben als:

$$P(y|x) = \frac{\prod_{C_e \in C} \prod_{\Psi_c \in C_e} \Psi(x_c, y_c; \theta)}{\sum_y \prod_{C_e \in C} \prod_{\Psi_c \in C_e} \Psi(x_c, y_c; \theta)}, \tag{3.25}$$

wobei es sich beim Nenner um die Partitionsfunktion, der Form

$$Z(x) = \sum_y \prod_{C_e \in C} \prod_{\Psi_c \in C_e} \Psi(x_c, y_c; \theta), \tag{3.26}$$

handelt. Wird Gleichung 3.26 in Gleichung 3.25 eingesetzt, dann folgt daraus

$$P(y|x) = \frac{1}{Z(x)} \prod_{C_e \in C} \prod_{\Psi_c \in C_e} \Psi(x_c, y_c; \theta). \tag{3.27}$$

In diesem speziellen Fall wird der Graph G über seine Cliques C beschrieben. Eine solche Clique ist in Abbildung 3.10 über $y|_t \wedge y|_{t+1}$ als farbige Markierung dargestellt. Für die Potenzialfunktionen gilt:

$$\Psi(x_c, y_c; \theta) = \exp \left\{ \sum_{k=1}^{K(e)} \lambda_{ek} f_{ek}(x_c, y_c) \right\} \tag{3.28}$$

Hierbei ist f_{ek} die Funktion der Features, λ_{ek} sind die entsprechenden Wichtungparameter und θ bezeichnet die Modellparameter.

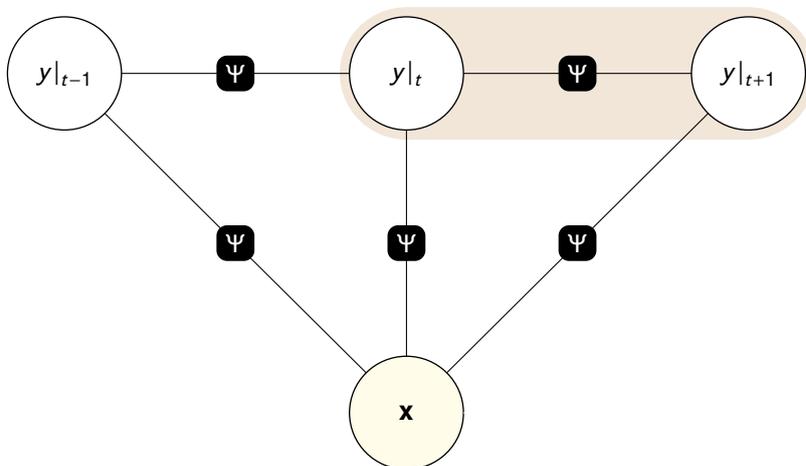


Abbildung 3.10: Modell eines CRF Faktorgraphen für den Sonderfall eines linear chain CRF. Quelle: Eigene Darstellung.

Der entscheidende Vorteil bei CRF besteht darin, dass lediglich die Beziehungen der Labels zueinander, modelliert werden müssen. Den Labels entsprechen in Abbildung 3.10 die Knoten y , der Knoten Vektor x entspricht der beobachteten Sequenz. Damit müssen die Beziehungen der einzelnen x untereinander nicht modelliert werden; aber die Möglichkeit, für die Voraussage eines Labels alle Werte des Vektors x miteinzubeziehen. Der wesentlichste Unterschied eines CRF zu einem MEMM liegt in der Normalisierung der Werte. Bei einem CRF wird global normalisiert, wodurch die Anzahl der Folgeknoten eines Zustandes keine direkte Rolle bei der Normalisierung spielt und das Label-Bias-Problem somit nicht auftritt. Trainiert

werden Conditional Random Fields anhand des Viterbi Algorithmus. Näheres dazu ist in Sutton und McCallum (2012)⁷² sowie McCallum, Freitag und Pereira (2000)⁷³ zu finden. Die hier dargebotenen Ausführungen basieren auf diesen beiden Werken.

⁷²Vgl. Sutton/McCallum (2012)

⁷³Vgl. McCallum/Freitag/Pereira (2000)

3.6 Künstliche neuronale Netze

Eng verbunden mit der Nutzung von stochastischen Modellen sind künstliche neuronale Netze. Diese finden ebenfalls in vielen Domänen Anwendung und sind ein wichtiges Werkzeug der künstlichen Intelligenz, des maschinellen Lernens, des NLP und somit auch der IE. In diesem Abschnitt soll die Funktion neuronaler Netze kurz beschrieben werden, um diese im späteren Verlauf für den IE Task nutzen zu können. Insbesondere sogenannte Recurrent Neural Networks finden bei der Zuordnung von Labels zu einer Sequenz von Worten – dies ist die Aufgabe, mit der sich diese Arbeit im Wesentlichen auseinandersetzt – häufig Anwendung. Es soll mit einem einfachen, einlagigen Perzeptron begonnen und darauf aufbauend hin zu komplexeren Formen der neuronale Netze (NN) geführt werden. Neue komplexe Formen und Architekturen, die auf neuronalen Netzen aufbauen, sind aktuell Forschungsgegenstand verschiedener Disziplinen des NLP und erreichen State-of-the-Art-Ergebnisse in Precision, Recall und F-Score (siehe hierzu Kapitel 3.3).⁷⁴

3.6.1 Einlagiges Perzeptron

Die Abbildung 3.11 zeigt, wie man sich ein einzelnes Neuron mathematisch vorstellen kann. Die Eingaben des Modells stellt der Vektor $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$ dar. Diese Eingaben werden im Weiteren mit den Wichtigkeitsfaktoren $\boldsymbol{\omega} = \{\omega_1, \omega_2, \dots, \omega_n\}$ multipliziert.

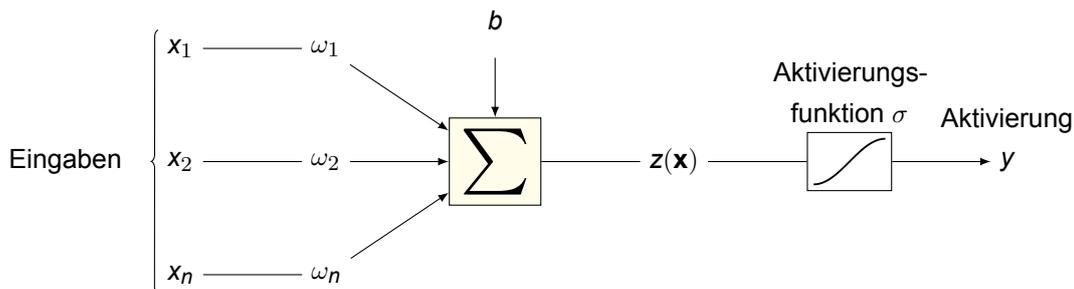


Abbildung 3.11: Das mathematische Modell eines künstlichen Neurons. Quelle: Eigene Darstellung.

Im folgenden Knoten werden die eingehenden Werte aufsummiert. Damit ergibt sich am Ausgang dieses Knotens

$$z(\mathbf{x}) = \sum_{i=1}^n \omega_i x_i + b. \quad (3.29)$$

Dieser Wert ist daraufhin der Eingangswert der Aktivierungsfunktion, b ist der sogenannte Bias der Funktion. Die Aktivierungsfunktion wird mit σ bezeichnet. Die Aktivierungsfunktion kann unterschiedlicher Natur sein: Es kann sich bei ihr um einen Sprung (Heaviside-Funktion) um eine S-Kurve (Sigmoid) oder auch um eine einfache oder gleichgerichtete lineare Funktion handeln. Handelt es sich bei der Aktivierungsfunktion um eine Schwannenhalskurve (Sigmoid), dann entspricht dieser Aufbau in Analogie einer Nervenzelle im menschlichen Gehirn. Eine Heaviside-Funktion – Unit-Step oder auch Einheitssprung genannt – ist für Eingangswerte unter einem gewissen Schwellwert 0 und darüber 1. Auch bei den Neuronen im menschlichen Gehirn kann man sich vorstellen, dass die eingehenden Signale aufsummiert werden und bei Überschreitung eines gewissen Schwellwertes das Neuron seinerseits feuert. Dies ist nur als stark vereinfachte Analogie zu verstehen, erklärt aber, warum es bei diesen künstlichen Neuronen abermals zu Vergleichen mit dem Menschlichen Gehirn kommt, in dem ohne Zweifel weit komplexere Vorgänge stattfinden.⁷⁵

⁷⁴Vgl. Arnold et al. (2016), S. 1-10.

⁷⁵Vgl. Frochte (2018), S. 161 ff.



Abbildung 3.12: Auf der linken Seite ist ein logisches UND und auf der rechten Seite ein logisches ODER. Beide Funktionen wurden mithilfe eines einzigen Perzeptron mit je 2 Eingängen und dem Bias erstellt. Quelle: Jurafsky/Martin (2018), S. 135. (modifiziert)

Im Folgenden wird anhand eines Beispiels, das sich an Jurafsky und Martin (2018)⁷⁶ orientiert, der Weg vom einfachen Feed Forward Network hin zum Recurrent Neural Network beschrieben. Das komplette Prinzip des Lernens von neuronalen Netzen und auch der meisten anderen Machine-Learning-Methoden basiert darauf, die Gewichte ω so anzupassen, dass y den gewünschten Wert annimmt. Das in Abbildung 3.11 dargestellte einlagige Perzeptron kann, indem zur Gleichung 3.29 die Aktivierungsfunktion σ hinzugefügt wird, mathematisch dargestellt werden. So ergibt sich

$$y = \sigma(z(\mathbf{x})) = \sigma\left(\sum_{i=1}^n \omega_i x_i + b\right), \tag{3.30}$$

mit y als letztgültige Ausgabe des Netzwerks. Zur einfacheren Darstellung dieses Sachverhaltes soll Obiges in Vektorschreibweise notiert werden als:

$$y = f(z) = \sigma(z(\mathbf{x})) = \sigma(\boldsymbol{\omega} \cdot \mathbf{x} + b), \tag{3.31}$$

wobei $\boldsymbol{\omega}$ der Vektor der Wichtungsfaktoren, \mathbf{x} der Vektor der Eingangswerte, b der Bias und σ die Aktivierungsfunktion ist. Aus diesem Zusammenhang folgt aus einer Aktivierung z in σ der Ausgang y . Als Aktivierungsfunktion werden häufig eine Sigmoid-Funktion, Tangens Hyperbolicus (TanH) oder Rectified Linear Unit (ReLU) verwendet.⁷⁷

Die Sigmoid-Funktion trat bereits bei der logistischen Regression auf. Sie hat die Form

$$y_{Sigmoid} = f(z) = \frac{1}{1 + e^{-z}} \tag{3.32}$$

und zeichnet sich durch einen Wertebereich von 0 bis +1 sowie eine stetige Differenzierbarkeit aus. Wird aus Gleichung 3.31 hier eingesetzt, ergibt sich:

$$y_{Sigmoid} = f(z) = \frac{1}{1 + \exp(-(\boldsymbol{\omega} \cdot \mathbf{x} + b))}. \tag{3.33}$$

Der TanH bildet hierbei auf einen Wertebereich von -1 bis +1 ab und teilt den S-Förmigen Verlauf eines Sigmoids. Er hat die Form

$$y_{TanH} = f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{3.34}$$

und kommt häufiger zur Anwendung als Sigmoid. Die wahrscheinlich am häufigsten verwendete Aktivierungsfunktion ist die ReLU, die zudem eine der Einfachsten ist. ReLU ist gleich $\boldsymbol{\omega} \cdot \mathbf{x}$, wenn $\boldsymbol{\omega} \cdot \mathbf{x}$ größer als 0 ist und 0 sonst, somit kann der Ausgangswert y_{ReLU} dieser Funktion beschrieben werden mit:

$$y_{ReLU} = \max(\boldsymbol{\omega} \cdot \mathbf{x}, 0). \tag{3.35}$$

⁷⁶Vgl. Jurafsky/Martin (2018)

⁷⁷Vgl. Jurafsky/Martin (2018), S. 132.

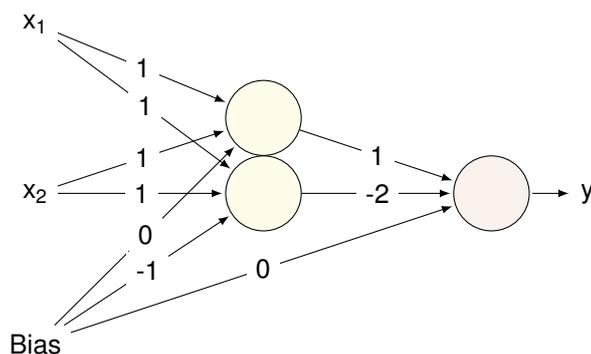


Abbildung 3.13: Darstellung eines logischen XOR als Verbindung mehrerer Perzeptren. Quelle: Jurafsky/Martin (2018), S. 136. (modifiziert)

Diese Aktivierungsfunktionen haben unterschiedliche Eigenschaften, aufgrund derer sie bei verschiedenen Anwendungen oder Architekturen nützlich sind. Zum Beispiel hat ReLU die vorteilhafte Eigenschaft, dass sie einer linearen Funktion sehr nahekommt. Sigmoid und TanH begrenzen die Werte von y , was beim Lernen ein Nachteil sein kann. ReLU hingegen begrenzt die Werte von y nicht, das kann in bestimmten Situationen ein Vorteil sein. Es kann gezeigt werden, dass ein einlagiges Perzeptron dieser Form mithilfe seiner Wichtungsfaktoren ein logisches UND sowie ein logisches ODER abbilden kann, nicht aber ein logisches explizit ODER (XOR), dies kann in Abbildung 3.12 nachvollzogen werden. Die Lösung für dieses Problem sind Netzwerke von Perzeptren. Hintereinander geschaltete Perzeptren können ein logisches XOR abbilden.⁷⁸

Diese Gegebenheit ist in Abbildung 3.13 dargestellt, für Näheres hierzu siehe Goodfellow⁷⁹.

3.6.2 Feed Forward Netzwerke

Der simpelste Aufbau eines neuronalen Netzes ist ein Feed-Forward-Netzwerk (FFN). Ein solches ist in Abbildung 3.14 dargestellt. Es besteht aus einem Input Layer, einem Output Layer und einer variablen

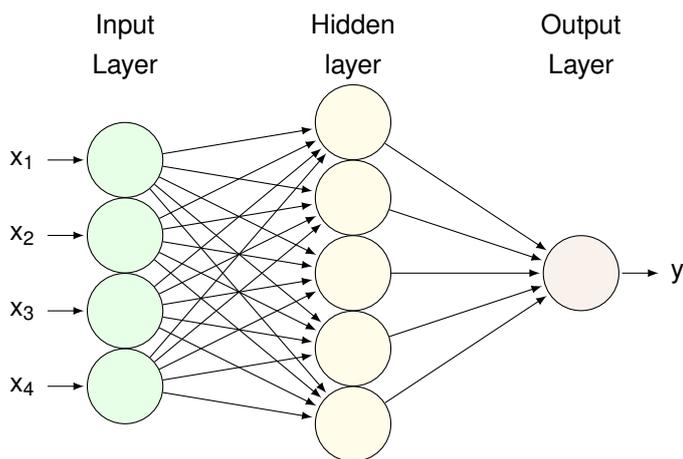


Abbildung 3.14: Neuronales Feed Forward Netzwerk mit einem Hidden Layer. Wie auch in den vorherigen Beispielen, können die Layer mit einem Bias versehen werden. Quelle: Eigene Darstellung.

Anzahl von Hidden Layers. Auch wenn es häufig auf diese Weise dargestellt wird, kann das Output Layer nicht nur aus einem einzelnen Knoten bestehen, zu dem alle Kanten weiterlaufen, sondern es können der Anwendung entsprechend mehrere Knoten verwendet werden. Namensgebend für FFN ist die Tatsache,

⁷⁸Vgl. Jurafsky/Martin (2018), S. 133 f.

⁷⁹Vgl. Goodfellow/Bengio/Courville (2016), S. 1 ff.

dass es keine Schleifen in diesen Netzwerken gibt. Das bedeutet, dass jeder Ausgang an ein folgendes Hidden oder Output Layer weitergeleitet, niemals jedoch an ein bereits durchlaufenes Layer zurückgeführt wird. Vorausgesetzt, dass jedes der Neuronen mit allen Neuronen des folgenden Layers verbunden ist, wird von Fully Connected Layers gesprochen. Dies ist der Standardfall und findet häufig Anwendung. Als Formel kann ein Hidden Layer beschrieben werden als:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}). \quad (3.36)$$

Hierbei steht \mathbf{h} für den Vektor der ausgegebenen Werte des Hidden Layers, \mathbf{W} ist eine Matrix, in der die jeweiligen Wichtungsfaktoren ω der Layer hinterlegt sind, \mathbf{x} ist ein Vektor, in dem die Eingangswerte übergeben werden, und \mathbf{b} ist ein Vektor, mit den Bias-Werten. Es muss angemerkt werden, dass jede Aktivierungsfunktion σ elementweise auf die einzelnen Werte der Berechnung angewendet wird. Im Falle von einfachen Fully Connected Layers kann der Ausgang eines weiteren Layers dadurch berechnet werden, dass der Ausgang des vorherigen Layers mit der Wichtungsmatrix des folgenden Layers multipliziert wird. Angenommen eine solche weitere Wichtungsmatrix ist eine Matrix \mathbf{U} , dann kann für den folgenden Output geschrieben werden:

$$\mathbf{z} = \mathbf{U}\mathbf{h} + \mathbf{b}, \quad (3.37)$$

wobei \mathbf{z} der Ausgang eben dieser Schicht ist. Dieser Vektor \mathbf{z} könnte in diesem Beispiel bereits zum Ergebnis werden. Angenommen bei der gestellten Aufgabe handelt es sich um eine Klassifizierung von Beobachtungen, dann kann die Wahrscheinlichkeitsverteilung über eine SOFTMAX-Funktion ermittelt werden. Für einen d dimensionalen Vektor \mathbf{z} ist der SOFTMAX definiert als:

$$\text{SOFTMAX}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^d e^{z_j}}, 1 \leq i \leq d \quad (3.38)$$

Als Ergebnis kann geschrieben werden:

$$y = \text{SOFTMAX}(z_i) \quad (3.39)$$

mit dem SOFTMAX als Aktivierungsfunktion und dem Ergebnis y .⁸⁰

Für die Aufgabe einer sequenziellen Verarbeitung von Text ist ein solches Netzwerk weniger geeignet. Das liegt daran, dass jede einzelne Schicht eines solchen Netzwerks zu jedem Zeitpunkt nur ihren aktuellen Eingangswert kennt. Bei der IE sind es gerade die Zusammenhänge zwischen den einzelnen Token, die zur Disambiguierung der Token führen.

Um auch kurz auf das Training von neuronalen Netzen einzugehen, eignet sich dieser einfache Fall eines FFN ebenfalls gut. Damit das Training das Ergebnis des FFN verbessert, müssen sich die Wichtungsfaktoren ändern. Die Änderung muss eine Verlustfunktion (eng. Loss Function) minimal werden lassen. Eine häufig verwendete Verlustfunktion ist die Kreuzentropie, Näheres hierzu in Kalinke und Seelen (1997)⁸¹. Eine der einfachsten Verlustfunktionen ist der sogenannte Mean Squared Error (MSE).

$$L_{\text{MSE}(\hat{\mathbf{y}}, \mathbf{y})} = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2 \quad (3.40)$$

Hierbei ist \hat{y}_i der vom Modell berechnete Wert, N ist gleich der Anzahl der Werte in Vektor \mathbf{y} welcher die Ergebnisse enthält und y_i der Wert, den das Modell bei einem Index i vorhersagen soll.

3.6.3 Recurrent Neural Networks

Bei Recurrent Neural Networks (RNN)⁸² handelt es sich um eine Alternative zu FFNs. Bei diesem war

⁸⁰Vgl. Jurafsky/Martin (2018), S. 137 ff.

⁸¹Vgl. Kalinke/Seelen (1997), S. 501-508.

⁸²Vgl. Hochreiter/Schmidhuber (1997), S. 1735-1780.

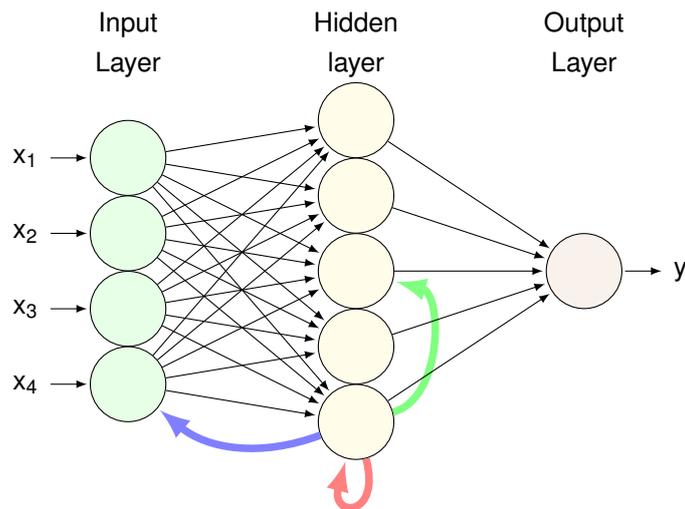


Abbildung 3.15: Neuronales RNN mit einem Hidden Layer. Jene Wege, die für Rückführungen verwendet werden können, sind als roter, blauer und grüner Pfeil dargestellt. Zusätzlich zu den Rückführungen der Werte aus dem Hidden Layer wäre es auch möglich, den Wert y an die vorherigen Layer zurückzuführen. Quelle: Eigene Darstellung

es nicht möglich, vergangene Zustände einer Sequenz in die Vorhersagen miteinzubeziehen, stattdessen wurde immer nur der aktuelle Wert am Eingang verarbeitet. Bei RNN wird der Ausgang eines Knotens, wie in Abbildung 3.15 gezeigt werden soll, an den Eingang des aktuellen Knotens (rote Kante), an den Eingang eines benachbarten Knotens (grüne Kante) oder an den Eingang eines Knotens eines vorhergehenden Layers (blaue Kante) zurückgeführt. Mit dieser Methode ist es RNN möglich, die vorhergehenden Elemente einer Sequenz in die aktuelle Beobachtung miteinzubeziehen. Wird an ein solches Netzwerk eine Sequenz der Form $\mathbf{x} = \{x_1, x_2, \dots, x_T\}$, bestehend aus T Elementen x_t gelegt, dann wird der zurückgeführte Ausgang berechnet mit:

$$h_t = \sigma(\mathbf{W}x_t + \mathbf{U}h_{t-1} + b_t). \quad (3.41)$$

Hierbei werden h_t als Hidden States bezeichnet – wie im Text zu Abbildung 3.15 beschrieben hätte hier auch der Ausgang y anstelle von h_t zurückgeführt werden können – der Bias als b_t , die Variablen mit den Wichtungsfaktoren sind die Matrizen \mathbf{W} und \mathbf{U} , während es sich bei σ um die Aktivierungsfunktion handelt.⁸³ Wird nun angenommen, dass die einzelnen Elemente x_t Zahlen sind, die einem Wort eindeutig zugeordnet sind, dann gibt ein solches RNN, nachdem es trainiert worden ist, ein Modell wieder, das die Wahrscheinlichkeit für das Auftreten eines Wortes basierend auf den vorhergehenden Worten wiedergibt. Ein Recurrent Neural Network stellt damit eine bedeutende Weiterentwicklung zu einfachen Feed Forward Networks dar, weil es auf eine gewisse Art und Weise ein Gedächtnis besitzt. Damit scheint es für Aufgaben, die Sequenzen verstehen sollen, eine gute Wahl zu sein. Aber auch die Nachteile von RNNs dürfen nicht unbeachtet bleiben. Das Modellieren einer Sequenz anhand eines bestimmten vorhergehenden Zustandes betrachtet eben genau diesen. Das Training einer solchen Architektur wird schwierig, wenn die Distanz der für den aktuellen Zustand relevanten Vorabinformation variiert oder sich über mehrere Zustände erstreckt.⁸⁴

Es ist eine einfache Methode, einem RNN ein Wort, einen Wortstamm, einen Satz oder gegebenenfalls ein ganzes Dokument in Form einer Zuordnung zu einer Zahl zu übergeben. Werden aber die große Menge an einzelnen Worten in einer Sprache und die möglichen Kombinationen dieser betrachtet – mit teils stark abweichender Semantik bei gleichen Wörtern – wird ersichtlich, dass auch alternative Methoden in Betracht

⁸³Vgl. Chung et al. (2014), S. 2-3.

⁸⁴Vgl. Bengio/Simard/Frasconi et al. (1994), S. 157-166.

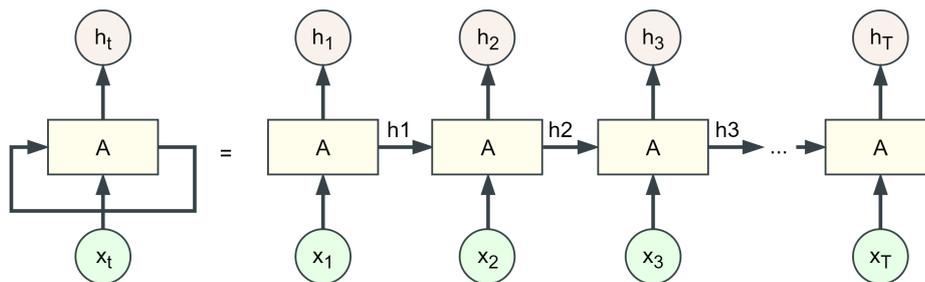


Abbildung 3.16: Darstellung eines, über die Zeit abgerollten, Recurrent Neural Network. Quelle: Olah (2015) (modifiziert)

zu ziehen sind, um Wörter zur weiteren Verarbeitung in neuronalen Netzen zu repräsentieren (siehe hierzu Seite 37).

3.6.4 Long Short Term Memory

Bei der Verarbeitung sequenzieller Daten, also dann, wenn Daten Dimensionen wie Zeit (t) oder Position [n] beinhalten, ist es häufig nicht hinreichend, die diskreten Einträge als einander unabhängig anzunehmen. Wie im vorherigen Abschnitt mit RNN erwähnt wurde, können neuronale Netze versuchen diese Abhängigkeiten mittels Rückführung einer Funktion ihres Ausgangs zu modellieren. Dieser Ansatz hat sich bei sequenziellen Daten als State of the art etabliert, jedoch werden hierbei keine gewöhnlichen RNNs verwendet, sondern komplexere Strukturen wie LSTM⁸⁵ (Long Short Term Memory) Networks oder GRU (Gated Recurrent Units). Erstere sollen hier kurz vorgestellt werden, nähere Informationen sind Sherstinsky (2018) zu entnehmen.

Die folgenden Ausführungen basieren auf Olah (2015).

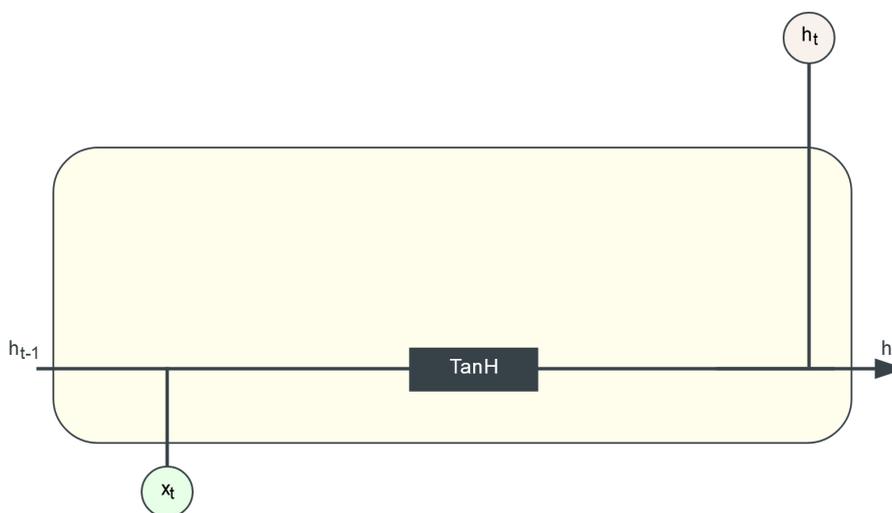


Abbildung 3.17: Innenleben eines Recurrent Neural Network mit TanH als Aktivierungsfunktion. Quelle: Olah (2015) (modifiziert)

Als Ausgangspunkt soll, wie in Abbildung 3.16 zu sehen ist, ein Teil A eines Neuronale-Netze (NN) dienen, mit seinem Eingang x_t und den Hidden State h_t als Ausgang. Um der folgenden Berechnung den Ausgangswert der Berechnung zur Verfügung zu stellen, wurde eine Schleife eingezeichnet, die links in Abbildung 3.16 zu sehen ist. Rechts daneben ist ersichtlich, wie dies, wenn die Schleife über die Dimension der Sequenz – in diesem Beispiel t – abgerollt wird, veranschaulicht werden kann.

⁸⁵Vgl. Hochreiter/Schmidhuber (1997), S. 1735-1780.

In Abbildung 3.17 ist hierzu zu sehen, wie das Innere eines gewöhnlichen RNN aussehen würde. Dabei steht das Symbol TanH für ein neuronales Netzwerk, das einen TanH (Tangens Hyperbolicus) als Aktivierungsfunktion besitzt. Der TanH bildet auf einen Wertebereich zwischen -1 und +1 ab. Ein Sigmoid, hier dargestellt durch das Symbol SIGMOID , bildet auf einen Wertebereich zwischen 0 und +1 ab. Also beseitigt der Sigmoid negative Werte, was bei gewissen Architekturen gewünscht sein kann und bei anderen wiederum nicht. Hierin liegt eine jener Überlegungen, die beim Erstellen von neuronalen Architekturen eine tragende Rolle spielt. Bei diesen gewöhnlichen RNNs kommt an dieser Stelle das Vanishing-Gradient-Problem⁸⁶ besonders stark zum Tragen. Das Modell hat in dieser Form also wenig bis keine Möglichkeit, ein Gedächtnis zu simulieren, das Ereignisse an unterschiedlich weit entfernten Positionen der Sequenz zu berücksichtigen vermag. Dieses Problem wird durch Long-Short-Term-Memory-Netzwerke behoben. Eine

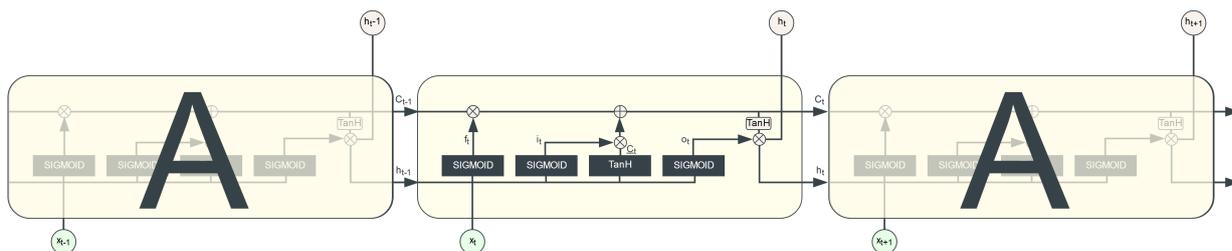


Abbildung 3.18: Darstellung eines, über die Zeit abgerollten, LSTM. Quelle: Olah (2015) (modifiziert)

schematische Darstellung der Aneinanderreihung von LSTMs ist in Abbildung 3.18 zu sehen. In dieser Abbildung wurde das RNN, analog zu Abbildung 3.16, über die Zeit abgerollt. Der Unterschied ist, dass es sich hierbei um ein LSTM-Netzwerk handelt. In Abbildung 3.19 ist der innere Aufbau einer solchen LSTM-Zelle

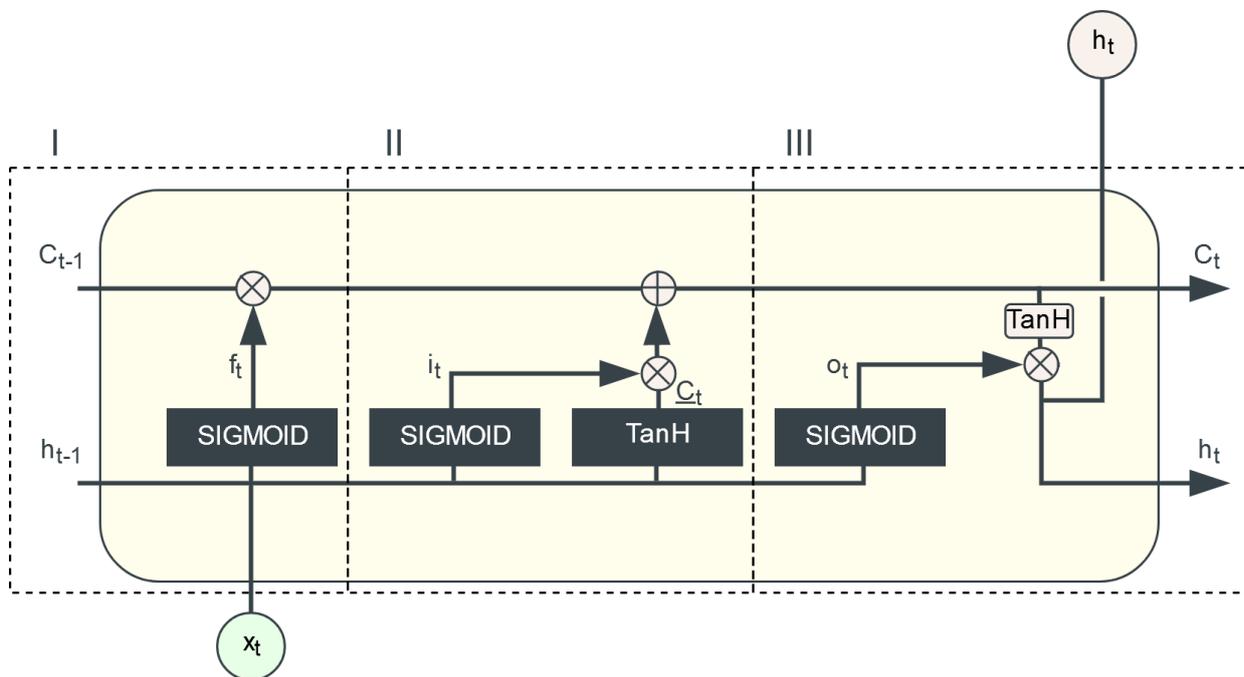


Abbildung 3.19: Innerer Aufbau eines LSTM, gegliedert in die Teilbereiche I) Forget, II) Update und III) Publish. Quelle: Olah (2015) (modifiziert)

dargestellt. Dieser ist in drei Teilbereiche aufgeteilt, die im Folgenden erklärt werden.

