Dirk Meyer
Master Informatik
Green IT und Hochleistungsrechnen
Sommersemester 2022

Masterarbeit

Situationsabhängige Bekleidungsmodellierung mit Hilfe von Machine Learning für die Erstellung von Avataren

Dirk Meyer

Abgabedatum: 15.04.2022

Informationen zur Verfügbarkeit des Programmcodes aus dieser Arbeit

Die im Rahmen dieser Arbeit entwickelten Programme und die Wissensdatenbank können über https://github.com/texttechnologylab/FashionNet eingesehen werden.

Zusammenfassung

Bei der Bekleidungsmodellierung geht es um den Entwurf von Bekleidung von Personen, die beispielsweise in Szenen dargestellt werden können. Dabei stützt sich der Entwurf auf Informationen aus einer Datengrundlage. Die Darstellung von Szenen, in denen Personen dargestellt werden, stellt sich grundsätzlich als Zusammenspiel komplexer Teilaspekte dar. Dabei wird die Nachvollziehbarkeit einer modellierten Szene oder modellierter Avatare im Auge des Betrachters ganz wesentlich durch den Faktor passend gewählter Kleidung bestimmt.

In dieser Arbeit werden Ansätze und Verfahren vorgestellt, die zur Bekleidungsmodellierung auf Grundlage von Textdokumenten basieren. Dafür werden Möglichkeiten erörtert, die es erlauben Informationen aus Texten zu extrahieren und für die Modellierung einzusetzen.

Zur Bearbeitung der Aufgabenstellung wird zunächst ein aus dem *Machine Learning* bekanntes kontextuelles Modell hinsichtlich einer Mehrklassen-Klassifizierung trainiert und angewendet. Daraufhin wird die Erstellung einer eigenen Wissensressource, die sich auf textlicher Ebene mit dem Thema der Bekleidung auseinandersetzt, aufgebaut und mit zahlreichen Informationen aus bereits bestehenden Ressourcen popularisiert. Die neue Ressource wird in Form einer Graphdatenbank entworfen. Dabei werden Relationen zwischen den einzelnen Elementen mithilfe von statischen Modellen sowie einem kontextuellen Modell, dem BERT-Modell, erstellt. Schließlich wird auf Grundlage der entwickelten Graphdatenbank ein in der Programmiersprache *Python* entwickeltes Programm vorgestellt, dass Eingabetexte unter Hinzunahme der Informationen und Relationen innerhalb der Graphdatenbank verarbeitet und Kleidungsstücke detektiert.

Nach der theoretischen Aufarbeitung der entwickelten Ansätze werden die daraus resultierenden Ergebnisse diskutiert und bestehende Problematiken bei der Bearbeitung der Aufgabenstellung angesprochen. Abschließend wird die Arbeit zusammengefasst und Anregungen für die weitere Bearbeitung dieser Thematik vorgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

NLP Natural Language Processing

POS Part-of-Speech

NER Named Entity Recognition

KI Künstliche Intelligenz

KNN Künstliche neuronale Netze

CNN Convolutional Neural Network

FNN Feed-Forward Neural Network

RNN Recurrent Neural Network

BERT Bidirectional Encoder Representations from Transformers

LSTM Long Short-Term Memory

MLM Masked Language Modeling

NSP Next Sentence Prediction

MNLI Multi-Genere Natural Language Inference

SQuAD Stanford Question Answering Dataset

TF Term Frequency

IDF Inverse Document Frequency

RDBMS Relational Database Management Systems

1. Einleitung

In diesem Kapitel wird eine Einführung in die Thematik anhand der Aufgabenstellung dieser Arbeit gegeben. Daraufhin werden Motive vorgestellt, die der Bearbeitung der Aufgabenstellung zugrunde liegen.

1.1. Aufgabenstellung

Das Ziel dieser Arbeit ist es, eine Ressource zu schaffen, um aus einem gegebenen Text die darin enthaltenen oder beschriebenen Kleidungsstücke zu extrahieren und somit die Möglichkeit zu eröffnen, eine beschriebene Person passend zu kleiden. Daraus ergibt sich, dass zunächst geprüft werden muss, welche Verfahren und Möglichkeiten bereits bestehen, um Kleidung in Texten zu identifizieren und welche Faktoren für die Akzeptanz einer Bekleidungsmodellierung relevant sind. Dies schließt die Recherche nach sinnvoll einsetzbaren Ressourcen ein, die sich in diesem Fall vornehmlich auf Datenbanken mit Texten beschränken ein. Datensätze, die neben reinen Texten möglichst qualitativ und quantitativ hochwertige Zusatzinformationen beinhalten, stellen dabei das Optimum für passende Datenressourcen dar.

Anschließend soll ein eigenes Verfahren entwickelt werden, dass die Entwicklung einer geeigneten und grundlegenden Datenstruktur beinhaltet. Der selbstständig entwickelte Ansatz soll daraufhin anhand eines geeigneten Datensatzes getestet und evaluiert werden und soll sich auf bereits etablierte Ansätze stützen und diese sinnvoll integrieren. Neben der reinen Evaluation soll der Ansatz auch angewendet und die Vorgehensweise dabei beschrieben und erläutert werden.

Für die Entwicklung des eigenen Ansatzes zur Bearbeitung der Aufgabenstellung sollen möglichst aktuelle Verfahren und Modelle vorrangig aus dem *Machine Learning* einbezogen werden und diese in die eigenen Überlegungen und praktischen Realisierung integriert werden.

1.2. Motivation

Die automatische Generierung von Szenen, in denen Personen auftreten, erfordert die Erfüllung zahlreicher Faktoren, damit die generierte Szene vom späteren Betrachter als stimmig und nachvollziehbar bewertet wird.

Ein zentraler Faktor stellt die Bekleidung der repräsentierten Personen dar. In einem Film wirken die Schauspieler erst dann realistisch, wenn sie geschminkt, frisiert und vor allen Dingen kostümiert sind. Die Bekleidung einer Person kann darüber hinaus weitere Anhaltspunkte beispielsweise über den sozialen Status oder kulturelle Einflüsse geben.

Es stellt sich nun die Frage, wie aus einem Text, in dem die Bekleidung einer Person beschrieben wird, die einzelnen Kleidungsobjekte extrahiert werden können, um diese anschließend für die Erstellung einer Szene präsentieren zu können. Viel mehr noch ist die Fragestellung, welche Faktoren und Rahmenbedingungen dafür entscheidend sind.

Für die Bearbeitung dieser Fragestellung wird schnell ersichtlich, dass es zahlreiche offensichtliche Faktoren wie das Alter, das Geschlecht oder die Größe einer Person gibt, die die Bekleidung grundlegend bestimmen, doch schon immer beschreibt der Begriff der Mode die Vielseitigkeit und damit einhergehend die Anwesenheit zahlreicher weiterer Faktoren.

In dieser Arbeit wird ausgehend von verfügbaren Datensätzen eine Datenbank aufgebaut, die es später ermöglichen soll, Kleidung auf Textebene zu detektieren unter Berücksichtigung verschiedener Faktoren.

Beim Aufbau der Datenbank werden mehrere und im Kern unterschiedliche Daten und Relationen zueinander mit einbezogen, sodass auch mehrere Faktoren abgebildet werden können. Bei den einzelnen Erhebungen werden zum einen bereits vorliegende Daten und zum anderen speziell für die Aufgabenstellung berechnete Daten, die mithilfe von Modellen und Algorithmen aus dem *Machine Learning* berechnet wurden, einbezogen. Die daraus entstehenden Menge an Daten kann anschließend genutzt werden, um passende Ergebnisse zu ermitteln.

2. Stand der Wissenschaft und Technik

In diesem Kapitel werden einige für diese Arbeit grundlegende Konzepte und Begriffe erklärt und vorgestellt. Dabei werden nur die Themen aufgeführt, die eine Relevanz für diese Arbeit darstellen und deren Verständnis wichtig sind.

Zunächst wird in den Unterkapiteln *Natural Language Processing* (Abschnitt 2.1), *Künstliche Intelligenz* (Abschnitt 2.2) und *Machine Learning* (Abschnitt 2.3) eine Einordnung dieser Arbeit in die betreffenden Forschungsdisziplinen gegeben.

Eine grundlegende Rolle für die darauffolgenden Themen nehmen die Unterkapitel *Neuronale Netze* (Abschnitt 2.4) und *Transformer* (Abschnitt 2.5) ein. Sie werden daher darauffolgend entsprechend vorgestellt.

Für diese Arbeit stellt das Unterkapitel BERT einen zentralen Bestandteil dar. Sodass in Abschnitt 2.6 entsprechend detaillierte Erklärungen hierzu folgen. Ebenfalls eine wichtige Rolle spielen Modelle die zur Erstellung statischer Wort-Repräsentationen genutzt werden können. In Abschnitt 2.7 werden die für diese Arbeit relevanten Modelle vorgestellt und erklärt.

Zum Abschluss dieses Kapitels werden weitere Ressourcen beschrieben, die in dieser Arbeit genutzt wurden, thematisch jedoch außerhalb der bereits angesprochenen Unterkapitel liegen (Abschnitt 2.8).

2.1. Natural Language Processing

Die Verarbeitung natürlicher Sprache wird in der Wissenschaft als *Natural Language Processing* bezeichnet, kurz *NLP*. Die in der *NLP* beinhalteten Aufgabenstellungen sollen dazu dienen, Computersysteme zu befähigen, menschliche Sprache zu verstehen und somit auch Interpretationen dieser möglich zu machen (Hirschberg und Manning (2015, S.261)).

Da die natürliche Sprache extrem komplex und häufig mehrdeutig (Jurafsky und Martin (2008, S.4), Jusoh (2018)) ist, stellen sich verschiedenste Problematiken für die automatisierte Sprachverarbeitung in den Weg. Zusätzlich verfügen Computersysteme nicht grundlegend über gesammelte Erfahrungen und Vorinformationen zum Verständnis von Sprachen, sodass eine kollaborative Zusammenarbeit zwischen einzelnen Fachdisziplinen benötigt wird (Khurana et al. (2017, S.3)), um diese Lücke schließen zu können.

Während die Verarbeitung natürlicher Sprache bereits seit vielen Jahren als wissenschaftliche Disziplin besteht (Khurana et al. (2017, S.7), Jurafsky und Martin (2008, S.9)) und verschiedene Wissenschaftler an einzelne Aufgabenstellungen arbeiten oder bestehende An-

sätze versuchen zu verbessern, haben sich die Voraussetzungen dafür in den letzten Jahren deutlich verbessert (Hirschberg und Manning (2015, S.261)). Die immer weiter steigende Verfügbarkeit großer Datenmengen über das Internet (Jurafsky und Martin (2008, S.8)), eine weiter steigende Rechenleistung sowie die Verbesserung bestehender Algorithmen und Ansätze tragen dazu bei, dass diese Disziplin immer stärker in den Fokus rückt. Dies gilt für die Wissenschaft genauso wie für Wirtschaftsunternehmen, die sich durch die Anwendung der entwickelten Verfahren Vorteile versprechen (Dale (2019, S.2-3), Prasad und Korrapati (2017)).

Während die anfänglichen Bemühungen in der *NLP* noch hauptsächlich statistische Verfahren nutzten, werden heute zumeist Methoden aus dem *Machine Learning* (Abschnitt 2.3) und *Deep Learning*[1] verwendet. Grundlage für diese Methoden stellen zumeist *neuronale Netze* (Abschnitt 2.4) dar (Xu und Sun (2017, S.24)). Ein wesentlicher Grund für diese Veränderung stellt der immense Aufwand dar, der für die Erstellung von Annotationen und Anreicherung der jeweiligen Texte mit spezifischen Daten erfordert (Xu und Sun (2017, S.24), Chai und Li (2019)).

Für die Verarbeitung von natürlicher Sprache wurden verschiedene grundlegende Verfahren entwickelt, die dabei helfen, die Daten für weiterführende Anwendungen vorzubereiten. Besonders zu erwähnen sind die Tokenisierung (*tokenization*), Lemmatisierung (*lemmatization*), *Part-of-Speech*-Tagging (*POS-Tagging*) und *Named-Entity-Recognition*, die neben weiteren Verfahren von zentraler Bedeutung sind (Manning et al. (2014, S.57)).

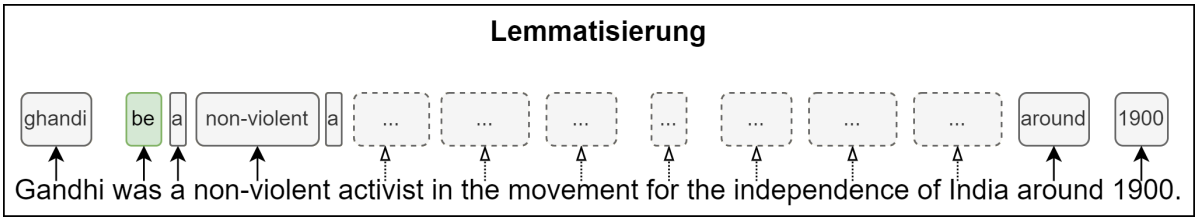

Abbildung 2.1.: Prinzip der Lemmatisierung

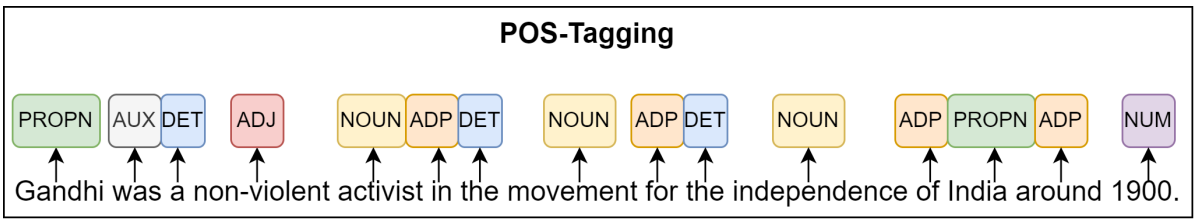

Abbildung 2.2.: Prinzip des *POS-Taggings* in Anlehnung an Hirschberg und Manning (2015, S.262)

[1] *Deep Learning* stellt einen Teilbereich des *Machine Learnings* dar und nutzt große Datenmengen sowie *Neuronale Netze*, um Entscheidungen oder Prognosen generieren zu können (Kelleher (2019, S.6-10)).

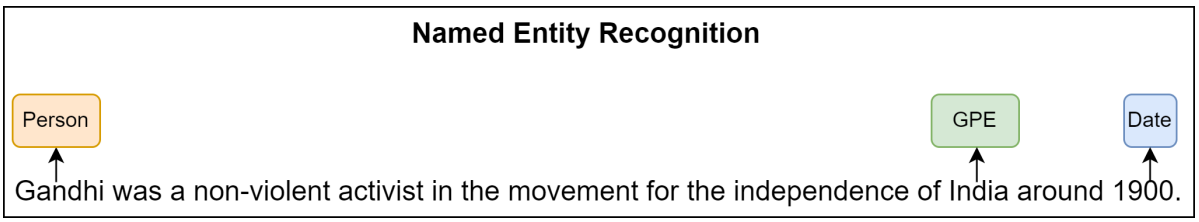

Abbildung 2.3.: Prinzip der *Named Entity Recognition* in Anlehnung an Hirschberg und Manning (2015, S.262)

Bei der Tokenisierung wird ein Eingabe-Satz in einzelne Token zerlegt, dieses Verfahren stellt zumeist den initialen Schritt für die Vorverarbeitung dar. Die Lemmatisierung hingegen generiert die Basisform (Lemma) eines gegebenen Tokens (Abb. 2.1). Beim *POS-Tagging* wird für jedes Token eine Zuordnung zur jeweiligen Wortart vorgenommen (Abb. 2.2). Schließlich können mittels *NER*-Verfahren einzelne Token zu Entitäten wie Organisationen, Personen oder Orten getroffen werden (Abb. 2.3).

2.2. Künstliche Intelligenz

Nach Bhattacharyya et al. (2020) ist der Begriff der künstlichen Intelligenz (*Artificial Intelligence*) der Oberbegriff für die Simulation intelligenten Verhaltens und umfasst beispielsweise auch das maschinelle Lernen. Zumeist wird unter der künstlichen Intelligenz der Versuch verstanden, die Entscheidungsstrukturen des Menschen nachzubilden und somit Computersysteme zu befähigen, eigenständig Probleme zu lösen.

Eine genaue Definition für die künstliche Intelligenz (*KI*) zu finden ist schwierig, da es sich um kein geschlossenes Forschungsgebiet handelt. In zahlreichen Disziplinen wie der Psychologie (Crowder und Friess (2012), D'Alfonso (2020)), Neurologie (Ienca und Ignatiadis (2020)) oder Medizin (Krittanawong et al. (2017), Ahmed et al. (2020)) profitiert die Forschung von der Unterstützung von KI-Systemen und umgekehrt. Beispielsweise spielen neuronale Netze im Bereich der künstlichen Intelligenz eine tragende Rolle und begründen ihren Ursprung aus der Neurophysiologie (Traeger et al. (2003)).

So vielseitig der Austausch mit anderen Forschungsdisziplinen ist, so divers sind auch die Anwendungsmöglichkeit von *KI*-Systemen. Von wissensbasierten Systemen (Futia und Vetrò (2020)) über Anwendungen in der *Robotik* (Roy et al. (2020)) oder im Zusammenhang mit autonomen Fahrsystemen (Ma et al. (2020)) können durch den Einsatz von *KI*-Systeme wichtige Fortschritte in einzelnen Forschungsgebieten erzielt werden.

2.3. Machine Learning

Der Begriff der *künstlichen Intelligenz* ist ein Oberbegriff für alle Verfahren und Ansätze, um einem Computersystem menschliches und damit komplexes Denken beizubringen. Dabei geht es nicht länger darum nach vordefinierten Regeln und vorprogrammierten Funktionen Aufgaben abzuarbeiten, stattdessen soll ein Computersystem aus Erfahrungen lernen und die gesammelten Ergebnisse auf neue Situationen anwenden können (Bhattacharyya et al. (2020)).

Ein Teilgebiet der *KI* stellt das maschinelle Lernen (*Machine Learning*) dar. *Machine Learning* beinhaltet den Ansatz aus einer möglichst großen Datenmenge Muster zu entdecken und daraus wiederum Regeln abzuleiten (Bhattacharyya et al. (2020)). Der Prozess, der für *Machine-Learning*-getriebene Computer-Systeme verwendet wird, kann als automatische Optimierung durch Erfahrungen und später erfolgende Implementierung als Lernprozess verstanden werden (Ayodele (2010)).

Das System arbeitet somit nicht exakt, schafft es aber je nach Forschungsstand und Anwendungsgebiet gute bis sehr gute Annäherungen zu erzielen. Eine Besonderheit hierbei ist, dass einmal trainierte Modelle in der Lage sind weiter zu lernen und somit immer besser abgestimmte Regeln zu definieren und die daraus resultierenden Ergebnisse zu verbessern. Einmal trainierte Modelle können dann durch eine weitere gezielte Lernphase auf ähnliche Fragestellungen aus der gleichen Domäne angewendet werden (Alpaydin (2021, S.27-34)).

Eine Unterscheidung gibt es bei den Lernmethoden, die verwendet werden können. Beim überwachten Lernen (*supervised learning*) wird manuelle menschliche Vorarbeit benötigt, die dem System mitteilt, welcher Sachverhalt gelernt werden soll. Daten müssen entsprechend mit zusätzlichen Informationen (*Annotationen*) angereichert werden, sodass das Computersystem Zuordnungen zwischen Eingabe und Ausgabe verstehen und später auf neue Daten anwenden kann. Bei der anderen Methode handelt es sich um unüberwachtes Lernen (*unsupervised learning*). Hierbei wird einem Computersystem die Aufgabe gestellt, unstrukturierte Daten zu interpretieren und Merkmale zu finden. Die Ergebnisse können anschließend von Menschen überprüft und interpretiert werden (Alpaydin (2021, S.142-151)).

2.4. Neuronale Netze

Ein wichtiges Konzept des *Deep Learnings* stellen neuronale Netze (*neural networks*) dar. Mithilfe von neuronalen Netzen ist es möglich, große Mengen an unstrukturierten Daten auszuwerten und Muster zu entdecken. Auch wenn es einige verschiedene Varianten von künstlichen neuronalen Netzen (KNN) gibt, teilen sich diese die gleichen Grundbestandteile. Sie bestehen aus den einzelnen Neuronen, deren Verbindungen und Lernregeln innerhalb des Netzes selbst (Kelleher (2019, S.65-68)).

Für die Verarbeitung von eingehenden Informationen innerhalb eines Neurons wird ein zwei-

stufiger Prozess durchlaufen. Ziel der Verarbeitung ist es, aus den Eingabedaten eine entsprechend Ausgabe abzubilden. Im ersten Schritt wird die gewichtete Summe über alle Eingabedaten berechnet. Anschließend wird durch eine zweite Funktion das errechnete Ergebnis auf eine Ausgabe abgebildet. Dabei kann es sich bei der zweiten Funktion um einfache Entscheidungsregeln oder um hochkomplexe Modelle handeln. Da der Ausgabewert eines Neurons häufig als Aktivierungswert bezeichnet wird, wird die zweite Funktion in der Regel als Aktivierungsfunktion bezeichnet (Kelleher (2019, S.73-76)).

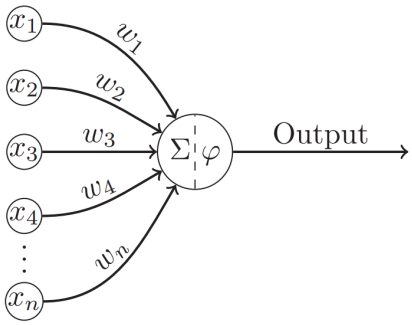

Abbildung 2.4.: Struktur der Informationsverarbeitung durch ein künstliches Neuron (Kelleher (2019, S.72))

In Abb. 2.4 ist die grundlegende Struktur der Informationsverarbeitung eines Neurons dargestellt. Dabei steht $\sum$ für die Berechnung der gewichteten Summe über die Eingangsdaten $[x_1, ..., x_n]$ mit den Gewichten $[w_1, ..., w_n]$ und ϕ für die Aktivierungsfunktion. Mathematisch lässt sich dies durch die Gleichung (2.1) formulieren. Mit der Anwendung der Aktivierungsfunktion folgt dann daraus die Gleichung (2.2), wobei die Aktivierungsfunktion je nach Aufgabenstellung unterschiedlich ausfallen kann (Kelleher (2019, S.75-76)).

$$z = \sum_{i=1}^{n} x_i \cdot w_i \tag{2.1}$$

$$Output = Aktivierungsfunktion(\sum_{i=1}^{n} x_i \cdot w_i) \tag{2.2}$$

Der Aufbau eines neuronalen Netzes besteht aus insgesamt drei Ebenen-Typen und wird in Abb. 2.5 bildlich dargestellt. Dabei bestehen die einzelnen Ebenen wiederum aus einer Menge der bereits erwähnten Neuronen. Einer Eingabe-Ebene (*Input-Layer*), die Daten entgegennimmt. Einer oder auch mehreren Zwischenebenen (*Hidden-Layer*), in denen Berechnungen durchgeführt werden. Die Anzahl an Zwischenebenen hängt von der Komplexität der vorliegenden Aufgabenstellung ab. Und einer Ausgabe-Ebene (*Output-Layer*), welche in der Lage ist sämtliche Ergebnisrepräsentationen abzubilden und diese gegebenenfalls weiterzuleiten (O'Shea und Nash (2015, vgl.)).

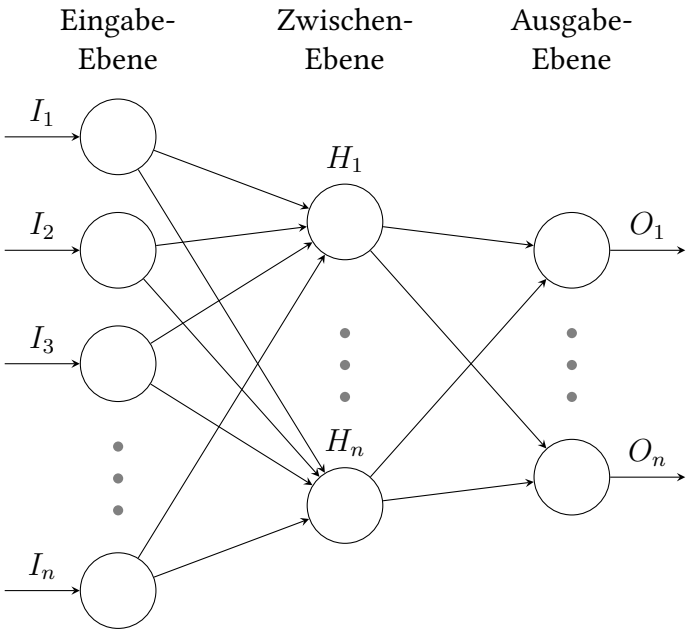

Abbildung 2.5.: Aufbau der Ebenen von neuronalen Netzen in Anlehnung an O'Shea und Nash (2015, S.2)

Alle Ebenen bestehen aus Neuronen, die sich untereinander beeinflussen. Je näher sich zwei Neuronen sind, desto stärker fällt diese Beeinflussung aus. Die konzeptionelle Umsetzung für die Beeinflussung wird in neuronalen Netzen mithilfe von Verbindungsgewichten realisiert. Diese Gewichte liegen jeweils zwischen zwei Neuronen. Die Gewichtsstärke beschreibt hierbei den Einfluss eines Neurons auf ein anderes Neuron. Neuronale Netze müssen trainiert werden. Ziel hierbei ist es, die Gewichte zu optimieren, sodass das Modell in der Lage ist, immer bessere Vorhersagen treffen zu können.

Nach Staudemeyer und Morris (2019) sind die grundlegendste Art von neuronalen Netzen sogenannte *feed-forward neuronal networks* (*FNN*). Sie sind jedoch auf eine statische Abbildung von Eingabe zu Ausgabe limitiert und können keine dynamischen Klassifizierungen durchführen. Um Informationen von vorherigen Eingaben mit in die Berechnungen einfließen lassen zu können, wurden *recurrent neural networks* entwickelt, die in Abschnitt 2.4.1 beschrieben werden.

2.4.1. Rekurrente neuronale Netze

Im Folgenden werden rekurrente neuronale Netze nach Schmidt (2019) kurz skizziert. Neuronale Netze sind in vielen verschiedenen Anwendungsgebieten zum Einsatz gekommen und sind dementsprechend fortlaufend weiterentwickelt worden. Eine dieser Varianten stellen *rekurrente* neuronale Netze (*recurrent neural networks*, kurz *RNN*) dar.

Die Unterscheidung zu den klassischen *FNN* liegt in der Informationsübertragung zwischen den einzelnen Schichten. Während bei *FNN* die Informationen von Schicht zu Schicht fortlau-

fend weitergegeben werden, können diese bei rekurrenten Netzen zyklisch bzw. rückwärtsgewandt ausgetauscht werden. Dadurch wird es ermöglicht, nicht nur die aktuellen Informationen mit einzubeziehen, sondern auch Informationen, die von vorherigen Eingabe-Daten stammen. In Abb. 2.6 ist der Unterschied zwischen beiden Arten von neuronalen Netzen bildlich nach Schmidt (2019, S.2) dargestellt.

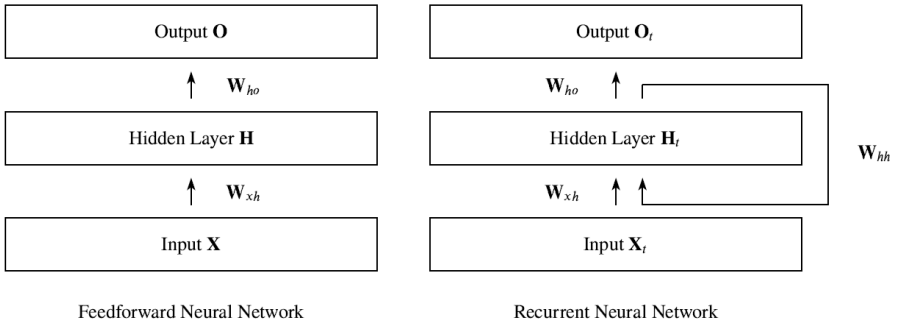

Abbildung 2.6.: Visualisierung der Unterschiede zwischen FNN und RNN (Schmidt (2019, S.2))

Auch wenn die Beachtung vergangener Daten in aktuelle Berechnungen vorteilhaft ist, entsteht dadurch wiederum ein weiteres Problem. Das *RNN* ist teilweise nicht mehr in der Lage, alle Informationen abzuspeichern und verliert darüber hinaus die Zusammenhänge zwischen einzelnen Informationen, was darauf zurückzuführen ist, dass die rückgekoppelten Signale entweder extrem groß (*exploding gradients*) oder verschwindend gering werden und gegen null tendieren (*vanishing gradients*). Dabei führen *exploding gradients* zu sehr stark schwankenden Gewichten, wohingegen *vanishing gradients* den Lernprozess extrem verlangsamen oder gar zum Erliegen bringen (Staudemeyer und Morris (2019, S.17-18)). Eine Lösung für dieses Problem wird in Abschnitt 2.4.1 erläutert.

Long Short-Term Memory

Mit dem Begriff *Long Short-Term Memory* (*LSTM*) werden spezielle Zellen in einem neuronalen Netz beschrieben, die entscheiden können, welche Informationen weiterhin gespeichert bzw. welche vergessen werden sollen (Hochreiter und Schmidhuber (1997)). *LSTMs* werden im Folgenden nach Staudemeyer und Morris (2019) kurz erklärt.

Der Aufbau einer *LSTM* -Zelle besteht aus drei Toren (*Gates*). Einem Eingangstor (*Input Gate*), welches Eingangssignale in eine Zelle aufnimmt und bestimmt in welchem Umfang dies geschieht. Das *Forget Gate* dient der Entscheidung, ob eine Information in einer Zelle beibehalten werden soll oder vergessen werden soll. Das Ausgabetor (*Output Gate*) bestimmt schließlich, welche in der Zelle berechneten und enthaltenen Werte ausgegeben werden sollen. Die eben genannten Bestandteile werden über eine Logik aus Informationsflüssen und Speichervorgängen realisiert, die wiederum auf Matrix- und Vektoroperationen aufbauen.

Auch wenn *LSTMs* das Problem von *RNN* mindern, so bleibt doch das Problem, dass die Wahrscheinlichkeit für die Erhaltung des Kontexts zwischen zwei sehr weit entfernten Wörtern fortlaufend stark abnimmt. Somit können lange Eingabesätze nur unzureichend behandelt werden. Zusätzlich ergibt sich das Problem, dass derartige Modelle nur sehr schwer parallelisierbar sind, da jedes Eingabewort einzeln in die Berechnungen aufgenommen werden muss. Insgesamt ergeben sich drei wesentlich Problematiken. Zum einem wird die Parallelisierbarkeit durch die sequenzielle Berechnung gehemmt. Darüber hinaus gibt es keine explizite Modellierung von Abhängigkeiten über kurze oder lange Distanzen und der Abstand zwischen einzelnen Positionen ist linear.

2.4.2. Convolutional Neural Networks

Eine weitere Variante von neuronalen Netzen stellen *Convolutional Neural Networks* (*CNN*) dar. Diese sollen nach den Arbeiten von O'Shea und Nash (2015) kurz erklärt werden. Da *CNN* jedoch nur eine geringe Rolle im Rahmen dieser Arbeit spielen, werden die Ausführungen sich auf das Wesentliche beschränken.

Als einer der ersten Arbeiten zu *CNN* gilt die Arbeit von LeCun, Bengio et al. (1995) in der bereits klar wird, dass sich *CNN* vorrangig der maschinellen Verarbeitung von Bild- oder Audiodaten widmen. Trotzdem konnten auch in der Spracherkennung bemerkenswerte Ergebnisse erzielt werden.

Der Aufbau eines *CNN* besteht in der Regel aus einem oder mehreren *Convolutional Layer*, dem sich ein *Pooling Layer* anschließt. Dabei ist es möglich, dass sich dieses Element wiederholt. Ab einer bestimmten Anzahl von Wiederholungen ist in diesem Zusammenhang auch die Rede von einem *Deep Convolutional Neural Network*. Der sogenannte *Fully-connected Layer* schließt den Aufbau ab.

Die *Convolutional* Schicht erkennt und extrahiert über eine diskrete Faltung über die Eingabedaten Muster. Dabei wird eine Faltungsmatrix in kleinen Schritten über die Eingabe bewegt. Hierdurch werden benachbarte Neuronen beeinflusst. Um die Ergebnisse auf die wesentlichen Informationen zu reduzieren, werden in der *Pooling*-Schicht bspw. über das *Max-Pooling* nur diejenigen Neuronen-Ergebnisse aussortiert und für die weitere Verarbeitung genutzt, die den höchsten Aktivitätsstatus besitzen.

Auch wenn in diesem Schritt theoretisch Informationen verloren gehen, bleiben die wesentlichen Eigenschaften erhalten. Außerdem überwiegen die daraus entstehenden Eigenschaften, die vor allen Dingen einen geringeren Speicherbedarf und daraus resultierend eine geringere Zeit für die Berechnung bedeuten.

Die abschließende *Fully-Connected*-Schicht wird vorrangig für die Klassifizierung genutzt, dessen Größe sich nach den vom *CNN* zu unterscheidenden Klassen richtet und variiert somit je nach Aufgabenstellung.

In Abb. 2.7 ist eine grob vereinfachte und grundlegende Architektur eines *CNN* bildlich dargestellt.

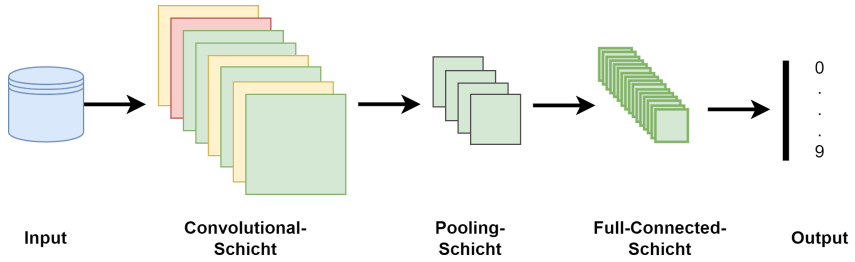

Abbildung 2.7.: Grundlegende Architektur eines *CNN* in Anlehnung an O'Shea und Nash (2015, S.4)

Bei der Verwendung von *CNNs* wird auf überwachtes Lernen gesetzt. Für spezielle Anwendungsgebiete bieten *CNN* durch die Verringerung der zu erlernenden Gewichte große Vorteile gegenüber klassischen *KNN*, da sie weniger Speicherbedarf benötigen und strukturell relativ leicht umsetzbar sind.

2.5. Transformer

Transformer entwickelten sich aus neuronalen Netzen. Auf sie bauen viele Ansätze im Zusammenhang mit *Natural Language Processing* heute auf. In der Arbeit von Vaswani et al. (2017) wird ein *Transformer* vorgestellt, der basierend auf der Architektur von neuronalen Netzen zusätzlich einen sogenannten *Self-Attention* Mechanismus einsetzt, der speziell für das natürliche Sprachverständnis entwickelt wurde.

Die Besonderheit liegt hierbei darin, dass eine Sequenz ähnlich wie bei *sequence-to-sequence*-Modellen, verarbeitet und in eine neue Sequenz transformiert wird, in diesem Fall aber der *Self-Attention*-Mechanismus zusätzlich angewendet wird. Dadurch kann im Gegensatz zu Ansätzen, die auf *RNN* oder *LSTM* aufbauen, die Parallelisierbarkeit deutlich erhöht werden und der gegebenenfalls besonderen Bedeutung einzelner Worte Rechnung getragen werden.

2.5.1. Architektur eines Transformers

Im Folgenden wird die *Transformer*-Architektur nach Vaswani et al. (2017) vorgestellt. Wie die meisten neuronalen Modelle für die Sequenztransduktion wird auch bei einem *Transformer* eine *Encoder-Decoder*-Struktur verwendet.

Hierbei wird als Eingabe eine ganze Sequenz $x = (x_1; ...; x_n)$ simultan entgegen genommen und in eine Sequenz von kontinuierlichen Darstellungen $z = (z_1; ...; z_n)$ übertragen und dem *Decoder* überreicht. Der *Decoder* erzeugt anschließend aus z eine Ausgangsfolge $(y_1, ..., y_n)$. Die Länge der Ausgabe-Sequenz muss hierbei nicht der Länge der Eingabe-Sequenz entsprechen. Dies erklärt sich damit, dass beispielsweise bei Übersetzungsaufgaben der Eingabesatz

der Sprache L_1 nicht die gleiche Länge haben muss wie der Ausgabesatz aus einer Sprache L_2. Denn bei gleichbleibender Semantik eines Satzes können zwei verschiedene Sprachen $L_1 \neq L_2$ unterschiedlich viele Worte benötigen. Außerdem verarbeitet der *Decoder* in jedem Schritt jeweils nur ein y_i autoregressiv und berücksichtigt dabei vorher berechnete Ausgaben y_j. Hierbei muss die Bedingung $y_j < y_i$ gelten. Diese Eigenschaften werden in Vaswani et al. (2017) näher ausgeführt. In Abb. 2.8 ist die Architektur dargestellt.

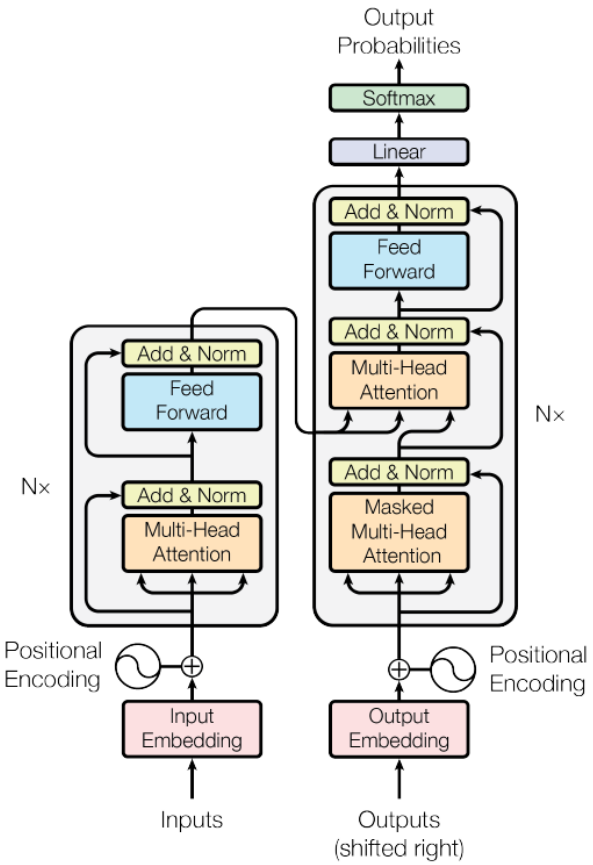

Abbildung 2.8.: Modell-Architektur eines *Transformers* nach Vaswani et al. (2017, S.3)

2.5.2. Encoder- und Decoder

Der *Encoder* besteht in den Arbeiten von Vaswani et al. (2017) aus $N = 6$ identischen Ebenen, auch Blöcke genannt. Jeder dieser Blöcke besteht wiederum aus zwei weiteren Blöcken. Zum Einen einem *Multi-Head Self-Attention*-Mechanismus und zum anderen aus einem *fully connected feed-forward network*. Daran wiederum ist jeweils eine Normalisierungschicht angeschlossen. Die nachfolgende Normalisierungsschicht berechnet jeweils die Summe $f(x)$ aus der jeweilig vorangegangenen Schicht und der ursprünglichen Eingabe x. Somit ergibt sich als Ausgabe für die Normalisierungsschicht $Norm(x + f(x))$, wobei $f(x)$ für den *Multi-Head Self-Attention*-Mechanismus oder *FNN* steht.

Der *Decoder* ist grundsätzlich ähnlich aufgebaut wie der *Encoder* mit dem Unterschied, dass

ein dritter Unterblock hinzugefügt wird, welcher den *Multi-Head Attention*-Mechanismus auf die Ausgabe des *Encoders* anwendet.

In *Encoder*- sowie *Decoder*-Block wird durch die Bereitstellung des originalen Inputs die Möglichkeit geschaffen, eine Verbindungsart zu überspringen. Dies wird *Residual Connection* genannt.

2.5.3. Self-Attention-Mechanismus

In der Arbeit von Vaswani et al. (2017) nimmt der Aufmerksamkeits-Mechanismus (*Attention-Mechanism*) eine zentrale Rolle ein und wird im Folgenden nach den Erläuterungen aus dieser Arbeit erklärt.

Der Mechanismus dient innerhalb des *Transformers* der Übertragung von Sequenzen unter Berücksichtigung der Wichtigkeit eines Wortes und wird auch als *Self-Attention*-Mechanismus bezeichnet, da er sich auf die eigene Eingabe bezieht.

Der Mechanismus wird durch die in Abb. 2.9 aufgeführte Struktur realisiert. Hierbei werden die einzelnen *Attention-Heads* aufgeführt, die jede für sich parallel zueinander das sogenannte *Scaled Dot-Product* berechnet. Die Anzahl der *Attention-Heads* wird in Abb. 2.9 durch h festgehalten und ist in der ursprünglichen Fassung auf $h = 8$ festgelegt.

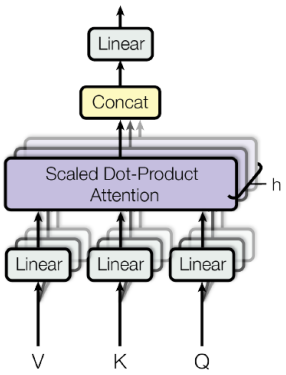

Abbildung 2.9.: Multi-Head Attention Vaswani et al. (2017, S.4)

In Abb. 2.10 ist wiederum die Struktur für die Berechnung des *Scaled Dot-Products* aufgeführt. Als Eingabe werden die Matrizen Q, K und V genutzt. Die Matrizen werden aus der originalen Eingabesequenz $x = (x_1, ..., x_n)$, die jeweils mit den gelernten Gewichtungs-Matrizen W^Q, W^K und W^V multipliziert werden, berechnet. Hierbei ist Q die *Query*-Matrix, K die *Key*-Matrix und V die *Value*-Matrix.

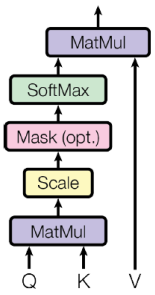

Abbildung 2.10.: Scaled Dot Attention Mask Vaswani et al. (2017, S.4)

Zusammengefasst lässt sich die Berechnung der Ausgabe-Matrix wie in Gleichung (2.3) formulieren. Zunächst wird eine Matrixmultiplikation von Q und K^T durchgeführt und das Ergebnis durch $\sqrt{d_k}$ geteilt. Daraufhin wird die durch die *Softmax*-Funktion[2] normalisierte Matrix dann mit der Matrix V multipliziert. Als Resultat erhält man dann die Ergebnismatrix von $Attention(Q, K, V)$.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.3}$$

Das *Scaled Dot-Product* wird parallel in jedem der $h = 8$ *Attention-Heads* berechnet, wobei in jedem Schritt zufällig gewählte Gewichte bei der Initialisierung genutzt werden. Die so berechneten acht Ergebnis-Matrizen werden anschließen über alle *Attention-Heads* konkateniert. Die so generierte Matrix wird abschließend noch mit der Gewichtungsmatrix W^O multipliziert. Formell ist dies in Gleichung (2.4) und Gleichung (2.5) aufgeführt.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.4}$$

$$where\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.5}$$

In der Arbeit von Vaswani et al. (2017) wurde mit acht *Attention-Heads* gearbeitet. Die Dimensionen der Vektoren sind mit $d_k = d_v = d_{model}/h = 64$ angegeben. Durch die Reduktion der Dimensionen kann der Berechnungsaufwand für den *Multi-Head*-Mechanismus auf den einer einzelnen *Single-Head Attention* (bei voller Dimensionalität) reduziert werden.

Der ursprüngliche *Self-Attention*-Mechanismus arbeitet in der Eingabesequenz absolute Positionen um damit diese in den vorgestellten Berechnungen einberechnet werden können. Eine Erweiterung, sodass auch mit relativen Positionen gearbeitet werden kann, wurde in der Arbeit von Shaw et al. (2018) vorgestellt, soll aber in dieser Arbeit nicht weiter besprochen werden.

[2]Die *Softmax*-Funktion wird auch normalisierte Exponentialfunktion genannt und dient dazu, einen k-dimensionalen Vektor in einen ebenfalls k-dimensionalen Vektor mit reellen Komponenten aus dem Wertebereich {0,1} zu transformieren. Dabei entspricht die Summe der einzelnen Komponenten 1 (Bridle (1990)).

2.6. Bidirectional Encoder Representations from Transformers

In diesem Kapitel soll nach den Arbeiten von Devlin et al. (2018) das BERT-Modell vorgestellt werden. BERT ist ein bidirektionales Modell für die Repräsentation natürlicher Sprache. Im Gegensatz zu vorherigen Modellen wurde BERT entwickelt um unstrukturierte Texte auf allen Modellebenen kontextbezogen links- sowie rechtsseitig mit einzubeziehen. Die Abkürzung steht für **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.

BERT stellt verschiedene bereits vortrainierte Modelle zur Verfügung, die durch die Ergänzung einer Phase für Feinabstimmungen für spezifische Aufgaben erweitert und präzisiert werden können. Dabei entstehen wiederum Modelle, die für eine große Anzahl verschiedenster Aufgabengebiete genutzt werden können.

Die Aufgabengebiete sind im Bereich der Verarbeitung natürlicher Sprache nicht weiter begrenzt und lassen sich somit beispielsweise für die automatische Beantwortung von Fragen, die Vorhersage für fehlende Worte in einem Satz oder Text-Klassifizierungen nutzen (Devlin et al. (vgl. 2018, S. 1)).

Die folgenden Ausführungen werden nach Ravichandiran (2021) und Devlin et al. (2018) vorgestellt.

Ein Grund für den großen Erfolg von BERT ist, dass es sich um ein Kontext-basiertes Modell handelt. Als kurzes Beispiel, was mit *kontext*-basiert gemeint ist, stellen die folgenden zwei Sätze dar.

Satz A: „I have a new racket that is brilliant for playing on clay courts."
Satz B: „I can't concentrate with the racket from our neighbors."

In kontext-freien Modellen, wie beispielsweise Word2Vec (Abschnitt 2.7.1) würde das Wort „racket" jeweils auf die gleiche Weise repräsentiert werden. In einem kontext-basierten Modell wie dem BERT-Modell, wird für beide Sätze jeweils eine andere Wortrepräsentation generiert, da aus dem Kontext ersichtlich wird, dass sich einmal um einen Tennisschläger (Satz A) und einmal um Lärm, der aus der Nachbarschaft kommt (Satz B) handelt.

Um den Kontext zu verstehen und einbinden zu können, stellt BERT jedes Wort in Relation zu allen anderen Wörter in einem gegebenen Satz und prüft, welche Beziehung zwischen diesen Wörtern vorliegt. So sollte sich zum Beispiel ergeben, dass durch die Beziehung zwischen „playing" und „racket" klar wird, dass es sich bei der Bedeutung für das Wort „racket" nicht etwa um das Wort „Lärm", sondern vielmehr um einen Tennisschläger handelt.

BERT basiert auf dem *Transformer*-Modell, dass bereits in Abschnitt 2.5 beschrieben wurde. Ein wesentlicher Unterschied ist hier jedoch, dass nur der *Encoder*, nicht aber der *Decoder* genutzt wird. Bidirektional arbeitet das BERT-Modell, da bereits *Transformer*-Modelle grundsätzlich bidirektional arbeiten können, schließlich ist es möglich, einen Eingabesatz in beide Richtungen einzulesen.

Das Modell nimmt zunächst einen Eingabesatz entgegen und berechnet mithilfe des *Encoders* die kontextuelle Repräsentation für jedes Wort. Dafür wird der *Multi-Head Attention*-Mechanismus (Abschnitt 2.5.3) genutzt. Nun können N *Encoder* hintereinander geschaltet werden. In Abb. 2.11 wird der Aufbau bildlich dargestellt, wobei nur ein *Encoder* schematisch vollständig dargestellt ist. Hierbei entsprechen die Repräsentationen R_x jeweils der Repräsentation eines Wortes. Beispielsweise stellt R_{racket} die Repräsentation des Wortes „racket“ dar. Dabei entspricht die Größe der resultierenden Repräsentation der Größe der *Encoder*-Schicht.

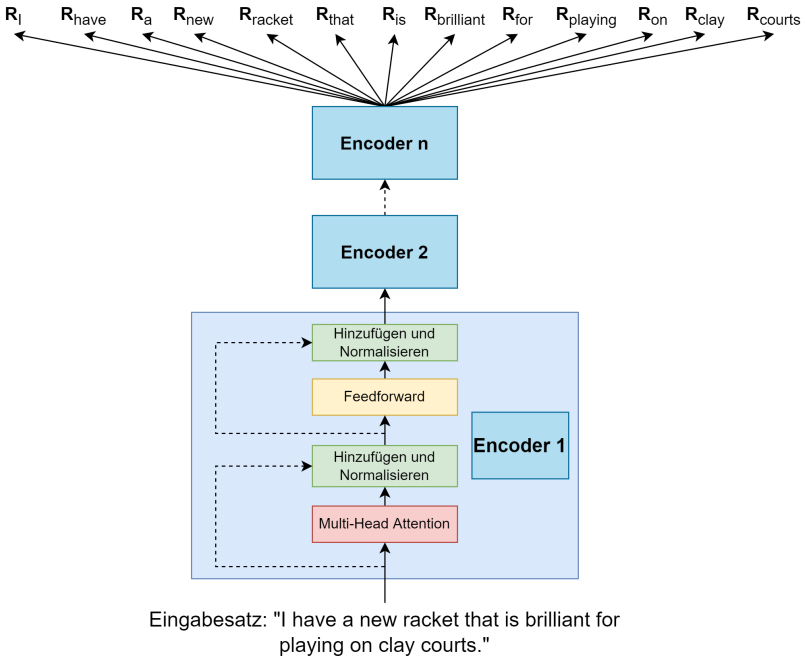

Abbildung 2.11.: Struktur für die Generierung von Wort-Repräsentationen eines Satzes in BERT in Anlehnung an Ravichandiran (2021, S.57)

2.6.1. Konfigurationen von BERT

Die in diesem Unterkapitel behandelten Informationen und Themen werden nach Ravichandiran (2021) vorgestellt. Es gibt verschiedene Konfigurationen, die sich mit dem BERT-Modell anwenden lassen. Diese unterscheiden sich in der Anzahl von *Encoder-Schichten* L, den *Attention-Heads* A und den *Hidden*-Schichten H. Die meistgebrauchten Modelle stellen *bert-base* und *bert-large* dar. Wobei das *bert-base*-Modell mit $L = 12$ *Encoder*-Schichten, $A = 12$ *Attention-Heads* und $H = 768$ *Hidden*-Schichten konfiguriert ist und insgesamt ca. 110 Millionen Parameter beinhaltet, damit ist auch die Größe der resultierenden Repräsentationen gleich der Anzahl an *Hidden*-Schichten und damit ebenfalls 768. Das *bert-large*-Modell kommt hingegen auf 340 Millionen Parameter, wobei $L = 24$, $A = 16$ und $H = 1024$ gilt.

Die Größe eines Modells bedeutet dabei nicht immer, dass es in jedem Fall eine höhere Ge-

nauigkeit aufweist. Wie in der Arbeit von Zhong et al. (2021) gezeigt wurde, verhält es sich zumeist eher so, dass zwar eine höhere Genauigkeit im Durchschnitt erreicht wird, jedoch für einzelne Instanzen durchaus auch weniger genaue Ergebnisse erzielt werden, sodass die Frage, welches Modell ausgewählt werden sollte, wiederum stark von der gegebenen Aufgabenstellung und den verfügbaren Daten abhängt.

Da die Modelle extrem viele Parameter aufweisen können und somit sehr umfangreich werden können, wurden weitere Ansätze entwickelt um aus der Trainingsphase die entscheidenden Informationen zu extrahieren und daraufhin zu komprimieren. Dieser Vorgang wird allgemein als *Destillation* bezeichnet. Je nach Anwendung können sich unterschiedliche Konfigurationen eignen. In der Arbeit von Turc et al. (2019) wurde festgestellt, dass sich die Vorteile von *Pre-Training*, *Fine-Tuning* und *Pre-Trained Destillation* gegenseitig verstärken können und somit eine Maximierung der Nutzung aller verfügbarer Ressourcen entstehen kann.

Es verwundert somit nicht, dass über die letzten Jahre entsprechend viele unterschiedliche Konfigurationen entstanden sind, die sich jeweils für unterschiedliche Anwendungsfälle am besten eignen.

Darstellung der Eingangsdaten von BERT

Damit BERT-Modelle überhaupt Daten entgegen nehmen können, müssen diese in eine entsprechend Repräsentation überführt werden. Dabei gibt es auf *Token*-, *Segment*- und Positions-Ebene verschiedene Repräsentationen, die im Folgenden nach Ravichandiran (2021) kurz erklärt werden.

Anhand des folgenden Zitats von Albert Einstein zitiert nach Ambler (1998, S.361) sollen nachfolgende Erklärungen gezeigt werden.

Satz A: „A clever person solves a problem."
Satz B: „A wise person avoids it."

Zunächst werden Eingabesätze tokenisiert und anschließend spezielle Token für den Eingabeanfang ([CLS]) und Satztrennungen ([SEP]) an die entsprechenden Stellen hinzugefügt.

Die Zerlegung einer Eingabe in Token wird in BERT mithilfe des *WordPiece Tokenizer* durchgeführt (Devlin et al. (2018)). Diese Methode teilt die Eingabe zunächst anhand von Leer- und Satzzeichen in Token auf. Anschließend wird für jedes generierte Token geprüft, ob dieses bereits im BERT-Vokabular beinhaltet ist. Falls ja, ist für diese Token die Prozedur bereits beendet. Ist dem jedoch nicht der Fall, wird das Token weiter unterteilt und abermals für die entstanden Sub-Token geprüft, ob es im Vokabular vorhanden ist. Dies wird im Zweifel so oft durchgeführt, bis man auf die Ebene einzelner Buchstaben angekommen ist. Mit dieser Methode kann effektiv dafür gesorgt werden, dass kein *out-of-vocabulary*-Fall auftritt und Probleme in der ersten Phase der Verarbeitung auftreten.