I) Forget

Im ersten Teilbereich des LSTM ist links oben als Eingang C_{t-1} zu sehen. Dieser Eingang läuft einmal

⁸⁶Vgl. Bengio/Simard/Frasconi et al. (1994), S. 157-166.

gerade durch die ganze Zelle und endet bei Ausgang C_t . Hierbei handelt es sich um den sogenannten *Cell State*, der von Zelle zu Zelle weitergegeben und bearbeitet wird. Die erste Änderung, mit der die Zelle den Cell State beeinflussen kann, ist f_t . Dieses kommt von unten und wird über den \otimes Operator – dieser soll hier für das elementweise Multiplizieren stehen – mit dem Cell State verbunden. Elementweises Multiplizieren beeinflusst den Cell State dabei wie ein Ventil, das über f_t geregelt werden kann. Wie viel der vorhergehenden Information C_{t-1} im Cell State bestehen bleiben soll, entscheidet das neuronale Netz, das vor f_t mittels Sigmoid Aktivierungsfunktion einen Wert zwischen 0 und 1 an den elementweisen Operator weiterleitet. Am Eingang des neuronalen Netzes liegt die Verkettung der Informationen h_{t-1} und x_t . Auf diese Weise wird in Abschnitt I) des LSTM-Moduls geregelt, wie viel der vorhergehenden Information im Cell State vergessen werden soll. Für f_t , also die Funktion Forget, kann beschrieben werden:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \quad (3.42)$$

wobei als Aktivierungsfunktion Sigmoid(σ) verwendet wird. Bei W_f handelt es sich um die Variablen des Netzes und bei b_f um den Bias.

II) Update

Für den zweiten Teilbereich wird in dieser Erklärung symbolisch der Begriff Update verwendet. In diesem Teilbereich wird dem Cell State über den Operator \oplus – dieser soll für Addition der Elemente stehen – Information hinzugefügt. So wird dieser sozusagen upgedated. Welche Information dem Cell State hinzugefügt werden soll, entscheidet \underline{C}_t . Dieser Wert wird von einem neuronalen Netz mit TanH als Aktivierungsfunktion erzeugt. Damit liegt dieser immer zwischen -1 und 1. Wie viel dieser Information dem Cell State hinzugefügt wird, wird über i_t entschieden; dies funktioniert analog zum ersten Teilbereich mittels elementweiser Multiplikation. Für die beiden neuronalen Netze kann geschrieben werden:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (3.43)$$

und

$$\underline{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C). \quad (3.44)$$

Damit ist der am Ausgang C_t anliegende Cell State für die folgende Zelle bereits vollständig und kann beschrieben werden mit:

$$C_t = f_t \otimes C_{t-1} + i_t \otimes \underline{C}_t, \quad (3.45)$$

wobei mit \otimes das Hadamard Product, also das elementweise Multiplizieren der Werte, gemeint ist. Im Cell State kann über die ersten drei neuronalen Netze entschieden werden, welche Information wann, wie berücksichtigt werden sollen. Damit hat diese Architektur eine Art variables Gedächtnis, womit das Vanishing-Gradient-Problem⁸⁷ behoben wird.

II) Publish

Der letzte Teilbereich wird in dieser Erklärung als Publish bezeichnet, weil es die Aufgabe dieser Architektur ist, die Information des Hidden States h_t zu publizieren. Wieder wird über einen Wert, in diesem Fall o_t , über die Variablen eines neuronalen Netzes entschieden, wie viel von C_t an den Hidden State h_t weitergegeben wird.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3.46)$$

⁸⁷Vgl. Bengio/Simard/Frasconi et al. (1994), S. 157-166.

Der Cell State kann an dieser Stelle zwischen -2 und +2 liegen, weshalb dieser mit einem TanH (an dieser Stelle nur die Funktion und nicht wie sonst ein neuronales Netz mit TanH als Aktivierungsfunktion) zwischen -1 und +1 begrenzt wird. Damit ergibt sich der Hidden State als

$$h_t = o_t \otimes \tanh C_t. \quad (3.47)$$

Bei LSTMs handelt es sich um eine Variante von RNNs. Neben der vorgestellten Variante existieren noch weitere Variationen, von denen an dieser Stelle die Architektur Gated Recurrent Unit⁸⁸ genannt sein soll. Diese kommt mit einem neuronalen Netz weniger aus als die hier gezeigte Variante und erfreut sich im Arbeiten mit Texten großer Beliebtheit.⁸⁹

⁸⁸Vgl. Cho et al. (2014), S. 1-10.

⁸⁹Vgl. Olah (2015)

positives oder negatives Sentiment des Textes schließen lassen. Ein Text wird danach je nach Häufigkeit der als positiv gelabelten hinterlegten Wörter im Verhältnis zu den als negativ gelabelten als positiv oder als negativ klassifiziert. Über Wörter, die nicht explizit hinterlegt sind, gibt ein solches Modell keine Auskunft. Sind die Wörter jedoch als Embeddings hinterlegt, können auch aus Wörtern, die eine semantische Nähe zueinander aufweisen, Rückschlüsse auf das Sentiment eines Textes getroffen werden. Deswegen, und weil für das Trainieren von Wortembeddings kein aufwendiges Zuweisen von Labels nötig ist, sind diese mittlerweile die bevorzugte Repräsentationsform für Wörter in NLP-Anwendungen. Im Allgemeinen kann bei Embeddings zwischen zwei Arten unterschieden werden, die im Folgenden kurz erklärt werden.⁹²

3.7.1 Term Frequency Inverse Document Frequency

Als Erstes soll das TF-IDF-Maß vorgestellt werden, für das ein Beispiel aus Jurafsky und Martin (2018)⁹³ als Vorbild diente. Der Ausdruck ist eine Abkürzung und steht für: term frequency bzw. inverse document frequency. Der Name beschreibt bereits, worum es dabei geht. Bei der term frequency⁹⁴ wird gezählt, wie oft ein bestimmtes Wort in einem Dokument vorkommt, und ein Faktor berechnet; mehr dazu auf Seite 40. Werden verschiedene Dokumente hinsichtlich des quantitativen Vorkommens bestimmter Wörter verglichen, können bereits Rückschlüsse auf den Inhalt des Dokuments gezogen werden. Im Beispiel von Jurafsky und Martin ist dies nachvollziehbar. Diese zeigen ein Beispiel, in dem das Vorkommen der Wörter *battle*, *good*, *fool* und *wit* in Werken von Shakespeare gezählt wurde, und schreiben die Anzahl in der sogenannten term-document matrix nieder. Eine solche sieht aus wie in Tabelle 3.2, die Worthäufig-

	As You Like It	Twelfth Night	Julius Caesar	Henry V
wit	20	15	2	3
fool	36	58	1	4
good	114	80	62	89
battle	1	0	7	13

Tabelle 3.2: Darstellung einer term-document matrix. In den Spalten stehen Titel von Werken Shakespears und in den Zeilen die Anzahl der Vorkommen des entsprechenden Wortes. Quelle: Jurafsky/Martin (2018), S. 108. (modifiziert)

keit wird hierbei in Vektoren zu je vier Einträgen dargestellt. Der Vektor für das Werk *Henry V* würde die Form $[3, 4, 89, 13]$ haben. Auch wenn in dieser Form die Aussagekraft dieser Darstellung unter Umständen nicht sofort ins Auge sticht, wird diese ersichtlich, wenn die Werte, wie an Abbildung 3.21 zu sehen ist, im Zweidimensionalen dargestellt werden. Die Ähnlichkeit zwischen den beiden Komödien ist in der Darstel-



Abbildung 3.21: Die Werte für *battle* und *fool* als Vektoren dargestellt. Bei den Werken *As You Like It* und *Twelfth Night* handelt es sich um Komödien. Quelle: Jurafsky/Martin (2018), S. 109. (modifiziert)

lung bereits gut erkennbar. Im praktischen Einsatz haben diese Dokumentvektoren eine wesentlich größere

⁹²Vgl. Jurafsky/Martin (2018), S. 107.

⁹³Vgl. Jurafsky/Martin (2018), S. 109 ff.

⁹⁴Vgl. Luhn (1957), S. 309-317.

Anzahl an Einträgen als vier. Wird $|\mathbf{v}|$ als die Länge des Vektors notiert, umfasst diese im Allgemeinen zwischen 10.000 und 50.000 der häufigsten Wörter. Wird beachtet, dass die meisten Wörter nur in gewissen Dokumenten vorkommen, so erscheint es nachvollziehbar, dass die meisten Einträge dieser Vektoren 0 sind. Damit ergeben sich spärlich besetzte Vektoren, für die es allerdings effiziente Algorithmen gibt, um sie zu speichern und Berechnungen mit ihnen durchzuführen.⁹⁵

Die Ähnlichkeit zwischen solchen Vektoren mit 10.000 Einträgen ist jedoch visuell nicht mehr so gut erkennbar, wie dies in Abbildung 3.21 der Fall war. Deswegen muss ein anderes Maß angewandt werden, um die Ähnlichkeit zweier Vektoren auszudrücken. Der häufigste Weg, die Ähnlichkeit zweier Vektoren mathematisch zu berechnen, ist es, den Winkel, den die beiden einschließen zu berechnen. Wird in Abbildung 3.21 der Winkel betrachtet, den diese einschließen, erscheint dieses Vorgehen als plausibel. Der Kosinus, der über den Winkel zwischen zwei Vektoren berechnet werden kann, basiert auf dem inneren Produkt im Englischen dot-product:⁹⁶

$$\text{dot-product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^N v_i w_i = v_1 w_1 + v_2 w_2 + \dots + v_N w_N \quad (3.48)$$

wobei $v \in \mathbf{v}$ und $w \in \mathbf{w}$ für die Anzahl der Wörter an den entsprechenden Positionen der Vektoren stehen und N der Anzahl der Wörter im anzuwendenden Vokabular entspricht. Das innere Produkt wird dann groß, wenn die beiden Vektoren große Werte an denselben Positionen haben. Das innere Produkt würde in dieser Form aber lange Vektoren bevorzugen, was es in dieser Form noch nicht vollständig verwendbar macht. Um dies auszugleichen, kann das innere Produkt normalisiert werden. Dies ist am einfachsten zu erreichen, indem es durch die Länge der beiden Vektoren \mathbf{v} und \mathbf{w} dividiert wird. Die Länge eines Vektors \mathbf{v} ist definiert als:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^N v_i^2} \quad (3.49)$$

Für das innere Produkt zweier Vektoren kann geschrieben werden:

$$\mathbf{v} \cdot \mathbf{w} = |\mathbf{v}| \cdot |\mathbf{w}| \cos \theta. \quad (3.50)$$

Durch Umstellen der Gleichung ergibt sich die Form:

$$\frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}| \cdot |\mathbf{w}|} = \cos \theta. \quad (3.51)$$

Wenn in diese aus Gleichung 3.49 und aus Gleichung 3.48 eingesetzt wird, ergibt sich für den Kosinus die numerisch berechenbare Form:

$$\cos(\mathbf{v}, \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}| \cdot |\mathbf{w}|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N v_i^2} \sqrt{\sum_{i=1}^N w_i^2}} \quad (3.52)$$

Wird dieser Wert nun für die Werke *As You Like It*, *Twelfth Night* und *Julius Ceasar* aus Tabelle 3.2 unter Verwendung von Formel 3.52 berechnet, ergibt sich durch Einsetzen

$$\cos(\textit{As You Like It}, \textit{Twelfth Night}) = \frac{20 \times 15 + 36 \times 58 + 114 \times 80 + 1 \times 0}{\sqrt{20^2 + 36^2 + 114^2 + 1^2} \times \sqrt{15^2 + 58^2 + 80^2 + 0^2}} = 0.9499, \quad (3.53)$$

und für

$$\cos(\textit{Julius Caesar}, \textit{Twelfth Night}) = \frac{2 \times 15 + 1 \times 58 + 62 \times 80 + 7 \times 0}{\sqrt{2^2 + 1^2 + 62^2 + 7^2} \times \sqrt{15^2 + 58^2 + 80^2 + 0^2}} = 0.8089. \quad (3.54)$$

⁹⁵Vgl. Jurafsky/Martin (2018), S. 110.

⁹⁶Vgl. Jurafsky/Martin (2018), S. 111.

Der in den Gleichungen 3.53 und 3.56 errechnete Wert spiegelt zwar wider, dass sich die beiden Komödien *As You Like It* und *Twelfth Night* einander ähnlicher sind als die beiden Werke *Julius Caesar* und *Twelfth Night*, aber der Unterschied scheint entgegen der Erwartung nicht signifikant. Um zu Verstehen, woran dies liegen könnte, sollen die beiden Rechnungen nun ohne die Einträge für das Wort *good* durchgeführt werden. Diese würden folgendermaßen aussehen:

$$\cos(\textit{As You Like It}, \textit{Twelfth Night} |_{\textit{good}}) = \frac{20 \times 15 + 36 \times 58 + 1 \times 0}{\sqrt{20^2 + 36^2 + 1^2} \times \sqrt{15^2 + 58^2 + 0^2}} = 0.9676, \quad (3.55)$$

und

$$\cos(\textit{Julius Caesar}, \textit{Twelfth Night} |_{\textit{good}}) = \frac{2 \times 15 + 1 \times 58 + 7 \times 0}{\sqrt{2^2 + 1^2 + 7^2} \times \sqrt{15^2 + 58^2 + 0^2}} = 0.1998. \quad (3.56)$$

Hierbei wird nun ein deutlicher Unterschied erkennbar. Es handelt sich beim Wort *good* um ein Wort, das durch seine Allgegenwärtigkeit in allen Dokumenten eine geringe Aussagekraft über den Inhalt dieser zu liefern scheint. Wörter, die jedoch nur in einigen wenigen Dokumenten häufig auftreten, haben eine hohe Aussagekraft über deren Inhalt. Um diesem Zwiespalt Rechnung zu tragen, verwendet der tf-idf-Algorithmus zwei Terme, die jeweils eine der beiden Überlegungen widerspiegeln:⁹⁷

term frequency

Beim ersten zu berechnenden Faktor handelt es sich um die bereits am Anfang des Kapitels erwähnte term frequency. Hierbei wird die Gewichtung der Auftrittshäufigkeit eines bestimmten Wortes durch das Verwenden des Logarithmus leicht vermindert. Die Verwendung des Logarithmus ist nur eine Möglichkeit von vielen, das Ziehen der Quadratwurzel wäre eine weitere. Damit wird dem Umstand Rechnung getragen, dass ein Wort, das zehnmals häufiger auftritt als ein anderes Wort deswegen nicht zwingend über eine zehnmals höhere Aussagekraft verfügt. Die term frequency ist definiert als:

$$tf_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t, d) & \text{für } \text{count}(t, d) > 0 \\ 0 & \text{sonst,} \end{cases} \quad (3.57)$$

wobei die term frequency *tf* Funktion die Anzahl der Worte *t* in einem Dokument *d* darstellt.

inverse document frequency

Der zweite zu berechnende Faktor ist die idf (inverse document frequency). Hierbei wird der Tatsache Rechnung getragen, dass ein Wort, das nicht besonders häufig, dafür aber in nur einigen wenigen Dokumenten vorkommt, eine hohe Aussagekraft über den Inhalt dieser Dokumente liefert. Die document frequency ist hierbei die Anzahl an Dokumenten, in denen das Wort vorkommt. Für den Fall, dass *G* die Anzahl der Dokumente in der Kollektion ist und *df* die Anzahl an Dokumenten, in denen ein Wort *t* vorkommt, ist die inverse document frequency definiert als:

$$idf_t = \log_{10} \left(\frac{G}{df_t} \right). \quad (3.58)$$

Weil die meisten Kollektionen aus einer großen Anzahl einzelner Dokumente bestehen, wird auch hier der Logarithmus zum Ausgleichen verwendet.

Aus diesen beiden Faktoren ergeben sich demnach die tf-idf-Werte $w_{t,d}$ (*w* steht für weight) als:

$$w_{t,d} = tf_{t,d} \times idf_t, \quad (3.59)$$

Wenn die Formel aus Gleichung 3.59 nun auf Tabelle 3.2 angewendet wird, ergibt sich die Tabelle 3.3. Bei dieser fällt auf, dass sich die Aussagekraft des Wortes *good* zu 0 geändert hat. Bei der Berechnung ergeben sich damit Embeddings, in Form von Vektoren mit den Gewichtungen der Wörter. Die Länge der Vektoren beträgt $|\mathbf{v}|$ und diese sind schwach besetzt; die meisten Einträge sind – in der Praxis – also 0.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
wit	0.049	0.044	0.018	0.022
fool	0.019	0.021	0.0036	0.0083
good	0	0	0	0
battle	0.074	0	0.22	0.28

Tabelle 3.3: Tabelle der als tf-idf Maß bewerteten term document matrix. Quelle: Jurafsky/Martin (2018), S. 115. (modifiziert)

Eine Verwendung für ein tf-idf-Maß ist die Berechnung der Wortähnlichkeit. Damit können Wortparaphrasen gefunden werden, sich ändernde Semantik mitverfolgt werden oder die Bedeutungen von Wörtern in verschiedenen Korpora bestimmt werden. Das tf-idf-Maß kann auch verwendet werden, um zu entscheiden, ob und in welchem Maße sich zwei Dokumente ähnlich sind. Hierfür kann ein sogenannter Dokumentvektor berechnet werden. Dieses Maß findet häufig Anwendung in verschiedenen Domänen; als Beispiel sind Information Retrieval, Plagiatsdetektion und Empfehlungssysteme zu nennen.

3.7.2 Skip Gram with Negativ Sampling

Im vorhergehenden Abschnitt wurden Worteinbettungen in schwach besetzte Vektoren gezeigt; hierbei waren die meisten Einträge der erhaltenen Vektoren 0 und die Anzahl der Elemente lag bei einigen Tausend Stück. In diesem Kapitel sollen Worteinbettungen unter Verwendung des SGNS-(Skip Gram with Negative Sampling)-Algorithmus vorgestellt werden, das begleitende Rechenbeispiel basiert lose auf dem Beispiel in Jurafsky und Martin (2018)⁹⁸.

SGNS ist einer von zwei Algorithmen des *word2vec*-Softwarepakets und wird deswegen auch landläufig als *word2vec*⁹⁹ bezeichnet. Mit dieser Methode werden Wörter auf Vektoren abgebildet, die zwischen 50 und 500 Einträge besitzen, von denen die meisten ungleich Null sind. Es stellte sich heraus, dass diese so dicht gepackten Vektoren bessere Ergebnisse in sämtlichen NLP-Anwendungen erzielen. Warum dem so ist, kann derzeit nicht eindeutig bestimmt werden, aber es gibt Erklärungsansätze. Ein Grund könnte sein, dass es für einen lernenden Algorithmus besser funktioniert, die Gewichtungen für 100 Parameter zu bestimmen als für 10.000 Parameter. Ein weiterer Grund könnte sein, dass durch die geringere Anzahl an Parametern, das Overfitting des Modells an die eingehenden Daten vermindert wird. Dadurch könnte das Modell besser auf die zugrunde liegenden Gesetzmäßigkeiten generalisieren. Als weiteres Modell, das dicht gepackte Vektoren erzeugt, ist neben *word2vec* das ebenfalls sehr häufig verwendete *GloVe*. Die Idee hinter *word2vec* ist hierbei nicht, wie es bei tf-idf der Falle wäre, zu zählen, wie oft ein Wort neben einem anderen in der Sequenz vorkommt, sondern einen binären Klassifizierer darauf zu trainieren, zu erkennen, ob es wahrscheinlich ist, dass ein Wort in einer bestimmten Spanne um ein gesuchtes Wort auftritt. Als binärer Klassifizierer wird eine Art logistische Regression verwendet. Anstatt das Ergebnis des Klassifizierungsprozesses als Embedding zu verwenden, werden die erlernten Wichtungsfaktoren, die das Modell erlernt hat, als Embeddings verwendet. Ein wesentlicher Vorteil bei diese Methode ist, dass die Wortsequenz als solche für jeden zu erlernenden Token bereits eine richtige Antwort enthält. Ein sogenanntes Golden Sample (Eine verifizierte richtige Antwort) muss somit nicht von Hand gelabelt werden, sondern es wird einfach das entsprechende Folgewort des Satzes selbst verwendet. Somit kann jede Quelle mit geschriebenem Text dem Modell als Trainingsdaten dienen. Die aufwendige Zuweisung von Labeln per Hand entfällt hierbei vollständig.¹⁰⁰

⁹⁷Vgl. Jurafsky/Martin (2018), S. 113.

⁹⁸Vgl. Jurafsky/Martin (2018), S. 114 ff.

⁹⁹Vgl. Mikolov et al. (2013), S. 4 ff.

¹⁰⁰Vgl. Jurafsky/Martin (2018), S. 118-119.

Wenn das Skip-Gram-Modell am Beispiel des Satzes ‚Jeder wackere Bayer vertilgt bequem zwo Pfund Kalbshaxen.‘ illustriert werden soll, ergeben sich die in Abbildung 3.22 zu sehenden Wort-Tupel, die als Trainingsdaten für den Regressionsalgorithmus dienen.

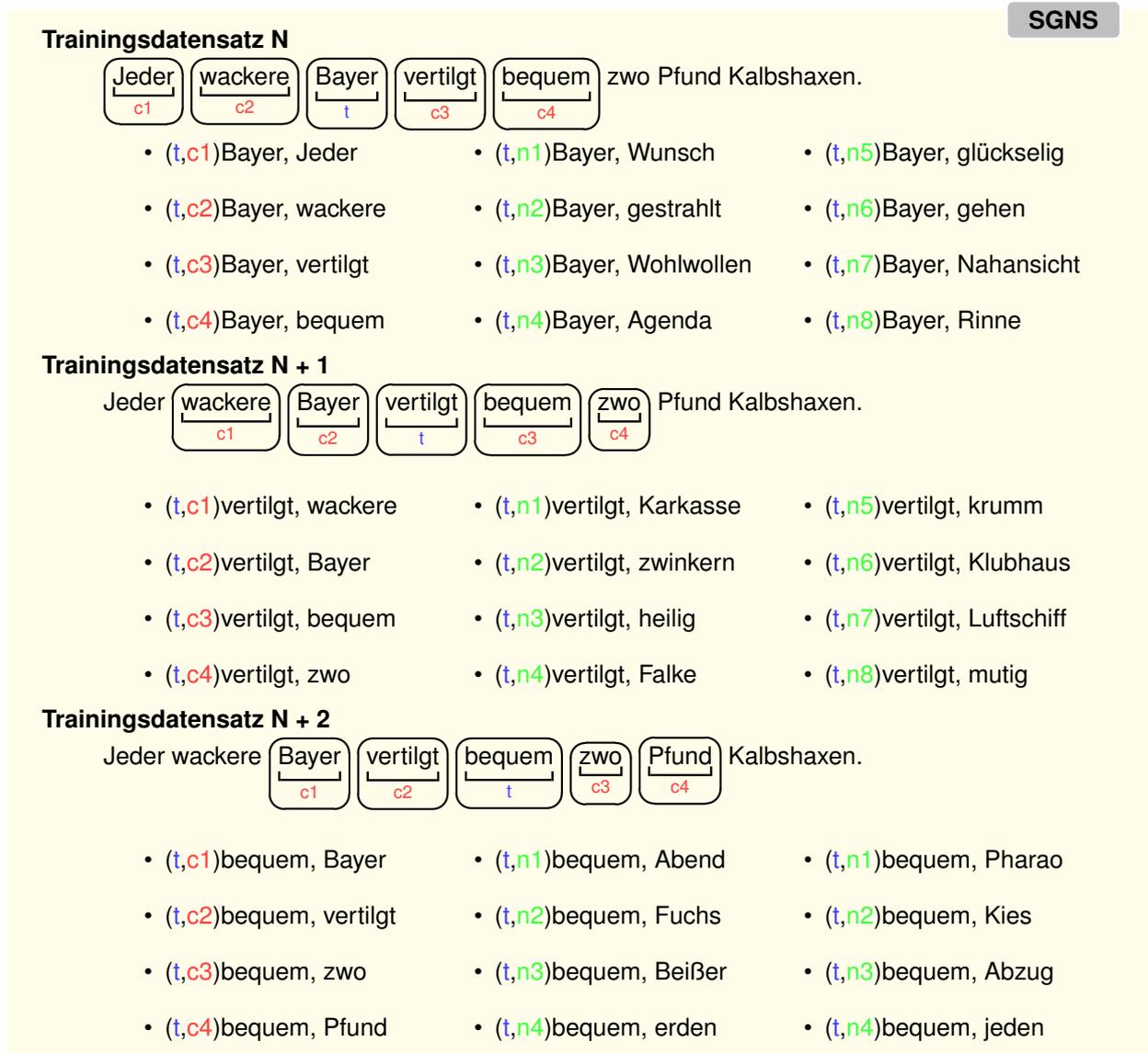


Abbildung 3.22: Tupel an Trainingsdaten, für eine Spanne von $L = \pm 2$ Token, bei einem SGNS Modell. Bei den als t bezeichneten Token handelt es sich um das Target-Word. Bei c handelt es sich um die Context-Words und bei n um Noise-Words welche der Generierung von $L \times k$, mit k negativen Samples dienen. In diesem Beispiel wurde k mit 2 gewählt. Quelle: Eigene Darstellung.

Hierbei wurde eine Kontextspanne von $L = \pm 2$ Wörtern, also insgesamt vier verwendet. Für jeden der auftretenden Tupel $a(t, c)$ – bei diesen handelt es sich um die positiven Beispiele für das gemeinsame Auftreten dieser beiden Wörter – soll das Modell die entsprechende Wahrscheinlichkeit

$$p(+|t, c) \tag{3.60}$$

berechnen. Wie sich die Tupel, die als Trainingsdaten dienen sollen zusammensetzen, soll in Abbildung 3.22 jeweils unter dem Satz als Aufzählung gezeigt werden. Das $+$ steht hierbei für positiv. Es bezeichnet jenen Fall, dass das Auftreten der Wortkombination des Tupels als wahrscheinlich klassifiziert werden soll. Wie der Name SGNS (Skip Gram with Negative Sampling) bereits andeutet, gibt es auch negative

Samples/Beispiele; diese werden jeweils aus t und einem n , das für Noise-Word steht, generiert. Die Wahrscheinlichkeit hierfür ergibt sich entsprechend als

$$p(-|t, c) = 1 - p(+|t, c). \tag{3.61}$$

Um zu berechnen, wie wahrscheinlich das Auftreten eines Beispiels ist, verwendet das Skip-Gram-Modell die Annahme, dass Wörter, bei denen es wahrscheinlich ist, dass sie nahe beieinander auftreten, eine hohe Ähnlichkeit aufweisen. Im vorhergehenden Kapitel wurde gezeigt, dass das innere Produkt als ein Maß für die Ähnlichkeit verwendet werden kann. Somit gilt auch hier

$$\textit{Similarity}(t, c) \propto t \cdot c. \tag{3.62}$$

Um dieses Skalarprodukt als Wahrscheinlichkeit zwischen 0 und 1 abzubilden, wird die logistische Funktion, die auch als Sigmoid bezeichnet wird, verwendet:

$$p(+|t, c) = \frac{1}{1 + e^{-t \cdot c}}. \tag{3.63}$$

Für den Fall, dass das Tupel als unwahrscheinlich klassifiziert werden soll, gilt:

$$p(-|t, n) = 1 - p(+|t, n) = \frac{e^{-t \cdot n}}{1 + e^{-t \cdot n}}. \tag{3.64}$$

Bis zu diesem Punkt kann nun eine Wahrscheinlichkeit für ein einzelnes Tupel berechnet werden. Benötigt wird für das SGNS die Wahrscheinlichkeit für das Auftreten des Tupel im gesamten Kontextfenster k , in dem dieses liegt. Hierfür macht das Skip Gram die Annahme, dass alle Context Words c oder für den negativen Fall n unabhängig voneinander sind. Beim positiven Beispiel gilt:¹⁰¹

$$p(+|t, c_{1:k}) = \prod_{i=1}^k \frac{1}{1 + e^{-t \cdot c_i}}, \tag{3.65}$$

woraus unter Zuhilfenahme der Rechenregeln für Logarithmen gebildet werden kann:

$$p(+|t, c_{1:k}) = \exp\left(\sum_{i=1}^k \ln\left(\frac{1}{1 + e^{-t \cdot c_i}}\right)\right). \tag{3.66}$$

Wie in Abbildung 3.22 zu sehen ist, werden bei Skip Gram mehr negative als positive Beispiele verwendet. Die Wörter n , die bei diesen Tupel zur Anwendung kommen, werden nach der gewichteten Häufigkeit ihres alleinigen Auftretens zufällig gewählt. Gewichtet wird hierbei, um auch seltene Noise Words eine leicht erhöhte Wahrscheinlichkeit zuzuweisen und diese somit zu berücksichtigen.¹⁰² Ziel des Algorithmus ist es, die Embeddings so anzupassen, dass die Ähnlichkeit bzw. das Skalarprodukt der t, c Tupel maximal und jenes der t, n Tupel minimal wird. Das Lernziel L ist damit

$$L(\theta) = \sum_{(t,c) \in +} p(+|t, c) + \sum_{(t,n) \in -} p(-|t, n), \tag{3.67}$$

wobei das Ziel ist, die Wichtungen/Embeddings θ so zu wählen, dass L maximal wird. Zum Trainieren kann der SGD verwendet werden. Abschließend soll hierzu erwähnt sein, dass das Skip-Gram-Modell für jedes Wort jeweils ein Embedding als Target t und ein Embedding als Context c erlernt. Von diesen kann dann entweder das Target Embedding einzeln oder die Summe der beiden Embeddings verwendet werden.¹⁰³ Um Einbettungen in Word2Vec visuell zu erkunden, kann derzeit der Tensorflow Embedding Projector als Onlinetool verwendet werden.¹⁰⁴

¹⁰¹Vgl. Jurafsky/Martin (2018), S. 119 ff.

¹⁰²Vgl. Rong (2014), S. 7 ff.

¹⁰³Vgl. Jurafsky/Martin (2018), S. 121-123.

¹⁰⁴Vgl. Smilov et al. (2016), S. 1-4.

4. STAND DER FORSCHUNG

In den vorhergehenden Kapiteln wurde ein Überblick über das Themengebiet des NLP gegeben. Begonnen wurde mit einer Erläuterung der Begrifflichkeiten und deren Abgrenzung. Danach wurde die Thematik der Daten an sich belichtet und wie diese für die maschinelle Verarbeitung vorbereitet werden können. Darauf aufbauend wurde gezeigt, wie ein Modell in Zahlen bemessen und bewertet wird. Das Bewerten der Methoden des NLP gibt Auskunft über die Leistungsfähigkeit aktueller Methoden bei deren Anwendung in bestimmten Aufgabengebiete im NLP. Die zugrundeliegende Theorie dieser Methoden wurden im Kapitel der theoretischen Grundlagen erläutert; einige dieser Methoden sind zurzeit Stand der Technik, andere wurden von neueren Methoden abgelöst oder verfeinert. Die im praktischen Teil der Arbeit verwendete Methodik, die in den nächsten Kapiteln folgt, baut auf diesen auf und erweitert sie gegebenenfalls. Dieses Vorgehen wurde gewählt, weil Natural Language Processing ein Themengebiet ist, das sich, zumindest zum jetzigen Zeitpunkt, in hohem Tempo weiterentwickelt. Ein Beispiel für diese schnelle Entwicklung soll Question Answering (QA), also das Beantworten einer in natürlichem Text geschriebenen Frage, dienen. Systeme, die dies leisten, werden an Datensätzen wie dem Stanford Question Answering Dataset 2.0 (SQuAD 2.0)¹⁰⁵ gemessen.

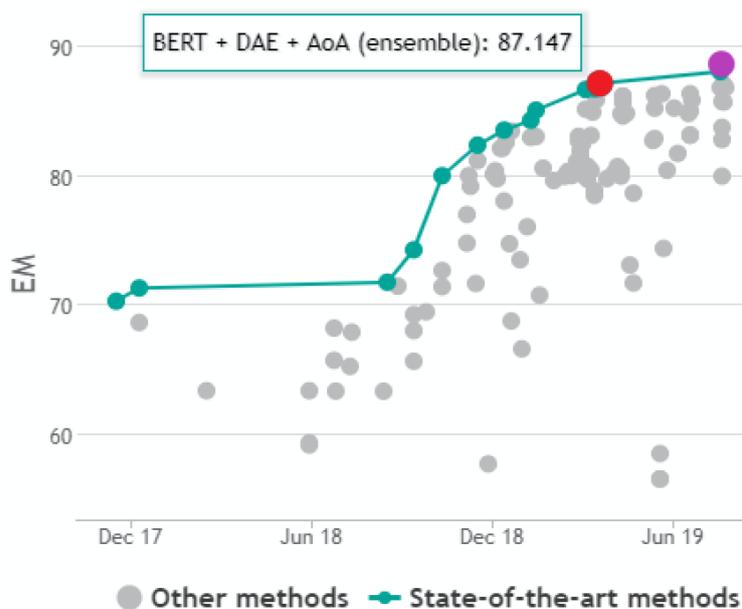


Abbildung 4.1: Leaderboard mit den State-of-the-Art (SOTA)-Methoden für QA, unter Verwendung des SQuAD 2.0. Quelle: Papers-With-Code (2019) (modifiziert)

In Abbildung 4.1 ist ein Ausschnitt aus dem SQuAD 2.0, über den Zeitraum Dezember 2017 bis Juni 2019, zu sehen. Im Dezember 2018 wurde der Score durch die Veröffentlichung des Bidirectional-Encoder-Representations-from-Transformers (BERT)-Modells¹⁰⁶ von 74,27 % auf 80,005 % gehoben. Anfang des Jahres 2019 erreichte Bidirectional Encoder Representations from Transformers (BERT) 87,147 % EM-Score, was dem roten Punkt in Abbildung 4.1 entspricht. EM steht für Exact Match, also den Fall, dass das Modell eine als richtig verifizierte Antwort genau wiedergibt. Im Juni 2019 veröffentlichte Google das Generalized-Autoregressive-Pretraining-for-Language-Understanding (XLNet)-Modell¹⁰⁷, das derzeit die BERT-Modelle mit einem Score von 88,592 % EM übertrifft; in Abbildung 4.1 entspricht dies dem violetten Punkt.

¹⁰⁵Vgl. Rajpurkar/Jia/Liang (2018), S. 1-9.

¹⁰⁶Vgl. Devlin/M. Chang et al. (2018), S. 1-16.

¹⁰⁷Vgl. Z. Yang et al. (2019), S. 1-18.

Diese Modelle wurden von Google veröffentlicht und stellen derzeit in dieser speziellen Disziplin den State of the Art (SOTA) dar. Es ist wahrscheinlich, dass zum Zeitpunkt der Veröffentlichung dieser Arbeit ebenfalls der von Generalized-Autoregressive-Pretraining-for-Language-Understanding (XLNet) erreichte Score übertroffen worden ist. NLP ist derzeit ein Forschungsgebiet, das sehr aktiv betrieben wird und in dem es keinen abzusehenden Stillstand gibt. Neben Question Answering gibt es viele weitere Disziplinen die dem NLP zuzurechnen sind. Einige Beispiele, für die es eigene Datensätze und Ranglisten gibt, sind in Tabelle 4.1 angeführt. Auch wenn es eine außergewöhnlich hohe Anzahl an verschiedenen Evaluierungen

Natural Language Processing			
Machine Translation	Sentiment Analysis	Reading Comprehension	Text Summarization
Question Answering	Text Classification	Named Entity Recognition	Information Retrieval
Language Modelling	Text Generation	Natural Language inference	Relation Extraction
Relational Reasoning	Dialogue	Semantic Textual Similarity	Dependency Parsing
Part-Of-Speech Tagging	Sentence Embeddings	Semantic Parsing	Conference Resolution

Tabelle 4.1: Beispiele für Teildisziplinen des NLP, für die es zum jetzigen Zeitpunkt Evaluierungen gibt. Insgesamt sind es mehrere Hundert Teildisziplinen, die zu Teilen ineinander greifen. Quelle: Eigene Darstellung.

gibt, sind die Modelle hinter den verschiedenen Disziplinen oft ähnlich. Bei den Modellen BERT und XLNet handelt es sich beispielsweise um Modelle, die beide auf Googles TransformerXL-Architektur¹⁰⁸ basieren und den Attention-Mechanismus¹⁰⁹ verwenden.

Bei der Suche nach dem Stand der Technik beim NLP oder in jenen Teilgebieten des NLP, die der strukturellen Informationsgewinnung aus Texten dienlich sind, kann derzeit nicht von einem einheitlichen Stand der Technik gesprochen werden. Vielmehr handelt es sich um eine Ansammlung verschiedener Methoden. Diese liefern entweder für sich allein genommen oder in Kombination miteinander anwendungsspezifische Ergebnisse, die es zu erkunden gilt. Während für das Extrahieren eines Preises der Form *100 € Regular expression (Regex)* das Mittel der Wahl sein könnte, können dies in anderen Gebieten statistische Modellgraphen wie CRFs sein. Deep Learning, Transfer Learning oder das Verwenden eines statischen Wissensgraphen sind weitere denkbare Ansätze. Dieser Vielschichtigkeit soll durch die Verwendung einer empirischen Vorgehensweise im Praxisteil der Arbeit Rechnung getragen werden.

¹⁰⁸Vgl. Dai et al. (2019), S. 1-20.

¹⁰⁹Vgl. Vaswani et al. (2017), S. 5998-6008.

5. PRAKTISCHE UMSETZUNG

Im folgenden Kapitel wird die praktische Umsetzung eines Dokument-Lesers mit Bearbeitungs- und Dokumentationsfunktion für das Legen von Angeboten bei Transformatoren und Drosseln in Komponenten erarbeitet. Ziel des Projektes und damit dieser Arbeit ist die prototypische Entwicklung eines arbeitsintegrierten digitalen Werkzeuges, das bei der technischen Angebotslegung unterstützt. Die Aufgabe des technischen Angebots ist es, auf Basis der Spezifikation, die der Kunde bereitstellt, ein Angebot für das Produkt zu erstellen und die Grundlage für eine interne Spezifikation zu schaffen. Bei diesem Prozess wird die erhaltene Spezifikation verarbeitet um Informationen über

- ähnliche, bereits durchgeführte Projekte,
- technische Kenngrößen des Produkts,
- geforderte Normen, die einzuhalten sind,
- relevante rechtliche Rahmenbedingungen,
- Anforderungen, die sich aus dem Transport des Produkts ergeben,
- Fertigbarkeit des Produkts an verschiedenen Standorten,
- Abschätzung des zu erwartenden Aufwandes,
- Komponenten, welche zuzukaufen sind,
- eine Abschätzung der Komplexität der Umsetzung,
- das damit verbundene Risiko

zu gewinnen und dem potenziellen, zukünftigen Kunden auf Grundlage des Erarbeiteten ein Angebot zu legen. Die Aufgaben des zu erarbeitenden Tools sind:

- die Dokumente einer Anfrage in einer Datenbank zu speichern,
- die Dokumente zu laden und die Anforderungen zu erkennen,
- die extrahierten Anforderungen den AnwenderInnen bereitzustellen,
- das Auflösen von Referenzen zu Dokumenten,
- die Bereitstellung von relevanten Dokumenten (z. B. Normblättern),
- das Anzeigen historischer ähnlicher Anforderungen und der gewählten Lösungen und
- das Vorschlagen von FachexpertInnen zu bestimmten Anforderungen, basierend auf inhaltlichen Merkmalen.

Damit soll den Anwenderinnen und Anwendern repetitive Arbeit abgenommen werden, damit sich diese auf das Lösen von neuen oder kritischen Anforderungen an Produkt und Umfeld fokussieren können.

5.1 Methodik

Die Fragestellung dieser Arbeit zielt darauf ab, Methoden des maschinellen Lernens zu erforschen und deren Eignung für die Implementation in das Tool zu bewerten. Zeitlich erstreckt sich die vorliegende Untersuchung über das erste Jahr eines dreijährigen Forschungsprojekts. Das Ziel nach diesem Jahr ist es, über einen Prototyp zu verfügen, der bereits in der Lage ist, relevante Informationen aus den Spezifikationen zu extrahieren und diese den AnwenderInnen bereitzustellen.

Um dies zu erreichen und als Zusatznutzen gegebenenfalls Methoden für weitere Bereiche im Unternehmen zu erarbeiten, wird im praktischen Teil schrittweise ein solches Tool entwickelt und es werden Lösungen für Teilbereiche erarbeitet und bewertet. Begonnen wird mit der Vorstellung einer möglichen Architektur für das Tool an sich, danach sollen Methoden des maschinellen Lernens implementiert werden. Abschließend soll der Entwicklungsstand zum Ende des ersten Jahres bewertet werden.

5.2 Software Architektur

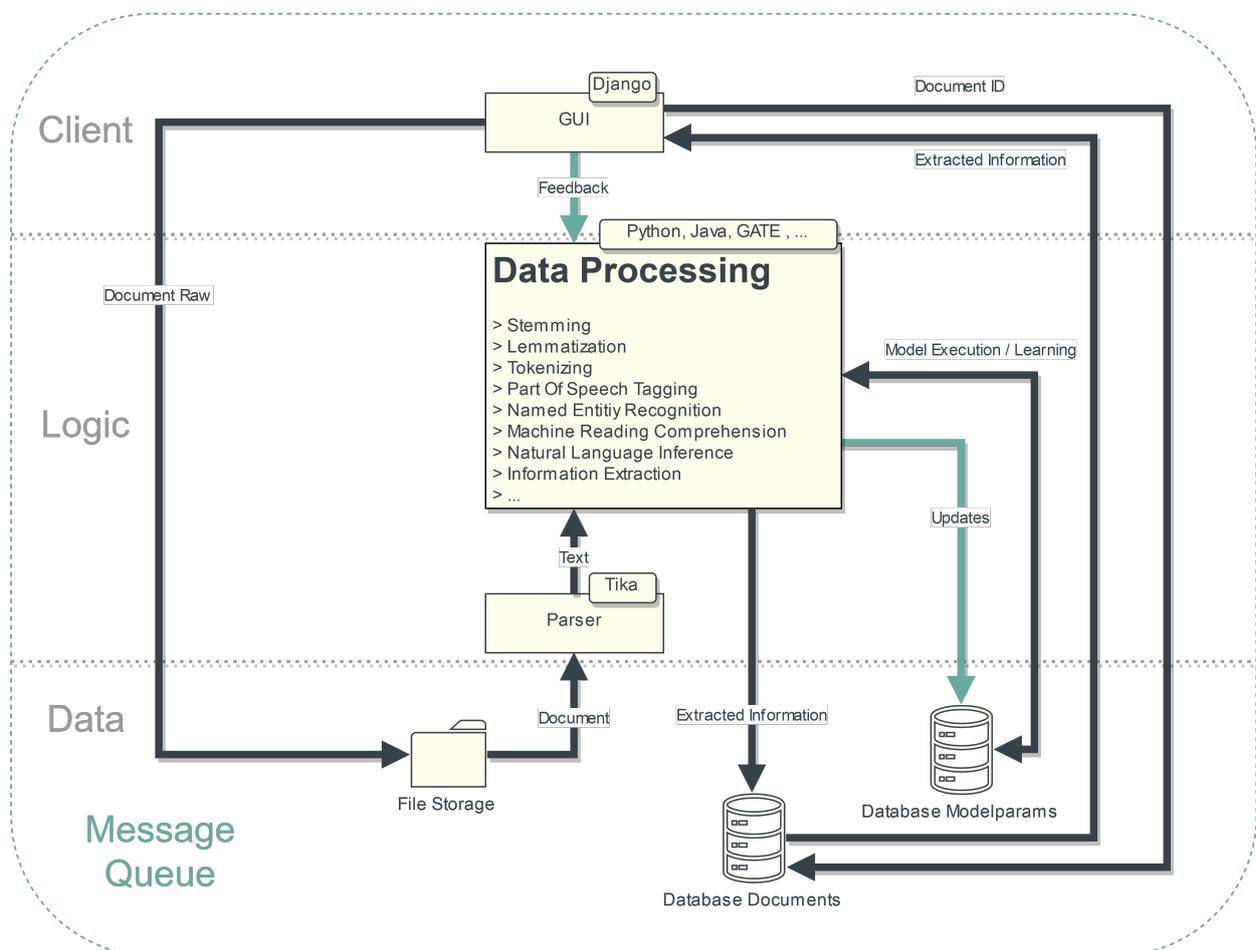


Abbildung 5.1: Schematischer Aufbau des Systems mit vom Client entkoppelter Datenverarbeitung. Die Message Queue übernimmt hierbei die Orchestration der Komponenten. Quelle: Eigene Darstellung.

Das Programm, das entwickelt wird, soll an mehreren Standorten im Unternehmen implementiert werden. Es besteht zu Projektbeginn bereits eine Softwarebasis, auf der das Projekt aufsetzen wird. Dabei handelt es sich um eine Webanwendung, die auf einem Server zentral zur Verfügung gestellt wird und an den Standorten über den Browser aufrufbar ist. In Abbildung 5.1 ist eine schematische Darstellung in einem

Schichtenmodell¹¹⁰ zu sehen. Dieser Aufbau entspricht nicht zwingend dem Aufbau des fertigen Produktes, wurde aber gewählt, weil er sich für die Entwicklung des NLP-Systems eignet.

Folgende Annahmen wurden hierzu getroffen:

- 1 Das fertige System wird in der Cloud des Unternehmens implementiert.
- 2 Die Benutzeroberfläche ist unabhängig von der Datenverarbeitung gestaltet.
- 3 Die Orchestration der Komponenten wird von einer Message Queue übernommen.
- 4 Die rohen Dokumente werden in einem File Storage abgelegt.
- 5 Die aus den Dokumenten gewonnenen Daten werden in einer relationalen Datenbank hinterlegt.
- 6 Die Modellparameter werden in einer eigenen Datenbank hinterlegt.

Aus diesen Annahmen ergibt sich als erstes der Vorteil, dass das *Data Processing* zu großen teilen unabhängig von der Benutzeroberfläche erarbeitet werden kann. In Abbildung 5.1 ist am Graphical User Interface, in der Ebene Client, die Anmerkung Django¹¹¹ zu sehen. Bei Django handelt es sich um ein Web

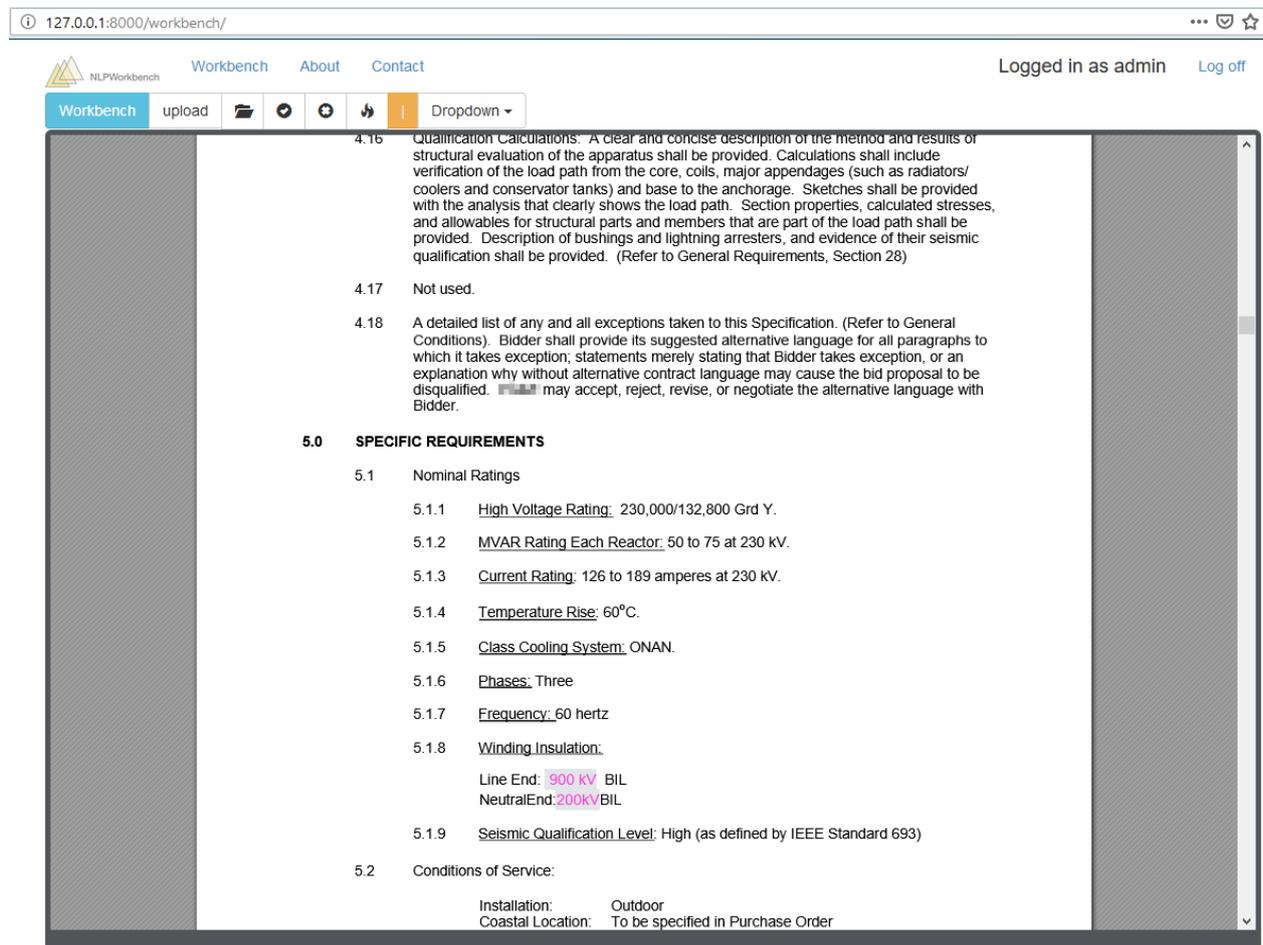


Abbildung 5.2: Benutzeroberfläche der in Django erstellten Testapplikation, die für die Entwicklung verwendet wird. Aufgerufen wurde diese im Webbrowser auf dem Localhost. In Rosa auf grauem Hintergrund hervorgehoben sind Werte, die automatisch erkannt wurden. Quelle: Eigene Darstellung.

Framework, mit dem sich das Backend einer Webanwendung in Python realisieren lässt; das Frontend kann

¹¹⁰Vgl. Buschmann et al. (1996), S. 7 ff..

¹¹¹<https://djangoproject.com>

ebenfalls mit am Server berechneten Daten versorgt werden. Der Grund für die explizite Erwähnung dieses Frameworks in der Abbildung ist, dass es mit wenig Aufwand möglich war, eine einfache Benutzeroberfläche für die Entwicklung einer Webanwendung zu erstellen, in der Benutzerverwaltung und auch der Aufbau und Zugriff auf die Datenbank unter Verwendung von Python als einzige verwendete Programmiersprache programmiert werden konnten. In Abbildung 5.2 ist ein Screenshot der mit Django erstellten Anwendung zu sehen. Für Buttons und die Navigationsleiste wurde das von Twitter entwickelte Bootstrap¹¹² verwendet. Die erstellte Testseite enthält einen Button *upload*, mit dem eine PDF-Datei in einen Ordner geladen werden kann, und der relative Pfad zu dieser Datei wird in einer SQL-Datenbank hinter einer *Projekt-ID* hinterlegt. Der am längsten dauernde Prozess beim Erstellen dieser Testumgebung war es, die PDF-Datei so darzustellen, dass in dieser bestimmte Wörter farbig hervorgehoben werden können. Um die erkannten Begriffe – in Abbildung 5.2 in Rosa auf grauem Hintergrund zu sehen – hervorzuheben, wurde das Programm *pdf2htmlEx*¹¹³ verwendet.

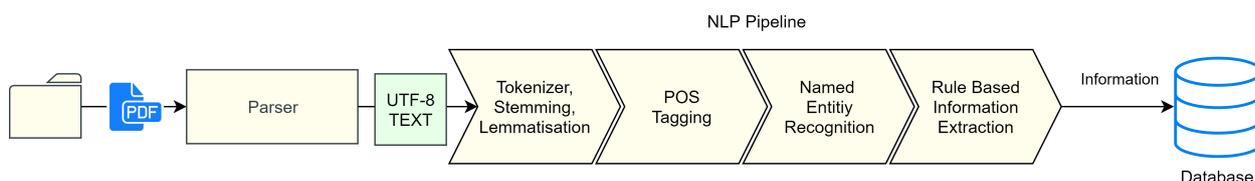


Abbildung 5.3: NLP Pipeline wie diese im Data Processing in Abbildung 5.1 vorkommen kann. Quelle: Eigene Darstellung.

Dieses wandelt das PDF in eine HTML-Datei um, die dem konvertierten PDF entspricht. In der erhaltenen HTML-Datei wurde dann der entsprechende Text mit einem Index und unter Verwendung der Python-Bibliothek *Beautiful Soup*¹¹⁴ an den entsprechenden Stellen mit dem farbigen Markup versehen. Mit dieser Testumgebung ist es somit möglich, ein PDF-Dokument abzuspeichern, einem Projekt zuzuweisen, die Benutzer zu verwalten und das Ergebnis der Datenverarbeitung, die von dieser Ebene unabhängig ist, visuell darzustellen. Die nächste Ebene in Abbildung 5.1 stellt die Programmlogik (engl. Logic) dar. Über die *Message Queue* können die dort implementierten Prozesse gesteuert und koordiniert werden. Eine weitere Aufgabe, die eine Queue hierbei übernehmen kann, ist das Loggen aller Ereignisse. Die Prozesse in dieser Logikebene sind jene, die im weiteren Verlauf dieser Arbeit erforscht werden. Bevor die Daten (engl. Data) aus den PDF-Dateien jedoch verarbeitet werden können, muss der Text aus den Dateien ausgelesen werden. Für das Auslesen der rohen Textdaten aus den Dateien wird das Tool *Apache Tika*¹¹⁵ verwendet. Apache Tika kann PDF-, Word-, E-Mail-, ODF- und über 1.400 weitere Dokumenttypen verarbeiten. In Abbildung 5.1 ist Apache Tika unter dem Element Parser eingezeichnet. Auf der Daten-Ebene sind neben dem Ordner mit den unverarbeiteten Dokumenten zwei Datenbanken eingezeichnet. Der Grund hierfür ist, dass es hinsichtlich der Performance sinnvoll sein kann, die Modellparameter der Machine-Learning-Modelle in einer NoSQL-(Not only SQL)-Datenbank abzuspeichern, während sich für die allgemeine Datenverwaltung eine SQL-Datenbank ausreichend gut eignet. Dieses Auslagern kann als sinnvoll angesehen werden; wird allerdings bedacht, dass ein Word Embedding wie zum Beispiel AllenNLPs ELMO¹¹⁶ ca. 1,6 Gigabyte groß ist, kann es sinnvoll sein, eine entsprechend angepasste Datenbank zu verwenden.¹¹⁷ In Abbildung 5.3 ist eine NLP-Pipeline dargestellt wie diese im Data Processing in Abbildung 5.1 auftreten könnte; mit dem Aufbau dieser wird sich diese Arbeit im Weiteren beschäftigen.

¹¹²<https://getbootstrap.com>

¹¹³<https://github.com/coolwanglu/pdf2htmlEX>

¹¹⁴Vgl. Richardson (2016)

¹¹⁵Vgl. Mattmann/Zitting (2011)

¹¹⁶Vgl. Peters et al. (2018), S. 1-15.

¹¹⁷Vgl. Moniruzzaman/Hossain (2013), S. 11 f.

den unteren von *Lexical Dispersion Plots* gesprochen. In den beiden oberen Diagrammen ist zu sehen, welche 30 Token die häufigsten sind und wie oft diese jeweils vorkommen. Darunter ist – ohne Entfernen der Stopwörter in Gelb und nach Entfernen der Stopwörter in Grün – zu sehen, an welchen Stellen im Datensatz die 20 häufigsten Stopwörter vorkommen. Bei Stoppwörtern handelt es sich im Allgemeinen

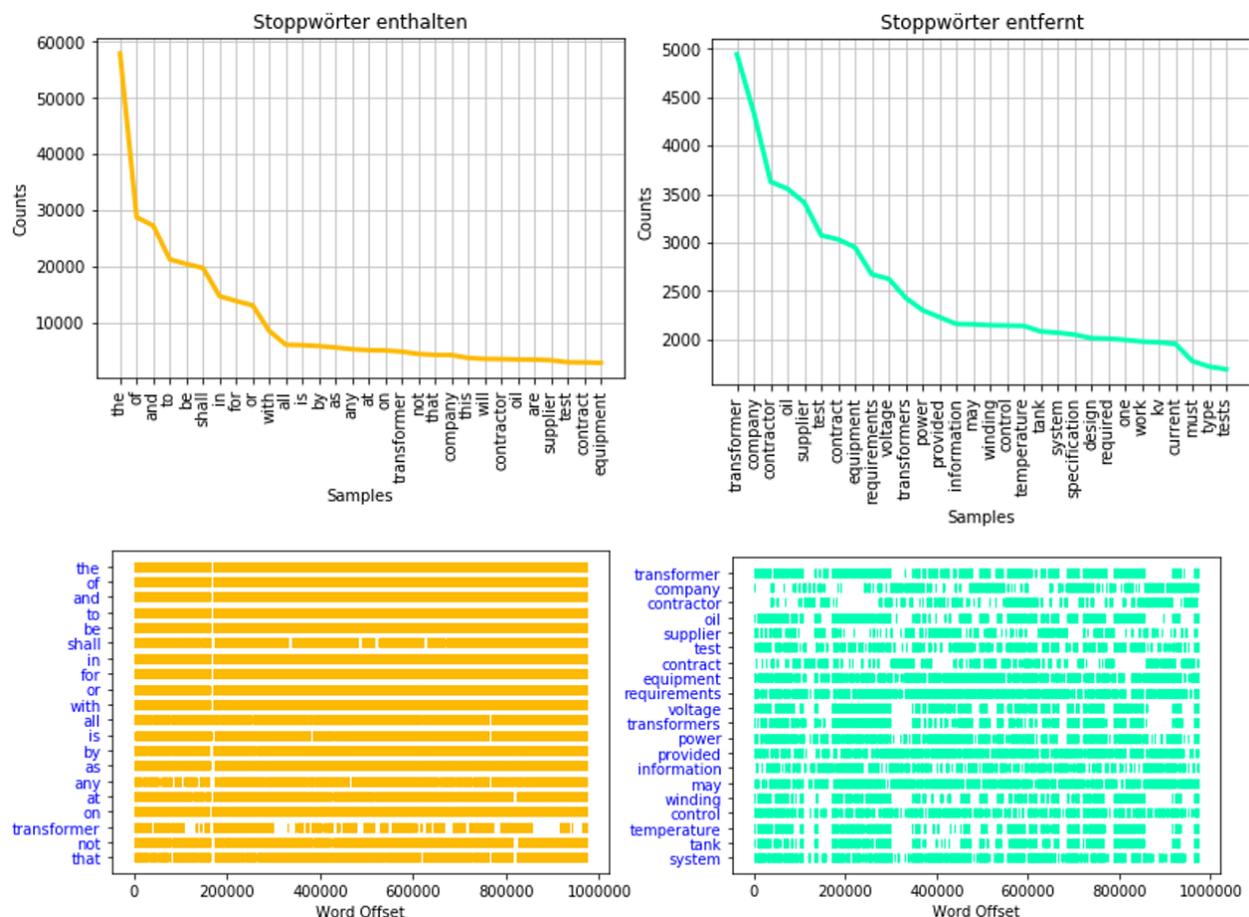


Abbildung 5.5: Auf der linken Seite in Gelb dargestellt sind oben die 30 häufigsten Token im Datensatz ohne das vorherige Entfernen der Stopwörter und rechts oben nachdem die Stopwörter entfernt wurden. Jeweils darunter ist das Auftreten der 20 häufigsten Token im Datensatz dargestellt. Quelle: Eigene Darstellung.

um Wörter, die nur sehr wenig zur Aussage eines Satzes beitragen. Auffällig und in dieser Darstellung gut sichtbar ist allerdings, dass nach dem Entfernen der Stopwörter der Token *not* nicht mehr zu sehen ist. Es wurden die standardmäßigen Stopwörter für Englisch, die in der Python-Bibliothek *NLTK* enthalten sind, verwendet. Der Token *not* ist jedoch ein Token, der einen relevanten Teil zur Bedeutung einer technischen Spezifikation beiträgt. Aus diesem Grund wird für die weitere Bearbeitung eine von Hand sortierte Liste an Stopwörtern verwendet. Eine weitere Aussage, die auf Basis von Abbildung 5.5 getroffen werden kann, ist, dass die Häufigkeit des Auftretens eines Tokens bereits innerhalb der ersten paar Dutzend Token stark abnimmt. Während der Token *transformer* noch nahezu 5.000 Mal auftritt, ist das Auftreten des Token *tests*, mit unter 1.000 Malen, bereits leicht aussagekräftiger. Wird der Token *transformer* im jeweils unteren Diagramm betrachtet, sind trotz seiner relativ hohen Häufigkeit Leerstellen zu erkennen. Wird der Datensatz an diesen Stellen untersucht, ist zu erkennen, dass es sich dabei um Dokumente im Anhang oder im Fall einer Spezifikation um Drosseln (engl. Reactor) handelt. Damit könnte im Weiteren – noch ohne komplexere Berechnungen wie Dokument-Ähnlichkeit auf Basis von Embeddings (z. B. Word2Vec) oder Ähnlichem – bereits anhand der Häufigkeit einzelner Token eine relevante Aussage über ein Dokument getroffen werden. Das Entfernen der Stopwörter an sich erscheint als nützlich. Wird, wie in Abbildung 5.5 zu sehen, der Anstieg der Fragmentierung in den beiden unteren Diagrammen betrachtet, ist nachvollziehbar, dass

die Daten für Mustererkennung auf Token-Basis in vielen Fällen besser geeignet sein können, wenn die Stopwörter entfernt werden.

5.4 Dokumentähnlichkeit

Beim Erstellen von Angeboten, Berechnungen oder Konstruktionen in Unternehmen werden nicht immer neue Ansätze entwickelt. Zumeist wird in internen Datenbanken, Mappen und anderen Erfahrungsspeichern nachgesehen, wie die gestellte Aufgabe in der Vergangenheit bewältigt wurde. Es wurde versucht,

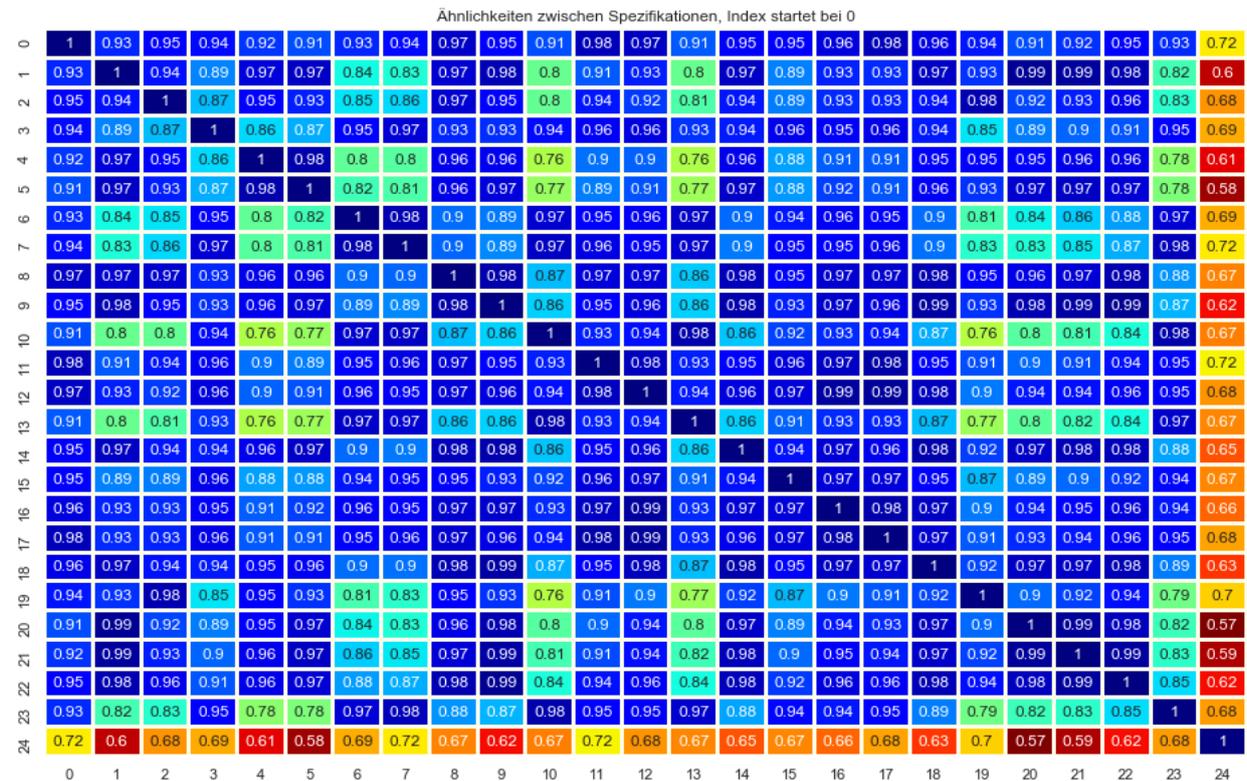


Abbildung 5.6: Korrelationsmatrix, die die Ähnlichkeiten der 25 Beispiele visualisiert. Der Index der Dokumente hat die Basis Null; das bedeutet, Dokument *KUNDE 00* liegt auf Eintrag Nummer 0. In der Diagonale zu sehen ist die maximale Ähnlichkeit von Eins, die jeweils zwischen dem Dokument und sich selbst auftritt. Quelle: Eigene Darstellung.

das prüfen von zwei Dokumente auf Ähnlichkeit, in diesem Beispielen mithilfe von Embeddings auf Dokumentenebene durchzuführen. Zu sehen ist eines der Ergebnisse in Abbildung 5.6. Dabei handelt es sich um eine Korrelationsmatrix. Um diese zu erstellen wurden:

- Alle Zahlen entfernt,
- alle doppelten Leerzeichen entfernt,
- Zeilenumbrüche und Tabulatoren entfernt,
- Stopwörter entfernt und
- Sonderzeichen und bedeutungslose Zeichenfolgen entfernt.