In den Arbeiten von Song et al. (2020) wurden bereits experimentelle Versuche erfolgreich unternommen, den Vorgang der *Tokenization* in linearer Zeit absolvieren zu können. Dies stellt aufgrund der teils enorm großen Eingabedaten, die für BERT-Modelle herangezogen werden, einen wichtigen Faktor dar, um die Berechnungszeit in Teilen reduzieren zu können.

```
tokens = [ [CLS], A, clever, person, solves, a, problem, .
↪   , [SEP], A, wise, person, avoids, it, .]
```

Jedes Token wird anschließend durch eine Einbettungs-Ebene in ein sogenanntes *token-embedding* überführt. Wobei E_{CLS} die Einbettung des Tokens $[CLS]$, E_{clever} die Einbettung des Tokens $clever$ usw. beinhaltet. In Abb. 2.12 ist dies schematisch dargestellt.

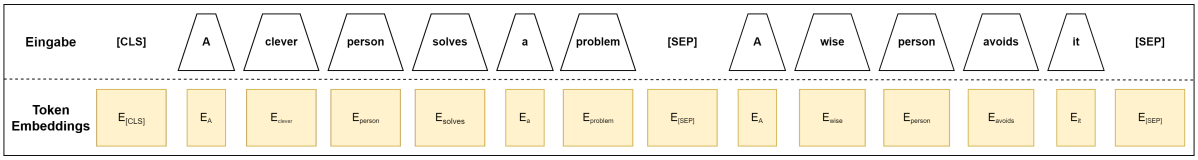

Abbildung 2.12.: Token-Embeddings in Anlehnung an Ravichandiran (2021, S.62)

Anschließend wird jedes Token einem Segment bzw. Satz zugeordnet. Dies ist schematisch in Abb. 2.13 dargestellt. Dabei ist zu beachten, dass die vorher für die Markierung eines Satzendes eingefügten Sonder-Token ($[SEP]$) jeweils als Ende des Satzes betrachtet werden und somit diesem Satz und nicht dem nächsten Satz zugeordnet werden.

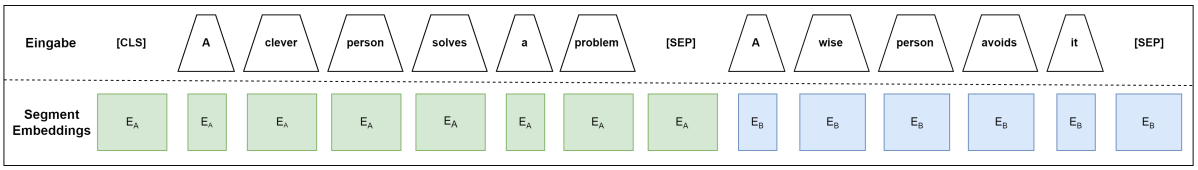

Abbildung 2.13.: Segment-Embeddings in Anlehnung an Ravichandiran (2021, S.63)

Da alle Wörter bzw. Token durch den *Encoder*, dessen Grundidee sich auch BERT bedient, parallel verarbeitet werden, verfallen die Informationen über die Positionierung des Tokens in den Eingabedaten. Diese Informationen werden gesondert durch das *Position embedding* erfasst und festgehalten (siehe Abb. 2.14).

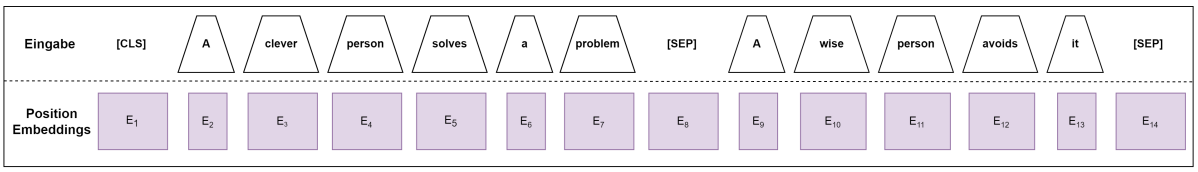

Abbildung 2.14.: Position-Embeddings in Anlehnung an Ravichandiran (2021, S.64)

Alle in Abb. 2.15 dargestellten *embeddings* werden nun aufsummiert als Eingabe an BERT übergeben. Durch dieses Vorgehen werden alle nötigen Informationen an BERT übergeben und alle anschließenden Operationen können entsprechend auf diese Grundinformationen zurückgreifen.

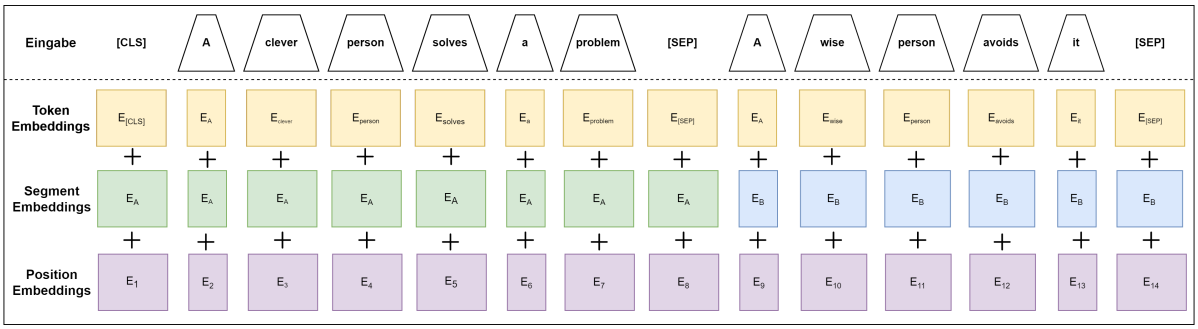

Abbildung 2.15.: Vollständig schematische Eingabe-Repräsentation in Anlehnung an Ravichandiran (2021, S.64)

2.6.2. Pre-Training

Dieses Kapitel beschreibt nach den Arbeiten von Devlin et al. (2018, S.4-5) und Ravichandiran (2021) das Thema des *Pre-trainings* in Bezug auf das BERT-Modell. Ein Modell sowie auch das BERT-Modell, muss vor der tatsächlichen Anwendung auf eine Aufgabenstellungen trainiert werden. In der Regel werden sogenannte *pre-trained models*, also bereits vortrainierte Modelle herangezogen, die anschließend nur noch auf eine bestimmte Aufgabenstellung mit dafür relevanten Daten fein abgestimmt werden müssen.

Zu Beginn wird ein Modell m auf einem sehr großen Datensatz trainiert. Anstatt für eine neue Aufgabenstellung ein neues Modell zu entwickeln und dieses wiederum von Grund auf neu trainieren zu müssen, wird das vorher bereits entwickelte Modell herangezogen und mit dessen Gewichten als Initialzustand gearbeitet. Anschließend aber mit einem Neuen für die neue Aufgabenstellung relevanten Datensatz abgestimmt (*Fine-Tuning*). Diesen Vorgang kann man als Transfer-Lernen bezeichnen (*Transfer-Learning*).

BERT-Modelle werden in der Trainingsphase mit einem großen Datensatz auf die Aufgaben *Language Modelling* (MLM) und *Next-Sentence Prediction* (NSP) trainiert. Durch diese Auf-

gabenstellungen kann dem BERT-Modell ein grundlegendes Sprachverständnis beigebracht werden. Dabei sorgt das Training über *MLM* für die Erfassung des bidirektionalen Kontexts und über *NSP* werden satzübergreifende Zusammenhänge erlernt.

Für das *Pre-Training* von BERT wurde auf den *BookCorpus* (Zhu et al. (2015)) der 800 Millionen und die Englische Wikipedia, die 2.500 Millionen Worte umfasst, zurückgegriffen.

Masked Language Modelling

Die folgenden Erläuterungen folgen den in Devlin et al. (2018, S.4) aufgeführten Inhalten. Bei der Aufgabe des *MLM* versucht das BERT-Modell für eine gegebene Eingabe ein vorher maskiertes Wort vorherzusagen. Dabei wird das Modell bidirektional einen Satz einlesen, um darauf basierend die Vorhersage durchzuführen. In der Trainingsphase werden 15% der Wörter aus den Eingabedaten maskiert. Das heißt, dass das Token selbst mit dem Token $[MASK]$ ausgetauscht wird. Die maskierten Token werden anschließend vorhergesagt. Da die Stellen der maskierten *Token* für BERT unbekannt sind, resultiert daraus eine kontextbezogene, gleichzeitig aber auch verteilte Repräsentation der Eingabedaten.

Durch dieses Vorgehen entsteht zwischen *Pre-Training* und *Fine-Tuning* eine Diskrepanz, die sich daraus ergibt, dass im *Pre-Training* maskierte *Token* vorkommen, jedoch in der Phase des *Fine-Tunings* keine maskierten *Token* vorhanden sind. Die entstehende Diskrepanz kann mit der sogenannten 80-10-10%-Regel beantwortet werden. Diese besagt, dass in nur 80% der Fälle für ein zufällig ausgewähltes Token, welches markiert werden soll, tatsächlich $[MASK]$ eingesetzt wird. In 10% der Fälle wird ein anderes zufälliges *Token* eingesetzt und in weiteren 10% wird das ursprüngliche *Token* nicht maskiert und verbleibt stattdessen in seinem Originalzustand.

Nach der *Tokenization* und der Maskierung einzelner *Token* werden die Eingabedaten an die in Abschnitt 2.6.1 beschriebenen Schichten und anschließend an BERT übermittelt. Daraus resultiert eine Repräsentation für jedes *Token* der Eingangsdaten, auch für $R_{[MASK]}$, das maskierte *Token*.

Um die Vorhersage für das maskierte *Token* durchzuführen, wird $R_{[MASK]}$ an das *FNN* mit einer *Soft-max*-Aktivierung übergeben. Für jedes im Vokabular von BERT vorhanden Token kann somit berechnet werden, wie hoch seine Wahrscheinlichkeit ist für das maskierte *Token* eingesetzt werden zu können. In Abb. 2.16 ist der beschriebene Vorgang schematisch dargestellt, wobei die in Abschnitt 2.6.1 beschriebenen Schritte für die Übersichtlichkeit nicht aufgeführt sind. Als Beispiel sollen erneut die folgenden Sätze fungieren. Wobei das Wort „problem“ maskiert wird.

Satz A: „A clever person solves a [MASK].“
Satz B: „A wise person avoids it.“

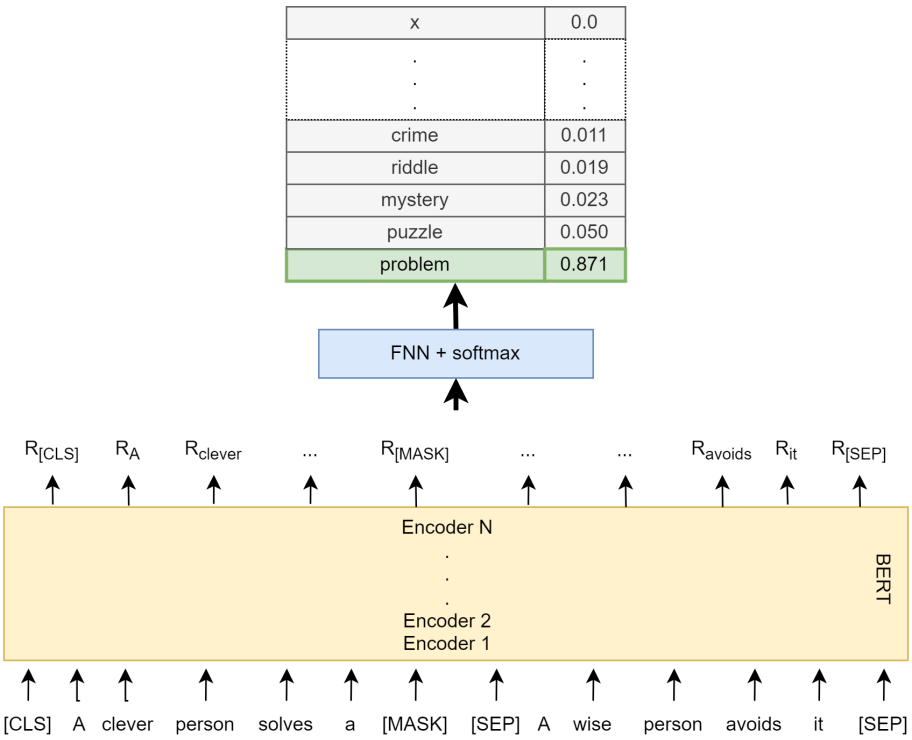

Abbildung 2.16.: Vorhersage eines maskierten *Token* in Anlehnung an Ravichandiran (2021, S.70)

Die in Abb. 2.16 aufgeführten Ergebnisse für die Vorhersage des maskierten *Token* entsprechen realen Werten, die aus dem Service von *Huggingface*[3] stammen und auf dem *bert-base-uncased* Modell basieren, welches bereits in Abschnitt 2.6.1 vorgestellt wurde und auf den Arbeiten von Devlin et al. (2018) basiert. Ein Modell wird in der *Pre-Training*-Phase initial noch nicht die richtigen Wahrscheinlichkeitswerte ausgeben, da die einzelnen Gewichtungen erst durch mehrere Iterationen und den *Backpropagation*-Mechanismus[4] im *FNN* und den Einbettungsebenen angepasst werden müssen.

Next sentence prediction

Eine weitere Methode während des *Pretrainings* von BERT-Modellen stellt die *Next Sentence Prediction* dar, die im Folgenden nach den Arbeiten von Devlin et al. (2018, S.4-5) erklärt wird. Bei der *Next sentence prediction*, kurz *NSP*, geht es um eine binäre Klassifizierungsaufgabe. Dabei werden als Eingabedaten zwei Sätzen an BERT übergeben und das Modell muss daraufhin entscheiden, ob der zweite Eingabe-Satz der nachfolgende Satz für den ersten Eingabesatz ist. Die beiden Klassen, in die das Modell unterscheidet, hängen davon ab, ob der zweite Eingabe-Satz der Folgesatz des ersten Eingabesatzes ist (*isNext*) oder nicht (*notNext*).

[3] https://huggingface.co/bert-base-uncased, Abrufdatum: 21.02.2022 11:31Uhr

[4] *Backpropagation ist ein Verfahren, dass aus dem supervised Learning stammt und die Fehlerrückführung für KNN umfasst, die zur Optimierung der einzelnen Gewichte im Netz selbst führt (Werbos (1990)).*

Hierzu werden aus einem monolingualen Korpus eine Reihe von zwei aufeinanderfolgenden Sätzen extrahiert und jeweils mit der Zusatzinformation *isNext* abgespeichert. Daneben werden weitere Einzelsätze aus dem Korpus extrahiert und mit einem zweiten Satz aus einem anderen Korpus verbunden, jeweils mit der Markierung *notNext* versehen und ebenfalls gespeichert. Hierbei ist es wichtig, dass jeweils 50% der Daten mit einer der beiden Klassen assoziiert sind.

Die so generierten Trainingsdaten werden ähnlich Abschnitt 2.6.2 durch die verschiedenen Einbettungschichten in eine für BERT kompatible Repräsentation übertragen und anschließend an das BERT-Modell übergeben. Die daraus resultierenden Token-Repräsentationen für alle Token der Eingabe werden bei der *NSP* jedoch nicht alle benötigt. Stattdessen sind in der Repräsentation für das *[CLS]*-Token ($R_{[CLS]}$) alle für die folgende Klassifizierung benötigten Informationen enthalten. Dies ist der Fall, da $R_{[CLS]}$ die Repräsentationen der weiteren *token* aggregiert beinhaltet.

$R_{[CLS]}$ kann nun dem *FNN* mit *softmax*-Funktion übergeben werden, woraufhin die Wahrscheinlichkeiten für die Klassenzugehörigkeiten ausgegeben werden können. Das Vorgehen für die *NSP* wird in Abb. 2.17 schematisch dargestellt. Äquivalent zu Abb. 2.16 sind auch in Abb. 2.17 keine Einbettungsschichten aufgeführt.

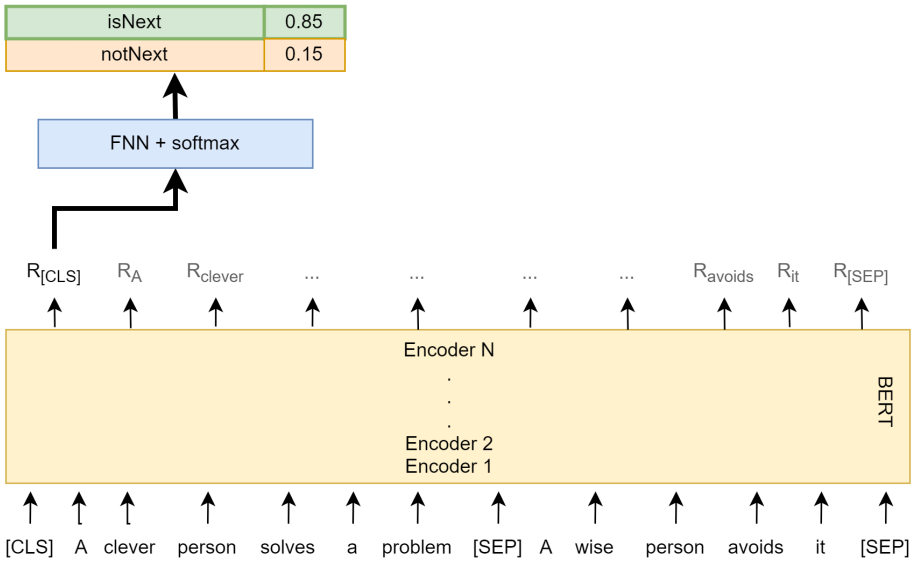

Abbildung 2.17.: Vorhersage der *NSP* in Anlehnung an Ravichandiran (2021, S.75)

2.6.3. Feinabstimmung

Die in Abschnitt 2.6.2 vorgestellte Prozedur ermöglicht es, ein bereits *pre-trained-model* für spezifischere Aufgabenstellungen mithilfe des *Fine-Tunings* nutzen zu können, welches nach Devlin et al. (2018, S.5) im Folgenden vorgestellt wird. Das *Fine-Tuning* stammt aus dem *Supervised Learning*, das bereits in Abschnitt 2.3 angesprochen wurde. Durch den in BERT enthaltenen *Self-Attention*-Mechanismus (Abschnitt 2.5.3) ermöglicht es das Modell flexibel auf

spezifische Aufgabenstellungen anzupassen und dabei auf bereits gelernte Zusammenhänge zurückgreifen zu können.

In Abb. 2.18 ist der schematische Zusammenhang zwischen *Pre-Training* und *Fine-Tuning* bildlich dargestellt. Bei den darin exemplarisch aufgeführten Aufgabenstellungen handelt es sich um *MNLI*[5], *NER*[6], *SQuAD*[7]

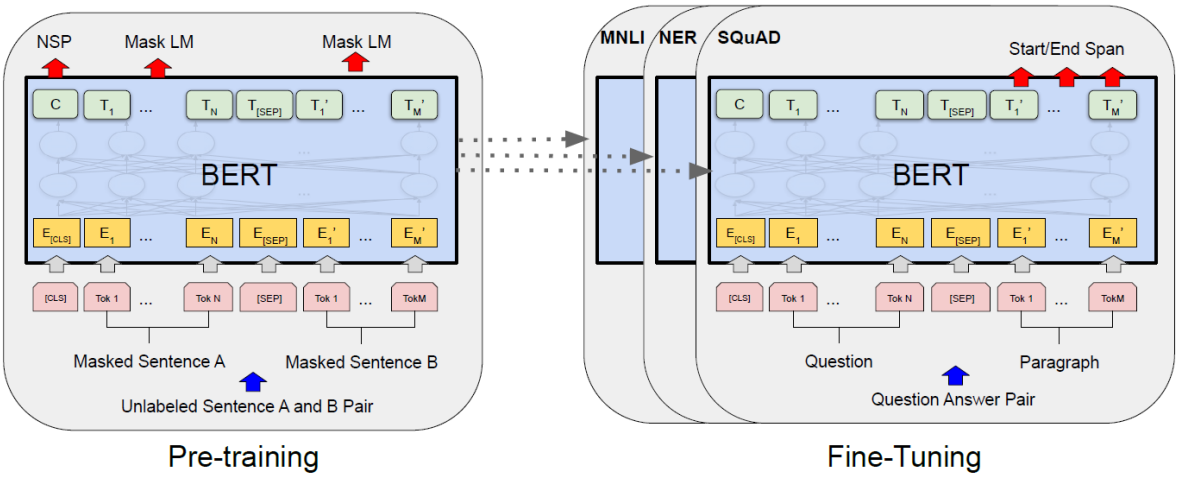

Abbildung 2.18.: Aufbau des *Pre-Trainings* und *Fine-Tunings* von BERT Devlin et al. (2018, S.3)

Für das *Fine-Tuning* wird auf die Architektur des *Pre-Trainings* zurückgegriffen. Allerdings wird ein neu initialisiertes *FNN* mit *softmax*-Funktion genutzt, welches der Ausgabe-Schicht hinzugefügt wird und für die Klassifizierung in der *Fine-Tuning*-Phase zuständig ist.

Durch die in der *Pre-Training*-Phase bereits vortrainierten Gewichte müssen diese in der *Fine-Tuning*-Phase lediglich auf die entsprechende Aufgabenstellung abgestimmt werden. Der zeitliche Aufwand ist hierbei verglichen zur *Pre-Training*-Phase erheblich geringer, was dazu führt, dass sich einmal vortrainierte BERT-Modelle sehr gut für neue Aufgaben nutzen lassen. Als Trainingsdaten für die *Fine-Tuning*-Phase werden für die jeweilige Aufgabenstellung spezifische Daten benötigt, die entsprechend relevante Zusatzinformationen enthalten.

[5]Multi-Genre Natural Language Inference ist ein großes, *crowdsourced* Klassifizierung-Aufgabe, bei der ähnlich der *NSP* für einen Folgesatz unterschieden werden soll, in welcher Beziehung dieser zum ersten Satz steht. Hierbei wird jedoch in mehrere Klassen unterschieden (Williams et al. (2017)).

[6]Bei der *Named Entity Recognition* geht es um die Klassifikation von Entitäten bspw. Personen, Organisationen oder Orte (Nadeau und Sekine (2007)).

[7]Das *Stanford Question Answering Dataset* ist eine Datenbank mit über 100.000 Fragen über Wikipedia-Artikel , das für die Aufgabe der automatischen Beantwortung von Fragen genutzt werden kann. Diese Transfer-Aufgabe wird *Question Answering (QA)* genannt und stellt eine grundlegende Aufgabenstellung dar (Rajpurkar et al. (2016)).

2.7. Statische Modelle

In der Verarbeitung natürlicher Sprache und der Verwendung von Methoden aus dem *Machine Learning* bilden Wort-Repräsentationen eine zentrale Basis, um auf mathematischer Ebene Berechnung durchführen zu können. Es gibt zahlreiche Repräsentationsarten, die über die letzten Jahre entwickelt und eingesetzt wurden. Die wohl einfachste Form stellt die Methode *Bag-of-Words* dar. Hierbei wird jedem Zeichen in einem Text ein diskreter Wert zugeordnet. Dadurch werden einfache Suchanfragen über Wortindexe möglich. In der Arbeit von Zhang et al. (2010) geht es zwar ursprünglich um die Repräsentation von Bildern, das Verfahren lässt sich aber auch auf Texte anwenden.

Natürlich sind für komplexere Fragestellungen Wort-Repräsentationen gefragt, die deutlich mehr Informationsgehalt in sich vereinen. Die für diese Arbeit relevanten Modelle werden im Folgenden kurz erklärt.

2.7.1. Word2Vec

In der Arbeit von Mikolov, Chen et al. (2013) stellen die Forscher ein distributionelles Modell vor, welches die semantische Bedeutung der in einem entsprechenden Text-Korpus vorliegende Wörter sequenziell verarbeitet und anschließend als Wort-Vektoren repräsentieren kann. Durch die sequenzielle Verarbeitung der Eingangsdaten können sprachliche Kontexte, die zwischen einzelnen Wörtern bestehen, in die Vektorrepräsentation mit aufgenommen werden. Als Eingabedaten wird ein Korpus aus voneinander getrennten Sätzen genutzt. Alle in einem Satz enthaltenen Wörter werden als n-dimensionale Vektoren dargestellt. Somit werden automatisch Wörter mit einem kontextuellen Bezug zueinander räumlich dicht aneinander platziert. Einem Zeichen wird kein diskreter, sondern ein kontinuierlicher Wert $v \in \mathbb{R}^n$ zugeordnet.

Durch die Vektorrepräsentation ist es möglich, mathematische Operationen auf diesen auszuführen und somit verschiedene Berechnungen wie die Ähnlichkeit von zwei Wörtern, zu berechnen. Die Ähnlichkeit von zwei Vektoren wird oftmals durch die Kosinus-Ähnlichkeit oder den euklidischen Abstand berechnet. Die Kosinus-Ähnlichkeit zweier Vektoren a und b entspricht dem Kosinus des eingeschlossenen Winkels θ (Gleichung (2.6)). Der euklidische Abstand ergibt sich in einem n-dimensionalen Raum $\mathbb{R}^n$ durch die Ermittlung des Differenzvektors zweier Vektoren (Gleichung (2.7)).

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2.6}$$

$$d(A, B) = |\vec{a} - \vec{b}| \tag{2.7}$$

Aus der Arbeit von Mikolov, Chen et al. (2013) ergibt sich, dass ein guter Wert für die Dimensionen bei 300 liegt. Die Trainingsdaten für das WORD2VEC-Verfahren benötigen lediglich

unstrukturierte Daten, sodass es zahlreiche mögliche Trainingsdaten gibt.

Die aus dem Verfahren entstehenden Vektoren besitzen mehrere interessante Eigenschaften. Neben der semantischen Ähnlichkeit wird auch die syntaktische Ähnlichkeit repräsentiert, wodurch es ermöglicht wird, aus Adjektiven, Steigerungsformen oder aus Verben die verschiedenen Zeitformen abzuleiten. Die semantischen Informationen, die in den Wortvektoren repräsentiert werden, führen dazu, dass bspw. *einfache* Gleichungen wie $Apple - iOS = Google - X$ durch die Auflösung mit Hilfe von Word2Vec zur Auflösung $X = Android$ gelangt.

Die Implementierung von Word2Vec basiert in der Regel auf einem *CBOW*- (*continuous bag of words*) oder einem *Skip-Gram*-Modell. Beim *CBOW*-Modell werden die im Kontext eines ausgewählten Wortes liegenden Wörter (Umgebungswörter) als Eingaben und damit das ausgewählte Wort als Ziel verwendet. Beim *Skip-Gram*-Modell ist diese Vorgehensweise vertauscht. Mithilfe eines ausgewählten Wortes wird von diesem versucht, auf die im Kontext umliegenden Worte zu schließen. In Abb. 2.19 ist der Unterschied zwischen beiden Ansätzen schematisch dargestellt.