Daraufhin wurden vortrainierte GloVe Embeddings verwendet, um die einzelnen Wörter mit Vektoren zu versehen. Unter Verwendung der Python-Bibliothek *SpaCy*¹¹⁹ wurden die Vektoren für die einzelnen Spe-

¹¹⁹Honnibal/Montani (2019)

zifikationen erstellt und zusammengefasst. Anschließend wurden alle möglichen Kombinationen der Dokumente miteinander hinsichtlich ihrer Ähnlichkeit bewertet. Das Ergebnis der Bewertung der Ähnlichkeit wird dabei in einer Zahl zwischen 0 und 1 ausgedrückt, wobei 0 für keine Ähnlichkeit und 1 für identische Dokumente steht. Diese Kombinationen sind in Abbildung 5.6 zu sehen und der Wert der Ähnlichkeit steht in den entsprechenden Feldern der Korrelationsmatrix. In den in Abbildung 5.6 dargestellten Korrelationen gehen einige der Werte gegen 1, aber es kommen auch Werte nahe der 0,5 vor. Werden die aus den PDF-Dateien extrahierten Daten ohne die in der vorherigen Aufzählung gezeigten Schritte für diese Analyse verwendet, ergeben sich Werte zwischen 1 und 0,98, was keine weitere Aussage zulässt. Ebenfalls erwähnenswert ist, dass Ausreißer, die einen besonders großen Unterschied zu den übrigen Dokumenten aufzuweisen scheinen, ebenfalls zu prüfen sind. In Abbildung 5.6 scheint sich das Dokument mit Index 24 von allen anderen Dokumenten besonders deutlich zu unterscheiden. Bei einer ersten Begutachtung wurde bemerkt, dass es sich um einen normalen Generator-Step-Up-Transformer (GSU) handelt und zudem sowohl die Anzahl der Dokumente als auch die Art des Aufbaus der Dokumente innerhalb der ersten dreitausend Token unauffällig scheint. Bei genauerer Betrachtung wurde festgestellt, dass dieses Dokument im hinteren

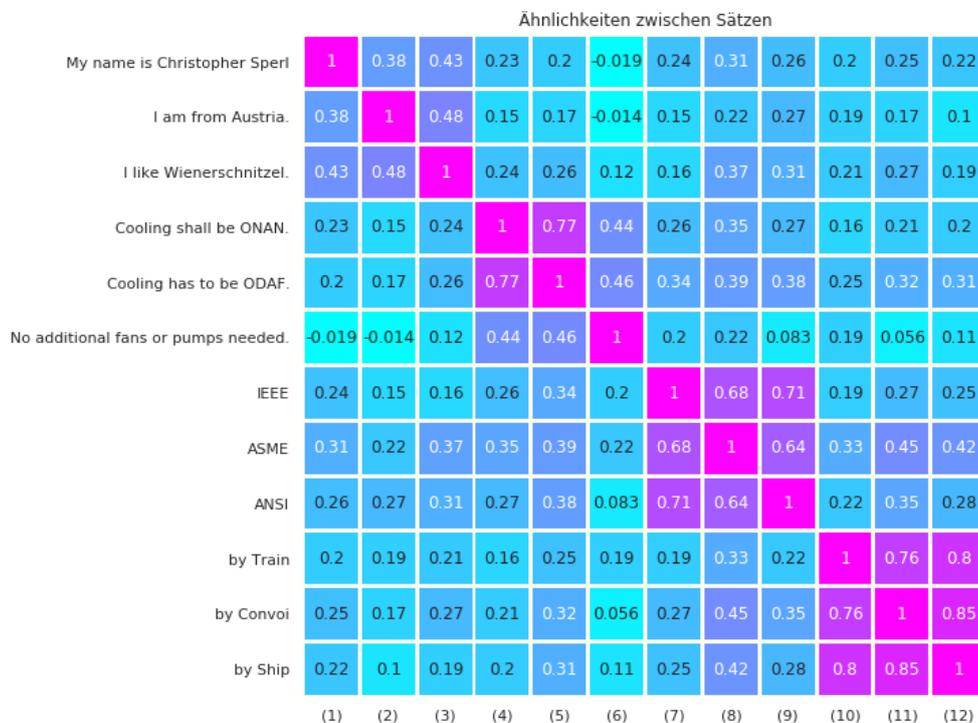


Abbildung 5.7: Korrelationsmatrix, die die Ähnlichkeiten zwischen Sätzen und Satzteilen visualisiert. Die Darstellung wurde unter Verwendung eines *Universal-Sentence-Encoder-LARGE*-Modells erstellt. Quelle: Eigene Darstellung.

Bereich über einen Anhang verfügt, der, anders als die restlichen Dokumente, nicht in Englisch verfasst ist. Wird dieser entfernt, ist auch der, in Abbildung 5.6 augenscheinlich wirkende, Unterschied nicht länger vorhanden. Beim Versuch, herauszufinden, warum Dokumente hohe Korrelation zueinander aufweisen, ergab sich, dass jene Spezifikationen, die vom selben Kunden stammen, mit konstanten Werten über 0,95 zwar relativ gute Werte aufweisen, aber das Vorhandensein bestimmter Anhänge und eines bestimmten Dokumentaufbaus einen wesentlich größeren Einfluss auf diese Korrelationen hat und Werte von 0,99 bei Dokumenten verursacht, die einander komplett verschieden sind. Als Beispiel sind die Dokumente 1 und 21 mit einer Ähnlichkeit von 0,99 zu nennen, bei denen es sich um einen großen Leistungstransformator und um eine Drossel handelt. Aufgrund dieses Sachverhaltes wird angenommen, dass Embeddings auf Dokumentenebene zur Bewertung der Ähnlichkeit nur bei exakter Übereinstimmung eine verlässliche Aussage

zulassen und ansonsten weiterer Optimierung bedürfen würden. In Tabelle 5.1 wird ein grober Überblick über die in Abbildung 5.6 verwendeten Daten vermittelt.

Ein weiterer Ansatz, der betrachtet wird, ist das Anwenden von Embeddings auf extrahierte Informationen aus den Dokumenten. Dabei kann es sich um ganze Sätze, Satzteile und auch einzelne Wörter handeln. In Abbildung 5.7 ist eine Korrelationsmatrix zu sehen, in der dies veranschaulicht werden soll. Erstellt wurde diese unter Verwendung eines *Universal-Sentence-Encoder-LARGE-Modells*¹²⁰, das vom Unternehmen Google auf dessen Webseite Tensorflow Hub¹²¹ zur Verfügung gestellt wird.

Datensatz Überblick									
0	SVC 250	5	SHR 110	10	AUT 180	15	SHR 100	20	AUT 300
1	GSU 600	6	SHR 50	11	SVC 200	16	SHR 100	21	SHR 200
2	AUT 375	7	GSU 160	12	GSU 400	17	AUT 375	22	GSU 150
3	AUT 500	8	GSU 750	13	GSU 315	18	AUT 400	23	GSU 300
4	AUT 330	9	AUT 300	14	SHR 70	19	AUT 515	24	GSU 120

Tabelle 5.1: Überblick über die im Datensatz enthaltenen Spezifikationen. Es sind der Index, gefolgt von der Art der Maschine und danach die Leistung in MVA für Volltrafo (TRA), Spartransformator (AUT) und Maschinentransformator (GSU) sowie in MVA_r für Drossel (SHR) und statischen Blindleistungskompensator (SVC) angegeben.

Um Platz zu sparen, wurde die Bezeichnung der Spalten abgekürzt. Der Satz in Reihe ‚My name is Christopher Sperl‘ ist somit der Spalte (1) zuzuordnen. Anzumerken ist, dass in der Abbildung 5.7 ein Zusammenhang zwischen der Semantik der Sätze und der Korrelation deutlich erkennbar ist. Die Sätze wurden so gewählt, dass diese in drei Gruppen zu drei Sätzen eine semantische Ähnlichkeit zueinander aufweisen sollten. Die Sätze (1) bis (3) sind Sätze über den Autor dieser Arbeit und stehen in keinem Zusammenhang mit Transformatoren. Die Ähnlichkeit dieser drei Sätze zueinander scheint etwas stärker ausgeprägt als zu den restlichen Sätzen. Dies ist zu erwarten, wenn davon ausgegangen wird, dass dieses Modell semantische Zusammenhänge erkennen soll. Die Sätze (4) bis (6) handeln vom Kühlkreislauf eines Transformators. Hier ist zu sehen, wie stark ein ähnlicher Satzbau das Ergebnis beeinflusst, aber auch der dritte Satz weist mit einem Score von rund 0,45 noch immer eine deutliche Korrelation zu den beiden Vorhergehenden auf. Bei (7), (8) und (9) handelt es sich um Normen und bei den letzten drei um Begriffe, die mit dem Transport eines Transformators assoziiert werden sollen. Die Korrelationen der Einträge sind in den Dreiergruppen besonders deutlich, aber teils auch zwischen Transport und Normen teils leicht erhöht. Wie bereits im Vorhinein zu erwarten war, scheint es für das Finden ähnlicher Dokumente nicht von Vorteil, das gesamte Dokument mit einem Word Embedding zu versehen und auf dessen Basis nach ähnlichen Dokumenten zu suchen. Das Suchen nach ähnlichen Dokumenten auf Basis von zuvor aus den Dokumenten extrahierten Werten soll angewendet werden, weil die erhaltenen Korrelationen aus den Modellen durch diese Vorgehensweise eindeutiger interpretierbar zu sein scheinen. Wie das Extrahieren von Daten bewerkstelligt werden kann und welche Aussagen anhand dieser Daten getroffen werden können, wird Thema der folgenden Kapitel sein.

¹²⁰Vgl. Cer et al. (2018a), S. 1-7.

¹²¹<https://tfhub.dev/google/universal-sentence-encoder-large/3>

5.5 Regelbasierte Informationsextraktion

„Regelbasierte Informationsextraktion ist tot! Lang lebe regelbasierte Informationsextraktion!“¹²² – Selbiges in englischer Sprache war der Titel eines von IBM-Mitarbeitern im Jahr 2013 veröffentlichten Artikels. Darin wird beschrieben, dass in akademischen Veröffentlichungen der Jahre 2003 bis 2012 in ca. 75 % der Fälle von auf maschinellem Lernen basierenden Systemen und nur in 3,5 % von regelbasierten Systemen geschrieben wurde. Bei den restlichen Werken ging es um hybride Ansätze. In der Industrie und vor allem bei größeren Unternehmen wurden zu diesem Zeitpunkt, mit einem Anteil von bis zu 67 %, überwiegend rein regelbasierte Systeme verwendet.¹²³

Gegenüberstellung		
	Vorteile	Nachteile
Regelbasierte Systeme	<ul style="list-style-type: none"> - deklarativ - einfach vergleichbar - gut zu warten - einfaches Einbringen von Domänenwissen - Fehler sind einfach zu finden und zu beheben 	<ul style="list-style-type: none"> - heuristisch - hoher Arbeitsaufwand bei vielen Regeln
ML Systeme	<ul style="list-style-type: none"> - trainierbar - wandlungsfähig - geringer manueller Arbeitsaufwand 	<ul style="list-style-type: none"> - benötigt gelabelte Daten - muss für jede Domäne neu trainiert werden - ML Wissen erforderlich - schwer nachvollziehbar

Tabelle 5.2: Gegenüberstellung der Vor- und Nachteile von regelbasierten IE-Systemen und auf maschinellem Lernen basierenden Systemen. Quelle: Chiticariu/Y. Li/Reiss (2013), S. 829. (modifiziert).

Einige Argumente – in Tabelle 5.2 dargestellt – die 2013 bereits valide waren, sind auch im Jahr 2019 noch immer gültig. Das hohe Interesse der Wissenschaft an maschinellem Lernen beschleunigt dessen Weiterentwicklung jedoch maßgeblich. Der Einsatz von maschinellem Lernen schafft einerseits manuellen Aufwand beim Erstellen der Trainingsdaten, andererseits kann auf das manuelle Erstellen der Regeln verzichtet werden. Diese Systeme sind hochgradig wandlungsfähig, wenn eine ausreichende Menge an Golden Samples – damit sind verifiziert richtige Beispieldaten gemeint – vorhanden ist. Ein sehr interessantes Argument für die hier angedachte Anwendung ist, dass das Einbringen von bereits vorhandenem Firmen- und Domänenwissen bei regelbasierten Systemen einfacher zu bewerkstelligen ist. Es soll untersucht werden, wie und ob möglicherweise das Beste aus beiden Bereichen in das System einfließen könnte. Hierzu wird mit dem Erstellen der regelbasierten Systeme, den regulären Ausdrücken, begonnen. Alle Versuche werden an einem frei erfundenen Datensatz, der relevante Teile wiedergibt, durchgeführt und beispielhaft auf den folgenden Seiten wiedergegeben. Quantitative Aussagen werden auf Basis des bereits im vorhergehenden Kapitel verwendeten Datensatzes, der aus 25 realen Spezifikationen besteht, getroffen (siehe hierzu Abbildung 5.1).

5.5.1 Reguläre Ausdrücke zur Informationsextraktion

Reguläre Ausdrücke (RegEx) werden im Folgenden immer zwischen zwei Schrägstrichen, **fett** und auf gelbem Hintergrund dargestellt. Mit dieser Konvention würde der RegEx zum Finden des Wortes *Transformer* das Aussehen **/Transformer/** haben. Erstmals beschrieben wurden RegEx vom Mathematiker und

¹²²Vgl. Chiticariu/Y. Li/Reiss (2013), S. 827-832.

¹²³Vgl. Chiticariu/Y. Li/Reiss (2013), S. 827 ff.

Logiker Kleene,¹²⁴ diese Definition übernahm Thompson (Bell Laboratories) und integrierte diese in einen Texteditor als Syntax für Textsuche.¹²⁵ Später übernahm er selbige Syntax im UNIX Editor *ed* und erstellte das weithin bekannte UNIX-Kommando GREP (Global Regular Expressions Print).¹²⁶ Eine Übersicht über die Syntax von RegEx ist Jurafsky und Martin (2018) zu entnehmen.

Unter Verwendung der Python-Bibliothek *re* (in Python Version 3.7.4) wurde als erstes der Ausdruck `^b[Tt]ransformer\b/` auf KUNDE 12 im Datensatz angewendet. Das Ergebnis dieser Suche ist in Abbildung 5.8 zu sehen.

Python 3.7

```

1  import re
2
3  # lead data
4  document = KUNDE[12]
5
6  # search for pattern
7  list_of_result = re.findall(r'\b[Tt]ransformer\b', document)
8
9  # count occurrence
10 cnt = len(list_of_result)
11 print('Das Wort << TRANSFORMER >> kommt {} mal vor.'.format(cnt))

out: Das Wort << TRANSFORMER >> kommt 402 mal vor.
```

Abbildung 5.8: Das Finden und Zählen eines Wortes in Text mittels RegEx in Python. Quelle: Eigene Darstellung.

Wird beachtet, dass diese – noch sehr einfach gehaltene – RegEx-Suche lediglich nach zwei Wörtern – nämlich Transformer und transformer – sucht, erschließt sich der Nutzen im Vergleich zu anderen Suchsystemen (wie in Windows meist *Strg + F*) zunächst nicht. Alles was bis hierhin an Information gewonnen werden kann, ist die *Entität* Transformer. Wird die gestellte Aufgabe von ‚finde das Wort Transformer‘ zu ‚finde ein Datum und/oder Uhrzeit‘ wird der Nutzen jedoch schnell ersichtlich. Schreibweisen für Datum und Uhrzeit variieren stark von Dokument zu Dokument und die Zeichenfolge *18/11/86* unterscheidet sich von der Zeichenfolge *5.7.2007* in jedem einzelnen Zeichen. Mit regulären Ausdrücken bieten sich hier beispielsweise folgende Möglichkeiten:

`/(\d+\d+\d+)/` würde das Datum *18/11/1968* finden, aber auch jedes weitere, das der Formatierung *dd/mm/yy* oder auch *mm/dd/yy* entspricht.

`^d{1,2}\.d{1,2}\.d{2,4}/` findet das Datum *5.7.2007*, würde aber auch *05.07.2007* oder *5.07.07* finden. Sollen Datum und Uhrzeit gefunden und mehrere Optionen für die Darstellung berücksichtigen werden, beginnen auch RegEx an Länge zu gewinnen:

`/([0-9]{1-9}|1[012])[-/](0[1-9]|[12][0-9]|3[01])[-/](0[9-4]{\s}((0[1-9]|1[012])\:(0[5-9]|(0[5-9])\s))([AM|PM]){2,2}))?/`

Dieser Ausdruck erkennt *11/18/1986 01:44:00 PM* oder *12-12-2000 11:42:42 AM* oder auch nur *18/11/1986* allein, ohne eine darauf folgende Uhrzeit. Auch wenn obiger Ausdruck kompliziert erscheint, handelt es sich um eine gute Methode, Datum und Uhrzeit in Texten zu finden. Das Wissen über die Entitäten Uhrzeit und Datum kann im weiteren Verlauf beim Finden von zum Beispiel Lieferterminen in Spezifikationen nützlich sein. Ein weiteres prominentes Beispiel für den Einsatz von RegEx ist:

`/([0-9A-Za-z](-\w)*[0-9A-Za-z])*@[([0-9A-Za-z](-\w)*[0-9A-Za-z]\.)+[A-Za-z]{2,10})/` dieser würde E-Mail-Adressen wie *Beispiel@Beispiel.at* oder *christopher.sperl@edu.campus02.at* finden. Dies kann ebenfalls

¹²⁴Vgl. Kleene (1951)

¹²⁵Vgl. Thompson (1968), S. 419-422.

¹²⁶Vgl. Pugh (1994)

eine *Entität* darstellen, die sowohl für das Finden ähnlicher Dokumente als auch für Informationsextraktion an sich von Interesse ist. Für das Suchen von starren Mustern wie Telefonnummern, Email-Adressen, Datum/Uhrzeit und Zahl-Einheit-Kombinationen (z.B. 750 kV) erwies sich die Nutzung von RegEx als praktikabel. Bei komplexeren Zusammenhängen, wie beispielsweise dem Finden von Kundenanforderungen, ist das Bewältigen der Aufgabe mit diesen allein wenig praktisch.

5.5.2 Programmbibliotheken zum Formulieren der Regeln

Vielmehr eine Erweiterung als eine Alternative zu regulären Ausdrücken stellen Programmbibliotheken dar. Diese bieten eine domänenspezifische Syntax und meist auch erweiterte Verfahren zum Formulieren der gewünschten Regeln. Als erstes Beispiel für eine solche Bibliothek soll Apache UIMA Ruta¹²⁷ (Rule-based Text Annotation) genannt sein. Bei der Unstructured Information Management Architecture (UIMA) handelt es sich um ein im Jahr 2005 von IBM gestartetes Projekt, das seit 2006 von der Apache Software Foundation betreut wird. Ziel des Projektes ist es, ein standardisiertes Framework zum Erstellen von Anwendungen zur Verarbeitung unstrukturierter Informationen, wie beispielsweise natürlichsprachlichem Text, zu bieten.¹²⁸ UIMA Ruta wurde in den Programmiersprachen C++ und Java entwickelt und wird in UIMA Pipelines in Java implementiert. Die Bibliothek wird als Ruta Workbench in die Eclipse-Entwicklungsumgebung implementiert, in der Regeln definiert und die Ergebnisse dieser als farbige Hervorhebung direkt im Text angezeigt werden können. Wie eine Regel, die Ruta verwendet, aussehen könnte, ist in Abbildung 5.9 zu

```

1 (NUM (("," | ".") NUM)*)
2     {-> AmountNumber };
3 (AmountNumber SPACE? CurrencyUnit)
4     {-> Money};

```

RUTA

Abbildung 5.9: Beispiel für eine Regel zum Finden von Währungsmengen mit Ruta. Gefunden werden würde zum Beispiel 31,50 €. Dabei ist das Leerzeichen zwischen Zahl und Einheit optional. Der Dezimalpunkt kann Komma oder Punkt sein. Quelle: Kluegl et al. (2016), S. 32. (modifiziert)

sehen. Bei einer einfachen Regel wie in dieser Abbildung ist der Vorteil einer Bibliothek zur Verwendung von RegEx im ersten Ansatz unter Umständen lediglich in der Zuweisung einer Extraktion zu einem Keyword zu erkennen (Zeilen 2 und 4 in Abbildung 5.9). UIMA Ruta ist jedoch weitaus vielschichtiger, als dieses Minimalbeispiel erkennen lässt. Weil die Testumgebung für die Darstellung der Dokumente in der Programmiersprache Python erstellt wurde, soll als zweites Beispiel nach UIMA Ruta die Python-Bibliothek *spaCy* verwendet werden. Diese besitzt einen sogenannten *Rule-based Matcher*, der ähnliche Funktionalitäten wie die Erstgenannte Bibliothek besitzt.

```

1 pattern = [{'LIKE_NUM': True},
2           {'IS_SPACE': True, 'OP': '?'},
3           {'POS': 'SYM'}]

```

Python 3.7 spaCy

Abbildung 5.10: Beispiel für eine Regel zum Finden von Währungsmengen mit *spaCy*. Quelle: Eigene Darstellung

In Abbildung 5.10 ist zu sehen, wie die in Abbildung 5.9 erstellte Regel in *spaCy* umgesetzt werden könnte. Ähnlich, auch zu RegEx, ist hierbei die Verwendung des Zeichens „?“ das das null- oder einmalige Auftreten des Leerzeichens beschreibt. Der große Unterschied zwischen der Verwendung von RegEx und der

¹²⁷Vgl. Kluegl et al. (2016), S. 1-40.

¹²⁸Vgl. Ferrucci/Lally (2003), S. 67-74.

Verwendung von Bibliotheken wird in Abbildung 5.10 Zeile 3 ersichtlich. Ebenso wie Apache Rute kann auch *spaCy* den POS-Tag des Tokens in das Suchmuster miteinbeziehen. In diesem Beispiel steht `{'POS': 'SYM'}` für den Part of Speech Tag *Symbol*. Der Vorteil gegenüber der in Abbildung 5.9 verwendeten Variante ist, dass die Liste von Währungseinheiten *Currency Unit* nicht extra formuliert werden muss. Stattdessen kann beispielsweise ein Conditional-Random-Field-Modell auf das Erkennen der Symbole trainiert werden. Der direkte Nachteil den eine dieser beispielhaften Regeln mit sich bringt, ist, dass auch „10 %“ erkannt werden würde. Jene Eigenschaften eines Token, nach denen ein Extraktionsmuster mit *spaCy* erstellt werden kann, sind in Tabelle 5.3 zu sehen. Während bei Eigenschaften wie *LOWER* noch keine weitere Intelligenz benö-

Attribut	Typ	Beschreibung
ORTH	UNICODE	Der exakte Aufbau des Textes.
TEXT	UNICODE	Der Text des Tokens.
LOWER	UNICODE	Der kleingeschriebene Text des Tokens.
LENGTH	INT	Die Anzahl der Zeichen im Token.
IS_ALPHA, IS_ASCII, IS_DIGIT	BOOL	Der Token besteht aus Text, ASCII Zeichen, Zahlen.
IS_LOWER, IS_UPPER, IS_TITLE	BOOL	Der Text des Tokens besteht aus Kleinbuchstaben, Großbuchstaben, der erste Buchstabe ist großgeschrieben.
IS_PUNCT, IS_SPACE, IS_STOP	BOOL	Der Token ist Punkt, Leerzeichen oder Stoppwort.
LIKE_NUM, LIKE_URL, LIKE_EMAIL	BOOL	Der Token ist numerisch, Internetadresse, E-Mail-Adresse.
POS, TAG, DEP, LEMMA, SHAPE	UNICODE	Der einfache und erweiterte Part Of Speech Tag, Dependency Tag, Lemma oder Form des Tokens.
ENT_TYPE	UNICODE	Entität des Tokens.
_	DICT	Liste mit Index.

Tabelle 5.3: Mögliche Tokeneigenschaften, die in *spaCy* v2.1 im Rule-Based-Matcher verwendet werden können. Quelle: Honnibal/Montani (2019) (modifiziert).

tigt wird, weil hierbei lediglich alle Buchstaben in Kleinbuchstaben umgewandelt werden müssen, benötigen Eigenschaften wie *POS* oder vor allem *ENT_TYPE* ein Machine-Learning-(ML)-Modell im Hintergrund, auf dessen Basis die Attribute der entsprechenden Token bestimmt werden. Es handelt sich also bei diesem System um einen hybriden Ansatz zwischen ML und regelbasierter Informationsextraktion. Die Qualität der Erkennung hängt also zu erheblichen Teilen vom zugrunde liegenden ML-Modell ab. Für die Eigenschaft *POS* ist es nicht unbedingt erforderlich, ein Modell von Null weg zu trainieren. Stattdessen können für viele gängige Sprachen bereits vortrainierte Modelle verwendet werden. Lediglich für Aufgaben wie NER sind diese Modelle zumeist nicht ausreichend, weil sie zumeist auf allgemeine Daten trainiert werden und somit für spezifische Domänen zu allgemein gehalten sind. Sowohl UIMA Ruta, *spaCy* oder auch andere Systeme wie zum Beispiel ANNIE¹²⁹, werden meist nicht als Standalone, sondern in Verbindung mit einem oder mehreren Sprachmodellen verwendet, auf deren Basis Informationen wie Part Of Speech oder Entität der Token bereitgestellt werden. Ein Beispiel für die Verwendung eines solchen hybriden Systems soll im folgenden Abschnitt veranschaulicht werden.

¹²⁹Vgl. Cunningham et al. (2002)

5.6 Informationsextraktion mit hybridem System

In diesem Abschnitt soll ein Beispiel für regelbasierte Informationsextraktion auf Basis eines hybriden Systems gezeigt werden. Dies bedeutet, dass das System einerseits auf Regeln, die kein ML-System benötigen, andererseits aber auch auf Informationen eines ML-Modells zurückgreifen kann. Für das Beispiel wird der in Abbildung 5.11 gezeigte Text verwendet. Es handelt sich um einen fiktionalen Text, in den Informationen, die es zu extrahieren gilt, eingebettet wurden.

The very first thing to find is a Date like 18.11.1986, and the second thing is an e-mail address like example@example.at.

Also very interesting are Norms like:

IEC 61181, IEC 6076-1, IEC 60935, EN 61347-2-13, UL 1310, ISO 9001:2018, ISO 26300, ANSI C57.13, ANSI/IEEE C57.19.00, ASTM D 3487, NEMA TR-1.

Other requirements refer to the rating of the transformer like 100MVA or 100 MVA or on Shunt Reactors one can ask for 75 MVAr. Different physical values can be 380 kV or maybe 110kV. The noise a transformer causes often shall not exceed a specific value, for example, 72 dB. Also very interesting 10 €/kV or 10.000 €.

Abbildung 5.11: Textpassage mit Informationen über Trafos, eingebettet in eine aus einer PDF-Datei erstellten HTML-Datei. Die Informationen werden mit der Python-Bibliothek *Beautiful Soup 4* aus dem HTML ausgelesen und in Python verarbeitet. Quelle: Eigene Darstellung.

Der Text wurde mithilfe von MS Word erstellt und als PDF-Datei abgespeichert. Um den Text und die Position des Textes in der Webumgebung möglichst einfach anzeigen zu können, wurde die PDF-Datei mithilfe von pdf2htmlEx in eine HTML-Datei umgewandelt und der Text aus der Datei unter Verwendung der Python-Bibliothek Beautiful Soup 4 aus dem HTML-Dokument ausgelesen. Anschließend wurde die Position des Textes im Originaldokument auf dem extrahierten Text mithilfe eines Python Dictionaries als Index zugeordnet, um in der Ausgabe diesen dann über seine Position in der HTML-Datei mittels Cascading Style Sheets (CSS) hervorzuheben. Bei dieser Methode handelt es sich um eine von zwei Methoden, die im Laufe des Projektes verwendet wurden, um Informationen in Dokumenten für die BenutzerIn hervorzuheben. Der Vorteil dieser Methode ist die Einfachheit der Umsetzung, die direkt in Python, mithilfe eines Dictionary, vorgenommen wurde. Zu beachten ist, dass bei dieser Methode das Erstellen der HTML immer auf dieselbe Art und Weise zu erfolgen hat, um eine allgemeingültige Zuordnung des Index zu gewährleisten. Das Zuordnen des Index auf den im HTML eingebetteten Text ist hierbei der herausfordernde Part. Ist dieser bewältigt, kann der aus dem HTML extrahierte Text direkt in spaCy eingelesen und verarbeitet werden.

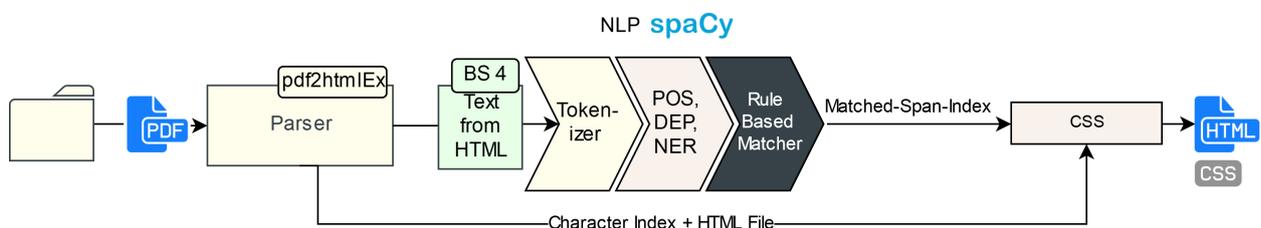


Abbildung 5.12: Darstellung des verwendeten Ablaufs. Das PDF-Dokument wird in eine HTML-Datei umgewandelt aus der der Text extrahiert und indiziert wird. Über eine NLP-Pipeline werden die Informationen extrahiert und in der HTML-Datei mit CSS hervorgehoben. Quelle: Eigene Darstellung.

Der Rule-Based-Matcher der Bibliothek spaCy liefert danach direkt jene Positionen, an denen der Text hervorzuheben ist. In Abbildung 5.18 ist ein Screenshot des Textes mit Hervorhebungen für den Anwender zu sehen. In Abbildung 5.12 ist der Ablauf schematisch dargestellt. Die Flexibilität beim Hervorheben von

Text, die CSS liefert, ist ein weiterer Vorteil dieser Methode. Aufgrund der Tatsache, dass der Text nach der Extraktion ohne weitere Vorverarbeitung in die NLP-Pipeline geladen wurde, ist es für das NLP-System schwierig, die Konsistenz der Indexierung zu erhalten.

```

1  # Blue Pattern
2  pattern = [
3      {'TEXT': {'IN': ['IEC', 'EN', 'ANSI', 'UL', 'ISO', 'ANSI', 'IEEE',
4          ... 'NEMA', 'ASTM', 'ASTM D', ]}},
5      {'ORTH': '/', 'OP': '*'},
6      {'TEXT': {'IN': ['IEC', 'EN', 'ANSI', 'UL', 'ISO', 'ANSI', 'IEEE',
7          ... 'NEMA', 'ASTM', ]}, 'OP': '?'},
8      {'IS_SPACE': True, 'OP': '?'},
9      {'IS_SPACE': False},]
10
11 matcher.add('BLUE_PATTERN', None, pattern)

```

Python 3.7 spaCy

Abbildung 5.13: Regel zum Finden von Normen. Das Keyword *IN* signalisiert das Folgen einer Liste, die Kandidaten bereitstellt. Über den Befehl *add* wird zum *Matcher* die Regel hinzugefügt. Quelle: Eigene Darstellung

Die Regel, die verwendet wird, ist in Abbildung 5.13 zu sehen. Wann immer der *Matcher* der Bibliothek *spaCy* eine Übereinstimmung mit einem Token findet, wird dieser in Abbildung 5.13 in der Farbe Blau hervorgehoben. Wird der Aufbau dieses Suchmusters betrachtet, fällt auf, dass es auch mit RegEx realisierbar gewesen wäre. Als Definition dient jener Teil, der in der Python-Liste *pattern* in den Zeilen 2 bis 9 definiert wurde. In Zeile 11 wird dieses dann über *matcher.add* zum Suchmuster *BLUE_PATTERN* hinzugefügt. In Abbildung 5.13, Zeile 3 und 4, wird definiert, dass nach jenem Text zu suchen ist, der in der Liste *IN* definiert wurde. Danach kann, um Kombinationen wie zum Beispiel *ANSI/IEEE* zu berücksichtigen, ein *,* kommen; mithilfe des Operator *'OP': '?'* wird festgelegt, dass dieser null- oder einmal vorkommen kann. Die Zeilen 6 und 7 sind nahezu identisch mit den Zeilen 3 und 4, aber ebenfalls als optional mit 0 oder 1 Vorkommen deklariert.

```

1  # Red Pattern
2  pattern_1 = [{'LIKE_NUM': True},
3              {'IS_SPACE': True, 'OP': '?'},
4              {'TEXT': {'IN': ['MVA', 'MVAr', 'kV', ]}},]
5
6  pattern_2 = [{'ENT_TYPE': 'MONEY'}]
7
8  matcher.add('RED_PATTERN', None, pattern_1)
9  matcher.add('RED_PATTERN', None, pattern_2)

```

Python 3.7 spaCy

Abbildung 5.14: Regel zum Finden von Kenngrößen und Geldbeträgen. In *pattern_2* wird dem *Matcher* ein Ergebnis der NER als Suchbegriff übergeben. Quelle: Eigene Darstellung

Um darauffolgend die komplexen, in Normen auftretenden Muster hinreichend abzubilden, wurde dem *Matcher* in den Zeilen 8 und 9 mitgegeben, dass ein Leerzeichen nach dem in den Listen deklarierten Textteilen optional ist und jedes weitere Auftreten eines Leerzeichens das Muster beendet. Wird das Ergebnis in Abbildung 5.18 betrachtet, fällt auf, dass das Muster am Zeichen *,* scheitert. Ein weiteres Problem sind Zeile-

numbrüche, weil diese sich von gewöhnlichen Leerzeichen unterscheiden. Besonders interessant ist, dass bei *ASTM D 3487* die Zahl nicht erkannt wurde. Der Grund hierfür wurde vorsätzlich in den Text in Form eines halben, gefolgt von einem normalen Leerzeichen in den Text eingebaut. An diesen Fehlstellen sind bereits Herausforderungen erkennbar, die das manuelle Erstellen der Regeln mit sich bringt. Als nächstes Beispiel wurde, in Abbildung 5.14 zu sehen, ein Suchmuster zum Finden von Kenngrößen wie physikalischen Einheiten und Geldbeträgen erstellt. Die physikalischen Kenngrößen wurden als Kombination einer Zahl, einem optionalen Leerzeichen und der in einer Liste vermerkten Einheit definiert. Anzumerken ist, dass das in Zeile 2 verwendete *LIKE_NUM* auch geschriebene Zahlen wie *two*, *three*, *four* erkennen würde. Das Wort *one* im Text wird also von diesem ebenfalls erkannt und nur mangels nachfolgender Einheit nicht hervorgehoben. Interessant ist, dass weder *100MVA* noch *110kV* von diesem Muster erkannt werden. Um dies zu verstehen, muss der Dependency Tree des Textes betrachtet werden. Dieser wurde mit der spaCy-Bibliothek DisplaCy erstellt und zeigt POS-Tag und DEP zwischen den einzelnen Token, zu sehen in Abbildung 5.15.

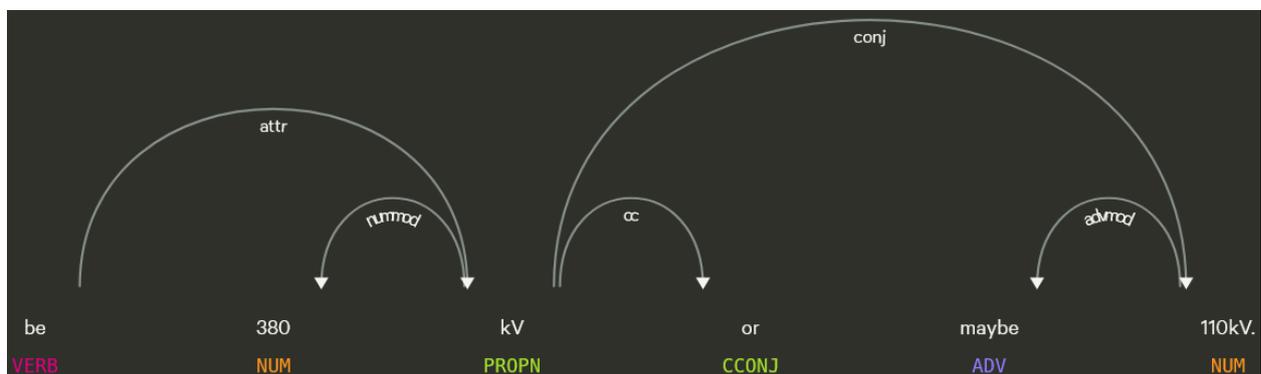


Abbildung 5.15: Dependency Tree einer Teilmenge des Textes. Quelle: Eigene Darstellung mit DisplaCy.

In Abbildung 5.15 ist zu sehen, dass das Modell (spaCy : en_core_web_md) den Token *110kV* als eigenes Wort mit POS-Tag *NUM* erkennt. Damit erweist sich in diesem Fall die Intelligenz des Modells als Nachteil. Selbiges gilt für den Token *100MVA*. Dieses Verhalten ist wenig verwunderlich, denn beim verwendeten Modell handelt es sich um ein allgemeines, nicht speziell an die Anwendung angepasstes und mit ca. 97 MB Größe mittelgroßes Modell.

```

1 # Yellow Pattern
2 pattern = [{'POS': 'VERB'}]
3
4 matcher.add('YELLOW_PATTERN', None, pattern)

```

Python 3.7 spaCy

Abbildung 5.16: Regel, die Zeitwörter anhand des POS-Tags hervorhebt. Quelle: Eigene Darstellung

In Abbildung 5.16 wurde festgelegt, dass das Modell die Zeitwörter hervorheben soll. Wird der Text in Abbildung 5.18 betrachtet, ist zu erkennen, dass diese sehr zuverlässig erkannt werden. Das Zuordnen der POS-Tags wird, zum jetzigen Stand beispielsweise im der sehr bekannten Python-Bibliothek *nltk*, sehr oft über Conditional-Random-Field-Modelle bewerkstelligt. Für diese Aufgabe existieren große Mengen an gelabelten Text, was die Aufgabe gut zu bewerkstelligen macht. Für die Verwendung von POS-Tags ist es somit meist nicht notwendig, ein eigenes Modell zu trainieren.

In Abbildung 5.17 wird nun abschließend eine Regel definiert, die einerseits E-Mail-Adressen erkennt und in einer zweiten Regel einen RegEx für das Finden von Datumsangaben in das Muster integriert. Aus diesem

```

1 # Green Pattern
2 pattern_1 = [{'LIKE_EMAIL': True},]
3 pattern_2 = [{'ORTH': {'REGEX': '^\\d{1,4}[-\\.]\\d{1,2}[-\\.]\\d{1,4}$'}}],]
4 # [{'ENT_TYPE': 'DATE'},]
5
6 matcher.add('GREEN_PATTERN', None, pattern_1)
7 matcher.add('GREEN_PATTERN', None, pattern_2)

```

Abbildung 5.17: Regel, die Mail-Adressen mittels in spaCy eingebauter Funktion findet. Mit `pattern_2` wird mittels RegEx das Muster für ein Datum definiert, auch hier wäre eine Suche nach der Entität `DATE` zum selben Ergebnis gekommen. Quelle: Eigene Darstellung

Grund kann behauptet werden, dass Frameworks nicht anstatt RegEx zum Einsatz kommen, sondern diese integrieren.

The very first thing to find is a Date like 18.11.1986, and the second thing is an e-mail address like example@example.at.

Also very interesting are Norms like:

IEC 61181, IEC 6076-1, IEC 60935, EN 61347-2-13, UL 1310, ISO 9001:2018, ISO 26300, ANSI C57.13, ANSI/IEEE C57.19.00, ASTM D 3487, NEMA TR-1.

Other requirements refer to the rating of the transformer like 100MVA or 100 MVA or on Shunt Reactors one can ask for 75 MVAr. Different physical values can be 380 kV or maybe 110kV. The noise a transformer causes often shall not exceed a specific value, for example, 72 dB. Also very interesting 10 €/kV or 10.000 €.

Abbildung 5.18: Textpassage mit hervorgehobenen Informationen über Trafos. Die Hervorhebungen wurden mittels CSS erstellt. Quelle: Eigene Darstellung.

Wie nun in Abbildung 5.18 zu sehen ist, können mithilfe einer Programmbibliothek und des Einsatzes einer Kombination aus regelbasiert und Machine Learning mit relativ wenig Code bereits einige Token erkannt und mit einem Label versehen werden. Nicht zu übersehen ist aber andererseits die Abhängigkeit von der Güte des Modells, die letzte, scheinbar einfach zu erkennende Entität `10.000 €` wurde aufgrund der Modellgüte nicht erkannt. Während POS- und DEP-Tags bereits bei vortrainierten Modellen respektable Ergebnisse liefern, sind diese für einen domänenspezifischen Einsatz zumeist erst nach entsprechendem Training des Modells ausreichend genau. SOTA POS-Tagger liefern einen Accuracy Score von über 97%.¹³⁰ Neben regelbasierten und hybriden Ansätzen gibt es auch Ansätze, die rein auf maschinellem Lernen basieren. Bei diesen Modellen kommt die charakteristischste Eigenschaft des maschinellen Lernen zum Tragen. Das Modell übernimmt die Formulierung der Regeln anhand von Beispielen, ohne, dass diese vom Menschen zu formulieren sind. Ein solcher Ansatz, der für das Auffinden und Extrahieren von Informationen nützlich sein könnte, wird im nächsten Kapitel behandelt.

¹³⁰Vgl. Akbik/Blythe/Vollgraf (2018), S. 1638-1649.

5.7 Open Information Extraction

Neben den im vorherigen Kapitel vorgestellten Hybriden aus regelbasierter Informationsextraktion und maschinellem Lernen kann auch maschinelles Lernen allein zur Informationsextraktion verwendet werden. Ein Ansatz hierfür ist es, ein NER-Modell zu trainieren, das jeweils die gesuchte Information als Entität erkennt. In der Literatur wird gerne von Open Information Extraction (OIE) und Open Relation Extraction (ORE) gesprochen; bei diesen Methoden werden Informationen anhand der Beziehungen der Token/Entitäten zueinander gewonnen. Diese Systeme existieren, wie es bei den meisten Machine-Learning-Bibliotheken der Fall ist, vorrangig für die englische Sprache. Als Beispiel für ein deutsches System soll deshalb GerIE¹³¹ genannt sein.

ClausIE

Input: The very first thing to find is a Date like 18.11.1986, and the second thing is an e-mail address like example@example.at.

Output:

```

1 # Semantic graph: [Date/NN
2 #           nsubj:[thing/NN
3 #           det:The/DT
4 #           amod:[first/JJ adumod:very/RB]
5 #           infmod:[find/VB aux:to/TO]]
6 #           cop:is/VBZ
7 #           det:a/DT
8 #           prep:[like/IN pobj:18.11.1986/CD]
9 #           cc:and/CC
10 #          conj:[address/NN
11 #              nsubj:[thing/NN det:the/DT amod:second/JJ]
12 #              cop:is/VBZ
13 #              det:an/DT
14 #              amod:e-mail/JJ
15 #              prep:[like/IN pobj:example@example.at/NN]]]
16
17 # Detected 2 clause(s).
18 # - SVC (C: Date@9, V: is@7, S: thing@4, A?: like@10)
19 # - SVC (C: address@20, V: is@17, S: thing@16, A?: like@21)
20
21 "The very first thing to find" "is" "a Date like 18.11.1986"
22 "The very first thing to find" "is" "a Date"
23 "the second thing" "is" "an e-mail address like example@example.at"
24 "the second thing" "is" "an e-mail address"

```

Abbildung 5.19: Beispiel zu ClausIE in dem der erste Satz aus dem Beispieltext des vorherigen Kapitels analysiert wird. Das Framework erkennt dabei die Beziehungen der Token untereinander und die darauf basierenden Aussagen. Quelle: Eigene Darstellung

Ein Vorteil dieser unüberwachten Systeme ist es, dass sie keine gelabelten Daten zur Informationsextraktion benötigen. Um herauszufinden, ob und inwiefern ein solches System in die angestrebte Lösung zu inte-

¹³¹Vgl. Bassa/Kröll/Kern (2018), S. 2-24.

grieren ist, wurde die Software ClausIE¹³² verwendet. Alternativen hierzu wären beispielsweise die Systeme GerIE,¹³³ Kraken,¹³⁴ TextRunner,¹³⁵ WOE^{POS},¹³⁶ R2A2,¹³⁷ OLLIE¹³⁸ und Wanderlust¹³⁹. Die Hauptziele von OIE-Systemen wie ClausIE sind Domänenunabhängigkeit, Skalierbarkeit für große Datenmengen und das unüberwachte Extrahieren von Informationen. Um Informationen eigenständig aus den Texten zu gewinnen, werden Tripel gebildet, die, wie in Abbildung 5.19 in den Zeilen 21–24 zu sehen ist, Informationen über den zugrunde liegenden Text wiedergeben.¹⁴⁰ Zu erwähnen ist, dass es sich an dieser Stelle bei OIE bereits um Tripel handelt, die auf für die Sprache typische Muster und Informationen enthalten, jedoch der Schritt der Relation Extraction ORE noch nicht eingeflossen ist. Zu sehen sind die Muster in den Zeilen 18 und 19 der Abbildung 5.19. SVC steht hierbei für Subject-Verb-Complement, und stellt eine der in ClausIE verwendeten Clauses dar. Alternative Clauses in ClausIE sind emphSV, SVA, SVO, SVOO, SVOA und SVOC mit:

- S als Subject,
- V als Verb,
- C als Complement,
- O als Object und
- A als Adverb.

Wie ebenfalls in Abbildung 5.19 zu sehen ist, sind die Aussagen, die OIE zu einem Satz zulässt, durchaus relevant. Eine mögliche Anwendung dieser Technik wäre gewesen, die extrahierten Informationen dem Benutzer direkt zugänglich zu machen. Wird die Methode auf ganze Spezifikationen angewendet, ergibt sich eine Informationsmenge, die das Dokument selbst in der Länge erheblich übertrifft. Deshalb wird OIE vorerst nicht direkt für Ausgaben an den Benutzer verwendet werden.

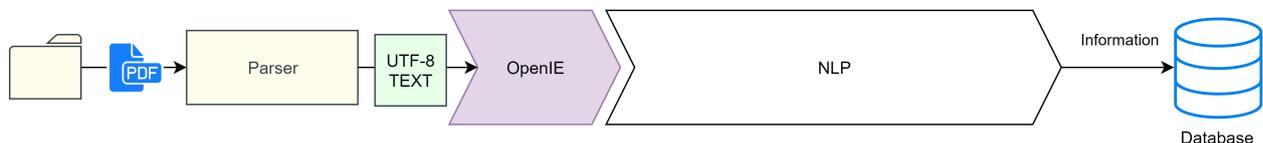


Abbildung 5.20: NLP Pipeline Schema mit OIE als Preprocessing um weiteren Anwendungen als zusätzliche Daten-Grundlage zu dienen. Quelle: Eigene Darstellung.

Von Interesse wird jedoch sein, ob und wie Informationen, die mittels OIE gewonnen werden, wie in Abbildung 5.20 angedeutet, für die weitere Datenverarbeitung von Nutzen sein könnten. Bis dato scheint OIE Informationen bereitzustellen, aber das Anwenden auf die realen Testdaten ergab eine kaum überschaubare Menge an Daten, die für den Nutzer so nicht weiter nützlich zu sein scheint. Um ein qualitatives Bild von diesem Umstand zu bekommen, soll als Beispiel genannt sein, dass ein Beispiel bestehend aus Rund 83.000 einzelnen Zeichen in der Eingabe in einer Ausgabe von knapp 800.000 Zeichen resultierte. Eine solche Datenmenge ist für einen Rechner gut auswertbar und könnte eine zusätzliche Grundlage für weitere Schritte darstellen. Eine Methode, die versucht Informationen unmittelbarer aus Text zu extrahieren ist Machine Reading Comprehension. Diese Aufgabe wird als Teilgebiet des Question Answering verstanden und in den folgenden Abschnitten behandelt.

¹³²Vgl. Del Corro/Gemulla (2013), S. 355-366.

¹³³Vgl. Bassa/Kröll/Kern (2018), S. 2-24.

¹³⁴Vgl. Akbik/Löser (2012), S. 52-56.

¹³⁵Vgl. Banko et al. (2007), S. 2670-2676.

¹³⁶Vgl. F. Wu/Weld (2010), S. 118-127.

¹³⁷Vgl. Etzioni et al. (2011), S. 3-10.

¹³⁸Vgl. Schmitz et al. (2012), S. 523-534.

¹³⁹Vgl. Akbik/Broß (2009)

¹⁴⁰Vgl. Del Corro/Gemulla (2013), S. 355.

5.8 Transfer Learning

Als letzte, und zum jetzigen Zeitpunkt aktuellste, Methode wird Transfer Learning betrachtet. Hierbei wird der Lernfortschritt eines Modells gespeichert und auf weitere Modelle als deren Grundlage übertragen. Damit müssen diese nicht von Null beginnend trainiert, sondern lediglich für deren spezifischen Anwendungszweck angepasst werden. Es soll herausgefunden werden, ob und wie die Methode eingesetzt werden kann und welche Voraussetzungen dafür gegeben sein müssen.

Recurrent Neural Networks wie z. B. LSTM-Networks (Long Short Term Memory) werden in verschiedenen Disziplinen des Natural Language Processing (NLP) eingesetzt; Vor allem in jenen Disziplinen, in denen es um Sprachverständnis geht – hierzu zählen unter anderem *Language Modeling*, *Machine Translation* und *Question Answering* – liefern diese zum jetzigen Zeitpunkt State of the Art (SOTA) Performance. Seit der Veröffentlichung des Papers *Attention is all you need* von Vaswani et al. (2017), das die von Google entwickelte Transformer-Architektur vorstellte, wurden LSTM-Netze, wie zum Beispiel ELMO,¹⁴¹ vor allem im Bereich des NLP durch diese an den Spitzen der Benchmarks abgelöst.

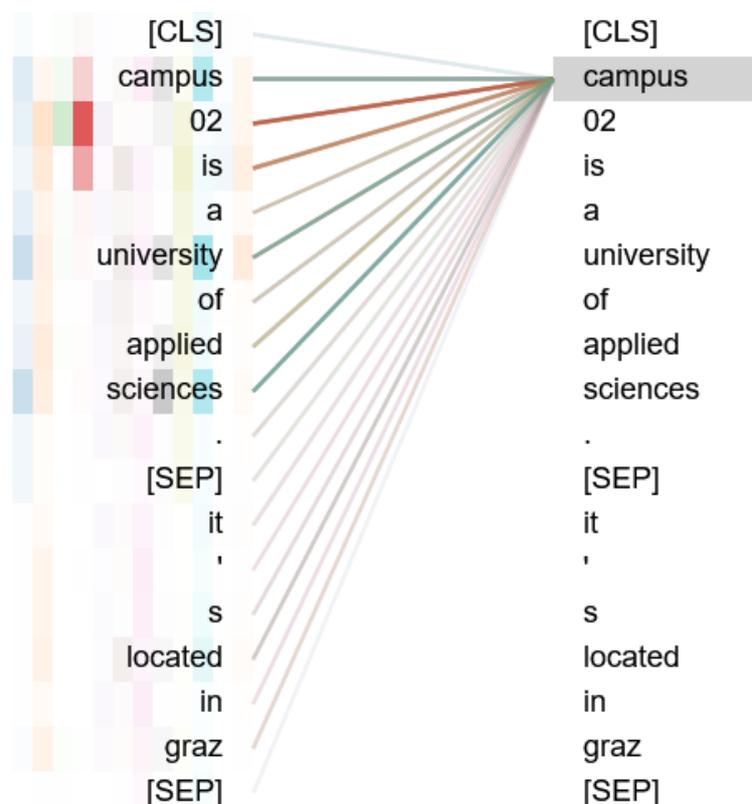


Abbildung 5.21: Visualisierung der zwölf Attention-Heads des BERT-Transformer-Modells. Jede Farbe stellt einen Head dar; die Intensität dieser ist Indikator für den Wert der Attention. In diesem Beispiel sind die Werte für das Wort *campus* gezeigt. Quelle: Vig (2019) (modifiziert)

Bei Transformer-Modellen handelt es sich um ein Encoder-Decoder-Netzwerk mit Attention-Mechanismus. Bekannte Vertreter, die Grundlegend auf dieser Architektur basieren, stellen derzeit BERT¹⁴², XLNet¹⁴³ und GPT-2¹⁴⁴ dar. Eine einfache qualitative Visualisierung des Attention-Mechanismus ist in Abbildung 5.21 gezeigt. Hierbei handelt es sich um das BERT-Modell welches auf den Text ‚Campus02 is a University of Applied Sciences. It’s located in Graz‘ angewendet wurde. Jede Zeile stellt hierbei zwei Token neben-

¹⁴¹Vgl. Peters et al. (2018)

¹⁴²Vgl. Devlin/M. Chang et al. (2018)

¹⁴³Vgl. Z. Yang et al. (2019)

¹⁴⁴Vgl. Radford/J. Wu et al. (2019)

einander dar, womit die Self-Attention sowohl zwischen einem Token und sich selbst als auch zu jedem andern Token visualisiert werden kann. Bei den Token in eckigen Klammern handelt es sich um spezielle Token, die das BERT-Modell benötigt. Die farbigen Kästchen auf der linken Seite visualisieren die Attention Heads (AHs) des Modells. Jeder der zwölf AHs des BERT-Modells hält in diesem Beispiel die Information eines Tensors mit 64 Einträgen. Somit handelt es sich um das kleinere der BERT-Modelle, das jeden Token auf 768 Dimensionen abbildet. Anhand dieser Zahlen lässt sich bereits vermuten, dass diese Modelle ein hohes Maß an Rechenleistung benötigen und für deren Einsatz die Berechnungen auf Graphics Processing Units (GPUs) oder Tensor Processing Units (TPUs) ausgelagert werden sollten. Als Testsystem für die Berechnungen wird aus diesem Grund entweder auf Nvidia-Grafikkarten der Pascal Generation oder auf TPUs in der Amazon Cloud zurückgegriffen. GPUs oder TPUs haben einerseits einen geringeren Takt pro Rechenkern als CPUs, dafür aber andererseits eine erheblich höhere Anzahl an Kernen. Berechnungen auf Vektoren, Matrizen und Tensoren können zumeist gut auf mehrere Kerne aufgeteilt werden, wodurch diese erheblich von der höheren Anzahl an Kernen profitieren.

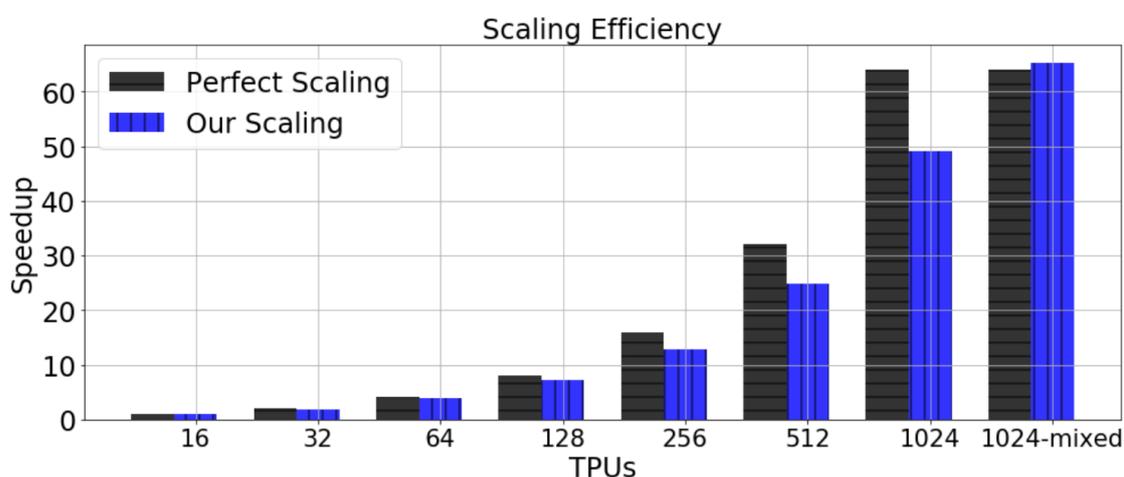


Abbildung 5.22: Darstellung der Skalierung der Lernrate eines BERT-Modells wenn man die Anzahl der TPUs erhöht. Eine Skalierung der Rechenleistung um das 64-fache ergibt eine um das 49-fache erhöhte Lernrate. Quelle: You et al. (2019), S. 8.

In Abbildung 5.22 ist die Skalierung der Lernrate eines BERT-Modells unter Verwendung der LAMB (Layer-wise Adaptive Moments optimizer for Batch training)¹⁴⁵ dargestellt. In diesem Paper wurde ein BERT-Modell in 76,19 Minuten auf einen F1-Score von 90,584 trainiert. Dies geschah unter Verwendung von 1024 TPUs. Unter der Annahme, dass eine TPU zum jetzigen Zeitpunkt ca. 8 \$ pro Stunde kostet, würde dieses Training ca. 10.300 \$ an Kosten verursachen. Bei LAMB soll es sich um eine besonders effektive Methode des Trainings handeln, welche ein vielfaches schneller ist als das Training der original veröffentlichten Modelle wie beispielsweise XLNet.

An dieser Stelle soll erwähnt werden, dass in diesem Projekt kein BERT- oder XLNet-Modell komplett neu von Null weg trainiert wird. Stattdessen werden die von Google als Free Open Source Software (FOSS) veröffentlichten, bereits vor-trainierten Modelle verwendet. Bei diesen Modellen wird empfohlen, unter Verwendung einer GPU oder TPU, diese über wenige Trainings-Epochen hinweg, auf den realen Daten ein Feintuning durchzuführen, näheres hierzu ist Devlin et al. (2019)¹⁴⁶ zu entnehmen.

¹⁴⁵Vgl. You et al. (2019), S. 1-36.

¹⁴⁶Vgl. Devlin/M.-W. Chang et al. (2019)

5.8.1 Transformer-Modell

Die in Vaswani et al. (2017) vorgestellte Transformer-Architektur stellt zum jetzigen Zeitpunkt eine der leistungsfähigsten Architekturen im Bereich des NLP dar.

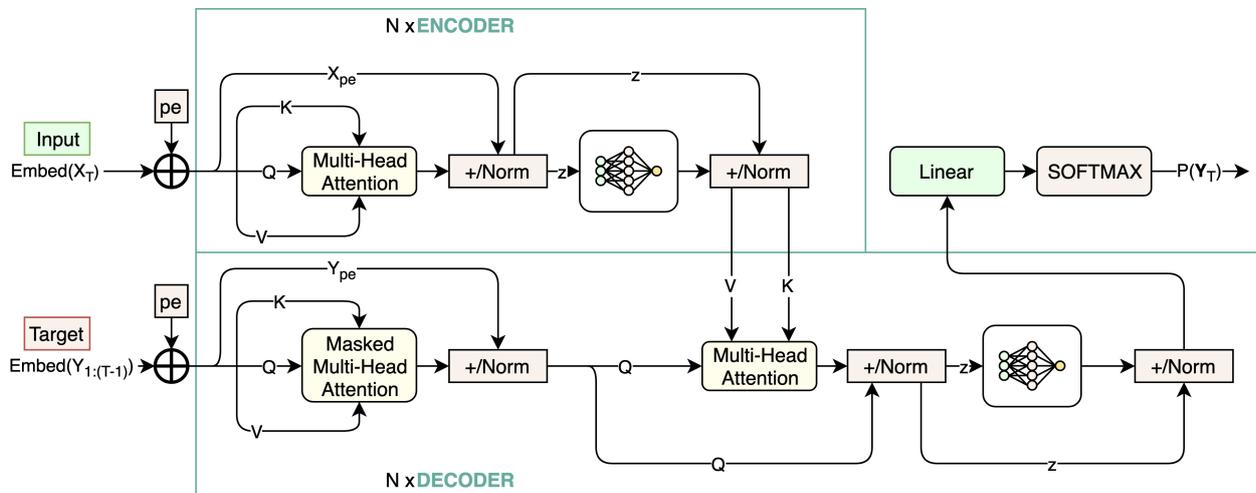


Abbildung 5.23: Schematische Darstellung eines Transformer-Modells. Quelle: Eigene Darstellung.

In Abbildung 5.23 ist der schematische Aufbau eines Transformer-Modells gezeigt. Diese Architektur liegt Modellen wie XLNet und BERT zugrunde. Die in diesen Modellen verwendete Multi-Head-Attention hat, in nahezu allen Bereichen des Natural Language Understanding, die zuvor verwendeten GRUs und LSTMs abgelöst. Bevor auf die Anwendung der Modelle, am Beispiel BERT, eingegangen wird, soll im Folgenden ein kurzer Überblick über den Transformer gegeben werden; dies wird unter der Verwendung von Beispiel-Code bewerkstelligt, der sich an den in Tensorflow-2.0rc (2019) verwendeten Methoden orientiert. Im Laufe der Bearbeitung dieser Arbeit stellte sich heraus, dass die Sprachverarbeitung der eingehenden Dokumente in Englisch zu erfolgen hat. Zudem wurde ersichtlich, dass nach dem Englischen die meisten Dokumente in Portugiesisch verfasst sind. Aus diesem Grund bietet es sich in diesem Beispiel an, das Transformer-Modell auf das Übersetzen von Portugiesisch auf Englisch anzuwenden. Die erhaltene Pipeline kann wie in Abbildung 5.25 abstrahiert werden. In Abbildung 5.23 ist das Transformer-Modell mit zwei Eingängen auf der linken Seite und einem Ausgang auf der rechten Seite dargestellt. An den Eingängen liegen Embeddings der Token an. Am oberen Eingang liegt X_T an, hierbei handelt es sich beispielsweise um eine Einbettung des Textes in der Ausgangssprache der gewünschten Übersetzung. Am unteren Eingang liegt dann der bereits übersetzte Teil des Satzes $Y_{1:T-1}$ an. In Abbildung 5.24 ist zu sehen, wie ein von Tensorflow Datasets (TFDS) geladener Encoder die Wörter in maschinenlesbare Zahlenrepräsentationen umwandelt. Würde zur weiteren Verdeutlichung, um was es sich bei diesem Encoding grundlegend handelt, der Befehl:

```
1 print(tokenizer_en.decode(7904)),
```

ausgeführt werden, würde in der Ausgabe der Text I ausgegeben werden.

Anschließend werden diese Encodings, bevor sie an den Eingang des Encoders (en Token) oder Decoders (pt Token) angelegt werden, in Batches zusammengefasst, mit deren Embedding zusammengeführt und mit einem Positional Encoding (pe) versehen. Dies ist in Abbildung 5.23 als pe , das über ein elementweises Plus mit dem Token zusammengeführt wird, dargestellt. Als wesentlicher Vorteil eines Transformers gegenüber Convolutional Neural Networks (CNN) und RNNs wird neben seiner guten Parallelisierbarkeit auf Rechenclustern und Grafikkarten, das Erlernen von long-term dependencies, also das Erlernen von Abhängigkeiten über viele Token hinweg, genannt. Dieser Vorteil, der sich aus der Verwendung des Multi-Head-Attention-Mechanismus ergibt, geht mit dem Nachteil einher, dass die Token keine Information über

```

1 ...
2 tokenizer_en = tfds.features.text.SubwordTextEncoder.build_from_corpus(
3     (en.numpy() for pt, en in train_examples), target_vocab_size=2**13)
4 tokenized_string = tokenizer_en.encode('I study at Campus02.')
5
6 for ts in tokenized_string:
7     print ('{:<5} --> {}'.format(ts, tokenizer_en.decode([ts])),end=' || ')

```

out:

```

7904 -> I || 7863 -> || 628 -> study || 38 -> at || 7898 -> C || 6465 -> amp || 171
-> us || 7879 -> 0 || 7881 -> 2 || 7877 -> . ||

```

Abbildung 5.24: Beispiel für das Zuweisen eines Satzes vor dem Positional Encoding (pe) am Decoder zu Token. Quelle: Eigene Darstellung.

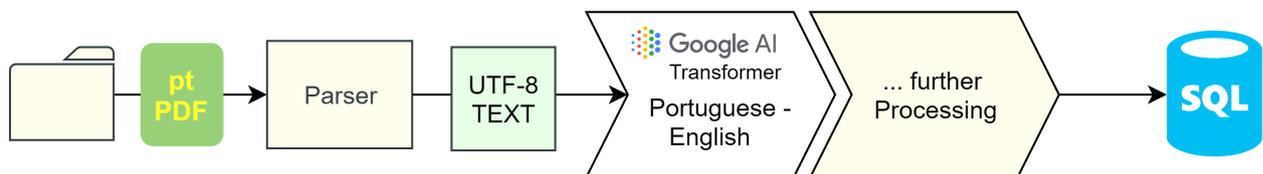


Abbildung 5.25: Schematische Darstellung der Pipeline-Komponente, die im Beispiel beschrieben wird. Der verwendete Transformer wird zum jetzigen Zeitpunkt von Google für Tensorflow in der Tensor2Tensor Library als FOSS bereitgestellt. Quelle: Eigene Darstellung

ihre Positionierung zueinander in der Sequenz enthalten. Um diesen Nachteil zu vermeiden, wird den Token – analog zur Vorgehensweise in Vaswani et al. (2017) – ein Positional Encoding in der Form

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right) \quad (5.1)$$

und

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right), \quad (5.2)$$

mitgegeben, wobei pos für die Position, i für die Dimension des Tokens in der Sequenz und d für Depth, also die Anzahl der Dimensionen, die das Modell hat, steht. Die Position Embeddings ergeben sich durch eine Aneinanderkettung der Sinus- und Cosinus-Funktion. Somit muss d im Fall der Input Embeddings des originalen BERT-Modells 512 betragen, damit das elementweise Zusammenfügen funktioniert. Wie die sich daraus ergebenden Positional Encodings aussehen, ist in Abbildung ?? zu sehen.

Gewählt wurde diese Methode um in der Lage zu sein, sehr lange Sequenzen mit Informationen über die Position eines Tokens zu versehen. Ein weiterer Vorteil dieser Methode ist, dass über das innere Produkt (Dot Product) zweier Token auf deren Nähe zueinander geschlossen werden kann.