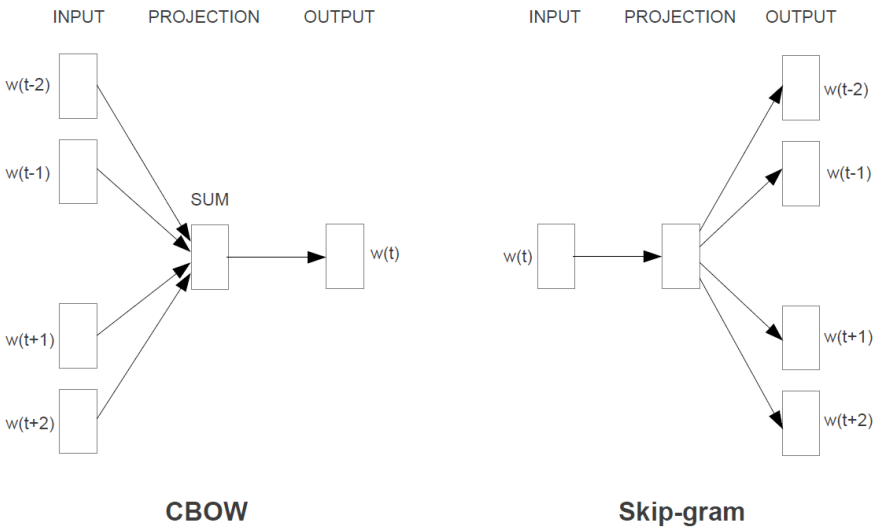

Abbildung 2.19.: Architekturen des CBOW- und des Skip-gram-Modells (Mikolov, Chen et al. (2013, S.5))

Das Word2Vec-Modell wurde durch die Arbeit von Mikolov, Sutskever et al. (2013) hinsichtlich der Repräsentationsqualität und Berechnungsgeschwindigkeit weiter verbessert. Hierfür wurden beispielsweise *Sub-Sampling*[8] für häufig auftretende Worte oder*negative sampling*[9] verwendet.

[8]Sub-Sampling ist ein Verfahren um frequenzabhängig Wörter zu ignorieren, um die Anzahl an Trainingsdaten zu reduziert (Mikolov, Sutskever et al. (2013)).

[9]Negative sampling ist ein Verfahren um die Menge an Gewichtsanpassungen zu reduzieren, indem negative Zwischenergebnisse nur teilweise übertragen werden (Mikolov, Sutskever et al. (2013)).

2.7.2. GloVe

Eine weitere Wort-Repräsentation stellt GLove dar, dass nach den Arbeiten von Pennington et al. (2014) kurz vorgestellt wird. GLove steht für *Global Vectors* und verwendet im Gegensatz zu vielen anderen Ansätzen keine neuronalen Netze. Stattdessen werden durch Minimierung einer Zielfunktion auf einer *Co-Occurence*-Matrix die Repräsentation von Wörtern als Vektoren berechnet. Es handelt sich dabei um einen *unsupervised learning*-Algorithmus (Abschnitt 2.3), der für die Berechnung von Wort-Repräsentationen genutzt werden kann.

Bei den verschiedenen Modellen für die Repräsentation von Wortvektoren lässt sich nach Pennington et al. (2014) in zählbasierte (*Count-Based*) und direkte Vorhersage-Modelle (*Direct Prediction*) unterscheiden. *Direct Prediction*-Modelle lassen sich sehr gut auf Aufgaben mit starkem Analogie-Bezug oder zur Erfassung komplexer linguistischer Muster anwenden. Ein Problem ist jedoch, dass derartige Modelle wiederkehrend neu berechnet werden müssen, sofern sich der abzubildende Korpus verändert.

Die *Count-Based*-Modelle können hingegen stets auf die einmalig generierte Statistik zurückgreifen. Die Berechnungszeit bleibt, solange die Matrix nicht zu groß wird, relativ kurz. Bei diesen Modellen werden jedoch semantische Bedeutungen von einzelnen Wörtern unbeachtet gelassen, lediglich Informationen über die Syntax bleiben erhalten. Ein zusätzliches Problem stellen sehr häufig vorkommende Wörter da, die jedoch nur wenig Bedeutung beinhalten (*Stop-Words*)[10]. Hierzu wurden jedoch bereits Ansätze entwickelt, die diesen Einfluss verringern (Rohde et al. (2006)).

Das GLove-Modell versucht beide vorher angesprochene Modelle zu kombinieren und daraus globale Repräsentationen für Wort-Vektoren abzuleiten.

Die *Co-Occurence*-Matrix ist der wesentliche Bestandteil des Verfahrens und sagt über einen gegebenen Textkorpus aus, in welcher Häufigkeit Wortpaare zusammen auftreten. Dabei repräsentiert jeder Wert der Matrix jeweils ein Wortpaar. Beispielhaft ist eine *Co-Occurence*-Matrix für die Beispielsätze A und B in Tabelle 2.1 aufgeführt.

Satz A: „Long nights are beautiful."
Satz B: „Beautiful nights are not always long."

[10] *Stoppwörter sind Wörter die in der Regel sehr häufig vorkommen, aber für die Verarbeitung natürlicher Sprache üblicherweise keine und nur sehr geringe Bedeutung besitzen (Sarica und Luo (2021)).*

Tabelle 2.1.: *Co-Occurence*-Matrix für die Sätze A und B

	always	**are**	**beautiful**	**long**	**nights**	**not**
always	0	1	0	1	0	1
are	1	0	2	1	2	1
beautiful	0	2	0	0	2	0
long	1	1	0	0	1	1
nights	0	2	2	1	0	1
not	1	1	0	1	1	0

Aus der *Co-Occurence*-Matrix lassen sich anschließend mit Hilfe der Gleichung (2.8) die Wahrscheinlichkeiten von Wortpaaren ermitteln.

$$P_{ij} = \frac{X_{ij}}{X_i} \tag{2.8}$$

Das allgemeine Ziel des GLove-Modells ist es, Wortvektoren so zu erlernen, dass der Logarithmus der Wahrscheinlichkeit für das gemeinsamen Auftretens von Wörtern gleich dem *Dot-Product* (Abschnitt 2.5.3) ist. Dies führt dazu, dass Vektorunterschiede entstehen, die zusätzliche Informationen beinhalten. Besonders auf Aufgabenstellungen, die den Fokus auf Analogien richten, schneidet GLove entsprechend gut ab.

2.7.3. fastText

Das von *Facebook AI Research* entwickelte Modell fastText kann zur Textklassifizierung und für die Erstellung morphologischer Wort-Repräsentationen genutzt werden (Joulin et al. (2016)). Anhand der Arbeiten von Bojanowski et al. (2017) wird das Modell kurz erklärt.

Die meisten Modelle für die Repräsentation von Worten weisen jedem im Vokabular befindlichen Wort eine Vektorrepräsentation zu. Dies geschieht beispielsweise auch bei Word2Vec. Problematisch wird dieser Umstand besonders bei Sprachen mit einem großen Vokabular in dem sich viele sehr selten vorkommende Wörter befinden. So kann es passieren, dass einzelnen Wörtern keine Vektor-Repräsentation zugewiesen wurde. Würde man es schaffen jedem noch so seltenen Wort eine Vektor-Repräsentation zuzuordnen, würde sich das Modell dann aufgrund der immensen Größe nur noch schlecht für die weitere Bearbeitung eignen, da die Zeit für die Verarbeitung zu groß wäre.

fastText nimmt sich diesem Problem an, indem nicht jedem Wort eine Vektor-Repräsentation zugeordnet wird, stattdessen werden Wörter als sogenannte *n-grams* auf Buchstabenebene repräsentiert. Dabei wird die Morphologie durch die Einbeziehung der Teilworte eines Wortes miteinbezogen und das Gesamtwort als Summe aus den *n-gram* der Teilworte auf Buchstabenebene berechnet. Das Modell basiert auf dem *skip-gram*-Modell (Mikolov, Sutskever et al. (2013)).

Als Modell für die Bildung der Teilworte, die jeweils als *n-gram* dargestellt werden, gilt

$3 <= n <= 6$. Spezielle Zeichen ($<, >$) werden für die Begrenzung am Wortanfang ($<$) und Wortende ($>$) eingesetzt. Das Gesamtwort wird ebenfalls in die Repräsentation mit aufgenommen. Beispielhaft ist eine Repräsentation des Wortes „daydream "mit $n = 3$ im Folgenden einzusehen:

$$< da, day, ayd, ydr, dre, rea, eam, am >$$

Eines der aufgeführten Teilworte ist das Wort „day", welches zwar ein eigenständiges Wort ist, allerdings in dieser Repräsentation als Teilwort nicht gleich der Repräsentation eines Gesamtwortes entspricht. Die Gesamt-Repräsentation des Wortes ergibt sich nun aus der Summe der einzelnen *n-gram*-Repräsentationen. G ist die Größe eines Wörterbuchs von *n-grams* und w ein gegebenes Wort, dessen Repräsentation gesucht wird. Dabei gilt, dass $G_W \supset 1, ..., G$ die *n-grams* darstellen, die in w enthalten sind. Zusätzlich wird als Vektor-Repräsentation z_g zu jedem *n-gram* g assoziiert. Dann lässt sich die Summe über diese Vektor-Repräsentationen der einzelnen *n-grams* wie in Gleichung (2.9) (Bojanowski et al. (2017, S.3)) aufgeführt als die Gesamt-Repräsentation eines Wortes bezeichnen.

$$s(w, c) = \sum_{g \in G_w} z_g v_c \tag{2.9}$$

Durch dieses Vorgehen können für selten vorkommende Wörter sehr gute Wort- Repräsentationen berechnet werden.

2.8. TF-IDF

Das statistische Verfahren TF-IDF gibt die Relevanz von einzelnen Wörtern, sogenannten *Keywords*, in einer Menge von Dokumenten an. Dabei kombiniert das Verfahren Werte für die Frequenz eines Terms (*TF*) und der inversen Dokument Frequenz (*IDF*) zu einem einzelnen Wert. Das Verfahren wird nun kurz nach Qaiser und Ali (2018) vorgestellt.

Die *Term Frequency*, kurz *TF*, gibt an, wie oft ein Term in einem Dokument vorkommt. Das bedeutet, dass bei einem gegebenen Dokument und dessen Gesamtanzahl an Wörtern n und einem gegebenen Wort w die Term-Frequenz durch Gleichung (2.10) angegeben werden kann.

$$TF = \frac{n}{w} \tag{2.10}$$

Neben der *TF* gibt die *Inverse Document Frequency*, kurz *IDF*, an, wie oft ein Term t in einer Menge von Dokumenten d ist. Dies lässt sich durch Gleichung (2.11) ausdrücken. Mit Hilfe der *IDF* kann auf einfache, aber effektive Weise festgestellt werden, welche Worte eine hohe und welche eine geringe Relevanz in der Gesamtmenge an Wörtern aller Dokumente besitzen. In diesem Zusammenhang werden auch sogenannte *Stop Words* behandelt.

$$IDF = log_e(\frac{d}{w}) \tag{2.11}$$

Die Kombination der Werte *TF* und *IDF* ergeben den Wert für *TF-IDF*. Dieser wird durch einfache Multiplikation beider Einzelwerte errechnet (Gleichung (2.12)).

$$TF - IDF = TF \cdot IDF \tag{2.12}$$

3. Grundlagen

In diesem Kapitel werden grundlegende Konzepte vorgestellt, die im Rahmen dieser Arbeit genutzt wurden. Die grundlegende Graphentheorie (Abschnitt 3.1) beschreibt die Regeln für die Erstellung von Graphen. Außerdem werden damit im Zusammenhang stehende Algorithmen beispielsweise für die Berechnung von kürzesten Wegen zwischen zwei Knoten vorgestellt. In Abschnitt 3.2 werden verwendete Datensätze behandelt, die für diese Arbeit genutzt wurden. Der darauffolgende Abschnitt 3.3 stellt die genutzten Softwarekomponenten vor. Abschließend werden in Abschnitt 3.4 weitere Ressourcen aufgegriffen, die im Rahmen dieser Arbeit verwendet wurden.

3.1. Graphentheorie

Da Graphen eine grundlegende Rolle im Zusammenhang mit dieser Arbeit spielen, werden die relevanten Grundlagen der Graphentheorie nach der Arbeit von Volker Turau (2009) und Stegbauer und Häussling (2010, S.345-353) im Folgenden kurz erklärt.

3.1.1. Ungerichtete und gerichtete Graphen

Ein *ungerichteter* Graph $G_u = (V_u, E_u)$ besteht aus einer Knotenmenge V_u und einer Kantenmenge E_u. Dabei werden jeder Kante $e \in E_u$ zwei Knoten aus V_u zugeordnet. Ein *gerichteter* Graph $G_g = (V_g, E_g)$ besteht ebenfalls aus einer Menge V_g an Knoten. Jedoch enthält die Menge an Kanten E_g in diesem Fall gerichtete Kanten, die sich aus einem geordneten Knotenpaar zusammensetzen. Es ist möglich, dass es zwischen zwei Knoten $u, v \in V$ mehrere Kanten gibt. Außerdem kann es Kanten $e \in E$ geben, deren Anfangs- und Endknoten identisch sind (Schleifen) (Volker Turau (vgl. 2009, S.18)).

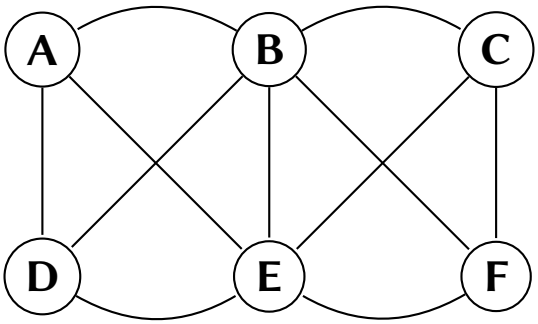

Abbildung 3.1.: Ein Beispiel für einen ungerichteten Graphen

In Abb. 3.1 ist ein Beispiel für einen ungerichteten Graphen mit der Knotenmenge V_u und Kantenmenge E_u aufgeführt.

$V_u = (A, B, C, D, E, F)$
$E_u = ((A, B), (A, D), (A, E), (B, D), (B, E), (B, F), (C, E), (C, F), (D, E), (E, F))$

Isomorphie und Teilgraphen

Teilgraphen beinhalten eine Teilmenge von Knoten und Kanten eines Originalgraphen. Dabei müssen alle Endknoten der Teilgraphen-Kantenmenge beinhaltet sein. Von besonderer Bedeutung ist der Begriff der *Isomorphie*. Sind zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ isomorph ($G_1 \cong G_2$), gibt es eine bijektive Abbildung $\phi : V_1 \longmapsto V_2$ die für alle $(u, v) \in E$ die Bedingung $(u, v) \in E_1 \longleftrightarrow (\phi(u), \phi(v)) \in E_2)$ erfüllt.

Damit sind zwei isomorphe Graphen G_1 und G_2 nicht zu unterscheiden, sofern Attribute und Identifikatoren entfernt werden. Über die *Isomorphie* und Teilgraphen im Allgemeinen lassen sich beispielsweise Untersuchungen bzgl. der Erfassung von *Cliquen*[1] in einem gegebenen Graphen durchführen (Stegbauer und Häussling (vgl. 2010, S.350-351)).

Grad eines Knoten

Der Grad eines Knotens $v \in V_u$ beschreibt mit $deg(v) = |e \in E : v \in e|$ bei einem ungerichteten Graphen die Anzahl der Kanten, an denen dieser Knoten beteiligt ist. Da die Kanten ungerichtet sind, wird keine Unterscheidung in Eingangsgrad oder Ausgangsgrad getroffen. Bei *gerichteten* Graphen wird dahingehend in Ein- bzw. Ausgangsknoten unterschieden, sodass sich daraus für den Eingangsgrad $deg^-(v) = |e \in E : e = (v, w)|$ und für den Ausgangsgrad $deg^+(v) = |e \in E : e = (u, v)|$ ableiten lässt (Stegbauer und Häussling (vgl. 2010, S.351)).

In Abb. 3.2 ist ein Beispiel für einen *gerichteten* Graphen aufgeführt. Beispielweise hat der Knoten B den Eingangsgrad $deg^-(B) = 3$, da drei Kanten von anderen Knoten in B eingehen. Der Ausgangsgrad ist $deg^+(B) = 1$, da der Knoten lediglich eine ausgehende Kante besitzt.

[1] Eine *Clique* heißt ein Teilgraph, in dem jeder Knoten mit jedem anderen Knoten verbunden ist. In der Forschung wird in der Regel versucht, möglichst große *Cliquen* zu finden (Stegbauer und Häussling (2010, S.351).

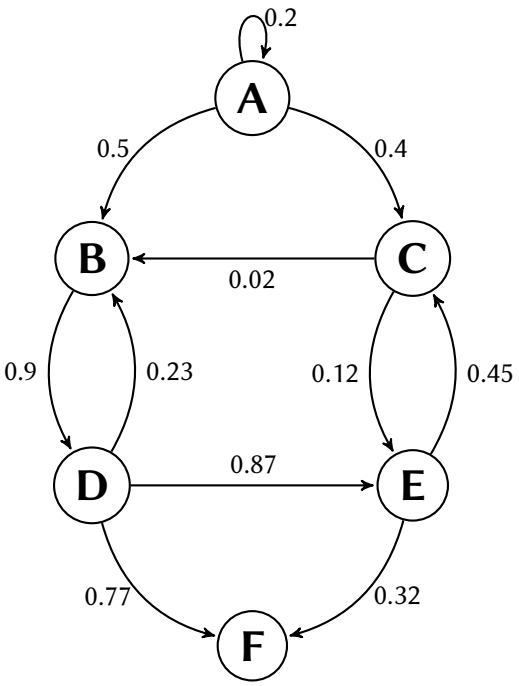

Abbildung 3.2.: Ein Beispiel für einen gerichteten Graphen

Kantengewichtete Graphen

Die in Abb. 3.2 enthaltenen Kanten verfügen darüber hinaus über sogenannte *Gewichte*. Graphen, deren Kanten gewichtet sind, ist jeweils eine reele Zahl als Kantengewicht zugeordnet, welche bei ungerichteten sowie gerichteten Graphen gleichermaßen auftreten können. Kantengewichte werden beispielsweise für die Modellierung von Beziehungen zwischen zwei Objekten genutzt. Das Kantengewicht kann dabei als eine Eigenschaft bezeichnet werden, die die Kante näher beschreibt. Damit lassen sich beispielsweise Distanzen zwischen zwei Knoten modellieren (Gumm und Sommer (2011, S.395-396)).

3.1.2. Wege und Zusammenhang

Als einen *Weg* bezeichnet man eine Folge von Knoten in einem Graphen, unter der Voraussetzung, dass es jeweils eine Kante zwischen den einzelnen Knoten des Weges gibt. Als ein *einfacher Weg* wird ein Weg von x nach y bezeichnet, sofern kein Knoten mehrfach darin enthalten ist. Ein *einfacher Weg* mit dem gleichen Start- und Endknoten x wird als *Zyklus* bezeichnet (Gumm und Sommer (2011, S.396)).

Ein ungerichteter Graph G wird als *zusammenhängend* bezeichnet, wenn es für jedes Knotenpaar, bestehend aus zwei verschiedenen Knoten, einen Weg gibt.

Ist ein ungerichteter Graph G zusammenhängend und zyklenfrei, heißt dieser *Baum*. Zu jedem *zusammenhängenden* Graphen gibt es einen erzeugenden *Baum*.

Algorithmen für die Berechnung kürzester Wege

In Graphen sind Algorithmen für die Berechnung von kürzesten Wegen zwischen Knoten für viele Aufgabenstellungen von hoher Bedeutung. In einem gewichteten Graphen wird dabei der *kürzeste* Weg zwischen einem Start- und Endknoten als der Pfad bezeichnet dessen aufsummierte Gewichte entlang seines Pfades, verglichen mit allen anderen möglichen Pfaden, am minimalsten sind.

Man kann das Problem der Berechnung kürzester Wege weiter unterscheiden. Wird der kürzeste Weg von einem Startknoten zu nur einem anderen Zielknoten gesucht, spricht man vom *Source Target Shortest Path Problem* (*STSP*). Werden hingegen alle kürzesten Wege von einem Startknoten zu allen anderen Knoten im Graphen gesucht, wird vom *Shortest Source Shortest Path Problem* (*SSSP*) gesprochen. Eine dritte Variante stellt die Suche nach den kürzesten Wegen für alle Knotenpaare dar, welche *All Pair Shortest Path Problem* (*APSP*) genannt wird (V. Turau (vgl. 1996, S.241-242)).

Es gibt einige Algorithmen, die sich für die Lösung eines oder mehrere der aufgeführten Problemstellungen eignen. Der *Dijkstra*-Algorithmus (Dijkstra et al. (1959)) ist für die Berechnung von kürzesten Wegen einer der bekanntesten Vertreter und löst das Problem für einen gegebenen Graphen $G = (V, E)$ in $\mathcal{O}(|V|log(|V|) + |E|)$. Der *Dijkstra*-Algorithmus wird auch in Abschnitt 3.3.1 zur Ermittlung kürzester Wege zwischen zwei Knoten verwendet (Needham und Hodler (2018)).

3.2. Datensätze

In diesem Kapitel werden die Datensätze, die im Rahmen dieser Arbeit genutzt worden sind, kurz vorgestellt.

3.2.1. DeepFashion2-Datensatz

Der in den Arbeiten von Ge et al. (2019) näher vorgestellte Datensatz beinhaltet 491.000 Bilder, auf denen 13 Kleidungsstücke enthalten sind. Die Bilder stammen von kommerziellen Shops oder Konsumenten. Zu jedem Bild gibt es eine korrespondierende Annotations-Datei (.json-Format).

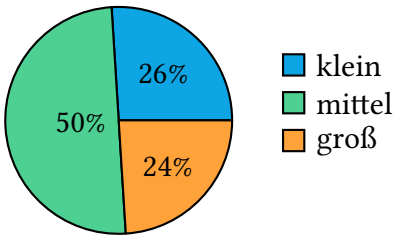

Abbildung 3.3.: Skalierungstufen in Anlehnung an Ge et al. (2019, S.4)

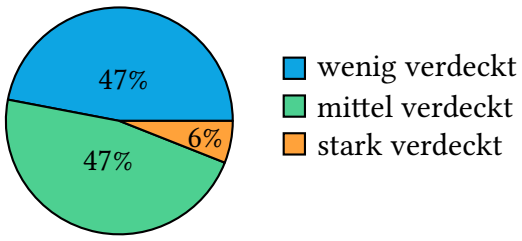

Abbildung 3.4.: Verdeckungsgrad in Anlehnung an Ge et al. (2019, S.4)

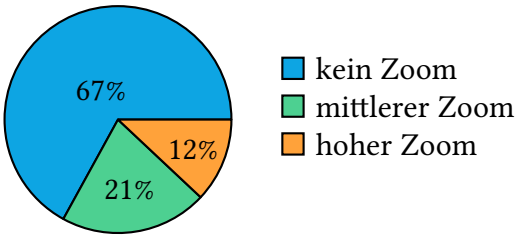

Abbildung 3.5.: Zoomstufen in Anlehnung an Ge et al. (2019, S.4)

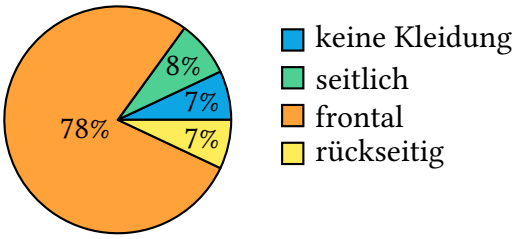

Abbildung 3.6.: Position in Anlehnung an Ge et al. (2019, S.4)

Die in der jeweiligen Annotations-Datei vorliegenden Daten beinhalten eine Kategorie-Zuweisung und einige weitere Daten (Bildabmessungen, Sichtbarkeit, Skalierung, Zoomstufe,...), die aber im Rahmen dieser Arbeit keine direkte Verwendung finden. Da diese Daten jedoch trotzdem eine gewisse Aussagekraft über die Qualität des Datensatzes haben, sind Statistiken über den Datensatz in Abb. 3.3, Abb. 3.4, Abb. 3.5 und Abb. 3.6 aufgeführt.

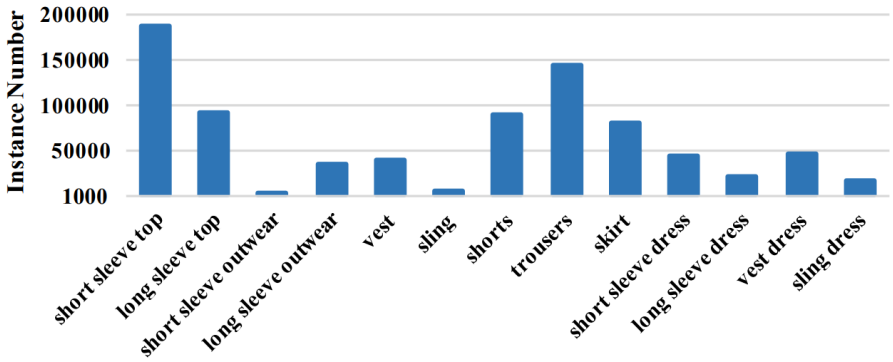

Abbildung 3.7.: Statistik über die Verteilung der 13 Kategorien im *DeepFashion2*-Datensatz (Ge et al. (2019, S.4))

Insgesamt wird im Datensatz zwischen 13 verschiedenen Kleidungsstücken unterschieden. In Abb. 3.7 sind die Kategorien aufgelistet und die Verteilung der Kategorien über alle im Datensatz enthaltenen Bilder aufgeführt.

3.2.2. Fashion Product Image Dataset

Ein weiterer Datensatz, mit dem in dieser Arbeit gearbeitet wurde, ist das *Fashion Product Image Dataset*, dass in den Arbeiten von Aggarwal (2019) vorgestellt wird.

Der Datensatz besteht aus 44.000 Bildern dem jeweils eine Annotations-Datei (.csv-Format) zugeordnet ist. In der Annotations-Datei sind jedem Bild Zusatzinformationen zugewiesen. Im Datensatz wird in den einzelnen Zuordnungen in 5 Geschlechter-Typen (Abb. 3.9), 7 Hauptkategorien (Abb. 3.10), 45 Subkategorien (Abb. 3.11), 143 Artikel-Typen (Abb. 3.8) und 47 Farben (Abb. 3.12) unterschieden. Alle aufgezählten Attribute wurden dabei manuell während der Katalogisierung der Bilder eingetragen.

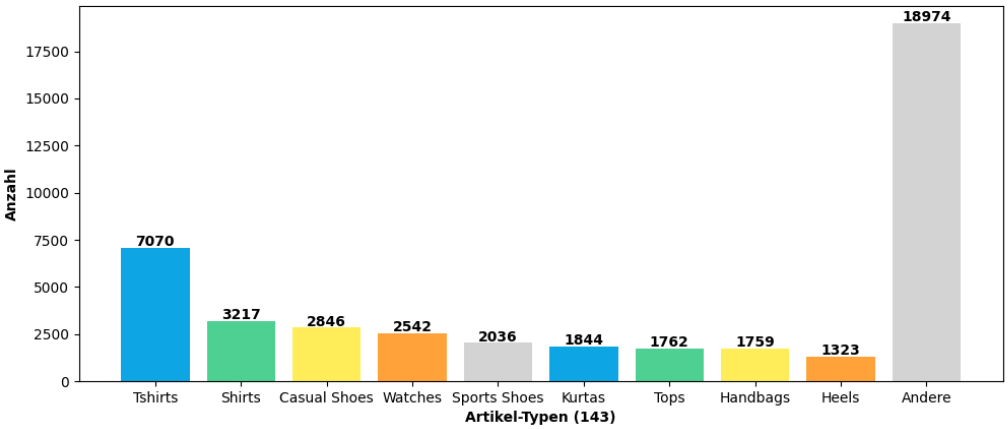

Abbildung 3.8.: Verteilung der Artikel-Typen

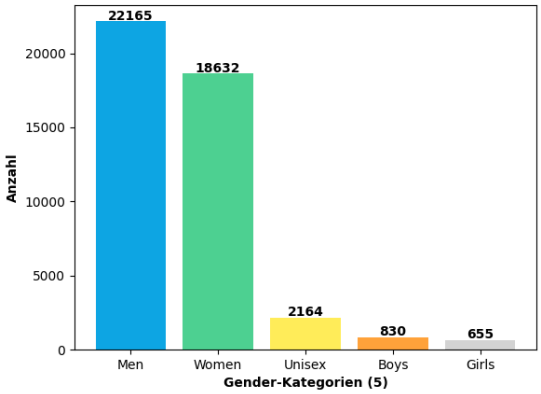

Abbildung 3.9.: Gender-Verteilung

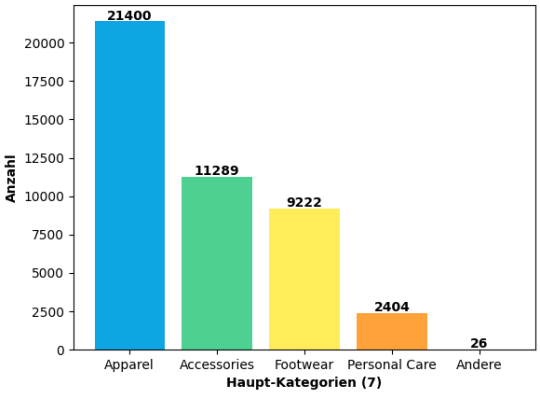

Abbildung 3.10.: Hauptkategorien-Verteilung

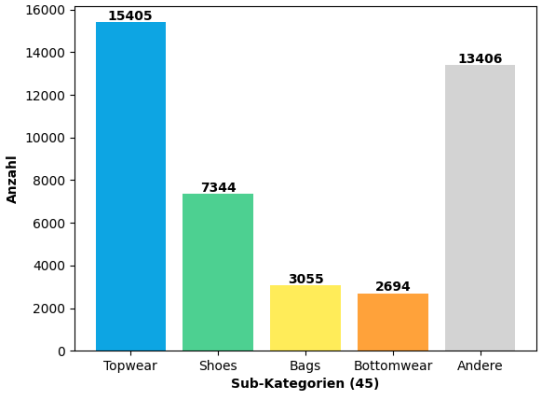

Abbildung 3.11.: Subkategorien-Verteilung

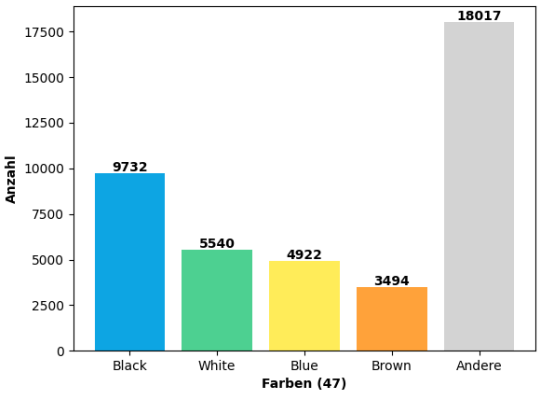

Abbildung 3.12.: Farb-Verteilung

3.2.3. Flickr Image dataset

Ein weiterer im Zuge dieser Arbeit verwendeter Datensatz ist das *Flickr Image dataset* (Sankesara (2018)). In den Arbeiten von Plummer et al. (2015) werden die einzelnen Verfahren, die für die Erstellung des Datensatzes genutzt worden sind, dargelegt.

Das *Flickr Image dataset* beinhaltet satzbasierte Bildbeschreibungen. Dabei sind 30.000 *FlickR*-Bilder insgesamt 158.000 Bildunterschriften zugeordnet. In Tabelle 3.1 sind die Daten zu einem Beispiel-Bild (Abb. 3.13) aufgeführt.

Abbildung 3.13.: Beispiel-Bild (ID:104835889) aus dem *FlickR Image dataset*

Tabelle 3.1.: Beispiel-Einträge aus dem *FlickR Image dataset*

Dateiname	**Kommentar-ID**	**Text**
104835889.jpg	0	*A samurai warrior in full black dress takes his sword from the sheath on an outdoor training mat.*
104835889.jpg	1	*A man in a black martial arts suit unsheathes a sword on a training mat in a field.*
104835889.jpg	2	*A guy with a sword and traditional fighting gear is getting ready for a fight.*
104835889.jpg	3	*A man in black wields a sword in front of rolling, green hills.*
104835889.jpg	4	*One man dressed in black doing karate in a green field.*

3.2.4. Fashionpedia

Anhand der Arbeiten von Jia et al. (2019) wird im Folgenden ein weiterer Datensatz vorgestellt. Beim *Fashionpedia*-Datensatz wurde auf Grundlage von 48.827 Bildern eine Ontologie[2] und Segmentierung von Kleidungsstücken abgebildet.

[2]Darstellung einer Menge von Objekten oder Begriffen in einer gegebenen Domäne D und der Relationen R in dieser Menge (Breitman et al. (vgl. 2007, S.17-19)).

Tabelle 3.2.: Oberkategorien und zugehörige Kleidungsstücke aus der Fashionpedia

Oberkategorie	**Anzahl Kleidungsstücke**
upperbody	6 (8)
lowerbody	3
wholebuddy	4
head	3 (5)
neck	1
arms and hands	2
waist	1
legs and feet	4 (5)
others	3
garment parts	7
closures	2
decoration	10

Die frei lizenzierten Bildern stammen dabei von verschiedenen Webseiten, wobei die Zusammenstellung der Bilder manuell von Modeexperten übernommen wurde. Über eine qualifizierte Gruppe an Nutzern wurde für jedes Bild eine Segmentierung manuell durchgeführt. Anschließend wurden die Bilder nochmals von einer Modeexperten-Gruppe mit detaillierten Attributs-Annotationen erweitert.

Insgesamt konnte dadurch in 46 Kleidungsstücke und weitere 92 Attribute unterschieden werden. Die Attribute wurden weiterführend in insgesamt 294 Sub-Attribute unterteilt. In Tabelle 3.2 sind die Hauptkategorie und die jeweilige Anzahl an zugeordneten Kleidungsstücken aufgelistet. Zu beachten ist, dass es zum Teil uneindeutige Bezeichnungen gibt. Ein Beispiel hierfür stellt der Eintrag „top, t-shirt, sweatshirt“ dar, der drei Kleidungsstücke zusammenfasst. Im Rahmen dieser Arbeit wurden diese mehrfachen Zuordnungen jeweils gesondert behandelt, sodass die Gesamtzuordnung statt 46 nun 51 Kleidungsstücke betrachtet. Die Neuzuordnung ist mit dem jeweiligen Wert in Klammern in Tabelle 3.2 dargestellt.

Neben dem Datensatz wurde in der Arbeit von Jia et al. (2019) auch ein Modell vorgestellt, welches für ein Bild eine Klassifizierung sowie Vorhersage für die Zuordnung zu Attributen durchführen kann. Das Modell basiert dabei auf einem *CNN*, das in Abschnitt 2.4.2 bereits erklärt wurde.

3.3. Softwarekomponenten

In diesem Kapitel werden die im Zusammenhang mit dieser Arbeit verwendeten Softwarekomponenten vorgestellt.

3.3.1. Neo4J

Bei Neo4J handelt es sich um eine Java-basierte *open-source graph database* die im Jahr 2007 veröffentlicht wurde. Mit Neo4J ist es möglich, vollständig transaktionale Datenbanken in Form von Graphen anstatt von Tabellen abzubilden. Die Software wird in zwei verschiedenen Versionen angeboten (*Community-, Enterprise*-Edition), die sich hinsichtlich ihrer Limitierungen für einzelne Teilbereiche und zusätzlichen Funktionen unterscheiden (Guia et al. (vgl. 2017, S.352)).

Graph-Datenbanken sind in den letzten Jahren zu einer guten Alternative gegenüber relationalen Datenbank-Management-Systemen (*RDBMS*) geworden. Während bei *RDBMS* die Speicherung, Bearbeitung und das Abrufen komplexer Daten vergleichsweise mühsam sind und durch die schemabasierten Datenmodelle wiederkehrende Anpassungen für neue Daten erfordern, sind Graphdatenbanken wie Neo4J hierfür flexibel einsetzbar. *RDBMS* sind für aggregierte Daten optimiert, Graphdatenbanken hingegen für stark vernetzte Daten (J. J. Miller (2013, vgl.)).

Bei Graphdatenbanken stehen insbesondere eine gute Lesbarkeit und Wartbarkeit im Fokus, auch wenn dies oftmals auf Kosten der Effizienz geht (Holzschuher und Peinl (2013)).

Es gibt viele Gründe, die für den Einsatz einer Graphdatenbank sprechen, besonders mit steigender Skalierung und größerer Anzahl an Relationen. Beispielsweise stellen die Optimierung hinsichtlich der Datenzusammenstellung, die leichte Erweiterbarkeit, der Support der *ACID*-Rules[3] und keine Notwendigkeit für eine feste Datentyp-Zuweisung einige dieser Gründe dar. Zusammengefasst lassen sich diese Gründe mit den Eigenschaften Performance, Flexibilität und Agilität zusammenfassen (Fernandes und Bernardino (vgl. 2018, S.2)).

Verglichen mit anderen Graphdatenbanken (*ArangoDB, OrientDB, AllegroGraph, InfiniteGraph*) schneidet Neo4J sehr gut ab. Besonders die eigene Abfrage-Sprache *CYPHER* macht es performant und gut einsetzbar, wie aus den Arbeiten von Francis et al. (2018) hervorgeht. Ein typischer Aufbau eines Graphen in Neo4J ist in Abb. 3.14 aufgeführt.

[3] ACID-Regeln steht für *atomicity, consistency, isolation* und *durability*, die für gewünschte Eigenschaften in DBMS-Systemen stehen und diese beschreiben. Atomität (*atomicity*): Entsteht ein Fehler bei einer Transaktion, bleibt der Datenbankstatus unverändert. Konsistenz (*consistency*): Jede Transaktion behält einen konsistenten Zustand der Datenbank bei. Isolation (*isolation*): Während einer Transaktion dürfen keine anderen Operationen auf die Daten zugreifen. Dauerhaftigkeit (*durability*): Das *DBMS* kann stets die Ergebnisse einer bestätigten Transaktion wiederherstellen (Dobler und Slopianka (1993)).

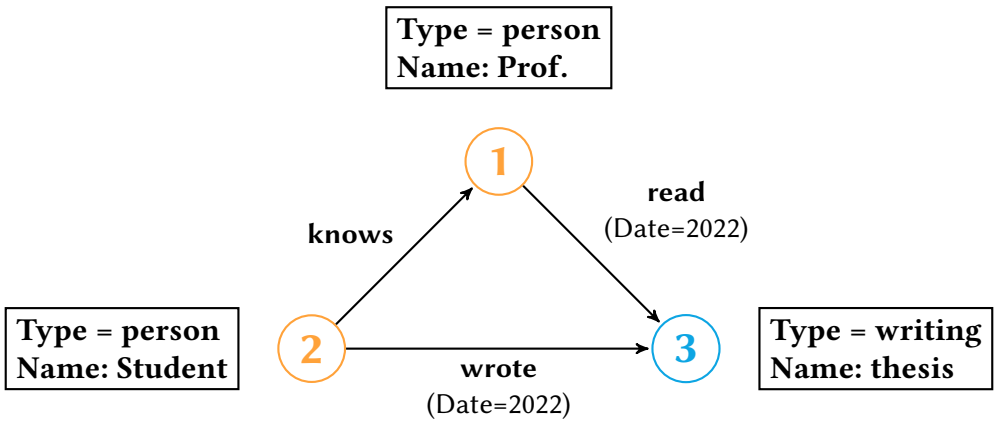

Abbildung 3.14.: Typischer Aufbau eines Graphen in Neo4J in Anlehnung an J. J. Miller (2013, S.141)

Das Grundkonzept in Neo4J (Abb. 3.15) besteht aus Knoten und Relationen. Knoten werden durch Relationen organisiert und können eigene Eigenschaften besitzen. Relationen wiederum können ebenfalls Eigenschaften besitzen, die eine detailliertere Beschreibung zulassen (J. J. Miller (vgl. 2013, S.142)). Mit den Eigenschaften von Relationen lassen sich beispielsweise auch Kantengewichte modellieren.

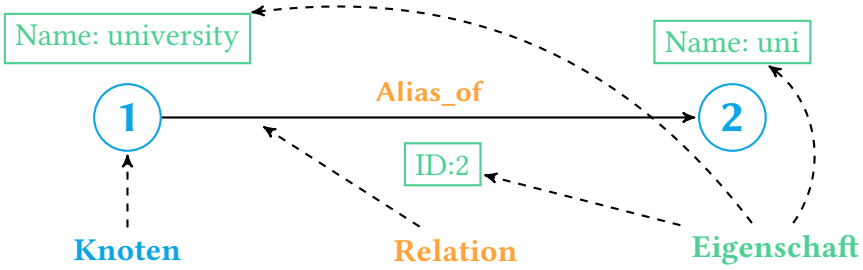

Abbildung 3.15.: Grundkonzept in Neo4J in Anlehnung an J. J. Miller (2013, S.142)

3.4. Weitere Ressourcen

Im Folgenden werden weitere Ressourcen vorgestellt, die im Rahmen dieser Arbeit verwendet wurden, sich thematisch jedoch nicht in die bisherigen Abschnitte eingliedern lassen.

3.4.1. wikidata

Die folgenden Ausführungen stammen aus den Arbeiten von Farda-Sarbas und Müller-Birn (2019). WIKIDATA ist eine frei bearbeitbare Wissensdatenbank, die strukturierte Daten bereithält. Als Schwesterprojekt von Wikipedia beinhaltet WIKIDATA für Menschen, aber auch für Maschinen lesbare Daten.

Das Projekt wurde im Jahr 2012 gestartet und wird seitdem von tausenden Benutzern gepflegt und erweitert. Ziel des Projekts ist es die Dateninkonsistenz aufgrund der verschiedenen Sprachversionen zu kompensieren. Die Daten werden in einer strukturierten Form (RDF-Format) gespeichert. Diese lassen sich über die graphenbasierte Abfragesprache *SPARQL* abrufen.

Nach Veen (2019), wird jedem Eintrag in WIKIDATA ein *Identifier* zugewiesen sowie auch jeder möglichen Relation zu anderen Einträgen. Mithilfe dieser *Identifier* ist es möglich, von einem Eintrag zu einem anderen zu gelangen. Zusätzlich können komplexe Abfragen geteilt und einfacher wiederverwendet werden. Es gibt zahlreiche Schnittstellen für eine Reihe von Programmiersprachen, um auf die in WIKIDATA enthaltenen Daten zugreifen zu können.

3.4.2. WordNet

WORDNET ist ein von der Princeton University entwickelte strukturierte lexikalisch-semantische Ressource für die englische Sprache, welche nun kurz nach den Ausführungen von George A Miller et al. (1990) und George A. Miller (1995) beschrieben werden. In dieser Ressource, dass auch als eine Art Wörterbuch verstanden werden kann, werden Wörter mit der gleichen semantischen Bedeutung (Synonyme) in *Synsets* zusammengefasst. Ein *Synset* organisiert Synonyme in einem Konzept. Jedem *Synset* sind weitere Relationen zugeordnet. Neben einer Kurzdefinition sind verschiedene Relationen darin aufgeführt.

WORDNET beinhaltet ca. 117.000 einzelne *Synsets*. Es wird unterschieden in Nomen, Verben, Adjektive, Adverbien und Funktionswörter. Nomen sind in WORDNET hierarchisch strukturiert. Verben sind durch Vererbungsrelationen organisiert. Adjektive und Adverbien hingegen sind als N-dimensionale Hyperräume organisiert. Als semantische Relationen zwischen den einzelnen *Synsets* können Synonymie[4], Meronymie[5], Hyponymie[6] und Antonymie[7] vorkommen.

[4]Wörter gelten als Synonyme, wenn sie ohne Kontextverlust durch ein anderes Wort ersetzt werden können (George A Miller et al. (1990, S.241)).

[5]Meronym-Relationen beschreiben Teil-Ganzes-Beziehungen (George A Miller et al. (1990, S.242-243)).

[6]Hyponymie-Relationen beschrieben das Verhältnis von Oberbegriffen zu einem gegebenen Wort (George A Miller et al. (1990, S.242)).

[7]Antonomyie-Relationen beschreiben Wörter, die eine gegensätzliche Bedeutung haben (George A Miller et al. (1990, S.242)).

4. Algorithmische Konzeption

In diesem Kapitel wird die konzeptionelle Vorgehensweise, die zur Bearbeitung der Aufgabenstellung durchlaufen wurde, beschrieben und erklärt. Zunächst wird die Bestimmung der einzelnen Faktoren besprochen, die Kleidung maßgeblich beschreiben und voneinander unterscheiden. Die bestimmten Faktoren werden anschließend in die weitere Bearbeitung miteinbezogen.

Zunächst wird auf die ersten Überlegungen und Ansätze (Abschnitt 4.2) eingegangen, die umgesetzt wurden und auf deren Erkenntnisse das weitere Vorgehen aufbaut. Anschließend wird der Aufbau einer Graphdatenbank in Neo4J als zentrale Ressource detailliert erklärt (Abschnitt 4.3), da hierauf aufbauend alle weiteren Arbeiten basieren. In dem darauffolgenden Abschnitt 4.4 wird der eigentliche Algorithmus für die Detektion von Kleidungsstücken in Texten und die darin enthaltenen Schritte vorgestellt. Schließlich werden in Abschnitt 4.5 Ansätze erläutert, die verworfen wurden.

4.1. Bestimmung der Einflussfaktoren für die Kleidungsdetektion

Für die Modellierung von Bekleidung mussten zunächst die Faktoren bestimmt werden, die einen Einfluss auf die Wahrnehmung für den Betrachter haben. Hierbei gibt es offensichtliche Faktoren wie die Größe der Kleidung oder die Farbe einer Kleidung, weit weniger offensichtlich sind beispielsweise Faktoren wie der Kleidungsstil oder die Zuordnung zu einer Altersgruppe oder einem Geschlecht.

Im Falle der Geschlechterzuordnung kann entweder binär (männlich, weiblich) oder noch detaillierter mit der Unterscheidung für Kinder- bzw. Erwachsenen-Bekleidung und geschlechtsneutrale Bekleidung (unisex) unterschieden werden. Eine weitere wichtige Unterscheidung kann in der Körperregion getroffen werden, für die die jeweilige Kleidung üblicherweise genutzt wird. Auch hier sind von groben Einteilungen (Kopf, Oberkörper, Unterkörper und Füße) bis hin zu deutlich detaillierten Unterteilungen viele Möglichkeiten valide.

Neben den bereits genannten Faktoren gibt es zahlreiche weitere Möglichkeiten, Unterscheidungen zu treffen, da insbesondere in der Kleidungsmode die Diversität und die Gestaltungsmöglichkeiten sehr vielseitig sind. Für Kleidung kann beispielsweise auch eine Unterscheidung in der Stilrichtung (modern, sportlich, formell u.w.) oder bezüglich des anvisierten Nutzungszwecks (Alltagskleidung, Arbeitskleidung, Schutzkleidung, Sportkleidung u.w) getroffen werden.

Darüber hinaus kann Kleidung auch kulturell oder religiös beeinflusst sein und sich dahingehend voneinander unterscheiden. Ein nicht unwesentlicher Punkt stellen Rollen dar, die

der Mensch automatisch mit bestimmten Kleidungsstilen assoziiert. Beispielsweise wird ein Anwalt üblicherweise in einem ordentlichen Anzug und ein Richter in einer Robe gesehen (Lehnert (vgl. 2014, S.7-13)).

Da im Rahmen dieser Arbeit die Einbeziehung all dieser Faktoren nicht möglich war, wurde eine Auswahl der Faktoren auf wesentliche Unterscheidungsmerkmale begrenzt. Diese stellen die Zuordnung zu einem Geschlecht und Alter sowie die Farbunterscheidung dar. Ausgenommen ist hier die Unterscheidung bei der Kleidungsgröße, da dies bei der Modellierung eine untergeordnete Rolle spielt und sich verhältnismäßig leicht durch eine entsprechende Skalierung ausdrücken lässt.

4.2. Mehrklassen-Textklassifizierung

Als erster Ansatz wurde auf Grundlage des in Abschnitt 3.2.2 vorgestellten Datensatzes *Fashion Product Image Dataset* ein bereits vortrainiertes BERT-Modell auf ein Klassifizierungsproblem mit mehrere Klassen abgestimmt.

Für das Training wurden die Bild-Beschreibungstexte und die Zuordnungen zu jeweils einem Artikel-Typen aus dem Datensatz extrahiert und gesondert abgespeichert. Über die Daten war es möglich, das BERT-Modell auf diese Klassifizierungsaufgabe hin zu trainieren. Hierbei stellt eine Klasse jeweils einen Artikel-Typen aus dem gesamten Datensatz dar (Abb. 3.8).

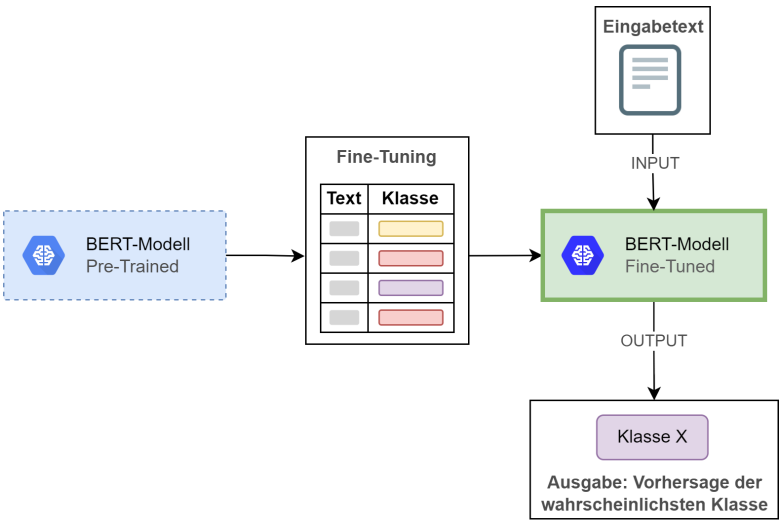

Abbildung 4.1.: Schematische Darstellung des *Fine-Tunings* des BERT-Modells basierend auf den Daten aus dem *Fashion Product Image Dataset (Abschnitt 3.2.2)*

Das über die Beschreibungstexte feiner abgestimmte Modell zur Klassifizierung in mehrere Klassen konnte anschließend genutzt werden, um für neue Sätze vorherzusagen, welcher Artikel-Typ beschrieben sein könnte.

Dieser Ansatz klassifiziert einen gegebenen Text in seiner Gesamtheit, sodass nicht festgestellt werden kann, ob mehrere Kleidungsstücke beschrieben werden. Es ist darüber hinaus auch nicht möglich festzustellen, wie viele und welche Kleidungsstücke in einem Text beschrieben werden.

Das konzeptionelle Vorgehen wird in Abb. 4.1 schematisch dargestellt. Die technische Realisierung dieses Vorgehens wird in Abschnitt 5.1 beschrieben. Die Ergebnisse des Ansatzes werden in Abschnitt 6.1 aufgeführt und besprochen.

4.3. Aufbau der Graphdatenbank in Neo4J

Neben der Berechnung eines eigenen BERT-Modells, dass für einen gegebenen Beschreibungstext ausgibt, welches Kleidungsstück am wahrscheinlichsten beschrieben wird, wurde im Rahmen dieser Arbeit eine Ressource aufgebaut, die Informationen über Kleidungsstücke und ihre Relationen untereinander beinhaltet. Auf Grundlage dieser Ressource sollen weitere Berechnungen aufbauen und Ergebnismengen eingeschränkt werden können, um die Ergebnisse zu verbessern. Der genaue Aufbau und die darin enthaltenen einzelnen Schritte werden in diesem Unterkapitel vorgestellt.

Wie bereits in Abschnitt 3.1 beschrieben, besteht ein Graph aus Knoten und Kanten, wobei die Kanten entsprechend die Relation zwischen den einzelnen Knoten beschreiben. Im Rahmen dieser Arbeit steht jeder Knoten für ein Wort. Kanten beschreiben Relationen zwischen den einzelnen Wörtern. Die Relationen wurden teilweise aus externen Ressourcen abgefragt oder eigens auf Grundlage verschiedener Modelle berechnet. Der erstellte Graph wurde in Neo4J (Abschnitt 3.3) erstellt und als Datenbank verfügbar gehalten.

Die für den Aufbau der Graphdatenbank programmierten Python-Skripte und einzelnen Vorverarbeitungsschritte werden nur konzeptionell in diesem Kapitel vorgestellt und werden in Kapitel 5 nicht genauer besprochen, da die wesentlichen Punkte durch die konzeptionelle Erklärung bereits abgedeckt wird. Alle verwendeten und geschriebenen Python-Skripte sowie relevanten *csv*-Dateien sind in Anhang B in einer Ordnerstruktur aufgeführt. Die Dateien sind entsprechend dieser Arbeit auf einem physischen Datenträger beigelegt oder auch über *Hessenbox* erreichbar. Außerdem ist in jedem Unterordner, der für einen wesentlichen Schritt in der Bearbeitungsreihenfolge steht, eine *readMe.txt* beigelegt, in der die relevanten Schritte zur Erstellung des Graphen beschrieben werden. Darüber hinaus ist auch eine Datenbank-Sicherung (*dump*) verfügbar, die direkt in Neo4J importiert werden kann und den gesamten Graphen beinhaltet.

4.3.1. Generierung einer Ausgangsmenge von Knoten

In einem ersten Schritt wurde eine Menge an Wörter aus der Kleidungsdomäne benötigt, die als Ausgangspunkt für alle folgenden Erweiterungen fungiert. Hierfür wurden aus dem *Fashion Product Image Dataset* (Abschnitt 3.2.2) alle 143 verfügbaren *Artikel-Types* verwendet,

die jeweils einzelne Kleidungsstücke repräsentieren.

Da im Datensatz zu diesen Kleidungsstücken weitere Informationen über die Zuordnung zu Altersgruppen, Geschlechtern, Haupt- und Subkategorien vorhanden sind, wurden diese Informationen in die Graphdatenbank als Eigenschaften der Knoten mit eingerechnet. Die Zusatzinformationen wurden über alle Einträge der Datenbank gesammelt und gemittelt den einzelnen Kleidungsstücken zugeordnet.

Da es sich hierbei um eine reine Datenerfassung handelt, wird das Vorgehen nicht weiterführend und detailliert erklärt.

4.3.2. Erweiterung der Grundmenge von Knoten und Kanten über WordNet

Nachdem in Abschnitt 4.3.1 eine Menge an Wörtern als Ausgangsmenge definiert wurde, wurden anschließend über WordNet (Abschnitt 3.4.2) alle Unterbegriffe durch die in WordNet eingetragenen *Hyponymie*-Relationen für jedes Wort aus der Ausgangsmenge zusammengetragen. Dieses Vorgehen wurde dreifach wiederholt, sodass für jeden der bereits erhaltenen *Hyponyme* nochmals die *Hyponyme* und von diesen ebenfalls die *Hyponyme* berechnet wurden. Somit konnte bereits eine Erweiterung der Ausgangsmenge mit weiteren relevanten Begriffen durchgeführt werden.