Visuell in einem Plot dargestellt ist dieser Zusammenhang in Abbildung ?? zu sehen. Bis hierhin wurde also gezeigt, dass an den Eingängen eines Transformer-Modells, wie es in Abbildung 5.23 zu sehen ist, Worteinbettungen angelegt werden. Weil diese d -dimensionalen Einbettungen – ein Beispielwert für d im Falle des originalen BERT-Modells wäre 512 – keine Information über die Reihenfolge der Token enthalten und weder der Multi-Headed-Attention-Mechanismus noch die neuronalen Feed-Forward-Netze im Inneren des Transformers Informationen über die Reihenfolge der Token bereitstellen, wurde diese über Winkel-funktionen direkt an die Token angefügt. Wird erneut Abbildung 5.23 betrachtet, zeigt sich, dass die mit der

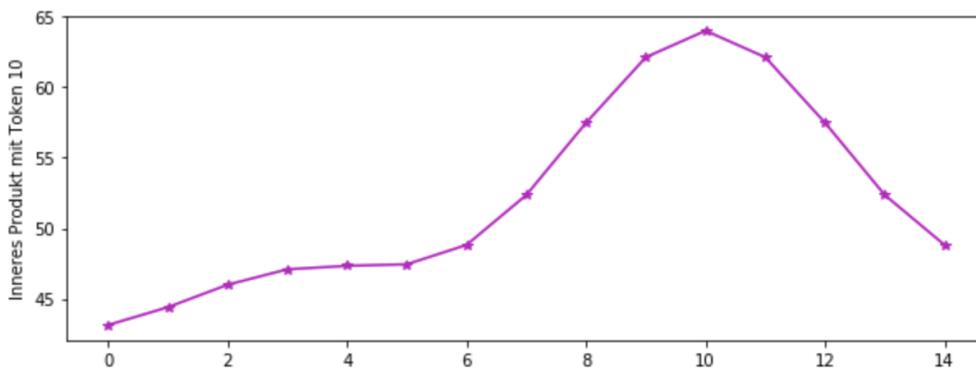


Abbildung 5.26: Nähe der umgebenden Token zu Token 10. Quelle: Eigene Darstellung

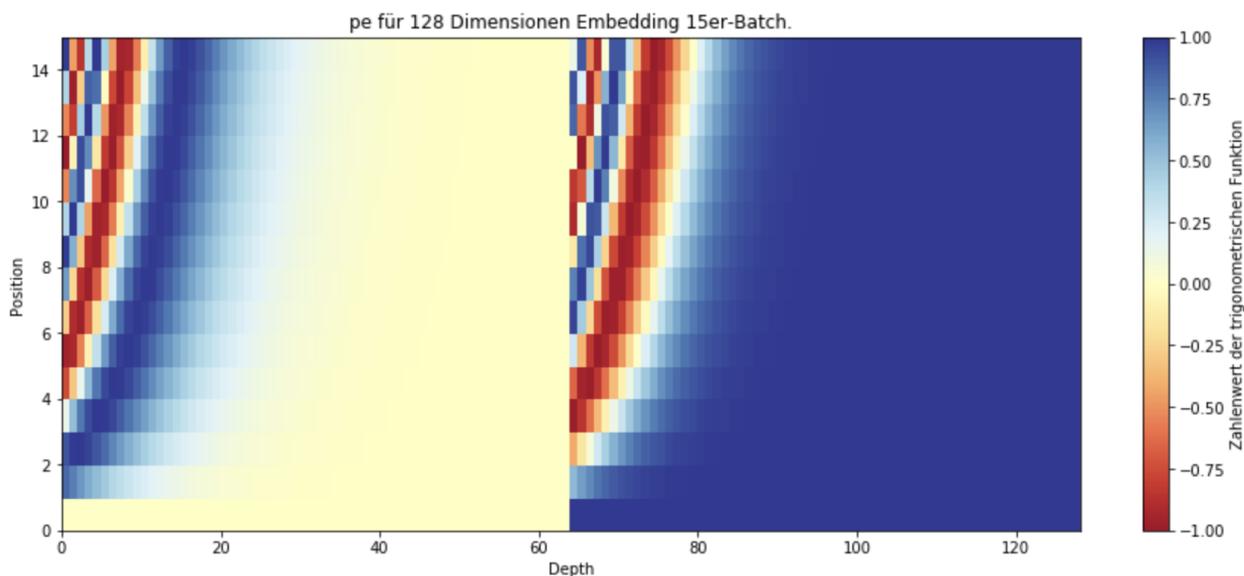


Abbildung 5.27: Plot der Werte der trigonometrischen Funktionen, die Werte-Ketten zu 128 Einträgen zeigt. Für ein reales Modell wäre ein realistischerer Wert 512 Einträge. Die Batchgröße wurde mit 15 gewählt was auch wenig ist, die kleineren Zahlen sollen der besseren Erkennbarkeit im Plot dienen. Quelle: Eigene Darstellung

Positionsinformation versehenen Token jeweils einmal sowohl an die Multi-Headed-Attention-Mechanismen als auch an die Feed-Forward-Netze hergeführt werden. Damit bleibt die Information erhalten. Die neuronalen Feed-Forward-Netze werden durch die wie neuronale Netze aussehenden Bilder auf weißem Hintergrund in Abbildung 5.23 dargestellt. RNNs und CNNs erhalten inhärent Informationen über die Position der Token zueinander, wodurch diese aber zugleich in der Parallelisierung der Teilschritte eingeschränkter sind als das Transformer-Modell. Mit diesem ersten Schritt sind die Daten nun bereit, um in die eigentliche Encoder-Decoder-Architektur des Transformers geführt zu werden.

5.8.2 Scaled Dot-Product Attention

In Abbildung 5.23 ist am Eingang zum Encoder und Decoder zu sehen, dass die Daten als erstes in einen Multi-Head-Attention-Block geführt werden. Die Formel für diesen Block lautet:

$$\text{Attention}(Q, K, V) = \text{SOFTMAX}\left(\frac{QK^T}{\sqrt{d_K}}\right)V. \tag{5.3}$$

Hierbei wird Q als Querys, K als Keys und V als Values bezeichnet. Unter d_K wird ein Skalierungsfaktor verstanden, der hier den Dimensionen von K entspricht. Um den Schritt zu Multi-Headed-Attention zu machen,

wird dieser Vorgang mehrere Male parallel ausgeführt. Um das Beispiel nachvollziehbar zu gestalten, sollte das Beispiel mit Zahlen untermauert werden. Angenommen der Token, der am Input in Abbildung 5.23 anliegt, wurde mit einem 512-dimensionalen Embedding plus einem 512-dimensionalen Positional Embedding beaufschlagt. Somit besitzt das am Attention-Head anliegende X_{pe} 512 Einträge. Als Anzahl der Attention-Heads wurde $h = 8$ gewählt. Als Größe für Q, K, V wurden 64 Dimensionen gewählt.¹⁴⁷ Um X_{pe} auf Q, K, V abzubilden, werden diese mit den Variablen des Modells W_i^Q, W_i^K, W_i^V multipliziert; i verweist hierbei auf den Attention-Head, zu dem die Variable gehört. Der Wert $\sqrt{d_K}$ ergibt in diesem Beispiel 8. Um die hier genannten $d_{model} = 512$ Einträge des Inputs auf die 64 Dimensionen von Q, K und V abzubilden muss für W_i^Q, W_i^K und W_i^V gelten:

$$\begin{aligned} W_i^Q &\in \mathbb{R}^{d_{model} \times d_K}, \\ W_i^K &\in \mathbb{R}^{d_{model} \times d_K}, \\ W_i^V &\in \mathbb{R}^{d_{model} \times d_K} \end{aligned} \tag{5.4}$$

und das für alle i , also für jeden Attention-Head gleich. Damit können die einzelnen Pfade der Attention Heads $head_i$ mit $i \in 1 \leq i \leq h$ zusammengeführt werden als:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{CONCAT}(head_1, \dots, head_h)W^O = Z, \\ \text{hierbei ist } head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{5.5}$$

mit $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ als Variable (Gewicht des Modells), die die Verkettung der einzelnen *heads* wieder auf die passende Form (Dimensionalität) zurückführt. Danach folgt in Abbildung 5.23 ein als *+Norm* bezeichneter Block. An dieser Stelle wird der Ausgang der Multi-Head-Attention Z mit X_{pe} addiert und normiert. Damit soll der Multi-Headed-Attention-Block in Abbildung 5.23 ausreichend beschrieben sein; eine vereinfachte, dafür aber intuitivere Illustration dieser Zusammenhänge kann in Alammari (2019) gefunden werden.

Eine weitere Frage, die sich in diesem Zusammenhang noch gestellt werden muss, ist, wo der Unterschied zwischen Query Q , Key K und Value V liegt. Im Falle der Encoder Attention gilt, dass es sich um Self-Attention handelt, und der Encoder kennt lediglich seinen aktuellen Token. Der Masked-Multi-Headed Attention Layer beim Input *Target* (Abbildung 5.23 links unten), kennt jene Token, die das gewünschte Ergebnis repräsentieren. Damit an dieser Stelle das Modell nicht einfach ‚nachsehen‘ kann, werden die Ergebnisse, deren Voraussage erlernt werden soll, maskiert. Für den maskierten Token, den Beginn und das Ende einer Sequenz und gegebenenfalls einen Separator zwischen den einzelnen Bestandteilen des Inputs gibt es eigene Token. Der Input für die Evaluierung eines Transformer-Modells könnte somit aussehen wie in Abbildung 5.21 gezeigt. Da es sich bei der linken und rechten Sequenz um den identischen Satz handelt, wird hierbei von Self-Attention gesprochen.



Abbildung 5.28: Darstellung eines Trainingstupel. Die Token stehen für SOS (Start of Sentence), EOS (End of Sentence), SEP (Seperator) und MASK für den maskierten Token. Die Bezeichnungen sind frei gewählt und variieren von Modell zu Modell. In diesem Beispiel hat der Transformer das Wort Campus02 zurückzugeben. Quelle: Eigene Darstellung.

Die entscheidende Verbindung ist jene zwischen Encoder und Decoder; an dieser Stelle liefert der Encoder *Key* und *Value* und der Decoder das *Query*. An dieser Stelle lernt das Modell den Zusammenhang zwischen Input und Target. In Abbildung 5.23 sind jeweils ein Encoder und ein Decoder dargestellt, was der besseren

¹⁴⁷Vgl. Vaswani et al. (2017), S. 5.

Übersichtlichkeit dient. Im originalen Transformer-Modell in Vaswani et al. (2017) wurden jeweils sechs Encoder und sechs Decoder in Serie geschaltet verwendet. Dieser Zusammenhang ist durch $N = 6$ in Abbildung 5.23 beschrieben.

5.8.3 FFN im Transformer

Das nächste Element, aus dem das in Abbildung 5.23 dargestellte Modell besteht, stellen die beiden (Point-wise) Feed-Forward-Netze dar.

Python 3.6

```

1  ...
2  def point_wise_feed_forward_network(d_model, d_ff):
3      return tf.keras.Sequential([
4          tf.keras.layers.Dense(d_ff, activation='relu'),
5          tf.keras.layers.Dense(d_model)
6      ])

```

Abbildung 5.29: Aufbau der FFNs, die im Transformer verwendet werden. Für die beiden Dense Layer wurde Keras verwendet. Diese Vorgehensweise wurde auch in der Tensorflow-Dokumentation verwendet und unter anderem deshalb angewendet, weil Keras ab Tensorflow 2.0 die offiziell in Tensorflow implementierte High-Level-API darstellt. Quelle: Eigene Darstellung.

Diese lassen sich erstellen wie in Abbildung 5.29 gezeigt und entsprechen der in Vaswani et al. (2017) gezeigten Formel:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2). \quad (5.6)$$

Damit entspricht deren Aufbau zwei durch eine Rectified Linear Unit (ReLU) verbundenen linearen Transformationen. W sind hierbei die Wichtungen (Variablen) des Netzes und bei b handelt es sich um den Bias. Jeder Token wird einzeln in den FFNs verarbeitet, daraus folgt die Bezeichnung Point-wise-Feed-Forward-Netzwerk.

5.8.4 Trainieren und Testen des Modells

Als Trainingsdaten wurde das Portugese-English translation dataset¹⁴⁸ verwendet. Die Daten wurde in 32er Batches an den Transformer geliefert und auf einer Nvidia Geforce Gtx 1070 der Pascal-Generation trainiert. Die Hyperparameter wurden an das Original angepasst:

- Layer = 4,
- Dimensionen Modell = 64,
- Dimensionen FFN = 256,
- Attention-Heads = 8,
- Dropout Rate = 0,1 und
- Epochen = 20.

¹⁴⁸Ye et al. (2018)

Pro Epoche dauerte das Training ca. 10 Minuten also 400 Minuten gesamt. Der Adam Optimizer und der Lern-Raten-Verlauf mit Warm-up (4000 Steps)

$$lrate = d_{model}^{-.5} * \min(step_num^{-.5}, step_num * warmup_steps^{-1.5}) \tag{5.7}$$

wurden unverändert aus Vaswani et al. (2017) übernommen. In Abbildung 5.30 wird dargestellt, wie sich Accuracy und Loss über die Trainingsepochen hinweg verhalten.

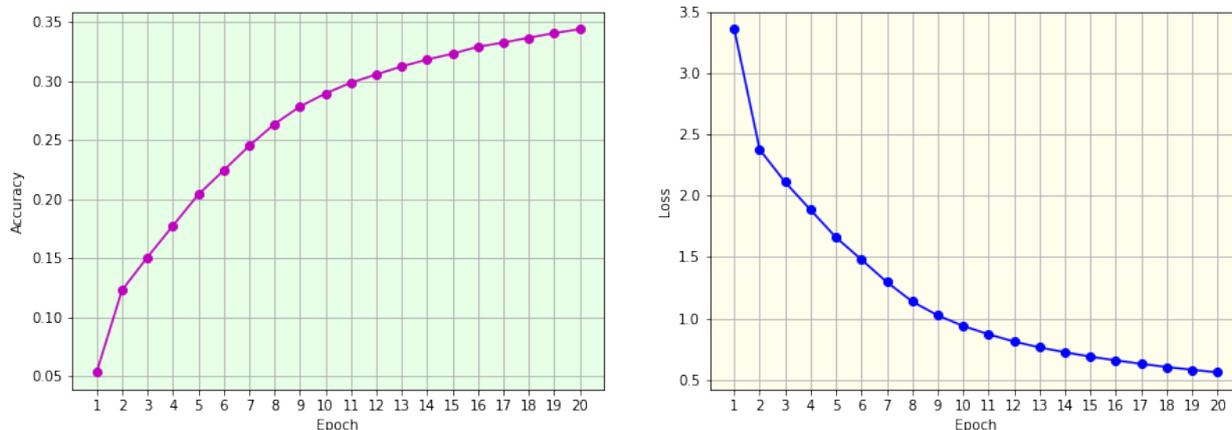


Abbildung 5.30: Im linken Plot ist der Verlauf der Accuracy über die 20 Trainings-Epochen hinweg abzulesen. Im rechten Plot ist der Verlauf des Loss abzulesen. Die in 20 Epochen erreichte Accuracy betrug 34,4 %. Quelle: Eigene Darstellung

Über die ersten Epochen hinweg nahm die Accuracy des Modells in rund 11 Epochen von 0 auf 30 % zu, in weiteren 9 Epochen wurden nur mehr 4 % hinzugewonnen. Der Loss ist wie zu erwarten der Accuracy indirekt proportional und die Minimierung von diesem ist das Trainingsziel des Modells. In Abbildung 5.31 ist dargestellt, welche Voraussage das über 20 Epochen trainierte Modell für einen Beispielsatz liefert.

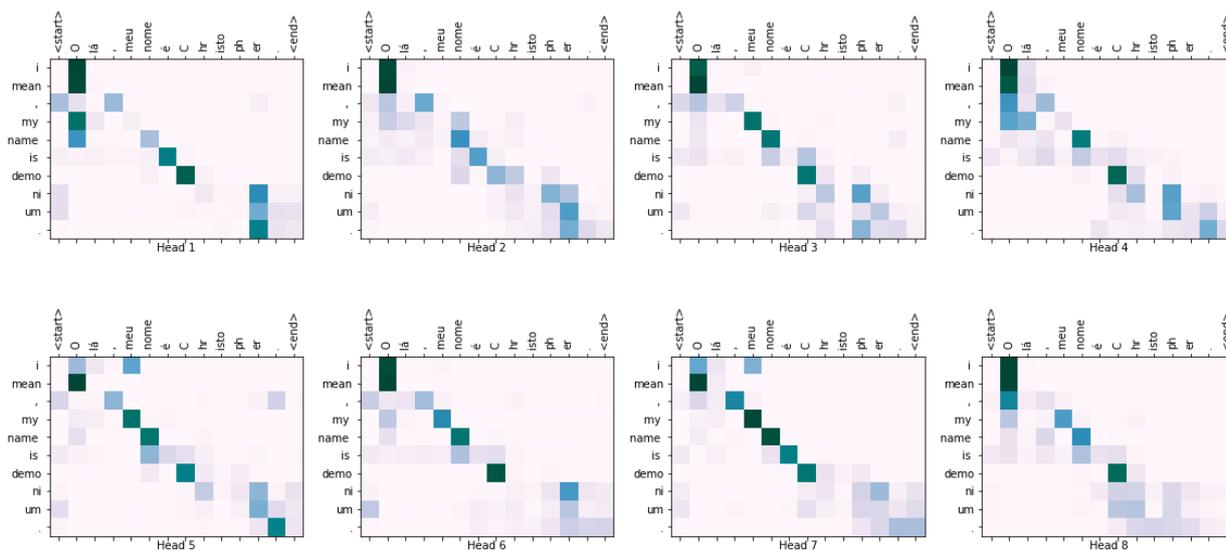


Abbildung 5.31: Visualisierung eines Attention Layers über alle 8 Heads für die erhaltene Übersetzung. Die Übersetzung ist nicht komplett richtig jedoch ist zu erkennen, dass semantische Bedeutung erlernt wurde. Quelle: Eigene Darstellung

Auch wenn der Satz nur zu teilen richtig übersetzt wurde, ist bei diesem als auch bei weiteren Sätzen, die getestet wurden, der Anschein entstanden, dass das Modell semantische Zusammenhänge verstehe.

Ebenfalls interessant in Abbildung 5.31 ist, dass in der besser übersetzten ersten Hälfte des Satzes alle acht Attention-Heads höhere Attention-Werte aufweisen als in der zweiten falsch übersetzten Satzhälfte. Eine Komponente der Transformer-Architektur wurde an dieser Stelle noch nicht erwähnt und zwar die Linearisierung und der folgende SOFTMAX am Ende des Modells. Aus diesem folgt eine Wahrscheinlichkeit $P(Y_T)$ oder genauer gesagt die Wahrscheinlichkeit für $P(Y_t|X_T, Y_{1:(T-1)})$, also die Wahrscheinlichkeit dafür, dass der gesuchte Token, der am Encoder zum Trainingszeitpunkt maskiert oder im Einsatz noch nicht vorhanden ist, einem bestimmten Wort einer anderen Sprache entspricht. Mit dem Erkenntnisstand aus diesem Versuch soll nun ein weiteres auf dem Transformer aufbauendes Modell auf dessen Anwendbarkeit zur Informationsextraktion getestet werden.

5.9 Bidirectional Encoder Representations from Transformers

In den vorherigen Kapiteln wurde gezeigt, wie Informationen aus Text extrahiert werden können und wie Text transformiert werden kann. Jene Eigenschaft, die die bis hierhin gezeigten Methoden (mit Ausnahme von OIE) gemeinsam haben, ist, dass die gesuchten Informationen beim Erstellen des Modells bereits bekannt sein mussten oder das Modell auf Daten, die zur Domäne passen, neu erstellt werden muss. Weil es nicht effektiv und meist auch nicht sinnvoll ist, immer von Neuem zu beginnen, wurde damit begonnen, Transfer Learning zu nutzen. Zu diesem Zweck wurden Modelle trainiert, die aus allgemeinen Texten wie Wikipedia oder auch dem Internet selbst die Zusammenhänge zwischen Wörtern und Sätzen erlernen.

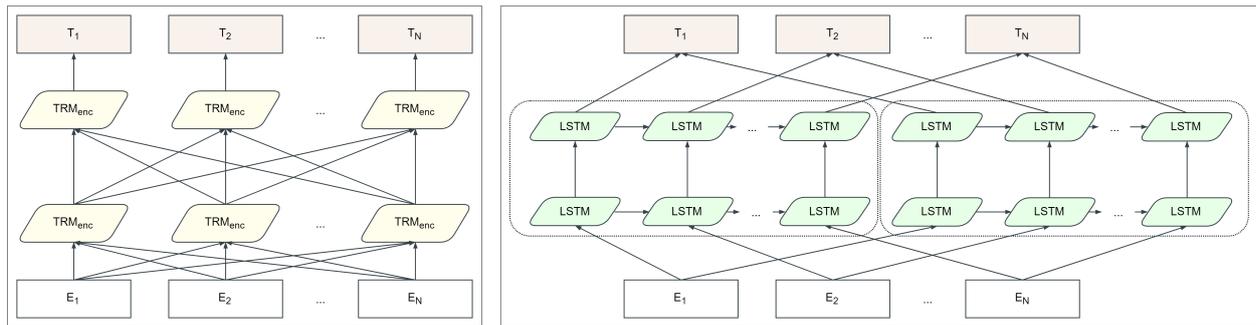


Abbildung 5.32: Links BERT das die Transformer Encoder Blöcke bidirektional in beide Richtungen verwendet und Rechts ELMo das LSTMs verwendet. Quelle: Eigene Darstellung.

Im Paper von Devlin et al. (2018) wurden Ende 2018 die sogenannten Bidirectional Encoder Representations from Transformers (BERT) vorgestellt und veröffentlicht. Darin wurde, ähnlich wie in Abbildung 5.32 dargestellt, ein Modell vorgestellt, das außerordentliche Verbesserungen vor allem in Bereichen des Natural Language Understanding (NLU) brachte. Während Modelle wie ELMo zu diesem Zeitpunkt Arten von RNNs in deren Architektur verwendeten, nutzte das Generative Pre-training for Transformers (GPT)¹⁴⁹ zu diesem Zeitpunkt bereits Attention. Der wesentliche Unterschied zwischen BERT und GPT war, dass BERT einen bidirektionalen Ansatz verwendete, was das Modell auf ein zu diesem Zeitpunkt unerreichtes Level hob. Ende Juni 2019 veröffentlichte Google in Yang et al. (2019) mit XLNet ein Modell, das die mit BERT erzielten Ergebnisse erneut übertraf. Dies wurde dadurch erreicht, dass die Sätze nicht nur bidirektional, sondern auf Permutationen dieser trainiert wurden. Im Wesentlichen sind die Ergebnisse, die die Modelle erzielen, mittlerweile wieder weniger verstreut und die Modelle verwenden die in BERT, GPT und XLNet verwendeten Ideen. Ein Weg der gegangen wird, um die Performance der Modelle zu erhöhen, ist die Verwendung von Ensembles. Auf diese Weise kann über die Kombination mehrerer Modelle die resultierende Performance der Modelle erhöht werden. Für den praktischen Einsatz im Anwendungsfeld dieser Arbeit sind diese Modelle keine Alternative, da bereits die Variante $BERT_{SMALL}$ erhebliche Rechenleistung benötigt. Deshalb wurden XLNet und Ensembles vorerst außen vor gelassen. Stattdessen werden BERT und der Universal Sentence Encoder^{150,151} verwendet. Ein weiterer erheblicher Grund für die Verwendung dieser Modelle ist, dass diese frei verwendbar und Open Source von Google veröffentlicht wurden. BERT ist beispielsweise unter <https://github.com/google-research/bert> [Stand: 03.09.2019] zu finden und eine XLNet-Implementierung auf <https://github.com/zihangdai/xlnet> [Stand: 03.09.2019]. Neben diesen beiden Tensorflow-Implementierungen sind auch Implementierungen für PyTorch, MXNet (GluonNLP), spaCy, Keras und weitere Frameworks vorhanden, was ein weiteres Argument für die Wahl dieser Modelle darstellte.

¹⁴⁹Vgl. Radford/Narasimhan et al. (2018)

¹⁵⁰Vgl. Cer et al. (2018b)

¹⁵¹Vgl. Y. Yang et al. (2019)

5.10 Machine Reading Comprehension

Eine jener Disziplinen, für die das BERT-Modell benutzt wurde, ist Machine Reading Comprehension (MRC). Bei MRC handelt es sich um eine Gattung des Question Answerings, bei dem der Algorithmus die Position einer Antwort in einer Textpassage herauszufinden versucht.

Rank	Model	Ranking Style	Submission Date
1	Enriched BERT base + AOA index + CAS Ming Yan of Alibaba Damo NLP	Full Ranking	August 20th, 2019
2	Enriched BERT base + AOA index V1 Ming Yan of Alibaba Damo NLP	Full Ranking	May 13th, 2019
3	BERTer pretraining (1)Rodrigo Nogueira, (2)Wei Yang, (3)Jimmy Lin, (4)Kyunghyun Cho - New York University(1,4), University of Waterloo(2,3), Facebook AI Research(4)	Full Ranking	May 21st, 2019
4	Enriched BERT base + AOA index V2 Ming Yan of Alibaba Damo NLP	Full Ranking	May 13th, 2019
5	BM25 + monoBERT + duoBERT + TCP Anonymous	Full Ranking	June 26th, 2019

Abbildung 5.33: Auszug aus dem MS MARCO Leaderboard für Passage Retrieval. Das Modell BERT ist auch in diesem öfters anzutreffen. [Stand: 02.09.2019] Quelle: Microsoft (2019).

Es existieren verschiedene Datensätze, anhand derer die Modelle trainiert werden können und ihre Performance in Ranglisten zur Schau stellen. Bekannte Vertreter dieser Datensätze sind das Stanford Question Answering Dataset (SQUAD), das Microsoft Machine Reading Comprehension Dataset (MS MARCO)¹⁵² und das ReAding Comprehension Dataset From Examinations (RACE)¹⁵³. Wie in Abbildung 5.33 zu sehen ist, kommt das BERT-Modell in diesen Ranglisten öfters vor, wenn auch immer in Verbindung mit zusätzlichen Techniken.

Machine Reading Comprehension ermöglicht dem Anwender die Antwort auf eine in natürlicher Sprache gestellte Frage in einem Text zu finden. Die gestellte Frage muss somit nicht wie bei den bisherigen Ansätzen bereits im Vorhinein bekannt sein. Auch für den Fall, dass diese bekannt sein müsste, ist die Möglichkeit, die Antwort auf eine definierte Frage in einem Text zu finden, eine interessante Anwendung. Mit dieser Technologie könnte es einem Techniker möglich sein, eine Frage, die sich ihm beim Bearbeiten einer Spezifikation stellt, direkt in dieser zu suchen.

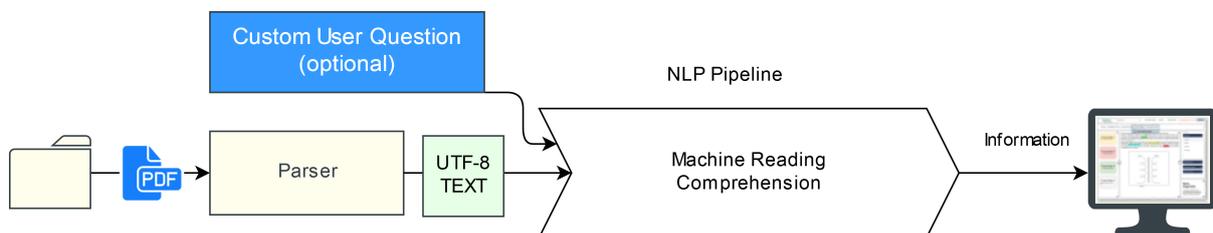


Abbildung 5.34: Schematische Darstellung einer Pipeline wie MRC in das System integriert sein könnte. In einem solchen System wären beliebige vom Anwender selbst erstellte natürlichsprachlich gestellte Fragen (Querys) möglich. Die Information soll dem Benutzer in einer Web-App ausgegeben werden. Quelle: Eigene Darstellung.

Beim Question Answering – auch als Machine Reading Comprehension (MRC) bezeichnet – werden dem BERT-Modell zwei weitere Lagen hinzugefügt, deren Aufgabe es ist, Anfangs- und Endposition der Ant-

¹⁵²Vgl. Nguyen et al. (2016)

¹⁵³Vgl. Lai et al. (2017)

wort in der Passage vorauszusagen. In Abbildung 5.34 ist schematisch dargestellt, wie eine Pipeline bzw. Komponente oder Microservice – je nachdem wie die Implementierung ausgeführt und benannt wird – aufgebaut sein könnte. Um diese Aufgabe zu bewältigen und weitere Erkenntnisse über dieses Gebiet des NLP zu erlangen, wurde als erstes eine Implementierung anhand des SQUAD Datensets gewählt. Durch dieses Vorgehen kann im ersten Schritt auf das Vorhandensein eigener Daten verzichtet werden. Ein wesentlicher Punkt für die Implementierung solcher Systeme ist das Vorhandensein von geeigneten Datensätzen. In den meisten Fällen ist ein solcher, zumindest bei Projektbeginn, nicht vorhanden, wie es auch bei dieser Umsetzung der Fall ist. Ein MRC-System benötigt für das Training Passagen, anhand derer es ein Feintuning erfahren kann. Bei einem solchen Vorgehen bietet sich beispielsweise das BERT_{SMALL}-Modell als Ausgangsbasis an, das dann auf domänenspezifischen Daten zur Einsatzreife für die entsprechenden Domäne gebracht wird.

SQUAD 2.0

Paragraph:

An example of a decision problem is the following. The input is an arbitrary graph. The problem consists in deciding whether the given graph is connected, or not. The formal language associated with this decision problem is then the set of all connected graphs—of course, to obtain a precise definition of this language, one has to decide how graphs are encoded as binary strings.

EXAMPLE 1

Question:

What kind of graph is an example of an input used in a decision problem?

Ground Truth Answers:

arbitrary graph <68-84>, arbitrary <79-84>, arbitrary <79-84>

EXAMPLE 2

Question:

What type of graph is an example of an output used in a decision problem?

Ground Truth Answers:

<No Answer>

Abbildung 5.35: Beispielhafter Auszug aus dem SQUAD 2.0 Datensatz. Zu sehen ist ein Paragraf mit Informationen und zwei Fragestellungen zu diesem. Im ersten Beispiel ist die Antwort im Paragrafen zu finden und die Position der Antwort hinterlegt. Für das zweite Beispiel enthält der Absatz keine Antwort. Die wesentlichste Neuerung von SQUAD 1.1 auf SQUAD 2.0 war, dass es nicht mehr für jede Frage eine Antwort gab und somit <No Answer> zu einer möglichen Antwort wurde. Quelle: Rajpurkar/Jia/Liang (2018) (modifiziert).

In Abbildung 5.35 ist zu sehen, wie ein Datensatz, der sich für ein Feintuning eines Modells eignet, aufgebaut ist. Aspekte, die sich während der Bearbeitung der Thematik als bedeutsam herausstellten, sind:

- Die Antwort auf ein und dieselbe Frage wurde von verschiedenen Personen zum Teil in verschiedenen Abschnitten des Textes gefunden. Der Grund hierfür waren persönliche Expertise der Personen und die Anwendung von Domänenwissen.
- Das Bereitstellen mehrerer Ground Truth Answers aus dem Text stellte sich als Aufgabe heraus, die eine sorgsame Schulung der ausführenden Personen bedarf. Deshalb, und aus Gründen der Datensicherheit, wurde auf Optionen wie Mechanical Turk verzichtet.
- Die Option <No Answer> stellte sich in der Praxis als unabdingbar heraus. Damit einher geht, dass eine weitere, nicht auf Text basierende, relationale, auf vorherigen Extraktionen basierende Wissensdatenbank, ebenfalls vorhanden sein muss.

- Die Länge einer Passage, die eine Antwort umgibt und damit ebenfalls gesichert werden muss, ist bei Texten, die auf Absätzen basieren, verhältnismäßig einfach zu definieren. Sind Antworten in Tabellen oder Abbildungen hinterlegt, ist eine klare Festlegung des umgebenden Textes schwierig.
- Teils wurden Antworten auf Kombinationen aus Informationen zurückgeführt, die in weit auseinandert liegenden Textpassagen gefunden wurden. Das Erstellen entsprechender Trainingsdaten ist hierbei durch das Entfernen von dazwischenliegendem Text teilweise möglich.

Ein System, das als Beispiel zur Informationsextraktion mittels Machine Reading Comprehension verwendet wurde, ist der Universal-Sentence-Encoder-Multilingual-QA^{154,155}.

Python 3.6

```

1  ...
2  def answerQuestion(args):
3      '''questions, responses, response_contexts'''
4      g = tf.Graph()
5      with g.as_default():
6          module = hub.Module(
7              ... "https://tfhub.dev/google/universal-sentence-encoder-multilingual-qa/1"
8              question_embeddings = module(dict(input=args[0]),
9                                          signature="question_encoder",
10                                         as_dict=True)
11
12             response_embeddings = module(dict(input=args[1],
13                                             context=args[2]),
14                                         signature="response_encoder",
15                                         as_dict=True)
16  ...

```

Abbildung 5.36: Beispiel für die Einbindung des Universal Sentence Encoders. Dieser kann vom Tensorflow Hub als Modul geladen und verwendet werden. Auf der Webseite wird die Performance am SQUAD Dataset mit dev | train Precision 53.2 | 43.3 angegeben. Quelle: Eigene Darstellung.

In Abbildung 5.36 wird gezeigt, wie der Universal Sentence Encoder QA in ein Python-Programm als Modul implementiert werden kann. In Abbildung 5.37 ist ein Beispiel-Ergebnis eines MRC-Laufs auf dem SQUAD Dataset zu sehen. Bei den Versuchen mit dem SQUAD-Datensatz fiel auf, dass die erhaltenen Ergebnisse in weniger als 50 % der Fälle (87/200 Abfragen) richtig waren. Dies erscheint bei den gemachten Precision-Angaben in Tensorflow Hub plausibel.

In Abbildung 5.37 ist zu sehen, wie unter Verwendung von Deep Learning mittels Bibliothek eine Antwort und mögliche Antwortpassagen ausgegeben werden. Die subjektiv geringe Genauigkeit dieser Systeme auf den Testdatensätzen erscheint als Nachteil; selbst diese Ergebnisse konnten außerhalb von großen und gut annotierten Datensätzen wie SQUAD, MS MARCO oder RACE nicht erreicht werden. Ein Umstand der die Lösung trotzdem als interessant klassifiziert ist die Tatsache, dass nicht nur die Antwort allein, sondern eine Reihe an Passagen, die die mögliche Antwort enthalten, ausgegeben wird. Damit ist es möglich, sofern ausreichend Trainingsdaten vorhanden sind, dem Nutzer eine Auswahl an potenziell interessanten Passagen, die die Antwort auf eine Frage enthalten könnten, anzuzeigen. Die Auswahl einer geeigneten Passage an sich durch den Benutzer kann in diesem Fall als weiteres Trainingssample für das Modell verstanden werden. Neben dem Universal Sentence Encoder wurden auch die Bibliotheken Closed Domain

¹⁵⁴Vgl. Cer et al. (2018b)

¹⁵⁵Vgl. Y. Yang et al. (2019)

Random Question from SQuAD:

To what did Tesla attribute the unknown signals his radio received?