Algorithmus 1 : Berechnung der Begriffsmenge über *Hyponomie*-, *Hypernomie*- und *Meronymie*-Relationen

Input : Menge an Kleidungsrelevanten Begriffen
Output : Begriffsmenge und Relationen über Hypo-, Hypernomie sowie Meronymie

```
1  words = [w_0, w_1, w_2, ...w_n] // Alle Wörter aus der Anfangsmenge
2  relations = ∅
3  for i ∈ words do
4      h = wordNet.get_hyponyms(i) // Frage Hyponyme für i ab
5      words.append([h_0, h_1, ..., h_n])
6      relations.append([(i, h_0), (i, h_1), ..., (i, h_n)])
7  for j ∈ words do
8      hy = wordNet.get_hypernyms(i) // Frage Hypernyme für i ab
9      words.append([hy_0, hy_1, ..., hy_n])
10     relations.append([(j, hy_0), (j, hy_1), ..., (j, hy_n)])
11 for k ∈ words do
12     m = wordNet.get_meronyms(i) // Frage Meronyme für i ab
13     words.append([m_0, m_1, ..., m_n])
14     relations.append([(k, m_0), (k, m_1), ..., (k, m_n)])
15 return words und relations
```

Aus der *Hyponomie*-Relation lässt sich aufgrund der unterschiedlichen Begriffsumfänge von

Unterbegriffen (*Hyponyme*) und Obergriffen (*Hypernyme*) jedoch nicht ableiten, dass sich diese immer gegensätzlich exakt gleich verhalten. Beispielsweise ist jeder Fußballer ein Sportler, aber nicht jeder Sportler ist auch ein Fußballer. Entsprechend wurden anschließend für jeden der durch die dreistufige *Hyponymie*-Relationen generierten Begriffe die jeweilige Menge an verbundenen *Hypernymen* berechnet.

Schließlich wurden zusätzlich für jeden Knoten alle Begriffe abgefragt, die in einer *Meronymie*-Relation zum jeweiligen Knoten stehen und der Begriffsmenge hinzugefügt. Da Kleidungsstücke sich häufig aus mehreren Bestandteilen zusammensetzen und die *Meronym*-Relation genau diese *Teil-Ganzes*-Relation abbildet, wurden über die Menge an bis dahin generierten Begriffen alle in Relation stehenden Begriffe über WordNet abgefragt. Die nach diesem Vorgehen zusammengetragenen Begriffe ergeben die Basis für alle weiteren Berechnungen.

In Algorithmus 1 ist in Pseudocode das Vorgehen für die Generierung der zusätzlichen Wörtern über die entsprechenden Relationen beschrieben. Dabei sind in $words$ alle Kleidungsstücke aus dem *Fashion Product Image Dataset* als Ausgangsmenge enthalten. In $relations$ werden alle Relationen zwischen einem Starknoten (Wort) und seinem Endknoten (Wort) den berechneten Relationen entsprechend gespeichert. Die Menge an Wörter ($words$) sowie die Relationen ($relations$) werden durch die darunter aufgeführten Abfragen mit den jeweiligen *Hyponymen*, *Hypernymen* und Meronymen erweitert.

4.3.3. Erweiterung der Knotenmenge über Alias-Relationen in wikidata

Da Begriffe oftmals Alias-Bezeichnungen besitzen und diese ebenfalls abgebildet werden sollen, wurde über wikidata zu jedem Begriff aus der Graphdatenbank die Menge an Alias-Namen und damit einhergehend die Relation zwischen dem jeweiligen Wort und seinen Alias-Bezeichnungen abgefragt und in die Graphdatenbank aufgenommen. In Algorithmus 2 ist in Pseudocode das Vorgehen für die Abfrage der Alias-Bezeichnungen aufgeführt.

Algorithmus 2 : Abfrage der Alias-Bezeichnungen für jeden Knoten über wikidata

Input : Knotenmenge aus der Graphdatenbank
Output : Alias-Namen zu jedem Knoten und Alias-Relationen

```
1 nodes = [n_0, n_1, ..., n_n] // Alle Knoten aus dem Graphen
2 alias_nodes = ∅
3 alias_relations = ∅
4 for i ∈ nodes do
5     a = wikidata.get_alias(i) // Frage Wikidata-Alias für i ab
6     alias_nodes.append([a_0, a_1, ..., a_n])
7     alias_relations.append([(n_i, a_0), (n_i, a_1), ..., (n_i, a_n)])
8 return alias_nodes, alias_relations
```

Die Variable $nodes$ beinhaltet alle Wörter, die durch den vorherigen ersten Verarbeitungs-

schritt mit Hilfe von WordNet generiert wurden. Über diese Menge wird für jedes Element die Menge an Alias-Bezeichnungen abgefragt und diese in $alias\_nodes$ gespeichert. Die entsprechenden Relationen zwischen Wort und Alias-Bezeichnungen werden in $alias\_relations$ gespeichert.

Die Daten-Anfragen an wikidata wurden mithilfe der Python-Bibliothek *pywikibot*[1] umgesetzt. Hierdurch können Anfragen in die von wikidata verwendete Abfragesprache *SPARQL* übersetzt und leichter verarbeitet werden.

4.3.4. Farberweiterung und POS-Tagging

Da Farben häufig in Zusammenhang mit Kleidungsstücken genannt werden und in diesem Fall genügend Datenressourcen verfügbar sind, wurden über eine Liste von Farbbezeichnungen[2], die auf Grundlage des Wikipedia-Artikels (*List of Colors*[3]) zusammengestellt wurde, diese Bezeichnungen in die Graphdatenbank mitaufgenommen.

Außerdem wurden die *Part-of-Speech*-Zuordnungen (Abschnitt 2.1) für jeden Knoten über die *nltk*-Bibliothek[4] abgefragt und anschließend als Knoten-Eigenschaft jedem Knoten hinzugefügt. Der Algorithmus hierzu wird nicht weiter ausgeführt, da sich dieser lediglich auf einfache Funktionsaufrufe und Listenoperationen beschränkt.

4.3.5. Einbeziehung von Daten zur Ontologie von Kleidungsstücken

Der in Abschnitt 3.2.4 vorgestellte *Fashionpedia*-Datensatz beinhaltet eine Zuordnung von insgesamt 46 Artikel-Typen zu sogenannten *Superkategorien*. Diese lassen sich als eine Zuordnung von Kleidungsstücken bzw. Artikel-Typen zu einzelnen Körperregionen verstehen. Beispielsweise wird ein „shirt“ der Superkategorie *upperbody* zugeordnet.

Diese Daten wurden durch eine einfache Datenerfassung über eine *.json*-Datei, zusammengetragen und in die Graphdatenbank eingefügt. Zu Beachten ist an dieser Stelle, dass nicht alle Artikel-Typen aus der initial zusammengestellten Menge an Begriffen rund um Kleidungsstücke einen entsprechenden Eintrag in dem *Fashionpedia*-Datensatz besitzen, sodass nur eine Teilmenge der als Kleidungsstücke deklarierten Knoten in der Graphdatenbank diese gesammelten Informationen enthalten.

Nach der Datenerfassung wurde für jede *Superkategorie* ein gesonderter Knoten in der Graphdatenbank angelegt und jeder *Article-Typ* (Kleidungsstück) mit diesen gemäß der erfassten Daten verbunden.

[1] https://github.com/wikimedia/pywikibot, Abrudatum: 27.02.2022 17:26Uhr.
[2] https://github.com/codebrainz/color-names/tree/master/output, Abrufdatum: 27.02.2022 17:30Uhr.
[3] http://en.wikipedia.org/wiki/List_of_colors, Abrudatum: 27.02.2022 17:31Uhr.
[4] https://www.nltk.org/, Abrufdatum: 27.02.2022 18:15Uhr.

4.3.6. Berechnung der n-ähnlichsten Worte über statische Modelle

Nach der Erstellung der Grundmenge an Begriffen bzw. Knoten wurden mit Hilfe der in Abschnitt 2.7 beschriebenen statischen Modelle Ähnlichkeiten zwischen den einzelnen Begriffen berechnet und jeweils die $n = 5$ ähnlichsten Begriffe zu einem Ausgangsbegriff gespeichert und über eine entsprechende Relation in die Graphdatenbank aufgenommen.

Hierbei wurde die Ähnlichkeit zweier Vektoren bei allen drei verwendeten Modellen über die Kosinus-Ähnlichkeit (Gleichung (2.6)) berechnet. In Algorithmus 3 ist der Algorithmus in Pseudocode für die Berechnung der $n = 5$ ähnlichsten Begriffe über die jeweilige Vektorrepräsentation aufgeführt.

Algorithmus 3 : Berechnung der n-ähnlichsten Knoten nach den Modellen Word2Vec, GLove und FastText für jeden Knoten aus der Graphdatenbank

Input : Vektormodell (Word2Vec oder GLove oder FastText) und Menge an Eingabeknoten

Output : Menge der n-ähnlichsten Knoten nach dem gewählten Modell für jeden Eingabeknoten

```
1  nodes = [n_0, n_1, ...n_n] // Alle Knoten aus dem Graphen
2  nodes_vectors = ∅ // Vektorrepräsentationen jedes Knoten (Wortes)
3  for i ∈ nodes do
4      nodes_vectors.append(model.get_vector(n_i))
5  top_n_nodes = ∅
6  top_n_relations = ∅
7  for j ∈ nodes_vectors do
8      node_distances = ∅
9      for k ∈ nodes_vectors ≠ j do
10         distance = cos_sim(n_j→, n_k→) // Berechne Kosinus-Ähnlichkeit
11         node_distances.append(distance)
12     top_n_nodes.append(nearest_n(node_distances))
13     top_n_relations.append(nearest_n(n_j→, n_y→))
14 return top_n_nodes, top_n_relations
```

In Algorithmus 3 werden zunächst alle Knoten (Wörter) aus der Graphdatenbank geladen und in $nodes$ gespeichert. Anschließend wird für jeden Knoten n_x die Vektorrepräsentation aus dem jeweiligen Modell geladen. Anschließend wird für jeden Wort-Vektor mithilfe der Kosinus-Ähnlichkeit (Gleichung (2.6)) die Ähnlichkeit mit jedem anderen Wort-Vektor berechnet und die $n = 5$ ähnlichsten Wort-Vektoren sowie die dazu gehörende Relation in top_n_nodes bzw. $top_n\_relations$ gespeichert.

In diesem Schritt wurden drei verschiedene Modelle (Word2Vec, GLove und fastText) verwendet, wobei sich der grundlegende Algorithmus nur hinsichtlich der verwendeten Vek-

torrepräsentationen unterscheidet. Hierbei wurde für WORD2VEC das vortrainierte Modell *word2vec-google-news-300*[5] verwendet, dass drei Millionen Vektorrepräsentationen umfasst. Für GLOVE wurde hingegen das vortrainierte Modell *glove-wiki-gigaword-50*[6] verwendet, das auf der Grundlage von Wikipedia-Artikeln trainiert wurde. Schließlich wurde für FASTTEXT das vortrainierte Modell *wiki-news-300d-1M*[7] verwendet, das unter anderem auf Wikipedia-Artikeln aus dem Jahr 2017 trainiert wurde und 16 Milliarden Token umfasst.

Um die Ergebnisse der drei zuvor besprochenen statischen Modelle bezüglich der Ähnlichkeit von Begriffen zusammenzufassen und auf einen einzelnen Wert zu reduzieren, wurde der Algorithmus (Algorithmus 3) auf die 100 ähnlichsten Vektoren für jedes Modell erweitert. Dadurch sollten die unterschiedlichen Ergebnisse aus den verschiedenen Modellen zu einem höheren Maß berücksichtigt werden. Anschließend wurden die Ergebnisse der drei Modelle miteinander verrechnet und ein Durchschnittswert über alle Modelle berechnet, welche in einem letzten Schritt wiederum auf die n-ähnlichsten Wortpaare reduziert wurden.

Somit konnte die Ähnlichkeit von Wörtern auf Basis von einzelnen statischen Modellen auf einen kombinierten Gesamtwert über alle drei Modelle erweitert werden. Der Algorithmus für die Berechnung der n-ähnlichsten Worte über alle drei Modelle wurde durch den in Pseudocode dargestellten Algorithmus 4 umgesetzt.

Algorithmus 4 : Berechnung der n-ähnlichsten Knoten als kombinierten Wert über alle statischen Modelle

Input : 100-ähnlichste Worte aller drei statischen Modelle
Output : Menge der n-ähnlichsten Knoten über die Komibination der statischen Modelle

$top100\_word2vec = [[n_0, [w(x_0), ..., w(x_{100})], ..., [n_n, [w(x_0), w(x_1), ..., w(x_{100})]]]]$
$top100\_glove = [[n_0, [g(x_0), ..., g(x_{100})], ..., [n_n, [g(x_0), g(x_1), ..., w(g_{100})]]]]$
$top100\_fasttext = [[n_0, [f(x_0), ..., f(x_{100})], ..., [n_n, [f(x_0), f(x_1), ..., f(x_{100})]]]]$
$nodes = [n_0, n_1, ..., n_n]$
$top_n\_combined = \emptyset, top_n\_combined\_relations = \emptyset$
for $i \in nodes$ **do**
    $all\_values = \emptyset$
    $all\_values.append(sum(top100\_word2vec, top100\_glove, top100\_fasttext)/3)$
    $top_n\_combined.append(top_n(all\_values))$
    $top_n\_combined\_relations.append(top_n(n_i, n_x))$
return $top_n\_combined, top_n\_combined\_relations$

Der Algorithmus lädt zunächst die 100 ähnlichsten Wörter für jeden Knoten (Wort) aus den drei Modellen (WORD2VEC,GLOVE und FASTTEXT) und speichert diese in den Variablen

[5] https://huggingface.co/fse/word2vec-google-news-300, Abrufdatum: 13.03.2022 12:35Uhr.
[6] https://nlp.stanford.edu/projects/glove/, Abrufdatum: 13.03.2022 12:37Uhr.
[7] https://fasttext.cc/docs/en/english-vectors.html, Abrufdatum: 13.03.2022 12:40Uhr.

$top100\_word2vec$, $top100\_glove$ und $top100\_fasttext$.

Außerdem werden alle Knoten aus dem Graphen geladen und in der Variable $nodes$ gespeichert. Daraufhin wird über jedes Element aus $nodes$ iteriert, die drei entsprechenden Werte für die Ähnlichkeit nach den jeweiligen Modellen summiert und anschließend durch drei geteilt. Dadurch ist ein Durchschnittswert über alle Modelle berechnet worden. Die entsprechend $n = 5$ ähnlichsten Knoten bzw. die Relation zwischen einem Startknoten und den $n = 5$ ähnlichsten Worten werden abschließend in $top_n\_combined$ respektive $top_n\_combined\_relations$ gespeichert und in eine .csv-Datei geschrieben.

4.3.7. Berechnung der n-ähnlichsten Worte über BertMaskFlask und IDF

In Abschnitt 4.3.6 wurden bereits die n-ähnlichsten Worte zu jedem Wort über statische Modelle berechnet und anschließend in der Graphdatenbank über eine Relation verbunden. Da im Rahmen dieser Arbeit besonders Methoden aus dem *Machine Learning* (Abschnitt 2.3) verwendet und damit einhergehend deren Besonderheiten miteinbezogen werden sollten, wurden im nächsten Schritt die jeweils $n = 5$ ähnlichsten Worte basierend auf dem BERT-Modell berechnet.

Dabei wurde die bereits in Abschnitt 2.6.2 vorgestellte Aufgabe des *Masked Language Modeling* in einer erweiterten Variante genutzt. Als Datenbasis wurden die in dem *Flickr*-Datensatz vorliegenden Texte genutzt und jeweils geprüft, ob der Text zwei Wörter, die auch in der Graphdatenbank vorhanden sind, enthält. In der resultierenden Teilmenge der Gesamttexte wurde anschließend jeweils eines der Wörter maskiert und darüber hinaus eine weitere Version, in der beide Wörter maskiert werden, erstellt. Ein Beispiel soll an dieser Stelle die Vorverarbeitung der Texte zeigen. Durch diesen Prozess wurden insgesamt 50.000 vorverarbeitet und an das BERT-Modell übergeben.

Orginalsatz: „Socks and shoes are like jeans and trousers.“
Knoten in Graphdatenbank: „jeans“, „trousers “

Einfach maskierter Satz (**A**): „Socks and shoes are like jeans and [MASK].“
Umgekehrt einfach maskierter Satz (**B**): „Socks and shoes are like [MASK] and trousers.“
Zweifach maskierter Satz (**C**): „Socks and shoes are like [MASK] and [MASK].“

Resultierender Datensatz I: [”jeans”, A, C]
Resultierender Datensatz II: [”trousers”, B, C]

Aus der Vorverarbeitung können aus einem Eingabesatz gleich zwei für die Weiterverarbeitung nutzbare Datensätze erstellt werden, indem die einfache Maskierung in zwei Varianten durchgeführt wird und jeweils den zwei Zielwörtern aus dem Eingabesatz zugeordnet werden.

Die derart generierten Sätze wurden anschließend an den BertMaskFlask-Service[8] übergeben und die Rückgabewerte entsprechend entgegengenommen und abgespeichert. Dabei wurde über den Parameter *targets* bestimmt, dass die Lösungsmenge der Vorhersage auf die Knoten aus der Graphdatenbank beschränkt ist und über den Parameter $top_k = 5$, dass pro Anfrage nur die fünf wahrscheinlichsten Worte zurückgegeben werden.

Da die Verteilung und Häufigkeit der einzelnen Wörter in der Menge der Texte sehr unterschiedlich sein können, wurde der *IDF*-Wert (Abschnitt 2.8) für jedes Wort über den *FlickR*-Datensatz berechnet und mit den Ergebnissen des BertMaskFlask-Service multipliziert. Außerdem wurden die aus den vorherigen Schritten berechneten Werte gemittelt und in die Ergebnismenge aufgenommen, da es für jedes Wort mehr als nur einen vorverarbeiteten Text geben kann.

Die Berechnung der n-ähnlichsten Worte über das BERT-Modell und damit einhergehende Vorgehensweise wird in Algorithmus 5 in Pseudocode aufgeführt.

Algorithmus 5 : Berechnung der n-ähnlichsten Knoten (Worte) über das BERT-Modell unter Verwendung des BertMaskFlask-Service mit der Einbeziehung der *IDF*-Werte.

Input : Liste an maskierten Texten ($input$) und IDF-Werte für jedes Wort (idf_values)
Output : Die n-wahrscheinlichsten Worte die für die maskierte(n) Stelle(n) nach dem BERT-Modell eingesetzt würden sowie deren Relationen zum Starknoten

```
1  input = [[word_x, maskedSentence_x, doubleMaskedSentence_x], ..., []]
2  idf_values = {w_0 : x_0, w_1 : x_1, ..., w_n : x_n} // IDF-Werte für jedes Wort
3  top_n_bert_words = dict{∅} // Dictionary mit keys=words
4  top_n_bert_relations = [∅]
5  for s ∈ input do
6      top_n_s = bertMaskFlask_getValues(s) // bertMaskFlask-request
7      top_n_bert_words[s_word] = (top_n_s)
8  for key, val ∈ top_n_bert_words.items() do
9       // Berechnung des Durschnittswert und IDF-Multiplikation
10     top_n_bert_words[key] = avg(val) · idf_values[key]
11     top_n_bert_relations.append(top_n(n_key, nx))
12 return top_n_bert_words und top_n_bert_relations
```

[8]Der BertMaskFlask-Service erweitert die Aufgabe für die Vorhersage von maskierten Wörtern, sodass mehrere maskierte Wörter in einem Satz von einem BERT-Modell vorhergesagt werden können. Das hier genutzt Programm für die Umsetzung ist auf der Seite der Texttechnologie der Goethe Universität Frankfurt am Main unter https://gitlab.texttechnologylab.org/henlein/bertmaskflaskservice erreichbar (Henlein (2021)).

4.4. Algorithmus zur Detektion von Kleidungsstücken in Texten

Die in Abschnitt 4.3 vorgestellte Erstellung eines Graphen beinhaltet eine Vielzahl von Informationen und Relationen über Wörter, die eng mit dem Thema der Kleidung verbunden sind. Auf der Grundlage dieses Graphen wurde ein Python-Programm programmiert, dass in der Lage ist, aus einem gegebenen Text beschriebene Kleidungsstücke zu detektieren. Hierbei kann unterschieden werden in die direkte Detektion auf Wortebene und die indirekte Detektion, die von den im Eingabetext vorhanden Wortinformationen über den Graphen ein potenziell beschriebenes Kleidungsstück findet und ausgibt.

Die zwei folgenden Beispiele sollen den Unterschied zwischen direkter und indirekter Detektion deutlich machen.

Eingabetext: „A woman in a beautiful dress is going to a party today. “
Direkte Detektion: „dress “$\longrightarrow$ [Kleidungsstück: dress]

Eingabetext: „The floral pattern on the breast pocket make it a real eye-catcher. “
Indirekte Detektion: „breast pocket “$\rightarrow$[IS_PART_OF] $\rightarrow$ [Potentielles Kleidungsstück: T-Shirt]

Die beiden Beispiele sind hierbei vereinfachte Darstellungen, das Prinzip jedoch ermöglicht es aus den Wortinformationen des Textes auf ein Kleidungsstück zu schließen, wofür die Graphdatenbank die nötigen Relationen zwischen einzelnen Wörtern liefert.

Die Funktionsweise des Programms ist unterteilt in drei Schritte. Zunächst wird in einem ersten Schritt der Eingabetext eingelesen und vorverarbeitet. Dabei werden einzelne Wörter tokenisiert, lemmatisiert (Abb. 2.1), auf die Unterscheidung zwischen Groß- und Kleinschreibung verzichtet und Sonderzeichen entfernt. Anschließend wird jedes vorverarbeitete Wort mit den Knoten aus der Graphdatenbank verglichen, um Übereinstimmungen zu finden. Sofern ein Wort auch in der Graphdatenbank vorhanden ist, werden zu diesem Wort alle vorliegenden Knoteneigenschaften abgefragt und ausgegeben.

In einem zweiten Schritt werden die Wörter gemäß der direkten Detektion geprüft, sollte ein Kleidungsstück wörtlich im Text vorkommen, wird dieses ausgeben. Andernfalls wird für jedes Wort, das einen übereinstimmenden Knoten in der Graphdatenbank enthält, der kürzeste Weg zu einem Kleidungsstück berechnet. Dabei kann jedes Element aus der Menge der Kleidungsstücke der Zielknoten sein. Sofern ein Weg zu einem Kleidungsstück besteht, wird dieses unter der Angabe der einzelnen Relationen und Knoten auf dem Pfad ausgegeben. Die Länge des Pfades gibt damit gleichzeitig einen Hinweis über die Wahrscheinlichkeit an, ob der indirekt detektierte Zielknoten bzw. das Zielwort eine treffende Lösung darstellt.

In einem letzten Schritt werden zusätzlich Informationen über die Zugehörigkeit zu Altersgruppe und Geschlecht ausgegeben. Diese Informationen basieren wie in Abschnitt 4.3.1 beschrieben auf dem *Fashion Product Image Dataset* und den darin vorliegenden Daten.

Zu den detektierten Kleidungsstücken werden zusätzlich über eine interne Dependenzrelation, diejenigen im Text vorhandenen Attribute ausgegeben, die in direkter Beziehung das Kleidungsstück näher beschreiben.

Die folgende beispielhafte Ausgabe soll das Vorgehen verdeutlichen.

Eingabesatz: „The girl on the bench wears fancy red high heels and a blue jeans."

Detected Items: 1
Name: jeans
Associated Attributes: blue

→**Item-Properties (jeans)**:
AgeGroup: [['Adults-Men', '86.57%'], ['Adults-Women', '9.14%'],...]
Gender: [['Men', '88.41%'], ['Women', '9.14%'],...]
Category: Apparel
Sub-Category: Topwear

Potential Items: 1
Target-Name: shoes
Source-Name: high-heels
Path-length: 2
Node-Path: ['high heels', 'heels', 'shoes']
Relationship-Path: ['BMF_IDF', 'BMF_IDF']

Aus der beispielhaften Ausgabe geht hervor, dass aus dem Eingabetext ein Kleidungsstück (*Detected Items 1*) mit dem Namen „jeans "erkannt wurde. Hierzu werden alle relevanten zusätzlichen Informationen über Altersgruppen- und Geschlechtszuordnung sowie Haupt- und eine Subkategorie ausgegeben. Außerdem wird unter *Potential Items* ein möglicherweise im Text beschriebenes Kleidungsstück ausgegeben. Dabei stellt das Wort „high-heels "das Ausgangswort dar, von dem aus über einen Pfad der Länge zwei zweimalig über die Relation *BMF_IDF* auf das Kleidungsstück „shoes "geschlossen werden konnte.

Über das gegebene Beispiel wird die Wichtigkeit von Relationen deutlich sichtbar. Bei der Relation *BMF_IDF* handelt es sich um eine Relation, die aufgrund der Berechnungen über das BERT-Modell (Abschnitt 4.3.7) in den Graphen aufgenommen wurde.

4.5. Verworfene Ansätze

In diesem Kapitel werden verworfene Ansätze, die zur Bearbeitung der Aufgabenstellung angestrengt wurden, kurz vorgestellt.

DeepFashion2 - ImageRecommendation

Auf Grundlage des DeepFashion2 Datensatzes wurde ein Programm geschrieben, dass die im Datensatz enthaltenen Zuordnungen von Bildern zu Kleidungs-Kategorien und Attributen bei wortgetreuer Übereinstimmung findet und aus dem Datensatz ein passendes Bild bzw. passende Bilder anzeigt. Dabei wurde die Suche nach passenden Bildern durch im Text auftretende Attribute ergänzt und damit eingeschränkt.

Da hier jedoch lediglich ein wortgetreuer Vergleich mit einer Liste von Kategorie-Bezeichnungen vorgenommen wurde und der *DeepFashion2*-Datensatz bezüglich der eingetragenen Annotationen zu jedem Bild als auch die Qualität der Bilder im Datensatz Mängel aufweisen, wurde dieser Ansatz nicht weiter verfolgt.

Erstellung einer Klassenhierarchie für Kleidung durch WIKIDATA

Ein interessanter und erstrebenswerter Punkt für die Detektion von Kleidung stellt die Zuordnung detektierter Bestandteile zu einem kompletten Outfit dar. Hierbei ist es nötig, eine Hierarchie möglichst vieler Kleidungsstücke und deren Bestandteile zu erstellen.

Dies wurde versucht, durch die in WIKIDATA enthaltenen Zuordnungen von einzelnen Entitäten zu seinen Unterklassen (WIKIDATA: *instance_of, wikidata-identifier: P31*) und der Zuordnung der Entität selbst als Instanz von einer Entität höherer Ebene (WIKIDATA: *subclass_of, wikidata-identifier: P279*). Das zugrunde liegende Schemata wird im folgenden Beispiel kurz dargestellt.

Wort: „blue “
instance of (P31): „color “
subclass of (P279): „light “

Hieraus ließe sich theoretisch eine Hierarchie abfragen bzw. erstellen. Allerdings stellten sich die erzielten Ergebnisse als nicht ausreichend dar, da diese Daten nur sehr sporadisch zu den einzelnen Entitäten vorliegen und sich teilweise sogar widersprechen, sodass sich daraus keine Hierarchie ableiten ließ.

5. Technische Realisierung

In diesem Kapitel werden verschiedenen Programme und Ansätze aus Kapitel 4 aus technischer Sicht erklärt. Dabei wird auf den Aufbau der Graphdatenbank (Abschnitt 4.3) nicht weiter eingegangen, da die einzelnen Schritte bereits konzeptionell erklärt wurden und die technische Umsetzung durch die Aufführung der algorithmischen Darstellung in Pseudocode bereits erklärt wurde. Stattdessen wird der Ansatz für die Mehrklassen-Klassifizierung (Abschnitt 4.2) sowie das Programm für die Detektion von Kleidungsstücken (Abschnitt 4.4) bezüglich ihrer technischen Umsetzungen erklärt.