Answer:

communications from another planet

Retrieved sentences :

1. Tesla investigated atmospheric electricity, observing lightning signals via his receivers.
2. During his time at his lab, Tesla observed unusual signals from his receiver which he concluded may be **communications from another planet**.
3. The great distances and the nature of what Tesla was detecting from lightning storms confirmed his belief that the earth had a resonant frequency.
4. Tesla was incorrect in his assumption that high frequency radio waves would penetrate water but Émile Girardeau, who helped develop France's first radar system in the 1930s, noted in 1953 that Tesla's general speculation that a very strong high frequency signal would be needed was correct stating "(Tesla) was prophesying or dreaming, since he had at his disposal no means of carrying them out, but one must add that if he was dreaming, at least he was dreaming correctly.
5. It has been hypothesized that he may have intercepted Marconi's European experiments in July 1899—Marconi may have transmitted the letter S (dot/dot/dot) in a naval demonstration, the same three impulses that Tesla hinted at hearing in Colorado—or signals from another experimenter in wireless transmission.

Abbildung 5.37: Beispiel für ein Ergebnis einer Machine Reading Comprehension auf dem SQUAD Dataset. Das System wurde so genutzt, dass sowohl die fünf wahrscheinlichsten Paragraphen, die die Antwort enthalten, als auch die Antwort selbst ausgegeben werden. Quelle: Eigene Darstellung.

Question Answering (cdQA)¹⁵⁶, GluonNLP und BERT (Pytorch-Transformers¹⁵⁷) getestet. Für cdQA konnten keine Ergebnisse auf dem SQUAD-Datensatz erzielt werden; dies sollte grundsätzlich möglich sein, gelang aber nicht. Das System brachte dafür aber bereits eine grafische Oberfläche mit sich, in der die Fragen visualisiert werden. Die Antworten, die das System auf dem BNP-Paribas-Datensatz gegeben hat, konnten zu großen Teilen mithilfe des Datensatzes als richtig verifiziert werden.

```
python finetune_squad.py --bert_model bert_24_1024_16
                        --optimizer adam
                        --accumulate 2
                        --batch_size 6
                        --lr 3e-5
                        --epochs 2
                        --gpu 0
                        --null_score_diff_threshold -2.0
                        --version_2
```

Python GluonNLP

Abbildung 5.38: Die auf der Webseite <https://gluon-nlp.mxnet.io/> empfohlenen Parameter zum Finetuning von BERT auf dem SQUAD 2.0 Datensatz. Batch Size und Accumulate wurde angepasst, um auf der GTX 1070 mit dem Speicher auszureichen. Quelle: Eigene Darstellung.

In Abbildung 5.38 sind die Einstellungen abgebildet, die benötigt werden, um das BERT-Modell mit Gluon NLP auf dem SQUAD-2.0-Datensatz zu trainieren. Die Einstellungen mussten angepasst werden, um auf einer Nvidia GTX 1070 zu laufen. Daran konnte abermals erkannt werden, wie hoch der Speicherbedarf von Modellen wie BERT ist. Der erreichte F1 Score, sollte um die 77,90 liegen, wenn diese Einstellungen mit derselben GPU und GluonNLP ausgeführt werden.

In diesem Kapitel wurde ein kurzer Überblick über Machine Reading Comprehension gegeben. Im Rahmen der Versuche hat sich ergeben, dass die Technik für die Informationsgewinnung von großem Nutzen ist. Die ausführende Person könnte eine Liste mit möglichen Antworten auf in natürlicher Sprache gestellte oder

¹⁵⁶Vgl. Mikaelian et al. (2019)

¹⁵⁷Vgl. Debut (2019)

auch für den Arbeitsprozess vordefinierte Fragen erhalten und damit eine Vorauswahl für die gesuchten Informationen erhalten. Damit könnte der Arbeitsprozess gut unterstützt werden. Die Nachteile der Methode liegen im mangelnden Vorhandensein von Daten. Aus dieser Erkenntnis konnte die nachhaltige Speicherung der Daten in einem für maschinelles Lernen geeigneten Format als ein Ziel des Projektes abgeleitet werden.

5.11 Visualisierung des Dokuments

Bereits bei Projektbeginn war ein Prototyp vorhanden, der Wörter, die in einer Liste hinterlegt wurden, farblich hervorhebt. Um dies in dieser ersten Version zu erreichen, wurde der Text aus der PDF-Datei extrahiert und von jeder Seite der PDF-Datei ein Bild dieser gespeichert. Zusätzlich zum Text wurde auch die Position des Textes unter Verwendung einer Java-Bibliothek gespeichert. Mithilfe des gespeicherten Textes und der Positionen wurde dieser auf dem GUI mittels eines farbigen Rechtecks, das über die entsprechenden Koordinaten gelegt wurde, hervorgehoben. Wie zu Beginn des Praxisteils dieser Arbeit kurz vorgestellt wurde, wurde für Testzwecke und als Machbarkeitsnachweis eine prototypische Oberfläche erstellt, die anstatt des Bildes eine nach HTML-konvertierte PDF-Datei anzeigt und den Text über dessen Position im HTML und unter Verwendung von CSS hervorhebt (siehe hierzu Seite 48).

70206,	x1:328.11, y1:556.13,	x2:333.65, y2:565.47,	width:595.32, height:841.92,	pagenumber: 24
70207,	x1:333.74, y1:556.13,	x2:336.51, y2:565.47,	width:595.32, height:841.92,	pagenumber: 24
70208,	x1:336.51, y1:556.13,	x2:339.27, y2:565.47,	width:595.32, height:841.92,	pagenumber: 24
70209,	x1:339.27, y1:556.13,	x2:344.81, y2:565.47,	width:595.32, height:841.92,	pagenumber: 24
70210,	x1:344.88, y1:556.13,	x2:351.53, y2:565.47,	width:595.32, height:841.92,	pagenumber: 24

Abbildung 5.39: Mit der Python-Bibliothek PDFMiner erstellter Positions-Index über den kompletten Text einer PDF Datei. Hierbei wurde ein Ausschnitt gewählt, der einem aus fünf Zeichen bestehenden Wort zugeordnet ist. Quelle: Eigene Darstellung.

Die in der ersten Version verwendete Vorgehensweise funktioniert, benutzt aber eine Bibliothek, deren fortwährende Kompatibilität mit dem lebendigen PDF-Standard nicht sichergestellt ist.

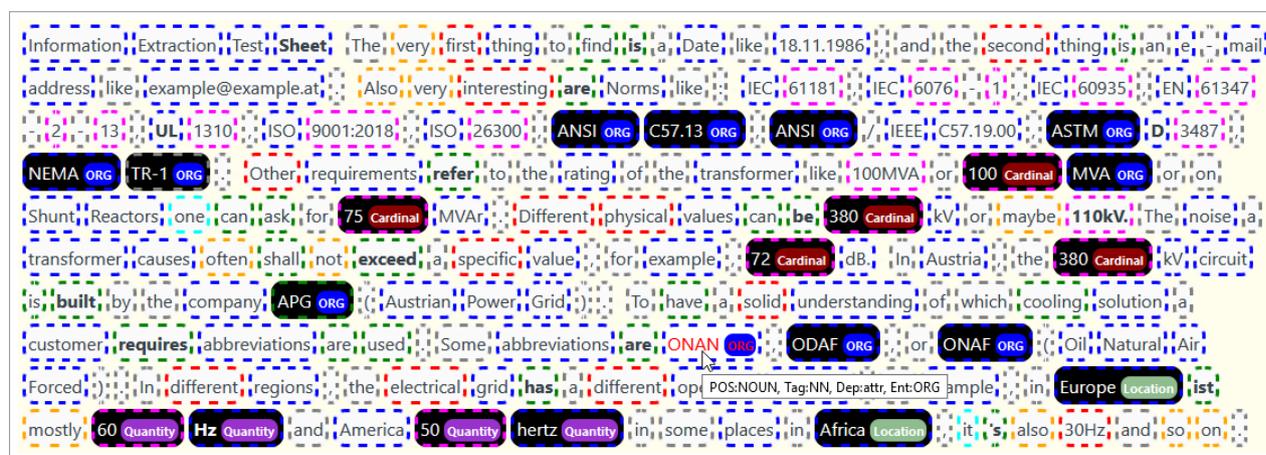


Abbildung 5.40: Als zusätzliche Ausgabe zur Beispielvisualisierung wird eine Visualisierung der einzelnen Worttoken erzeugt. In dieser sind die POS-Tags als Rahmenfarbe und erkannte Entitäten direkt am Token mit schwarzem Hintergrund hervorgehoben. Die Idee hinter dieser Visualisierung ist es beim Erstellen von Regeln, während der Entwicklung, zu unterstützen. Quelle: Eigene Darstellung.

Deshalb, und um eine Vorlage für eine verbesserte Oberfläche zu entwickeln, wurde auf die Extraktion der Koordinaten des Textes aus dem PDF und auf die Verwendung des gängigen, von Mozilla entwickelten, browserbasierten PDF-Viewer PDF.js (<https://mozilla.github.io/pdf.js/>) gewechselt. In Abbildung 5.39 ist zu sehen, wie die Positionsdaten zu einem aus fünf Zeichen bestehenden Wort gespeichert wurden.

Zu diesen Daten wurde ebenfalls der Text extrahiert, der das gesuchte Wort in exakt derselben Zeichenspanne beinhaltet. Angenommen das gesuchte Wort ist *shall*, dann ist das 70.206. Zeichen in diesem extrahierten Text ein *s* und das 70.210. Zeichen das abschließende *l* des Wortes *shall*. Wortbeginn und Wortende wurden in Abbildung 5.39 Rot hinterlegt. Soll nun das Wort, wie in Abbildung 5.41 dargestellt, mit

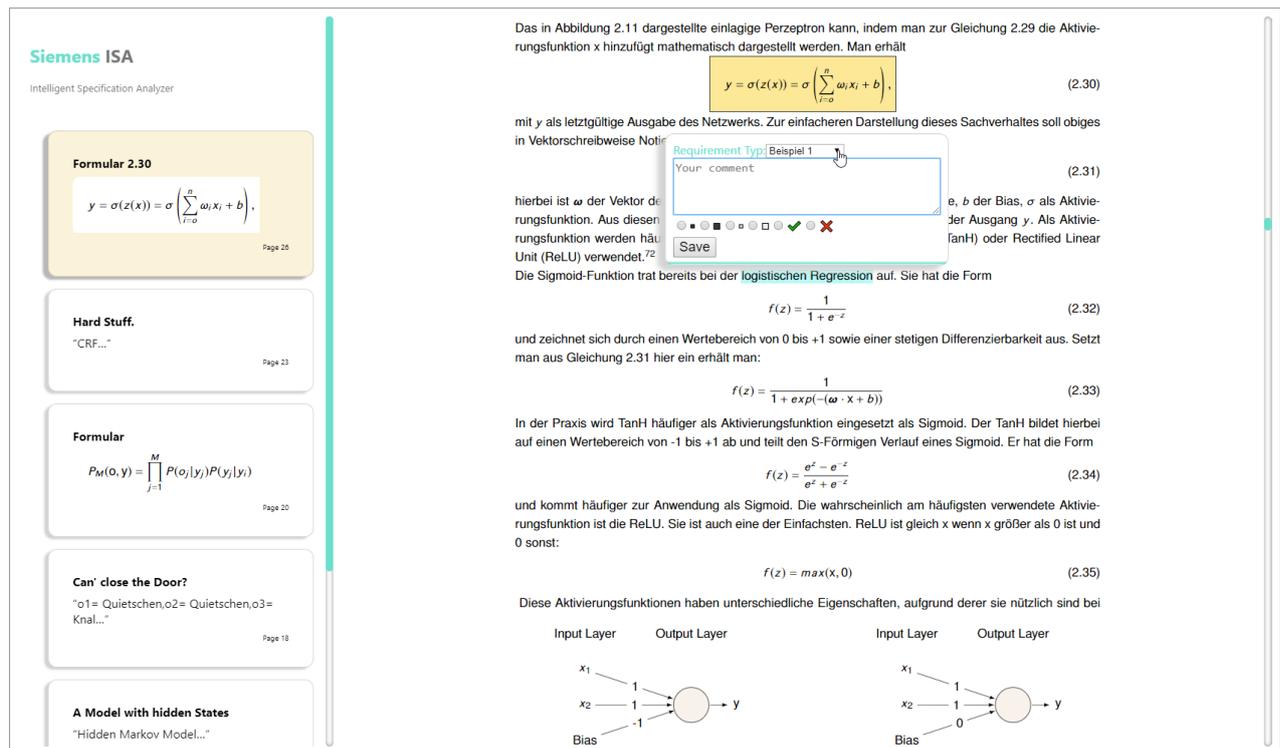


Abbildung 5.41: Prototyp der Oberfläche, der als Teil des Projekts erstellt wurde und die PDF über PDF.js direkt in der Web-App rendert und Annotationen ermöglicht. In der Seitenleiste können gefundene Annotation sowie selbst erstellte Annotationen dargestellt und ausgewählt werden. Außerdem kann über diese zur entsprechenden Stelle im PDF gesprungen werden. Quelle: Eigene Darstellung.

einem Highlight versehen werden, dann muss das Highlight bei den Start-Koordinaten des ersten Buchstaben beginnen (Grün hinterlegt) und bei den Endkoordinaten des letzten Buchstaben (Gelb hinterlegt) enden. In Abbildung 5.41 ist ein Bild der Oberfläche zu sehen. Diese zeigt im Browser die PDF auf der rechten Seite an und eine Sammlung der Hervorhebungen in einer Seitenleiste links. Wird auf die Miniatur in der Seitenleiste geklickt, springt die PDF automatisch an diesen Punkt. Die PDF-Datei wird auf der rechten Seite unter Verwendung von PDF.js auf ein HTML-Canvas-Element gerendert. Der Originalprototyp hatte Probleme mit dem Zoomen der Dateien, weil dieser die PDF aus Bildern aufgebaut hat. Zudem konnte er auch keinen Text im PDF-Dokument markieren, weil es sich um ein Bild handelte. Dies funktioniert mit PDF.js beides, wie auch in PDF-Viewern, die außerhalb des Browsers arbeiten. Ein weiterer Vorteil von PDF.js ist, dass dieser der Standard PDF-Viewer des Firefox Browsers ist und somit einer aktiven Entwicklung unterliegt. Aus diesem Grund ist abzusehen, dass zukünftige Änderungen am PDF-Standard vom Entwickler in PDF.js berücksichtigt werden. Um die Highlights im PDF anzuzeigen, wird ein Annotation Layer in PDF.js verwendet. Der Benutzer kann Text im PDF auswählen und, wie in Abbildung 5.41 zu sehen ist, das Highlight einer Anforderung zuordnen und einen Kommentar hinterlegen. Für Annotation und Texthighlighting wurden React-Komponenten verwendet, bei denen es sich um eine Java-Script-Bibliothek handelt, die einen komponentenbasierten Ansatz einfacher gestaltet und HTML und Code in einer Datei zulässt. Die Komponenten zeichnen sich durch Wiederverwendbarkeit aus.

Bereits in den frühen Entwicklungsphasen des Projekts dienen die erstellten Visualisierungen dazu, einerseits einen Ausblick auf das fertige Produkt zu vermitteln und andererseits NLP-Algorithmen zu testen. Durch eine grafische Ausgabe der Ergebnisse kann ein besseres Verständnis der Ergebnisse gewonnen werden. Auch wenn das schlussendliche Aussehen des fertigen Tools zu diesem Zeitpunkt noch nicht absehbar ist, kann angenommen werden, dass die in Abbildung 5.41 gezeigte Komponente für den Part des

PDF-Viewers wegweisend sein wird. Neben den verwendeten Techniken des NLP sind die Visualisierung und die Interaktion mit den angezeigten Daten ein wichtiger Bestandteil zur Unterstützung des Anwenders.

6. CONCLUSIO/ERGEBNIS

In den vergangenen Kapiteln wurde beispielhaft gezeigt, welche Methoden zur Informationsextraktion aus natürlichsprachlichem Text getestet wurden. Es wurde versucht, einen Überblick über die Methodik der einzelnen Ansätze zu geben und deren Eignung für eine Implementierung zu ergründen. Zusätzlich ist anhand von Beispielprogrammen getestet worden, ob und wie sich die Ansätze mit verschiedenen Programmierumgebungen umsetzen lassen und wie hierzu eine Visualisierung realisiert werden kann.

Beim Thema Visualisierung wurde versucht, den Inhalt von PDF-Dokumenten im Webbrowser anzuzeigen und bestimmte Wörter hervorzuheben. Zwei verschiedene Ansätze wurden hierzu getestet. Der erste Ansatz war das Konvertieren des Dokumentes in ein HTML-Dokument. Aus diesem wurde der Text extrahiert und über einen Index wurden die einzelnen Highlights mittels CSS auf den entsprechenden Wörtern realisiert. Der zweite Ansatz bestand darin, die Positionen eines Wortes auf der jeweiligen Seite direkt aus dem PDF-Dokument zu extrahieren und diese Information auf Buchstabenebene den Wörtern zuzuordnen. Das Hervorheben der Wörter wurde hierbei über das Einbetten von Highlights in einen HTML-PDF-Renderer realisiert. Beide Methoden haben in den gezeigten Beispielen funktioniert, in längeren PDF-Dokumenten stellte sich die zweite Methode jedoch als stabiler heraus. Der Grund hierfür ist, dass der Index, der bei der ersten Methode verwendet wurde, sowohl bei Übergängen in Kopf- und Fußzeilen als auch beim Übergang vom Inhaltsverzeichnis in das Dokument Zeichen nicht richtig berücksichtigte, was Probleme mit verschobenen Hervorhebungen verursachte. Ein weiterer Nachteil war, dass die Highlights von hinten nach vorne in das Dokument über CSS eingefügt werden mussten, um den Index nicht zu verschieben. Sich überlappende Hervorhebungen führten hierbei zu Schwierigkeiten. Die Methode der Hervorhebung über Koordinaten auf dem gerenderten PDF im Webbrowser war diesbezüglich stabiler und wird deswegen für einen solchen Anwendungsfall empfohlen.

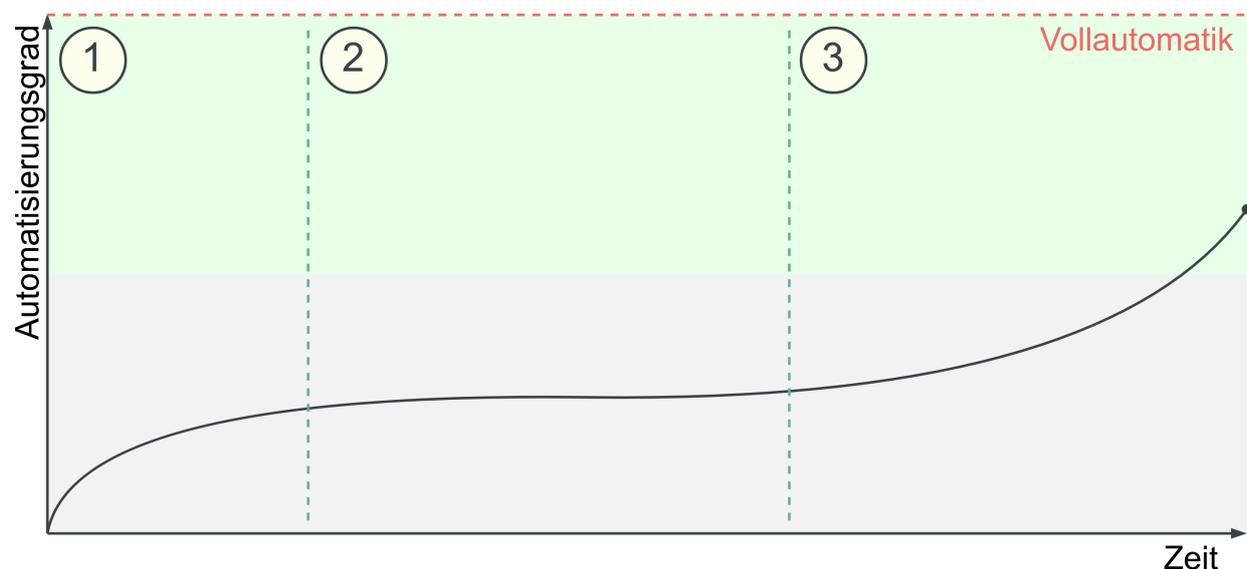


Abbildung 6.1: Qualitative Darstellung eines möglichen Verlaufs des Automatisierungsgrades, der durch den Einsatz eines auf maschinellem Lernen basierenden Werkzeuges zu erwarten ist. Quelle: Eigene Darstellung.

Für die Extraktion von Informationen aus den einzelnen Texten wurde versucht einen inkrementellen Ansatz zu finden. Hierzu wurde mit klassischen, regulären Ausdrücken begonnen. Diese funktionieren für sich wiederholende Muster wie E-Mail-Adressen und Telefonnummern oder Normen gut. Ebenfalls gut funktionierte das Erkennen von Zahlen mit nachfolgenden Einheiten; die Herausforderung liegt hierbei darin,

dass sich gewisse Zahl-Einheit-Kombinationen mehrere Hundert Mal im Dokument finden und die Auswahl des gesuchten Wertes weiterhin schwierig war. Als nächste Methode wurde die Verbindung von regulären Ausdrücken mit Programmbibliotheken, die ein Sprachmodell beinhalten, getestet. Dies wurde als hybrider Ansatz bezeichnet und auf Basis der Annahme, dass es für POS-Tagging – auch ohne entsprechendes Training auf der Anwendungsdomäne – entsprechend gute Modelle frei verfügbar gibt, getestet. An dieser Stelle wurde versucht einen Nutzen aus den gut passenden POS-Tags zu ziehen, was sich als schwierig herausstellte. Einfacher zu nutzen sind Entity Tags, die aber ein zusätzliches Training, auf entsprechend aufbereiteter Domäne, benötigen würden. Weil diese Trainingsdaten derzeit noch nicht vorliegen, konnte der hybride Ansatz keine wesentlichen Vorteile gegenüber dem alleinigen Einsatz von regulären Ausdrücken bieten. Nach einem Versuch, Open Information Extraction anzuwenden, was aufgrund der Einfachheit des durchgeführten Versuchs noch keine zuverlässige Aussage zuließ, wurde probiert die Eignung modernster Deep-Learning-Methoden zu ergründen. Um einen allgemeinen Einstieg in die Thematik zu finden, wurde versucht sich am Beispiel von Übersetzungen an die Themen Transfer Learning und Attention-Mechanismus anzunähern. Um diese Technologien im Bereich der Informationsextraktion einzusetzen, wurde der Machine Reading Comprehension (MRC) Task als geeignete Methode identifiziert. Aufgrund des Vorhandenseins vieler frei verfügbarer Implementierungen und Datensätze konnte die Methode am SQUAD-Datensatz mit einem für den praktischen Einsatz als ausreichend anzusehenden Score validiert werden. Unter der Annahme, dass in absehbarer Zukunft der Mensch als letzte Entscheidungsinstanz nicht zu ersetzen sein wird und dieser damit die Auswahl aus einer Sammlung möglicher Antworten an das Produkt vornehmen kann, stellt die Implementierung solcher Methoden ein als sinnvoll erscheinendes Langzeitziel dar.

Es wird auf Basis der durchgeführten Versuche ein inkrementeller Ansatz empfohlen. Dieser sollte bei der Implementierung gängiger Methoden wie einer Search Engine mit Indexierung beginnen, um schnelle Suchanfragen auf großen Datensätzen zu ermöglichen. Eine entsprechende Speicherung und Aufbereitung der im laufenden Betrieb anfallenden Daten soll hierbei zu einer Datenbasis führen, die Methoden wie Named Entity Recognition und Machine Reading Comprehension in zukünftigen Iterationen des Werkzeuges ermöglicht. Der in Abbildung 2.1 dargestellte Automatisierungsgrad illustriert auf qualitative Weise den Verlauf des Nutzens über diese Phasen hinweg. Der grau hinterlegte Bereich soll dabei den Bereich der Routineaufgaben darstellen und der grün hinterlegte Bereich jenen Teil der Arbeit, der Spezialwissen und Erfahrung benötigt. Die Zahlen auf gelbem Hintergrund beschreiben Implementierungsphasen – das meint jene Phasen ab Beginn des Produktiveinsatzes – die ein solches Werkzeug durchlaufen müsste. In der ersten Phase sind es regelbasierte Methoden, die die AnwenderInnen damit unterstützen, dass ihnen wichtige Passagen hervorgehoben, passende Begleitdokumente wie Normen oder bereits erarbeitete Lösungen zur Verfügung gestellt werden und ein optimierter Arbeitsablauf des Befüllens einer Key-Data-Matrix bereitgestellt wird. Auf die initialen Verbesserungen durch „digital ergonomische Arbeitsabläufe“, das Vermeiden von Fehlern und das Teilen von Wissen stellt sich eine zweite Phase der Sättigung und des Datensammelns ein, der – wenn entsprechende Daten gesammelt wurden – eine dritte Phase folgen kann. In einer solchen können den NutzerInnen weitere Komfortfunktionen geboten werden, indem sie von, durch den Einsatz von Deep Learning, nochmals intelligenter wirkenden Systemen unterstützt werden.

7. AUSBLICK

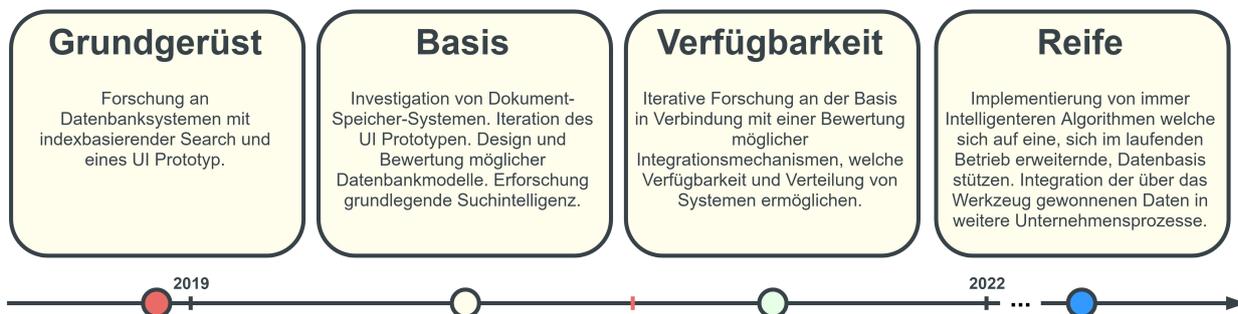


Abbildung 7.1: Lebenszyklus eines auf maschinellem Lernen basierenden Softwareprojektes. Quelle: Eigene Darstellung.

Mit Abgabe der vorliegenden Arbeit befindet sich das Projekt, im Rahmen dessen diese Arbeit erstellt wurde, in etwa an der Stelle, wo sich der rote Strich in Abbildung 3.1 befindet. Mit der vorliegenden Arbeit wurden Konzepte aufgezeigt, die in verschiedene Phasen eines solchen Projektes Einzug finden. Nicht erwähnt wurde hierbei – da nicht Teil der Zielsetzung – dass sich bei Einführung eines solchen Werkzeuges Synergien und Seiteneffekte ergeben, aus denen zusätzlicher Nutzen gezogen werden kann. Durch die Anbindung von ML-Systemen an ein solches Werkzeug ergibt sich inhärent, dass Möglichkeiten zur Datenakquisition zur Verfügung stehen. Die gewonnenen Daten können in Systemen, wie beispielsweise einem Knowledge Graph, verwendet werden. Ganz allgemein sind in den Daten gegebenenfalls Informationen zu finden, mit denen auf den ersten Blick nicht gerechnet werden würde. Möglicherweise sind anhand der erhaltenen Spezifikation Rückschlüsse auf die Wahrscheinlichkeit eines erfolgreichen Angebots möglich, ohne diese jemals manuell gelesen zu haben. Darüber hinaus können über eine Verbindung der Daten aus bereits erfolgreich abgeschlossenen Aufträgen gegebenenfalls Qualitätsverbesserungen in der Angebotslegung selbst erreicht werden. Die Möglichkeiten solcher Systeme sind weitreichend und Data Warehousing ist eine Aufgabe, die viele Wirtschaftsbereiche in den kommenden Jahren wahrnehmen und als Chance begreifen müssen.

LITERATURVERZEICHNIS

Online-Quellen

Alammar, Jay (2019): *The illustrated transformer*. URL: <http://jalammar.github.io/illustrated-transformer/> [Stand: 01.10.2019].

Debut, Lysandre (2019): *PyTorch-Transformers*. URL: <https://github.com/huggingface/pytorch-transformers> [Stand: 05.09.2019].

Devlin, Jacob; Chang, Ming-Wei et al. (2018): *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805. URL: <http://arxiv.org/abs/1810.04805> [Stand: 13.08.2019].

Devlin, Jacob; Chang, Ming-Wei et al. (2019): *google research bert*. URL: <https://github.com/google-research/bert> [Stand: 30.09.2019].

Honnibal, Matthew; Montani, Ines (2019): *spaCy: Industrial-Strength Natural Language Processing in Python*. URL: <https://spacy.io/usage/rule-based-matching> [Stand: 19.08.2019].

Jurafsky, Dan; Martin, James H (Sep. 2018): *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. URL: <https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf> [Stand: 16.04.2019].

Microsoft (2019): *MS MARCO Passage Retrieval Leaderboard (10/26/2018 - 09/03/2019)*. URL: <http://www.msmarco.org/leaders.aspx> [Stand: 03.09.2019].

Mikaelian, Félix et al. (2019): *An End-To-End Closed Domain Question Answering System*. URL: <https://github.com/cdqa-suite> [Stand: 10.07.2019].

Olah, Christopher (2015): *Understanding lstm networks—colah's blog*. URL: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> [Stand: 19.07.2019].

Papers-With-Code (2019): *question-answering-on-squad20*. URL: <https://paperswithcode.com/sota/question-answering-on-squad20> [Stand: 16.06.2019].

Rajpurkar, Pranav; Jia, Robin; Liang, Percy (2018): *Know What You Don't Know: Unanswerable Questions for SQuAD*. arXiv: 1806.03822. URL: <http://arxiv.org/abs/1806.03822> [Stand: 13.08.2019].

Tensorflow-2.0rc (2019): *Google Tensorflow 2.0 docs*. URL: <https://github.com/tensorflow/docs/blob/r2.0rc/site/en/r2/tutorials/text> [Stand: 01.10.2019].

Gedruckte Werke

Akbik, Alan; Blythe, Duncan; Vollgraf, Roland (2018): „Contextual string embeddings for sequence labeling“. In: *Proceedings of the 27th International Conference on Computational Linguistics*.

Akbik, Alan; Broß, Jürgen (2009): „Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns“. In: *www workshop*. Bd. 48.

Akbik, Alan; Lösser, Alexander (2012): „Kraken: N-ary facts in open information extraction“. In: *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*. Association for Computational Linguistics.

Alaei, Ali Reza; Becken, Susanne; Stantic, Bela (2019): „Sentiment analysis in tourism: capitalizing on big data“. In: *Journal of Travel Research* 58.2.

Allahyari, Mehdi et al. (2017): „A brief survey of text mining: Classification, clustering and extraction techniques“. In: *arXiv preprint arXiv:1707.02919*.

Arnold, Sebastian et al. (2016): „Robust named entity recognition in idiosyncratic domains“. In: *arXiv preprint arXiv:1608.06757*.

Auer, Sören et al. (2007): „Dbpedia: A nucleus for a web of open data“. In: *The semantic web*. Springer.

Banko, Michele et al. (2007): „Open information extraction from the web.“ In: *Ijcai*. Bd. 7.

Bassa, A; Kröll, M; Kern, R (Jan. 2018): „GerIE - An open information extraction system for the german language“. In: *Journal of Universal Computer Science* 24.

Bengio, Yoshua; Simard, Patrice; Frasconi, Paolo et al. (1994): „Learning long-term dependencies with gradient descent is difficult“. In: *IEEE transactions on neural networks* 5.2.

Brémaud, Pierre (2017): *Discrete Probability Models and Methods - Probability on Graphs and Trees, Markov Chains and Random Fields, Entropy and Coding*. Volume 78. Berlin, Heidelberg: Springer. ISBN: 978-3-319-43476-6.

Buschmann, F et al. (1996): *Software Patterns*. John Wiley & Sons.

Cambria, Erik; White, Bebo (2014): „Jumping NLP curves: A review of natural language processing research“. In: *IEEE Computational intelligence magazine* 9.2.

Carstensen, Kai-Uwe et al., Hrsg. (2010): *Computerlinguistik und Sprachtechnologie*. 3. Aufl. Heidelberg: Spektrum. ISBN: 978-3-8274-2023-7.

Cer, Daniel et al. (2018a): „Universal sentence encoder“. In: *arXiv preprint arXiv:1803.11175*.

Cer, Daniel et al. (2018b): „Universal sentence encoder“. In: *arXiv preprint arXiv:1803.11175*.

Chen, Ming-Syan; Han, Jiawei; Yu, Philip S. (1996): „Data mining: an overview from a database perspective“. In: *IEEE Transactions on Knowledge and data Engineering* 8.6.

Chiticariu, Laura; Li, Yunyao; Reiss, Frederick R (2013): „Rule-based information extraction is dead! long live rule-based information extraction systems!“ In: *Proceedings of the 2013 conference on empirical methods in natural language processing*.