5.1. Mehrklassen-Textklassifizierung

Bei dem verwendeten vortrainierten Modell handelt es sich um das Modell *bert-base-uncased*, welches in Devlin et al. (2018) vorgestellt wurde. Das Modell unterscheidet nicht in Groß- und Kleinschreibung und wurde auf Texten in englischer Sprache ohne Annotationen oder Zusatzinformationen trainiert und wurde bereits in Abschnitt 2.6.1 näher beschrieben.

5.1.1. Fine-Tuning

Für das *Fine-Tuning* des BERT-Modells wurden Texte, die einem Kleidungsstück zugeordnet sind, verarbeitet und in eine für das BERT-Modell verständliche Form gebracht. Anschließend wurde das BERT-Modell auf diesen Daten trainiert bzw. abgestimmt.

Zunächst wurden die Textdaten durch den *Tokenizer* verarbeitet und die bereits in Abschnitt 2.6.1 angesprochenen speziellen Token ($[CLS], [SEP]$) eingefügt, die für BERT den Anfang des Eingabetextes bzw. die Satzenden symbolisieren. Der Zustand des $[CLS]$-Tokens entspricht dabei der aggregierten Sequenzdarstellung für die Klassifizierung.

Danach wurde über die Texte aus den Trainingsdaten die *Attention-Mask*, sowie eine *ID-Mask* angelegt. Die *Attention-Mask* zeigt dabei an, welche Token in den Trainingsdaten wie stark gewichtet werden sollen, die *ID-Mask* hingegen benötigt das Modell für die Zuordnung der einzelnen Token aus dem Eingabetext und entspricht somit einer numerischen Repräsentation der Token.

```
#Schleife über alle Eingabedaten bzw. die Beschreibungstexte
for i, phrase in enumerate(df["description"]):
  #Tokenisiere die Eingabedaten nach den gegebenen Parametern
  tokens = tokenizer.encode_plus(
    phrase, max_length=seq_len, truncation=True,
    padding="max_length", add_special_tokens=True,
    return_tensors="tf")
  #Erstellung der Input-Ids
```

```
    Xids[i, :] = tokens["input_ids"]
    #Erstellung der Attetion-Mask
    Xmask[i, :] = tokens["attention_mask"]
```

Da als Grundlage für das weitere Training auf dem *bert-base-uncased*-Modell aufgebaut wurde, wird jeder Token mit einer Vektor-Größe von 768 eingebettet. Dies ist den Arbeiten von Devlin et al. (2018, S.3) zu entnehmen. Als Eingabe wurden dem BERT-Modell anschließend die *Input*-IDs sowie die *Attention-Mask*, die durch die maximale Sequenzlänge von $seq\_len = 512$ in ihrer Größe bestimmt werden, übergeben. Der Wert für die maximale Sequenzlänge wurde maximal hoch angesetzt, da die zu verarbeiteten Textdaten größtenteils aus vielen langen Sätzen bestehen.

Nach der Verarbeitung der Eingabedaten für das BERT-Modell wurden eine Optimierungsfunktion, eine *loss*-Funktion sowie eine *accuracy*-Funktion bestimmt und an das Modell übergeben.

```
#Optimierungsfunktion
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-5,decay=1e-6)
#loss-Funktion
loss = tf.keras.losses.CategoricalCrossentropy()
#accuracy-Funktion
acc = tf.keras.metrics.CategoricalAccuracy("accuracy")
```

Schließlich wurden die Eingabedaten in Trainingsdaten (90%) und Daten für die Validierung (10%) aufgeteilt. Woraufhin das Modell mit dem *Fine-Tuning* beginnen konnte, wobei drei Durchgänge ($epochs = 3$) ausgeführt wurden, um die Genauigkeit des Modells zu verbessern.

```
#Model Training starten
history = model.fit(
  train_ds, #Trainingsdaten
  validation_data=val_ds, #Daten für die Validierung
  epochs=3) #Anzahl der festgelegten Durchgänge
#Speichern des berechneten Modells für die spätere Verwendung
model.save("category_model")
```

Der entspreche Code für die Berechnung des Models (*bert_fineTune.py*) und das berechnete Modell (Ordner: category_model) selbst sind in Anhang B hinterlegt. Der Datensatz (Abschnitt 3.2.2), der als Grundlage für das *Fine-Tuning* des BERT-Modells genutzt wurde, ist hingegen nicht hinterlegt, kann jedoch aus den Arbeiten von Aggarwal (2019) bezogen werden.

5.1.2. Ausführung des *Fine-Tuned BERT*-Modells

Für die Ausführung des im vorherigen Schritt berechneten BERT-Modells wird zunächst das Modell sowie das Mapping zwischen den Kleidungsstück-Bezeichnungen und der jeweiligen internen ID geladen. Anschließend erhält der Nutzer eine Abfrage, ob selbst ein Text eingegeben oder ein zufälliger Satz aus dem Datensatz ausgewählt werden soll. Nach der Eingabe

bzw. der Auswahl eines Eingabetextes wird über die Funktion *prep_data* die Eingabe in eine für das BERT-Modell verständliche Eingabe übertragen.

```
#Laden des BERT-Modells
model = tf.keras.models.load_model("../category_model/")

#Laden des Tokenizers
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

#Vorverarbeitung der Eingabe
def prep_data(text):
  tokens = tokenizer.encode_plus(text, max_length=512,
    truncation=True,
    padding="max_length",
    add_special_tokens=True,
    return_token_type_ids=False,
  return_tensors="tf")
  return {
    "input_ids": tf.cast(tokens["input_ids"], tf.float64),
    "attention_mask": tf.cast(tokens["attention_mask"],
     ↪  tf.float64)}

#Funktionsaufruf, um die Texteingabe für BERT vorzubereiten
data = prep_data(inputSequence)
```

Die Eingabe wird anschließend an das Model über die Funktion *Model.predict(data)* weitergegeben. Die Funktion ruft dabei eine Prozedur im Modell selbst auf, dass die wahrscheinlichsten Ausgabewerte in diesem Fall Kleidungsstücke, ausgibt. Anschließend wird aus den Rückgabewerten der wahrscheinlichste Wert über die Konsole ausgegeben.

```
#Vorverarbeitung des Eingabetextes
data = prep_data(inputSentence)
#Aufruf der Modell-Funktion zur Vorhersage der wahrscheinlichste
 ↪  Klasse
probs = model.predict(data)
#Berechnung des wahrscheinlichsten Werte aus der Menge der
 ↪  Ergebnisse
probsID = np.argmax(probs[0])
```

Der Programmcode für die Ausführung des BERT-Modells ist in dem Python-Skript (*ownModel.py*) abgelegt. Für die Ausführung wird der Zugriff auf das zuvor berechnete Modell benötigt (Anhang B).

5.2. Programm zur Detektion von Kleidungstücken in Texten

Das Programm ist in drei Python-Module unterteilt und verbindet sich durch Anfragen mit der Graphdatenbank in Neo4J. Entsprechend muss für die Ausführung die Graphdatenbank geöffnet und gestartet sein. Im Folgenden werden die drei Module vorgestellt.

5.2.1. Klassenmodul

Das Klassenmodul *fashionnet_classes* enthält die Klassendefinition $userInput$, in der die Eingabe des Nutzers nach einer Vorverarbeitung in eine Klasse übertragen wird.

userInput
sentences : int token : str pos_tags : str lemmata : str stemmer : str noun_chounks : list entities : boolean general : int dependencies : str neo_base_data : dict attributes : list
get_data_single_token(self, token) : dict

Abbildung 5.1.: UML-Klassendiagramm (userInput) aus dem Klassenmodul

Aus der Darstellung der Klasse in Abb. 5.1 wird ersichtlich, dass zunächst zahlreiche Informationen über die Eingabe des Nutzers gesammelt werden, die anschließend weiter genutzt werden können. Dabei werden die Satzzugehörigkeit, die einzelnen Token, die *POS-Tags* für die einzelnen Token sowie Lemmata und Stemma für jedes Token aus dem Eingabetext erfasst.

Außerdem werden Informationen über die im Text enthaltenen Nomen (*noun chunks*) erfasst, die das Nomen selbst und alle das Nomen beschreibenden Worte enthalten. Der Eingabetext wird darüber hinaus auf das Auftreten von Entitäten geprüft und das Ergebnis (*entities*) gespeichert. Generelle Informationen wie die Anzahl an Token im Eingabetext (*general*), beschreibende Attribute (*attributes*), möglicherweise bestehende Abhängigkeiten zu anderen Worten (*dependencies*) und alle Knoteninformationen aus der Neo4J-Graphdatenbank schließen die Liste ab. Mit der Klassenmethode *get_data_single_token()* können alle zuvor angesprochenen Informationen über das Token abgerufen werden.

5.2.2. Neo4J-Zugriffsmodul

Das Neo4J-Zugriffsmodul dient als Auslagerung für alle Funktionen, die sich auf die Graphdatenbank beziehen bzw. mit dieser interagieren. Hierbei wird zunächst über die Neo4J-Python-Bibliothek eine Verbindung mit der Datenbank aufgebaut. Unter Angabe der Zieladresse, über die die Graphdatenbank erreichbar ist, sowie der Zugangsdaten, kann mit der

Initialisierung des Treibers aus der Neo4J-Programmbibliothek eine Verbindung zur Graphdatenbank hergestellt werden.

```
#Zieladresse über die die Graphdatenbank erreichbar ist
uri = "bolt://localhost:7687"
#Treiberstart, um die Graphdatenbank erreichen zu können
driver = GraphDatabase.driver(uri, auth=("neo4j", "masterarbeit"))
```

Anschließend können über den initialisierten Treiber einzelne *sessions* gestartet werden, wobei hierin Datenbankabfragen in der von Neo4J verwendeten Abfragesprache *CYPHER* durchgeführt werden können. Die programmierten Funktionen werden in Anhang C detailliert beschrieben.

Durch die Funktionen können Daten über Knoten und Relationen aus der Graphdatenbank abgerufen werden und beispielsweise kürzeste Wege zwischen zwei Knoten oder Klassen von Knoten gefunden werden. Darüber hinaus ist es auch möglich, Einschränkungen bei der Suche nach kürzesten Wegen zu treffen. Zum Beispiel können den Relationsarten, aus dem ein kürzester Weg besteht, ausgeschlossen werden, sodass beispielsweise nur kürzeste Wege über Relationen gefunden werden, die mit Hilfe von statischen Modellen berechnet wurden.

5.2.3. Hauptmodul

Das Hauptmodul bildet den Kern des gesamten Programms und greift auf die beiden bereits vorgestellten Module (Abschnitt 5.2.2, Abschnitt 5.2.1) und die darin liegende Klassendefinition bzw. die Funktionen zum Informationsaustausch mit der Neo4J-Graphdatenbank zu. Hierbei wird zunächst die Eingabe eines Nutzers verarbeitet und die entsprechenden Berechnungen und Funktionsaufrufe umgesetzt. Die gesammelten Daten werden anschließend als Instanz der Klasse $userInput$ aus Abschnitt 5.2.1 gespeichert.

Anschließend wird für jedes im Eingabetext enthaltene Token mithilfe der Funktionen aus dem Neo4J-Zugriffsmodul geprüft, ob in der Graphdatenbank für dieses Token eine Repräsentation vorliegt. Sofern ein Token in der Graphdatenbank repräsentiert ist, wird anschließend geprüft, ob der zugehörige Knoten im Graphen mit dem Label *clothing* markiert ist. Somit kann festgestellt werden, ob der betreffende Knoten bereits ein Kleidungsstück repräsentiert. Bei einer Übereinstimmung werden aus der Klasseninstanz die zugehörigen Attribute und zusätzliche Informationen über Altersgruppen- und Geschlechterzuordnung sowie Haupt- und Subkategorie an den Nutzer in der Konsole ausgegeben. Gibt es hingegen eine Repräsentation eines Token, dass jedoch nicht mit dem Label *clothing* markiert ist, wird der kürzeste Weg von diesem Knoten aus zum nächsten Knoten gesucht, der als Kleidungsstück klassifiziert wurde.

Die gesammelten Informationen über detektierte Kleidungstücke respektive potenziell beschrieben Kleidungstücke (indirekte Detektion), werden schließlich gesammelt an den Nutzer über die Konsole ausgegeben.

6. Praktische Ergebnisse und Evaluierung

In diesem Kapitel werden die Ergebnisse, die sich aus den vorgestellten Modellen und Ansätzen ergeben haben, vorgestellt. Außerdem werden eine Evaluierung der beiden Hauptansätze auf der Grundlage zweier Testdatensätze durchgeführt und die daraus resultierenden Ergebnisse aufgezeigt.

6.1. Mehrklassen-Textklassifizierung

In einem ersten Ansatz wurde ein vortrainiertes BERT-Modell hinsichtlich einer Mehrklassen-Klassifizierung abgestimmt. Die Grundlage für das Training bildeten die Kleidungsstücke sowie zugewiesene Beschreibungstexte aus dem Datensatz aus Abschnitt 3.2.3.

Zunächst wurden hierfür die Daten in 90% Trainingsdaten und 10% Daten für die Validierung unterteilt. In insgesamt drei Trainingsdurchläufen (Epochen) wurden nacheinander die Gewichte innerhalb des BERT-Modells auf die spezifische Mehrklassen-Klassifizierung angepasst. In Tabelle 6.1 sind die resultierenden Werte aufgeführt, die bei der Berechnung des Modells entstanden sind.

Tabelle 6.1.: Gütewerte und beanspruchte Zeit für das Training des BERT-Modells aus Abschnitt 4.2

Epoche	Zeit (Stunden)	loss	accuracy	val_loss	val_accuracy
1	14:39h	1.5635	0.6340	0.7323	0.8204
2	14:41h	0.6209	0.8489	0.5044	0.8730
3	14:48h	0.4396	0.8885	0.3577	0.9066

Wie in Tabelle 6.1 zu sehen, wird die Genauigkeit des Modells über die Trainingsdaten (*accuracy*) und Validierungsdaten (*val_accuracy*) mit jedem Trainingsdurchgang erhöht, während die Werte der *loss*-Funktion über die Trainingsdaten (*loss*) sowie Validierungsdaten (*val_loss*) abnimmt. Dabei nimmt die vor dem Training definierte *loss*-Funktion direkten Einfluss auf die Parameter des BERT-Modells. Die Angabe der Genauigkeit hat hingegen keinen Einfluss während des Trainings und dient nur als Anzeigeparameter.

Das trainierte Modell kann für die Klassifizierung von gesamten Texten verwendet werden. Es ist nicht möglich, das Auftreten mehrerer Kleidungsstücke oder zugehörige Attribute zu identifizieren.

Das Modell wurde auf Grundlage einer Testmenge aus dem *Fashion Image Product Dataset*-Datensatz (Abschnitt 3.2.2) und Texten eines weiteren Datensatzes (*Fashion Clothing Products*

Dataset)[1] getestet. Die erzielten Werte für die Genauigkeit des Modells sind in Tabelle 6.2 aufgeführt.

Tabelle 6.2.: Angaben über die Genauigkeit des BERT-Modells bei der Mehrklassen-Klassifizierung

Datensatz	**FlickR Image Dataset**	**Fashion Clothing Products Dataset**
Textanzahl	10000	6850
Richtig klassifiziert	9071	6838
Falsch klassifiziert	929	12
Genauigkeit	90,70%	0,18%

In den Ergebnissen aus Tabelle 6.2 ist zu erkennen, dass der Transfer des trainierten BERT-Modells auf eine andere Testmenge an Texten sehr schlechte Ergebnisse produziert. Während die Mehrklassen-Klassifizierung auf Texteingaben aus dem für das Training verwendeten Datensatzes entsprechend der angegebenen Genauigkeit aus Tabelle 6.1 funktioniert und gute Ergebnisse erzielt, konnte dies auf eine andere Datenquelle nicht übertragen werden.

Der wesentliche Grund für die niedrige Genauigkeit des trainierten Modells auf einen zweiten Datensatz stellt der Inhalt des zum Training verwendeten Datensatzes dar, der hauptsächlich aus Waschanleitungen und weniger aus realen textlichen Beschreibungen von Kleidungsstücken besteht. Ein weiterer Grund ist die heterogene Verteilung der Texte zu den einzelnen Kategorien auf den Abschnitt 7.2 näher eingegangen wird. Außerdem ist die Anzahl der Klassen (143) deutlich zu hoch angesetzt, wobei ein möglicher Verbesserungsansatz in Abschnitt 7.3 vorgestellt wird.

Da bei der Recherche nach passenden Datensätzen für das Trainieren des BERT-Modells keine weiteren Datensätze gefunden werden konnten, die eine Verbesserung des Modells erwarten ließen, wurde dieser Ansatz schließlich verworfen. Für die Modellierung der Bekleidung einer Person reicht zudem die Klassifizierung eines Textes zu lediglich einem Kleidungsstück nicht aus, viel mehr wird die Detektion von mehreren im besten Fall jedem textlich beschriebenen Kleidungsstücken in einem Text benötigt.

Die gewonnenen Erkenntnisse aus dem Ansatz ein BERT-Modell auf eine Mehrklassen-Klassifizierungsaufgabe bezüglich einer zu bestimmenden Kleidungskategorie abzustimmen, wurde schließlich der Aufbau einer Graphdatenbank als eigenständige Ressource gewählt, um die fehlenden Informationen und die Themenkomplexität von Bekleidungen entsprechend abbilden zu können.

Auch wenn der in diesem Kapitel vorgestellt Ansatz gemessen an den erzielten Ergebnissen als misslungen bewertet werden kann, wurden hieraus die wesentlichen Erkenntnisse

[1] https://www.kaggle.com/datasets/shivamb/fashion-clothing-products-catalog, Abrufdatum: 22.03.2022 16:41Uhr

für die weitere Bearbeitung der Aufgabenstellung gewonnen. Zudem zeigt das Modell auf dem für das Training genutzten Datensatz durchaus gute Ergebnisse, sodass das Vorgehen, das in Abschnitt 4.2 und Abschnitt 5.1 detailliert vorgestellt wurde, bei einer neuen qualitativ passenden Datenquelle genutzt werden könnte und daraus auch beim Transfer auf andere Datensätze gute Ergebnisse erreicht werden könnten.

6.2. Neo4J Graphdatenbank

Aufgrund der Erkenntnisse aus Abschnitt 6.1 wurde in einem nächsten Schritt eine Graphdatenbank als Ressource für kleidungsspezifische Texte erstellt. Dabei wurden verschiedene Informationen über Kleidungsstücke mithilfe von WordNet, wikidata und Informationen aus verschiedenen Datensätzen in die Datenbank aufgenommen.

Anschließend wurden zunächst über die statischen Modelle Word2Vec, GLove und fastText, die Ähnlichkeiten unter den einzelnen Knoten innerhalb der Datenbank berechnet. Zusätzlich wurde die Ähnlichkeit einzelner Knoten auch über ein BERT-Modell mithilfe des BertMaskFlask-Service unter Verrechnung der IDF-Werte für jedes Wort berechnet. Durch die einzelnen Schritte konnte die Datenbank aufgebaut und popularisiert werden.

Tabelle 6.3.: Relationsarten und zugehörige Anzahl in der Neo4J-Graphdatenbank

Relation	**Anzahl**	**Beschreibung**
fastText	48270	n nächste Worte (FastText)
STATIC_MODELS_COMBINED	47158	n nächste Worte (GLove)
GLove	45555	n nächste Worte (Word2Vec)
Word2Vec	44322	n nächste Worte über alle statischen Modelle
BMF_IDF	39117	n nächste Worte BERT-Modell
HAS_HYPONYM	7145	Oberbegriff zu Unterbegriff (Hyponym)
HAS_HYPERNYM	5232	Unterbegriff zu Oberbegriff (Hypernym)
IS_PART_OF	1140	Teil-Ganzes-Beziehung (Meronym)
IS_COLOR	826	Relation zur Hauptkategorie: Farbe
IS_CLOTHING	126	Relation zur Hauptkategorie: Kleidung
HAS_SUPERCATEGORY	42	Relation zur Superkategorie (Körperregion)
IS_ALIAS_OF	29	Alias zu Hauptwort

Tabelle 6.4.: Anzahl und Art der Knoten in der Neo4J-Graphdatenbank

Label	**Anzahl**	**Beschreibung**
ITEM	8751	Nicht näher bestimmte Wort-Knoten
COLOR	825	Farben
CLOTHING	126	Kleidung
SUPERCATEGORY	12	Körperregion

Die Graphdatenbank enthält insgesamt 9736 Knoten, die jeweils ein Wort repräsentieren (Tabelle 6.4). Außerdem wurden insgesamt 238.962 Relationen (Tabelle 6.3) zwischen den einzelnen Knoten eingetragen. Auf Grundlage der erstellten Graphdatenbank lassen sich Modelle anreichern oder begrenzen.

6.3. Hauptprogramm

Auf der Grundlage der entworfenen Graphdatenbank wurde das Hauptprogramm entworfen, dass die in einem Eingabetext beschriebenen bzw. enthaltenen Kleidungsstücke detektiert. Dabei werden die direkte als auch indirekte Detektion genutzt, wobei bei letzterer Variante die zahlreichen Graphrelationen eine tragende Rolle spielen.

Im Folgenden werden die Ergebnisse der Evaluierung des Programms in Tabelle 6.5 vorgestellt. Für die Evaluierung wurden 10.000 Sätze aus dem *Fashion Clothing Products Dataset* genommen. Die Texte sind in diesem Datensatz jeweils nur einer Kleidungskategorie zugeordnet, sodass die Menge an direkten und indirekten Detektion des Programms dahingehend geprüft wird, ob die zugeordnete Kategorie enthalten ist.

Zusätzlich wurde das Programm während der Validierung hinsichtlich der verfügbaren Relationen aus der Graphdatenbank eingeschränkt. Somit kann eine Einschätzung über die Qualität der gezogenen Relationen über statische bzw. das kontextbezogene BERT-Modell getroffen werden.

Tabelle 6.5.: Testergebnisse des Hauptprogramms auf dem Test-Datensatz

Besonderheiten	**Richtig**	**Falsch**	**Genauigkeit**
Nur Relationen statischer Modelle	825	175	82,5%
Nur Word2Vec-Relationen	822	178	82,2%
Nur GLove-Relationen	815	185	81,5%
Nur FastText-Relationen	806	194	80,6%
Nur kombinierte-Relationen (statische Modelle)	806	194	80,6%
Keine Einschränkungen	771	229	77,1%
Nur über BERT-Relationen	770	230	77,0%

Die in Tabelle 6.5 vorgestellten Ergebnisse zeigen, dass die berechneten Relationen über statische Modelle die besten Werte erzielen. Hierbei ist jedoch zu beachten, dass die als Knoten dargestellten Wörter aus der Graphdatenbank als sehr spezifisches Vokabular verstanden werden können. Diese wurde über den BertMaskFlask-Service bzw. dem zugrundeliegenden BERT-Model entsprechend häufig von Subwörtern ersetzt. Dies führt dazu, dass die Ergebnismenge aus den Berechnungen auf eine kleinere Anzahl an Wortknoten reduziert wurde und damit eine Reduzierung möglicher Zielknoten. Eine Anpassung des Vokabulars war nicht möglich, da sich der Service auf die verfügbaren vortrainierten Modelle stützt und keine direkte Erweiterung erlaubt. Bei einer entsprechende Anpassung des zugrunde liegenden Modells, könnte dadurch eine erhöhte Genauigkeit erzielt werden.

6.4. Performance und Zusammenhänge mit Green-IT

Im Rahmen dieser Arbeit wurden verschiedene Verfahren und Modelle genutzt mithilfe derer neue Informationen über eine Datenbasis erlangt wurden. Die verwendeten Verfahren, speziell die statischen sowie kontextuellen Modelle aus dem Bereich des *Machine Learning* berechnen dabei Zusammenhänge über eine sehr große Menge an Daten. Hierbei wird eine hohe Rechenleistung benötigt. Daraus resultiert in der Regel eine hohe Berechnungszeit. Diese steht wiederum in direktem Zusammenhang mit der benötigten Energie.

In Zeiten des Klimawandels hat ökologisches Verhalten in allen Lebensbereichen zunehmend an Bedeutung gewonnen. Dies spiegelt sich auch in der Informationstechnologie und den damit verbundenen Hardware- sowie Softwareprodukten wider. Weltweit werden alleine ca. 10% des Energiebedarfs durch IT-Systeme beansprucht (Verdecchia et al. (2021, S.7)). Somit besteht die Notwendigkeit, IT-bezogene Arbeiten hinsichtlich des Energieverbrauchs zu optimieren.

Die Forschungsdisziplin *Green-IT* beschäftigt sich mit dem möglichst effizienten und effektiven Design von Soft- und Hardwarekomponenten, um die Auswirkungen auf die Umwelt so gering wie möglich zu halten (Murugesan und Gangadharan (2008, S.2)).

In diesem Zusammenhang werden die benötigten Zeiten für die Berechnung der vorgestellten Schritte tabellarisch aufgeführt (Tabelle 6.6), wobei nur jene Schritte beachtet werden, die ein signifikantes Volumen an Rechendauer und Leistung beansprucht haben. Insgesamt ergibt sich eine Gesamtzeit von 74:23:58h für die Berechnungen der vorgestellten Verfahren und Modelle in dieser Arbeit.

Tabelle 6.6.: Übersicht über die benötigte Zeit zur Berechnung verschiedener Aufgaben über mehrere Modelle

Aufgabe der Berechnung	**Verwendetes Modell**	**Berechnungszeit**
Berechnung der ähnlichsten Knoten	GLove	1:15:44h
Berechnung der ähnlichsten Knoten	Word2Vec	1:13:22h
Berechnung der ähnlichsten Knoten	FastText	1:29:39h
Berechnung der ähnlichsten Knoten	BERT	26:13:34h
Fine-Tuning: Mehrklassen-Klassifizierung	BERT	44:11:39h
	Gesamtzeit	**74:23:58h**

Das BERT-Modell, welches in Abschnitt 4.2 vorgestellt wurde, wurde auf einem Computer mit den Spezifikationen, die in Tabelle 6.7 aufgeführt sind, trainiert. Ebenso alle weiteren Berechnungsschritte. Da für die eingebundene Programmbibliothek *Tensorflow*, über das das Modell geladen wurde, keine Erweiterung hinsichtlich der Unterstützung von GPU-basierten Modell-Berechnungen für die genutzte GPU zur Verfügung stand, wurden die Berechnungen auf der CPU ausgeführt. Dementsprechend erklärt sich die hohe Berechnungszeit.

Tabelle 6.7.: PC-Spezfikationen die für die Berechnungen genutzt wurden

Hardware	**Modellbeschreibung**
Prozessesor	AMD Ryzen 5 3600 6x 3.60GHz (65W)
Mainboard	MSI B450 Gaming Plus MAX AMD (95W)
Grafikkarte	8GB AMD Radeon 5700 XT(225W)
Arbeitsspeicher	32GB (2x 16384MB) DDR4-3200 (<5W)

Aus den angegebenen Spezifikationen und der benötigten Berechnungszeit lässt sich der Energieverbrauch jedoch nicht zuverlässig berechnen. Da hierzu aufgrund fehlender Möglichkeiten softwareseitig keine verlässliche Erfassung der Leistungsaufnahme für die verwendete Hardware während der Berechnungen protokolliert werden konnte, wird an dieser Stelle auf eine exakte Berechnung des Energieverbrauchs verzichtet. Als obere Schranke des Energieverbrauchs ergeben die angegebenen $65W$ für die maximale Leistungsaufnahme der CPU und der Schätzung von $100W$ für alle weiteren Komponenten unter Einbeziehung der benötigten Zeit $\approx 74,4h$ einen Energieverbrauch von $\approx 12,276\ kWh$ (Gleichung (6.1)). Dabei bezieht sich die Berechnung auf eine CPU unter Volllast während der gesamten Laufzeit.

$$Energieverbrauch = \frac{Leistung(W) \cdot Zeit(h)}{1000} = \frac{165W \cdot 74,4h}{1000} = 12,276\ kWh \quad (6.1)$$

Aus den vorherigen Ausführungen ergibt sich ein dringender Bedarf zur Optimierung der einzelnen Berechnungsschritte hinsichtlich der benötigten Berechnungsdauer. Eine Möglichkeit zur Reduktion der Berechnungszeit könnte die Verwendung der GPU als primäre Berechnungsgrundlage darstellen. Hierbei ist jedoch zu beachten, dass eine GPU in der Regel einen deutlich höheren Energiebedarf als eine CPU aufweist (Tabelle 6.7). Insgesamt zeigen die erfassten Daten, dass die Entwicklung der hier vorgestellten Programme hinsichtlich eines ökonomischem Verhaltens optimiert werden sollten.