Cho, Kyunghyun et al. (2014): „Learning phrase representations using RNN encoder-decoder for statistical machine translation“. In: *arXiv preprint arXiv:1406.1078*.

Chung, Junyoung et al. (2014): „Empirical evaluation of gated recurrent neural networks on sequence modeling“. In: *arXiv preprint arXiv:1412.3555*.

- Cohen, Aaron M; Hersh, William R (2005): „A survey of current work in biomedical text mining“. In: *Briefings in bioinformatics* 6.1, S. 57–71.
- Cunningham, Hamish et al. (2002): „GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications“. In: *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*.
- Dai, Zihang et al. (2019): „Transformer-xl: Attentive language models beyond a fixed-length context“. In: *arXiv preprint arXiv:1901.02860*.
- Del Corro, Luciano; Gemulla, Rainer (2013): „Clausie: clause-based open information extraction“. In: *Proceedings of the 22nd international conference on World Wide Web*. ACM.
- Dietrich, Rainer (2010): *Automatische Textwörterbücher: Studien zur maschinellen Lemmatisierung verbaler Wortformen des Deutschen*. Bd. 2. Walter de Gruyter. ISBN: 978-3-111-92565-3.
- Dinov, Ivo D (2018): *Lazy Learning: Classification Using Nearest Neighbors*. Springer.
- Do, Minh N; Vetterli, Martin (2002): „Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance“. In: *IEEE transactions on image processing* 11.2.
- Džeroski, Sašo (2009): *Relational data mining*. Springer. ISBN: 978-3-540-42289-1.
- Erkan, Günes; Radev, Dragomir R (2004): „Lexrank: Graph-based lexical centrality as salience in text summarization“. In: *Journal of artificial intelligence research* 22.
- Etzioni, Oren et al. (2011): „Open information extraction: The second generation“. In: *Twenty-Second International Joint Conference on Artificial Intelligence*.
- Fayyad, Usama M; Piatetsky-Shapiro, Gregory; Smyth, Padhraic et al. (1996): „Knowledge Discovery and Data Mining: Towards a Unifying Framework.“ In: *KDD*. Bd. 96.
- Fayyad, Usama; Piatetsky-Shapiro, Gregory; Smyth, Padhraic (1996): „From data mining to knowledge discovery in databases“. In: *AI magazine* 17.3.
- Feldman, Ronen; Dagan, Ido (1995): „Knowledge Discovery in Textual Databases (KDT).“ In: *KDD*. Bd. 95.
- Ferrucci, David; Lally, Adam (2003): „Accelerating corporate research in the development, application and deployment of human language technologies“. In: *SEALTS '03: Proceedings of the HLT-NAACL 2003 workshop on Software engineering and architecture of language technology systems*. Morristown, NJ, USA: Association for Computational Linguistics. DOI: <https://dx.doi.org/10.3115/1119226.1119236>.
- Frawley, William J; Piatetsky-Shapiro, Gregory; Matheus, Christopher J (1992): „Knowledge discovery in databases: An overview“. In: *AI magazine* 13.3.
- Frochte, Jörg (2018): *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. Volume 1. Carl Hanser Verlag GmbH Co KG. ISBN: 978-3-446-45291-6.
- Gantz, John; Reinsel, David (2012): „The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east“. In: *Technical Report 1. IDC*. URL: <http://informationwatch.org/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf> [Stand: 09.04.2019].

- Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016): *Deep Learning*. MIT Press. URL: <http://www.deeplearningbook.org> [Stand: 09.04.2019].
- Goutte, Cyril; Gaussier, Eric (2005): „A probabilistic interpretation of precision, recall and F-score, with implication for evaluation“. In: *European Conference on Information Retrieval*. Springer.
- Gundecha, Pritam; Liu, Huan (2012): „Mining social media: a brief introduction“. In: *New Directions in Informatics, Optimization, Logistics, and Production*. Informs.
- Hochreiter, Sepp; Schmidhuber, Jürgen (1997): „Long short-term memory“. In: *Neural computation* 9.8.
- Hoffart, Johannes et al. (2011): „Robust disambiguation of named entities in text“. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Hotho, Andreas; Nürnberger, Andreas; Paaß, Gerhard (2005): „A brief survey of text mining.“ In: *Ldv Forum*. Bd. 20. 1.
- Jiang, Jing (2012): „Information extraction from text“. In: *Mining text data*. Springer.
- Jivani, Anjali Ganesh et al. (2011): „A comparative study of stemming algorithms“. In: *Int. J. Comp. Tech. Appl* 2.6.
- Jungermann, Felix (2006): *Named entity recognition mit conditional random fields*.
- Jungermann, Felix (2007): „Named entity recognition without domain-knowledge using conditional random fields“. In: *Workshop Notes of the Machine Learning for Natural Language Processing Workshop*.
- Kalinke, Thomas; Seelen, Werner von (1997): „Kullback-Leibler Distanz als Maß zur Erkennung nicht rigider Objekte“. In: *Mustererkennung 1997*. Springer.
- Kasneji, Gjergji; Suchanek, Fabian; Weikum, Gerhard (2006): *Yago-a core of semantic knowledge*. Max-Planck-Institut für Informatik.
- King, Gary; Lowe, Will (2003): „An automated information extraction tool for international conflict data with performance as good as human coders: A rare events evaluation design“. In: *International Organization* 57.3.
- Kleene, Stephen Cole (1951): „Representation of events in nerve nets and finite automata“. In: *RAND PROJECT AIR FORCE SANTA MONICA CA*.
- Kluegl, Peter et al. (Jan. 2016): „UIMA Ruta: Rapid development of rule-based information extraction applications“. In: *Natural Language Engineering* 22 (01). ISSN: 1469-8110. DOI: 10.1017/S1351324914000114. URL: https://journals.cambridge.org/article_S1351324914000114.
- Krafft, Manfred (1996): *Der Ansatz der logistischen Regression und seine Interpretation*. Freiburg i.B.
- Lafferty, John; McCallum, Andrew; Pereira, Fernando CN (2001): „Conditional random fields: Probabilistic models for segmenting and labeling sequence data“. In: *Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001)*.
- Lai, Guokun et al. (2017): „RACE: Large-scale ReAding Comprehension Dataset From Examinations“. In: *arXiv preprint arXiv:1704.04683*.

- Lee, Jinhyuk et al. (2019): „BioBERT: pre-trained biomedical language representation model for biomedical text mining“. In: *arXiv preprint arXiv:1901.08746*.
- Li, Xin; Roth, Dan (2002): „Learning question classifiers“. In: *Proceedings of the 19th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics.
- Liu, Linqing et al. (2018): „Generative adversarial network for abstractive text summarization“. In: *Thirty-Second AAAI Conference on Artificial Intelligence*.
- Loper, Edward; Bird, Steven (2002): „NLTK: The Natural Language Toolkit“. In: *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*. Philadelphia: Association for Computational Linguistics.
- Lösch, Mathias (2011): „A multidisciplinary search engine for scientific open access documents“. In: *EBSLG Annual General Conference, 18.-21.05. 2010, Cologne. Selected papers*. Bd. 2.
- Lovins, Julie Beth (1968): „Development of a stemming algorithm“. In: *Mech. Translat. & Comp. Linguistics* 11.1-2.
- Luhn, Hans Peter (1957): „A statistical approach to mechanized encoding and searching of literary information“. In: *IBM Journal of research and development* 1.4.
- Maaten, Laurens van der; Hinton, Geoffrey (2008): „Visualizing data using t-SNE“. In: *Journal of machine learning research* 9.Nov.
- Manning, Christopher D. et al. (2014): „The Stanford CoreNLP Natural Language Processing Toolkit“. In: *Association for Computational Linguistics (ACL) System Demonstrations*. URL: <http://www.aclweb.org/anthology/P/P14/P14-5010>.
- Mattmann, Chris; Zitting, Jukka (2011): *Tika in action*. Manning Publications Co.
- McCallum, Andrew; Freitag, Dayne; Pereira, Fernando CN (2000): „Maximum Entropy Markov Models for Information Extraction and Segmentation.“ In: *Icml*. Bd. 17. 2000.
- Mikolov, Tomas et al. (2013): „Efficient estimation of word representations in vector space“. In: *arXiv preprint arXiv:1301.3781*.
- Moniruzzaman, ABM; Hossain, Syed Akhter (2013): „Nosql database: New era of databases for big data analytics-classification, characteristics and comparison“. In: *arXiv preprint arXiv:1307.0191*.
- Nguyen, Tri et al. (2016): „MS MARCO: A Human-Generated MACHine Reading COmprehension Dataset“. In: *ICLR 2017 conference submission*.
- Pang, Bo; Lee, Lillian et al. (2008): „Opinion mining and sentiment analysis“. In: *Foundations and Trends® in Information Retrieval* 2.1–2.
- Peters, Matthew E. et al. (2018): „Deep contextualized word representations“. In: *Proc. of NAACL*.
- Pugh, Kenneth (1994): *UNIX for the MS-DOS user*. PTR Prentice Hall.
- Rabiner, L.; Juang, B. (1986): „An introduction to hidden Markov models“. In: *ASSP Magazine, IEEE* 3.1.

- Radford, Alec; Narasimhan, Karthik et al. (2018): „Improving language understanding by generative pre-training“. In: *URL <https://s3-us-west-2.amazonaws.com/openai-assets/researchcovers/languageunsupervised/languageunderstandingpaper.pdf>*.
- Radford, Alec; Wu, Jeff et al. (2019): „Language Models are Unsupervised Multitask Learners“. In: *OpenAI Blog, 2019, 1. Jg., Nr. 8*.
- Raina, Rajat et al. (2004): „Classification with hybrid generative/discriminative models“. In: *Advances in neural information processing systems*.
- Rice, John A (2006): *Mathematical statistics and data analysis*. Cengage Learning. ISBN: 978-8-131-51954-7.
- Richardson, Leonard (2016): „Beautiful Soup: We called him Tortoise because he taught us“. In: *Crummy.com. Np*.
- Rong, Xin (2014): „word2vec parameter learning explained“. In: *arXiv preprint arXiv:1411.2738*.
- Schmitz, Michael et al. (2012): „Open language learning for information extraction“. In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics.
- Sebastiani, Fabrizio (2002): „Machine learning in automated text categorization“. In: *ACM computing surveys (CSUR)* 34.1.
- Sherstinsky, Alex (2018): „Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network“. In: *arXiv preprint arXiv:1808.03314*.
- Singh, Sukhpreet (2013): „Optical character recognition techniques: a survey“. In: *Journal of emerging Trends in Computing and information Sciences* 4.6.
- Smilkov, Daniel et al. (2016): „Embedding projector: Interactive visualization and interpretation of embeddings“. In: *arXiv preprint arXiv:1611.05469*. URL: projector.tensorflow.org.
- Smith, Ray (2007): „An overview of the Tesseract OCR engine“. In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. Bd. 2. IEEE.
- Sokolova, Marina; Japkowicz, Nathalie; Szpakowicz, Stan (2006): „Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation“. In: *Australasian joint conference on artificial intelligence*. Springer.
- Sutton, Charles; McCallum, Andrew (Apr. 2012): „An Introduction to Conditional Random Fields“. In: *Found. Trends Mach. Learn.* 4.4, S. 267–373. ISSN: 1935-8237. DOI: 10.1561/2200000013. URL: <http://dx.doi.org/10.1561/2200000013> [Stand: 16. 04. 2019].
- Thompson, Ken (1968): „Programming techniques: Regular expression search algorithm“. In: *Communications of the ACM* 11.6.
- Timischl, Wolfgang (1995): *Qualitätssicherung: Statistische Methoden*. Hanser. ISBN: 978-3-446-43238-3.
- Uysal, Alper Kursat; Gunal, Serkan (2014): „The impact of preprocessing on text classification“. In: *Information Processing & Management* 50.1.

- Vaswani, Ashish et al. (2017): „Attention is all you need“. In: *Advances in neural information processing systems*.
- Vig, Jesse (2019): „A Multiscale Visualization of Attention in the Transformer Model“. In: *arXiv preprint arXiv:1906.05714*. URL: <https://arxiv.org/abs/1906.05714>.
- Vijayarani, S; Ilamathi, Ms J; Nithya, Ms (2015): „Preprocessing techniques for text mining-an overview“. In: *International Journal of Computer Science & Communication Networks* 5.1.
- Webster, Jonathan J; Kit, Chunyu (1992): „Tokenization as the initial phase in NLP“. In: *COLING 1992 Volume 4: The 15th International Conference on Computational Linguistics*. Bd. 4.
- Willett, Peter (2006): „The Porter stemming algorithm: then and now“. In: *Program* 40.3.
- Wu, Fei; Weld, Daniel S (2010): „Open information extraction using Wikipedia“. In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics.
- Yahyapour, Ramin (2018): „E-Science Infrastrukturen“. In: *Informatik-Spektrum* 41.6.
- Yang, Yinfei et al. (2019): „Multilingual Universal Sentence Encoder for Semantic Retrieval“. In: *CoRR* abs/1907.04307. arXiv: 1907.04307. URL: <http://arxiv.org/abs/1907.04307>.
- Yang, Zhilin et al. (2019): „XLNet: Generalized Autoregressive Pretraining for Language Understanding“. In: *CoRR* abs/1906.08237. arXiv: 1906.08237. URL: <http://arxiv.org/abs/1906.08237> [Stand: 13. 08. 2019].
- Ye, Qi et al. (2018): „When and Why are pre-trained word embeddings useful for Neural Machine Translation“. In: *HLT-NAACL*.
- You, Yang et al. (2019): „Reducing BERT Pre-Training Time from 3 Days to 76 Minutes“. In: *arXiv preprint arXiv:1904.00962*.
- Young, Tom et al. (2018): „Recent trends in deep learning based natural language processing“. In: *ieee Computational intelligence magazine* 13.3.
- Zafarani, Reza; Abbasi, Mohammad Ali; Liu, Huan (2014): *Social media mining: an introduction*. Cambridge University Press. ISBN: 978-1-107-01885-3.
- Zheludkov, Michael; Isachenko, Timur et al. (2017): *High Performance in-memory computing with Apache Ignite*. Lulu. com. ISBN: 978-1-365-73235-5.

ABBILDUNGSVERZEICHNIS

1.1	Luftaufnahme, auf der der Standort Weiz der Siemens AG zu sehen ist. Quelle: Siemens-Pressefoto.	1
2.1	Darstellung eines Knowledge Discovery Prozesses. Quelle:Frochte (2018), S. 16.	5
3.1	Das Aufteilen eines Satzes in einzelne Wörter mit Python und NLTK. Quelle: Eigene Darstellung.	13
3.2	Beispiele für Stoppwörter mit Python und NLTK. Quelle: Eigene Darstellung.	15
3.3	Hier ist der Unterschied zwischen Affix- und Wörterbuchbasierter Grundformbildung zu sehen. Quelle: Eigene Darstellung.	16
3.4	Modell das einen Eingangsvektor einem Ausgangsvektor zuordnet mit Rückführung. Quelle: Eigene Darstellung.	18
3.5	Markov-Kette als Prozessschaubild dargestellt. Quelle: Eigene Darstellung.	19
3.6	Darstellung eines HMM mit den unsichtbaren Zuständen mit grünem Hintergrund und den beobachtbaren Zuständen auf gelbem Hintergrund. Quelle: Eigene Darstellung.	21
3.7	Modell eines HMM mit den Labeln auf weissem Hintergrund und den beobachtbaren Zuständen auf gelbem Hintergrund. Quelle: Eigene Darstellung.	22
3.8	Modell eines MEMM mit den Labeln auf weissem Hintergrund und den beobachtbaren Zuständen auf gelbem Hintergrund. Quelle: Eigene Darstellung.	23
3.9	Qualitativer Plot einer Sigmoid-Funktion p über die Werte z , im Deutschen auch mit Schwannenhalsfunktion oder S-Kurve benannt. Quelle: Eigene Darstellung.	25
3.10	Modell eines CRF Faktorgraphen für den Sonderfall eines linear chain CRF. Quelle: Eigene Darstellung.	26
3.11	Das mathematische Modell eines künstlichen Neurons. Quelle: Eigene Darstellung.	28
3.12	Auf der linken Seite ist ein logisches UND und auf der rechten Seite ein logisches ODER. Beide Funktionen wurden mithilfe eines einzigen Perzeptron mit je 2 Eingängen und dem Bias erstellt. Quelle: Jurafsky; Martin (2018), S. 135. (modifiziert)	29
3.13	Darstellung eines logischen XOR als Verbindung mehrerer Perzeptren. Quelle: Jurafsky; Martin (2018), S. 136. (modifiziert)	30
3.14	Neuronales Feed Forward Netzwerk mit einem Hidden Layer. Wie auch in den vorherigen Beispielen, können die Layer mit einem Bias versehen werden. Quelle: Eigene Darstellung.	30
3.15	Neuronales RNN mit einem Hidden Layer. Jene Wege, die für Rückführungen verwendet werden können, sind als roter, blauer und grüner Pfeil dargestellt. Zusätzlich zu den Rückführungen der Werte aus dem Hidden Layer wäre es auch möglich, den Wert y an die vorherigen Layer rückzuführen. Quelle: Eigene Darstellung	32
3.16	Darstellung eines, über die Zeit abgerollten, Recurrent Neural Network. Quelle: Olah (2015) (modifiziert)	33
3.17	Innenleben eines Recurrent Neural Network mit TanH als Aktivierungsfunktion. Quelle: Olah (2015) (modifiziert)	33
3.18	Darstellung eines, über die Zeit abgerollten, LSTM. Quelle: Olah (2015) (modifiziert)	34
3.19	Innerer Aufbau eines LSTM, gegliedert in die Teilbereiche I) Forget, II) Update und III) Publish. Quelle: Olah (2015) (modifiziert)	34
3.20	Visualisierung einer für das Wort <i>bridge</i> , mit Word2Vec erstellten, Einbettung. Quelle: Smilkov et al. (2016), S. 1-4.	37

3.21	Die Werte für <i>battle</i> und <i>fool</i> als Vektoren dargestellt. Bei den Werken <i>As You Like It</i> und <i>Twelfth Night</i> handelt es sich um Kömodien. Quelle: Jurafsky; Martin (2018), S. 109. (modifiziert)	38
3.22	Tupel an Trainingsdaten, für eine Spanne von $L = \pm 2$ Token, bei einem SGNS Modell. Bei den als <i>t</i> bezeichneten Token handelt es sich um das Target-Word. Bei <i>c</i> handelt es sich um die Context-Words und bei <i>n</i> um Noise-Words welche der Generierung von $L \times k$, mit k negativen Samples dienen. In diesem Beispiel wurde k mit 2 gewählt. Quelle: Eigene Darstellung.	42
4.1	Leaderboard mit den State-of-the-Art-(SOTA)-Methoden für QA, unter Verwendung des SQuAD 2.0. Quelle: Papers-With-Code (2019) (modifiziert)	44
5.1	Schematischer Aufbau des Systems mit vom Client entkoppelter Datenverarbeitung. Die Message Queue übernimmt hierbei die Orchestration der Komponenten. Quelle: Eigene Darstellung.	47
5.2	Benutzeroberfläche der in Django erstellten Testapplikation, die für die Entwicklung verwendet wird. Aufgerufen wurde diese im Webbrowser auf dem Localhost. In Rosa auf grauem Hintergrund hervorgehoben sind Werte, die automatisch erkannt wurden. Quelle: Eigene Darstellung.	48
5.3	NLP Pipeline wie diese im Data Processing in Abbildung 5.1 vorkommen kann. Quelle: Eigene Darstellung.	49
5.4	Wordcloud, die die häufigsten Begriffe im Datensatz in Form eines Transformators darstellt. Stoppwörter wie the, is, and und weitere wurden entfernt. Quelle: Eigene Darstellung.	50
5.5	Auf der linken Seite in Gelb dargestellt sind oben die 30 häufigsten Token im Datensatz ohne das vorherige Entfernen der Stoppwörter und rechts oben nachdem die Stoppwörter entfernt wurden. Jeweils darunter ist das Auftreten der 20 häufigsten Token im Datensatz dargestellt. Quelle: Eigene Darstellung.	51
5.6	Korrelationsmatrix, die die Ähnlichkeiten der 25 Beispiele visualisiert. Der Index der Dokumente hat die Basis Null; das bedeutet, Dokument <i>KUNDE 00</i> liegt auf Eintrag Nummer 0. In der Diagonale zu sehen ist die maximale Ähnlichkeit von Eins, die jeweils zwischen dem Dokument und sich selbst auftritt. Quelle: Eigene Darstellung.	52
5.7	Korrelationsmatrix, die die Ähnlichkeiten zwischen Sätzen und Satzteilen visualisiert. Die Darstellung wurde unter Verwendung eines <i>Universal-Sentence-Encoder-LARGE</i> -Modells erstellt. Quelle: Eigene Darstellung.	53
5.8	Das Finden und Zählen eines Wortes in Text mittels RegEx in Python. Quelle: Eigene Darstellung.	56
5.9	Beispiel für eine Regel zum finden von Währungsmengen mit <i>Ruta</i> . Gefunden werden würde zum Beispiel <i>31,50 €</i> . Dabei ist das Leerzeichen zwischen Zahl und Einheit optional. Der Dezimalpunkt kann Komma oder Punkt sein. Quelle: Kluegl et al. (2016), S. 32. (modifiziert)	57
5.10	Beispiel für eine Regel zum finden von Währungsmengen mit <i>spaCy</i> . Quelle: Eigene Darstellung	57
5.11	Textpassage mit Informationen über Trafos, eingebettet in eine aus einer PDF-Datei erstellten HTML-Datei. Die Informationen werden mit der Python-Bibliothek <i>Beautiful Soup 4</i> aus dem HTML ausgelesen und in Python verarbeitet. Quelle: Eigene Darstellung.	59
5.12	Darstellung des verwendeten Ablaufs. Das PDF-Dokument wird in eine HTML-Datei umgewandelt aus der der Text extrahiert und indexiert wird. Über eine NLP-Pipeline werden die Informationen extrahiert und in der HTML-Datei mit CSS hervorgehoben. Quelle: Eigene Darstellung.	59
5.13	Regel zum Finden von Normen. Das Keyword <i>IN</i> signalisiert das Folgen einer Liste, die Kandidaten bereitstellt. Über den Befehl <i>add</i> wird zum <i>Matcher</i> die Regel hinzugefügt. Quelle: Eigene Darstellung	60
5.14	Regel zum Finden von Kenngrößen und Geldbeträgen. In <i>pattern_2</i> wird dem <i>Matcher</i> ein Ergebnis der NER als Suchbegriff übergeben. Quelle: Eigene Darstellung	60
5.15	Dependency Tree einer Teilmenge des Textes. Quelle: Eigene Darstellung mit <i>DisplaCy</i>	61
5.16	Regel, die Zeitwörter anhand des POS-Tags hervorhebt. Quelle: Eigene Darstellung	61

5.17	Regel, die Mail-Adressen mittels in spaCy eingebauter Funktion findet. Mit <i>pattern_2</i> wird mittels RegEx das Muster für ein Datum definiert, auch hier wäre eine Suche nach der Entität <i>DATE</i> zum selben Ergebnis gekommen. Quelle: Eigene Darstellung	62
5.18	Textpassage mit hervorgehobenen Informationen über Trafos. Die Hervorhebungen wurden mittels CSS erstellt. Quelle: Eigene Darstellung.	62
5.19	Beispiel zu ClausIE in dem der erste Satz aus dem Beispieltext des vorherigen Kapitels analysiert wird. Das Framework erkennt dabei die Beziehungen der Token untereinander und die darauf basierenden Aussagen. Quelle: Eigene Darstellung	63
5.20	NLP Pipeline Schema mit OIE als Preprocessing um weiteren Anwendungen als zusätzliche Daten-Grundlage zu dienen. Quelle: Eigene Darstellung.	64
5.21	Visualisierung der zwölf Attention-Heads des BERT-Transformer-Modells. Jede Farbe stellt einen Head dar; die Intensität dieser ist Indikator für den Wert der Attention. In diesem Beispiel sind die Werte für das Wort <i>campus</i> gezeigt. Quelle: Vig (2019) (modifiziert)	65
5.22	Darstellung der Skalierung der Lernrate eines BERT-Modells wenn man die Anzahl der TPUs erhöht. Eine Skalierung der Rechenleistung um das 64-fache ergibt eine um das 49-fache erhöhte Lernrate. Quelle: You et al. (2019), S. 8.	66
5.23	Schematische Darstellung eines Transformer-Modells. Quelle: Eigene Darstellung.	67
5.24	Beispiel für das Zuweisen eines Satzes vor dem Positional Encoding (pe) am Decoder zu Token. Quelle: Eigene Darstellung.	68
5.25	Schematische Darstellung der Pipeline-Komponente, die im Beispiel beschrieben wird. Der verwendete Transformer wird zum jetzigen Zeitpunkt von Google für Tensorflow in der Tensor2Tensor Library als FOSS bereitgestellt. Quelle: Eigene Darstellung	68
5.26	Nähe der umgebenden Token zu Token 10. Quelle: Eigene Darstellung	69
5.27	Plot der Werte der trigonometrischen Funktionen, die Werte-Ketten zu 128 Einträgen zeigt. Für ein reales Modell wäre ein realistischerer Wert 512 Einträge. Die Batchgröße wurde mit 15 gewählt was auch wenig ist, die kleineren Zahlen sollen der besseren Erkennbarkeit im Plot dienen. Quelle: Eigene Darstellung	69
5.28	Darstellung eines Trainingstupel. Die Token stehen für SOS (Start of Sentence), EOS (End of Sentence), SEP (Seperator) und MASK für den maskierten Token. Die Bezeichnungen sind frei gewählt und variieren von Modell zu Modell. In diesem Beispiel hat der Transformer das Wort <i>Campus02</i> zurückzugeben. Quelle: Eigene Darstellung.	70
5.29	Aufbau der FFNs, die im Transformer verwendet werden. Für die beiden Dense Layer wurde Keras verwendet. Diese Vorgehensweise wurde auch in der Tensorflow-Dokumentation verwendet und unter anderem deshalb angewendet, weil Keras ab Tensorflow 2.0 die offiziell in Tensorflow implementierte High-Level-API darstellt. Quelle: Eigene Darstellung.	71
5.30	Im linken Plot ist der Verlauf der Accuracy über die 20 Trainings-Epochen hinweg abzulesen. Im rechten Plot ist der Verlauf des Loss abzulesen. Die in 20 Epochen erreichte Accuracy betrug 34,4 %. Quelle: Eigene Darstellung	72
5.31	Visualisierung eines Attention Layers über alle 8 Heads für die erhaltene Übersetzung. Die Übersetzung ist nicht komplett richtig jedoch ist zu erkennen, dass semantische Bedeutung erlernt wurde. Quelle: Eigene Darstellung	72
5.32	Links BERT das die Transformer Encoder Blöcke bidirektional in beide Richtungen verwendet und Rechts ELMo das LSTMs verwendet. Quelle: Eigene Darstellung.	74
5.33	Auszug aus dem MS MARCO Leaderboard für Passage Retrieval. Das Modell BERT ist auch in diesem öfters anzutreffen. [Stand: 02.09.2019] Quelle: Microsoft (2019).	75

5.34	Schematische Darstellung einer Pipeline wie MRC in das System integriert sein könnte. In einem solchen System wären beliebige vom Anwender selbst erstellte natürlichsprachlich gestellte Fragen (Querys) möglich. Die Information soll dem Benutzer in einer Web-App ausgegeben werden. Quelle: Eigene Darstellung.	75
5.35	Beispielhafter Auszug aus dem SQUAD 2.0 Datensatz. Zu sehen ist ein Paragraf mit Informationen und zwei Fragestellungen zu diesem. Im ersten Beispiel ist die Antwort im Paragrafen zu finden und die Position der Antwort hinterlegt. Für das zweite Beispiel enthält der Absatz keine Antwort. Die wesentlichste Neuerung von SQUAD 1.1 auf SQUAD 2.0 war, dass es nicht mehr für jede Frage eine Antwort gab und somit <No Answer> zu einer möglichen Antwort wurde. Quelle: Rajpurkar; Jia; Liang (2018) (modifiziert).	76
5.36	Beispiel für die Einbindung des Universal Sentence Encoders. Dieser kann vom Tensorflow Hub als Modul geladen und verwendet werden. Auf der Webseite wird die Performance am SQUAD Dataset mit dev train Precision 53.2 43.3 angegeben. Quelle: Eigene Darstellung.	77
5.37	Beispiel für ein Ergebnis einer Machine Reading Comprehension auf dem SQUAD Dataset. Das System wurde so genutzt, dass sowohl die fünf wahrscheinlichsten Paragrafen, die die Antwort enthalten, als auch die Antwort selbst ausgegeben werden. Quelle: Eigene Darstellung.	78
5.38	Die auf der Webseite https://gluon-nlp.mxnet.io/ empfohlenen Parameter zum Finetuning von BERT auf dem SQUAD 2.0 Datensatz. Batch Size und Accumulate wurde angepasst, um auf der GTX 1070 mit dem Speicher auszureichen. Quelle: Eigene Darstellung.	78
5.39	Mit der Python-Bibliothek PDFMiner erstellter Positions-Index über den kompletten Text einer PDF Datei. Hierbei wurde ein Ausschnitt gewählt, der einem aus fünf Zeichen bestehenden Wort zugeordnet ist. Quelle: Eigene Darstellung.	80
5.40	Als zusätzliche Ausgabe zur Beispielvisualisierung wird eine Visualisierung der einzelnen Worttoken erzeugt. In dieser sind die POS-Tags als Rahmenfarbe und erkannte Entitäten direkt am Token mit schwarzem Hintergrund hervorgehoben. Die Idee hinter dieser Visualisierung ist es beim Erstellen von Regeln, während der Entwicklung, zu unterstützen. Quelle: Eigene Darstellung.	80
5.41	Prototyp der Oberfläche, der als Teil des Projekts erstellt wurde und die PDF über PDF.js direkt in der Web-App rendert und Annotationen ermöglicht. In der Seitenleiste können gefundene Annotation sowie selbst erstellte Annotationen dargestellt und ausgewählt werden. Außerdem kann über diese zur entsprechenden Stelle im PDF gesprungen werden. Quelle: Eigene Darstellung.	81
6.1	Qualitative Darstellung eines möglichen Verlaufs des Automatisierungsgrades, der durch den Einsatz eines auf maschinellem Lernen basierenden Werkzeuges zu erwarten ist. Quelle: Eigene Darstellung.	83
7.1	Lebenszyklus eines auf maschinellem Lernen basierenden Softwareprojektes. Quelle: Eigene Darstellung.	85

TABELLENVERZEICHNIS

3.1 Konfusionsmatrix über Label der Daten und der tatsächlichen Beobachtung. Quelle: Eigene Darstellung.	16
3.2 Darstellung einer term-document matrix. In den Spalten stehen Titel von Werken Shakespears und in den Zeilen die Anzahl der Vorkommen des entsprechenden Wortes. Quelle: Jurafsky; Martin (2018), S. 108. (modifiziert)	38
3.3 Tabelle der als tf-idf Maß bewerteten term document matrix. Quelle: Jurafsky; Martin (2018), S. 115. (modifiziert)	41
4.1 Beispiele für Teildisziplinen des NLP, für die es zum jetzigen Zeitpunkt Evaluierungen gibt. Insgesamt sind es mehrere Hundert Teildisziplinen, die zu Teilen ineinander greifen. Quelle: Eigene Darstellung.	45
5.1 Überblick über die im Datensatz enthaltenen Spezifikationen. Es sind der Index, gefolgt von der Art der Maschine und danach die Leistung in MVA für Volltrafo (TRA), Spartransformator (AUT) und Maschinentransformator (GSU) sowie in MVA für Drossel (SHR) und statischen Blindleistungskompensator (SVC) angegeben.	54
5.2 Gegenüberstellung der Vor- und Nachteile von regelbasierten IE-Systemen und auf maschinellem Lernen basierenden Systemen. Quelle: Chiticariu; Y. Li; Reiss (2013), S. 829. (modifiziert).	55
5.3 Mögliche Tokeneigenschaften, die in spaCy v2.1 im Rule-Based-Matcher verwendet werden können. Quelle: Honnibal; Montani (2019) (modifiziert).	58

ABKÜRZUNGSVERZEICHNIS

HMM Hidden-Markov-Modell

MEMM Maximum-Entropy-Markov-Modell

POS Part of Speech

NB Naive-Bayes

LR Logistische Regression

CRF Conditional Random Fields

SGD Stochastic Gradient Descent

IE Information Extraction

IR Information Retrieval

OCR Optical Character Recognition

CLI Command Line Interface

BOW Bag-of-Words

LSI Latent Semantic Indexing

NLP Natural Language Processing

NER Named Entity Recognition

NED Named Entity Disambiguation

ACE Automatic Content Extraction

CoNLL Conference on Natural Language Learning

BioCreAtivE Critical Assessment of Information Extraction Systems in Biology

NN Neuronale-Netze

TanH Tangens Hyperbolicus

ReLU Rectified Linear Unit

FFN Feed-Forward-Netzwerk

MSE Mean Squared Error

RNN Recurrent Neural Networks

CNN Convolutional Neural Networks

LSTM Long Short Term Memory

QA Question Answering

SOTA State of the Art

SQuAD 2.0 Stanford Question Answering Dataset 2.0

BERT Bidirectional Encoder Representations from Transformers

XLNet Generalized-Autoregressive-Pretraining-for-Language-Understanding

RegEx Regular expression

GSU Generator Step-Up Transformer

UIMA Unstructured Information Management Architecture

ML Machine Learning

CSS Cascading Style Sheets

ORE Open Relation Extraction

OIE Open Information Extraction

AH Attention Head

GPU Graphics Processing Unit

TPU Tensor Processing Unit

FOSS Free Open Source Software

NLU Natural Language Understanding

GPT Generative Pre-training for Transformers

MRC Machine Reading Comprehension

cdQA Closed Domain Question Answering