6.5. Zusammenfassung der Ergebnisse

In diesem Kapitel wurden die Ergebnisse aus den einzelnen Modellberechnungen, dem Aufbau einer Graphdatenbank und eines Programms für die Detektion von Kleidungsstücken vorgestellt. Dabei zeigte der initiale Ansatz bezüglich des *Fine-Tunings* eines BERT-Modells auf eine Mehrklassen-Klassifizierung deutliche Schwächen beim Transfer auf neue Datensätze. Die Gründe hierfür liegen maßgeblich an dem verwendeten Datensatz, der für das *Fine-Tuning* verwendet wurde, sowie der Anzahl an Klassen.

Mangels alternativer Datensätze wurde daraufhin eine Wissenressource über eine Graphdatenbank erstellt, die mit einer Vielzahl von Knoten und Relationen über statische und kontextuelle Modelle angereichert wurde. Die Graphdatenbank wurde anschließend als Grundlage für die Erstellung eines Programms für die Detektion von Kleidungsstücken in Texten genutzt und zeigte bei der Evaluierung gute Ergebnisse.

7. Schlussfolgerung und Ausblick

In diesem Kapitel wird ein Fazit bezüglich der vorgestellten praktischen und theoretischen Ansätze und Umsetzungen gezogen. Darüber hinaus werden offene Probleme angesprochen, die im Rahmen dieser Arbeit nicht ausreichend bearbeitet werden konnten. Schließlich werden Anregungen für weitere Arbeiten vorgestellt, die auf Grundlage dieser Arbeit durchgeführt werden könnten.

7.1. Fazit

Bei dieser Arbeit wurde die Aufgabenstellung der Bekleidungsmodellierung in verschiedenen Ansätzen bearbeitet. Zunächst wurde ein BERT-Modell auf die Aufgabe einer Mehrklassen-Klassifizierung trainiert und abgestimmt. Aufgrund der schlechten Verfügbarkeit von Datensätzen, die in Qualität und Quantität für das Trainieren eines Modells ausreichend sind, wurde eine Wissensressource in Form einer Graphdatenbank entwickelt.

Dabei wurde erneut für die Berechnung der n-ähnlichsten Wortknoten das BERT-Modell in Form des BertMaskFlask-Service genutzt. Außerdem wurden für die gleiche Aufgabe statische Modelle (FastText, Word2Vec und GLove) herangezogen und zusätzlich ein kombinierter Wert über alle drei statischen Modelle gebildet. Des Weiteren wurde die Graphdatenbank mit Informationen aus verschiedenen zur Verfügung stehenden Ressourcen (*Fashionpedia Dataset*, wikidata, WordNet und einer Wikipedia Liste von Farbnamen) angereichert. Die Graphdatenbank wurde anschließend für die textbasierte Detektion von Kleidungsstücken genutzt. Hierbei konnten gute Ergebnisse erzielt werden.

Resultierend aus den weniger performanten Ergebnissen bei dem eigens trainierten BERT-Modell sowie der Berechnung von ähnlichsten Worten über BERT konnten Rückschlüsse auf die Begründung hierfür gezogen werden. Diese liegen hauptsächlich an dem sehr spezifischen Vokabular, welche im Kontext mit Bekleidungen auftreten, sowie der schlechten Verfügbarkeit von Trainings-Daten auf textlicher Basis. Darüber hinaus stellte sich die Heterogenität der verwendeten Datensätze als äußerst problematisch heraus, da so eine Zusammenführung der ohnehin raren Daten nur schwer umzusetzen war.

Trotzdem bieten die vorgestellten Ansätze wertvolle Erkenntnisse, auf denen aufbauen weitere Arbeiten erfolgen können. Sofern qualitativ hochwertige Datensätze verfügbar sind, könnten die Ansätze, die mithilfe des BERT-Modells durchgeführt wurden, vermutlich deutlich bessere Ergebnisse erzielen.

7.2. Offene Probleme

Im Folgenden werden die Probleme bei der Bearbeitung der Aufgabenstellung besprochen. Diese beziehen sich in der Regel auf alle durchgeführten Ansätze und Verfahren.

Domänenspezifisches Vokabular

Das hier behandelte Thema Kleidung inkludiert eine hohe Anzahl an domänenspezifischem Vokabular. Dieser Umstand führt dazu, dass die Worte bei vortranierten BERT-Modellen ersetzt werden und somit Informationen verloren gehen. Zum Beispiel wird das Wort „baby shoe"durch „baby" bzw. „shoe" ersetzt. Für die Berechnung des BERT-Modells bzw. die Verwendung eines vortrainierten BERT-Models über den BertMaskFlask-Service könnte das Vokabular des jeweiligen Modells um kleidungsspezifische Wörter ergänzt und neu darauf trainiert werden.

Inkonsistenzen über Kleidungsbezeichnungen

Wie bereits im vorherigen Abschnitt angedeutet, stellt sich das domänenspezifische Vokabular bezüglich Kleidung als äußerst divers heraus. Dies führt auch zu inkonsistenten Bezeichner für das gleiche Objekt bzw. zu unklaren Regularien für die Bezeichnung von Kategorien, Zuordnungen oder Kleidungsstücken selbst. Somit stellt sich die Problematik ein, dass Datensätze nur schwer miteinander vereinbar sind. Beispielsweise wird die Bekleidung für den Oberkörper in einem Datensatz mit „Topwear" und in einem anderen Datensatz als „Upperwear" bezeichnet.

Dies setzt sich auf der Ebene der Kleidungsstücke fort. So gibt es für ein T-Shirt mehrere Bezeichnungen über die Datensätze hinweg (*shirt, shirts, t-shirt, t-shirts, tshirt, tshirt*). Dabei werden zum Teil Zuordnungen getroffen, die nur schwer zu analysieren und erfassen sind, denn „shirt" beschreibt in der englischen Sprache in der Regel ein Hemd und kein T-Shirt. Erweitert auf umgangssprachliche Beschreibungen wie „tee" (für T-Shirt), wird die Zuordnung und damit einhergehend auch die Berechnungen über Modelle äußerst schwierig, da es für „tee" gleich mehrere Wortbedeutungen (Markierung, Verzweigung, u.w.) gibt, die mit Kleidungsstücken eher weniger in Verbindung stehen. Hierbei kann auch durch Lemmatisierung oder *Stemming*[1] nicht in jedem Fall das gewünschte Ergebnisse erzielt werden.

Tabelle 7.1.: Übersicht von Lemma und Wortstamm für verschiedene Begriffsvarianten eines T-Shirts

Wort	shirt	shirts	t-shirt	t-shirts	tshirt	tshirts
Lemma	*shirt*	*shirt*	*t-shirt*	*t-shirts*	*tshirt*	*tshirts*
Wortstamm	*shirt*	*shirt*	*t-shirt*	*t-shirt*	*tshirt*	*tshirt*

[1]Stemming beschreibt die Rückführung eines Wortes auf seinen Wortstamm (Jivani et al. (2011)).

Abschließend beinhalten neue Wortschöpfungen aus der Mode weitere Problematiken. Das Wort „jeggings“ beispielsweise stellt ein zusammengezogenes Wort aus den Wörter „jeans“ und „leggings“ dar, die Zuordnung hierfür unterliegt jedoch keinen klaren Regeln. Eine Zuordnung zum Wort „jeans“ oder „leggings“ wäre eben so zutreffend wie die Zuordnung zur Hauptkategorie Hose.

Zusammenführung einzelner Datensätze

Die in Abschnitt 7.2angesprochenen Problematiken spiegeln sich bei der Zusammenführung der einzelnen Datensätze direkt wider. Daraus ergeben sich Zuordnungsprobleme über inkompatible und sich teils widersprechende Bezeichnungen von Kategorien, Unterkategorien und Kleidungsbezeichnungen. Eine Zusammenführung von Daten aus verschiedenen Datensätzen ist entsprechend schwierig in der Umsetzung.

Qualität der Datensätze

Die im Rahmen dieser Arbeit verwendeten Datensätze (Abschnitt 3.2) stellen sich bezogen auf die Aufgabenstellung als qualitativ mangelhaft dar. Datensätze, die textliche Beschreibungen und zugewiesene Kleidungsstücke beinhalten, sind hauptsächlich auf Grundlage von Bilddatenbanken generiert worden und beinhalten häufig heterogene Textstrukturen bezüglich der Länge und Beschreibungsart. Zusätzlich beinhalten einige Datensätze unvollständige Einträge (kein Text, keine Annotationen oder kein Label), was die Menge an nutzbaren Daten stark reduziert. Darüber hinaus ist die Verteilung über die zugewiesenen Label oder Kategorien nicht homogen. So beinhaltet beispielsweise der *Fashion Product Image Dataset* ca. 7000 Einträge über T-Shirts, jedoch nur wenige (unter 10) über Hüte.

7.3. Anregungen für weitere Arbeiten

In diesem Kapitel werden aufbauend aus den Erkenntnissen dieser Arbeit Anregungen für weitere Arbeiten gegeben.

Aufbau einer Klassenhierarchie für Kleidungskategorien

Wie in Abschnitt 4.5 bereits angesprochen, wurde der Versuch unternommen eine Hierarchie unter den einzelnen Kategorien und Bezeichnungen für Kleidungsstücke herzustellen. Hierfür könnte die Menge an Kategoriebezeichnungen über den *coarse-to-fine*-Ansatz verarbeitet werden. Einen entsprechenden Ansatz wird in den Arbeiten von Mekala et al. (2021) vorgestellt.

Erweiterung des Vokabulars von BERT

Um die in Abschnitt 7.2 angesprochenen Probleme bezüglich des domänenspezifischen Vokabulars und damit einhergehend auch die Zusammenführung verschiedener Datensätze zu

verbessern, könnte ein Verfahren entwickelt werden, mit dem BERT-Modelle um dieses Vokabular entsprechend ergänzt werden. Hierzu wären Vorarbeiten aus Abschnitt 7.3 von großem Nutzen, um das spezifische Vokabular einzugrenzen und die Qualität zu steigern.

Bezug auf Situationen bei der Modellierung von Kleidung

Die im Rahmen dieser Arbeit vorgestellten Ergebnisse liefern nützliche Informationen über die in einem Eingabetext beschriebene Situation. Beispielsweise kann aus einem Text der das Tragen von Fußballschuhen beinhaltet darauf geschlossen werden, dass die Person sich in einer Situation befindet, bei der Fußballschuhe getragen werden. Daraufhin könnten Rückschlüsse über weitere Kleidungsstücke zur Vervollständigung eines gesamten Outfits gezogen werden. Ein möglicher Rückschluss ausgehend von Fußballschuhen wäre zum Beispiel, dass die Person auch ein Trikot, Stutzen und eine Trainingshose trägt. Basierend auf dieser Logik könnte in weiteren Arbeiten der Versuch unternommen werden, auf die Situation zu schließen.

Literaturverzeichnis

Hirschberg, Julia und Christopher D Manning (2015). „Advances in natural language processing“. In: *Science* 349 (6245), S. 261–266.

Jurafsky, Daniel und James H Martin (Feb. 2008). „Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing“. In: *Upper Saddle River, NJ: Prentice Hall* 2.

Jusoh, Shaidah (2018). „A study on NLP applications and ambiguity problems“. In: *Journal of Theoretical and Applied Information Technology*.

Khurana, Diksha et al. (2017). „Natural language processing: State of the art, current trends and challenges“. In: *arXiv preprint arXiv:1708.05148*.

Dale, Robert (2019). „NLP commercialisation in the last 25 years“. In: *Natural Language Engineering, Cambrigde University Press* 25 (3), S. 419–426.

Prasad, TVVV und Raghu B Korrapati (2017). „Computerized Applications of Natural Language Processing in Digital Economy: A Review“. In: *International Journal of Engineering and Management Research (IJEMR), Vandana Publications* 7 (1), S. 239–241.

Kelleher, John D (2019). *Deep learning*. The MIT Press. ISBN: 9780262537551.

Xu, Zongben und Jian Sun (2017). „Model-driven deep-learning“. In: *National Science Review* 5 (1), S. 22–24. ISSN: 2095-5138.

Chai, Junyi und Anming Li (2019). „Deep Learning in Natural Language Processing: A State-of-the-Art Survey“. In: *2019 International Conference on Machine Learning and Cybernetics (ICMLC)*. IEEE, S. 1–6.

Manning, Christopher D et al. (2014). „The Stanford CoreNLP natural language processing toolkit“. In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, S. 55–60.

Bhattacharyya, Siddhartha et al. (2020). *Deep Learning: Research and Applications*. Bd. 7. Walter de Gruyter GmbH & Co KG.

Crowder, James A und Shelli Friess (2012). „Artificial psychology: The psychology of AI“. In: *People* 2 (3), S. 4–5.

D'Alfonso, Simon (2020). „AI in mental health“. In: *Current Opinion in PsychologCurrent Opinion in Psycholog* 36, S. 112–117.

Ienca, Marcello und Karolina Ignatiadis (2020). „Artificial intelligence in clinical neuroscience: methodological and ethical challenges“. In: *AJOB neuroscience* 11 (2), S. 77–87.

Krittanawong, Chayakrit et al. (2017). „Artificial intelligence in precision cardiovascular medicine“. In: *Journal of the American College of Cardiology* 69 (21), S. 2657–2664.

Ahmed, Zeeshan et al. (2020). „Artificial intelligence with multi-functional machine learning platform development for better healthcare and precision medicine“. In: *Database, Oxford Academic*.

Traeger, M. et al. (2003). „Künstliche neuronale Netze“. In: *Der Anaesthesist* 52 (11), S. 1055–1061.

Futia, Giuseppe und Antonio Vetrò (2020). „On the integration of knowledge graphs into deep learning models for a more comprehensible AI—Three Challenges for future research“. In: *Information* 11 (2), S. 122.

Roy, Vincent Van et al. (2020). „AI and robotics innovation“. In: *Handbook of labor, human resources and population economics, Springer*, S. 1–35.

Ma, Yifang et al. (2020). „Artificial intelligence applications in the development of autonomous vehicles: a survey“. In: *CAA Journal of Automatica Sinica, IEEE* 7 (2), S. 315–329.

Ayodele, Taiwo Oladipupo (2010). „Machine learning overview“. In: *New Advances in Machine Learning* 2.

Alpaydin, Ethem (2021). *Machine Learning*. MIT Press. ISBN: 9780262529518.

O'Shea, Keiron und Ryan Nash (2015). „An Introduction to Convolutional Neural Networks“. In: *arXiv preprint arXiv:1511.08458*.

Staudemeyer, Ralf C. und Eric Rothstein Morris (2019). „Understanding LSTM–a tutorial into long short-term memory recurrent neural networks“. In: *arXiv preprint arXiv:1909.09586*.

Schmidt, Robin M (2019). „Recurrent neural networks (rnns): A gentle introduction and overview“. In: *arXiv preprint arXiv:1912.05911*.

Hochreiter, Sepp und Jürgen Schmidhuber (1997). „Long Short-Term Memory“. In: *Neural Computation* 9, S. 1735–1780.

LeCun, Yann, Yoshua Bengio et al. (1995). „Convolutional networks for images, speech, and time series“. In: *The handbook of brain theory and neural networks* 3361 (10), S. 1995.

Vaswani, Ashish et al. (2017). „Attention is all you need“. In: *Advances in neural information processing systems* 30.

Bridle, John S (1990). „Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition“. In: *Neurocomputing*. Springer, S. 227–236.

Shaw, Peter et al. (2018). „Self-Attention with Relative Position Representations“. In: *arXiv preprint arXiv:1803.02155*.

Devlin, Jacob et al. (2018). „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. In: *arXiv preprint arXiv:1810.04805*.

Ravichandiran, Sudharsan (2021). *Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT*. Packt Publishing Ltd.

Zhong, Ruiqi et al. (2021). „Are larger pretrained language models uniformly better? comparing performance at the instance level“. In: *arXiv preprint arXiv:2105.06020*.

Turc, Iulia et al. (2019). „Well-read students learn better: On the importance of pre-training compact models“. In: *arXiv preprint arXiv:1908.08962*.

Ambler, Scott W (1998). *Process Patterns: Building Large-Scale Systems Using Object Technology*. Cambridge University Press.

Song, Xinying et al. (2020). „Fast WordPiece Tokenization“. In: *arXiv preprint arXiv:2012.15524*.

Zhu, Yukun et al. (2015). „Aligning books and movies: Towards story-like visual explanations by watching movies and reading books“. In: *Proceedings of the IEEE international conference on computer vision*, S. 19–27.

Werbos, Paul J (1990). „Backpropagation through time: what it does and how to do it“. In: *Proceedings of the IEEE* 78 (10), S. 1550–1560.

Williams, Adina et al. (2017). „A broad-coverage challenge corpus for sentence understanding through inference“. In: *arXiv preprint arXiv:1704.05426*.

Nadeau, David und Satoshi Sekine (2007). „A survey of named entity recognition and classification“. In: *Lingvisticae Investigationes*.

Rajpurkar, Pranav et al. (2016). „Squad: 100,000+ questions for machine comprehension of text“. In: *arXiv preprint arXiv:1606.05250*.

Zhang, Yin et al. (2010). „Understanding bag-of-words model: a statistical framework“. In: *International Journal of Machine Learning and Cybernetics* 1 (1), S. 43–52.

Mikolov, Tomas, Kai Chen et al. (2013). „Efficient Estimation of Word Representations in Vector Space“. In: *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomas, Ilya Sutskever et al. (2013). „Distributed Representations of Words and Phrases and their Compositionality“. In: *Advances in neural information processing systems* 26.

Pennington, Jeffrey et al. (2014). „GloVe: Global Vectors for Word Representation“. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, S. 1532–1543.

Sarica, Serhad und Jianxi Luo (2021). „Stopwords in technical language processing“. In: *Public Library of Science San Francisco, CA USA* 16 (8).

Rohde, Douglas LT et al. (2006). „An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence“. In: *Communications of the ACM* 8 (627–633), S. 116.

Joulin, Armand et al. (2016). „FastText.zip: Compressing Text and Classification Models“. In: *arXiv preprint arXiv:1612.03651*.

Bojanowski, Piotr et al. (2017). „Enriching Word Vectors with Subword Information“. In: *Transactions of the association for computational linguistics* 5, S. 135–146.

Qaiser, Shahzad und Ramsha Ali (2018). „Text mining: use of TF-IDF to examine the relevance of words to documents“. In: *International Journal of Computer Applications* 181 (1), S. 25–29.

Turau, Volker (2009). *Algorithmische Graphentheorie*. Oldenbourg Wissenschaftsverlag. ISBN: 9783486593778.

Stegbauer, Christian und Roger Häussling (2010). *Handbuch Netzwerkforschung*. Springer. ISBN: 978-3-531-92575-2.

Gumm, Heinz Peter und Manfred Sommer (2011). *Einführung in die Informatik*. 9. Aufl. Oldenbourg Wissenschaftsverlag. ISBN: 978-3-486-59711-0.

Turau, V. (1996). *Algorithmische Graphentheorie*. Addison-Wesley Deutschland.

Dijkstra, Edsger W et al. (1959). „A Note and on Two and Problems in Connexion with Graphs". In: *Numerische Mathematik* 1 (1), S. 269–271.

Needham, Mark und Amy E. Hodler (2018). *Graph Algorithms in Neo4j: Shortest Path*. Abrufdatum: 04.02.2022 17:21Uhr. URL: https://neo4j.com/blog/graph-algorithms-neo4j-shortest-path/.

Ge, Yuying et al. (2019). „DeepFashion2: A Versatile Benchmark for Detection, Pose Estimation, Segmentation and Re-Identification of Clothing Images". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, S. 5337–5345.

Aggarwal, Param (2019). *Fashion Product Image Dataset*. Plattform: kaggle.com, Abrufdatum: 27.01.2022 15:21Uhr. URL: kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset.

Sankesara, Heet (2018). *Flickr Image dataset*. Version 1, Plattform: kaggle.com, Abrufdatum: 13.01.2022 12:25Uhr. URL: https://www.kaggle.com/hsankesara/flickr-image-dataset.

Plummer, Bryan A et al. (2015). „Flickr30k Entities: Collecting Region-to-Phrase Correspondences for Richer Image-to-Sentence Models". In: *Proceedings of the IEEE international conference on computer vision*, S. 2641–2649.

Jia, Menglin et al. (2019). „The Fashionpedia Ontology and Fashion Segmentation Dataset". In: *Cornell University*.

Breitman, Karin Koogan et al. (2007). „Ontology in computer science". In: *Semantic Web: Concepts, Technologies and Applications*, S. 17–34.

Guia, José et al. (2017). „Graph Databases: Neo4j Analysis." In: *ICEIS (1)*, S. 351–356.

Miller, Justin J (2013). „Graph database applications and concepts with Neo4j". In: *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*. Bd. 2324. 23.

Holzschuher, Florian und René; Peinl (2013). „Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j". In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, S. 195–204.

Dobler, Gabriele und Michael Slopianka (1993). „Kommunikationsunterstützung für verteilte Transaktionen mit Echtzeitanforderungen". In: *Pearl 93*. Springer, S. 74–83.

Fernandes, Diogo und Jorge Bernardino (2018). „Graph Databases Comparison: AllegroGraph, ArangoDB, InfiniteGraph, Neo4J, and OrientDB." In: *Proceedings of the 7th International Conference on Data Science, Technology and Applications (DATA 2018)*, S. 373–380.

Francis, Nadime et al. (2018). „Cypher: An Evolving Query Language for Property Graphs“. In: *Proceedings of the 2018 International Conference on Management of Data*, S. 1433–1445.

Farda-Sarbas, Mariam und Claudia Müller-Birn (2019). „Wikidata from a Research Perspective - A Systematic Mapping Study of Wikidata“. In: *arXiv preprint arXiv:1908.11153*.

Veen, Theo van (2019). „Communications Wikidata: From “an” Identifier to “the” Identifier“. In: *Information technology and libraries* 38 (2), S. 72–81.

Miller, George A et al. (1990). „Introduction to WordNet: An on-line lexical database“. In: *International journal of lexicography, Oxford University Press* 3 (4), S. 235–244.

Miller, George A. (1995). „WordNet: a lexical database for English“. In: *Communications of the ACM, ACM New York, NY, USA* 38 (11), S. 39–41.

Lehnert, Gertrud (2014). *Mode: Theorie, Geschichte und Ästhetik einer kulturellen Praxis*. Bd. 1. transcript Verlag.

Henlein, Alexander (2021). *BertMaskFlask-Service*. URL: gitlab.texttechnologylab.org/henlein/bertmaskflaskservice.

Verdecchia, R. et al. (2021). „Green IT and Green Software“. In: *IEEE Software*. Bd. 38. 6. IEEE, S. 7–15.

Murugesan, San und G. R. Gangadharan (2008). *Harnessing Green IT : Principles and Practices*. Bd. 10. 1. 24–33: IT professional, IEEE. ISBN: 978-1-1199-7005-7.

Jivani, Anjali Ganesh et al. (2011). „A comparative study of stemming algorithms“. In: *Int. J. Comp. Tech. Appl* 2 (6), S. 1930–1938.

Mekala, Dheeraj et al. (2021). „Coarse2Fine: Fine-grained Text Classification on Coarsely-grained Annotated Data“. In: *arXiv preprint arXiv:2109.10856*.

A. Verzeichnis der benutzten Hilfsmittel

Im Folgenden werden alle im Rahmen dieser Arbeit verwendeten Hilfsmittel aufgelistet. Sofern keine Lizenzinformationen bezüglich der verwendeten Datensätze verfügbar waren, wurden diese über die folgenden Rechtsgrundlage genutzt.

Rechtsgrundlage

Inwieweit Text und Data-Mining (kurz: TDM) rechtlich zulässig ist, wird durch das Urheber- und Datenschutzrecht bestimmt. Grundsätzlich ist Forschungsinstituten, die nicht gewinnorientiert handeln, Text- und Data Mining erlaubt, sofern die Daten mit Einwilligung des Urhebers zur Verfügung gestellt wurden oder frei (im Internet) zugänglich sind (§ 60d UrhG: privilegierte Nutzung für Forschungszwecke).

Die so erhobenen Daten können von Forschenden auch an beauftragte Dritte (z.B. Studierende oder Mitarbeitende der Bibliothek) weitergegeben werden. Die Auswertung der im Korpus gesammelten Daten ist nicht mehr urheberrechtsrelevant, also zulässig, sofern sie zu wissenschaftlichen Zwecken erfolgt.

Die Daten sind derzeit nach dem Urheber- und Datenschutzrecht zu löschen, sobald der Forschungszweck erfüllt ist. Allerdings muss es Forschungseinrichtungen möglich sein, Forschungsergebnisse i.S.d. Leitlinie 17 der DFG-Standards aufzubewahren, so dass hier Ausnahmen zum Löschungsgrundsatz aus §§ 60d, 60f, 60g UrhG hergeleitet werden.

Verwendete Programme

- Alle in dieser Arbeit erstellten Abbildungen, die keinen direkten Quellenverweis in der Bildunterschrift enthalten wurden mithilfe der browserbasierten Version der Open-Source-Software draw.io (http://draw.io) erstellt,
  EULA: https://desk.draw.io/support/solutions/articles/16000039574-draw-io-eula-terms-of-service.

- Für die Entwicklung der, in der Programmiersprache Python, programmierten Skripte und Programme wurde die Entwicklungsumgebung *PyCharm*.
  (https://www.jetbrains.com/de-de/pycharm/) in der Community-Version verwendet,
  EULA:
  https://sales.jetbrains.com/hc/de/articles/115001015290-Wo-finde-ich-die-Endbenutzer-Lizenzvereinbarung-EULA-.

- Für die Erstellung der Graphdatenbank wurde das Programm Neo4J () verwendet, EULA: https://neo4j.com/licensing/, Lizenz: GNU GPLv3.

Datensätze

- *DeepFashion2* https://github.com/switchablenorms/DeepFashion2, Lizenz: siehe Rechtsgrundlage.
- *Fashion Product Image Dataset* https://www.kaggle.com/datasets/paramaggarwal/fashion-product-images-dataset, Lizenz: siehe Rechtsgrundlage.
- *Fashion Clothing Products Dataset* https://www.kaggle.com/datasets/shivamb/fashion-clothing-products-catalog, Lizenz: CC0: Public Domain.
- *Fashionpedia* https://fashionpedia.github.io/home/Fashionpedia_download.html, Lizenz: Creative Commons Attribution 4.0 License.
- *FlickR Image dataset* https://www.kaggle.com/datasets/hsankesara/flickr-image-dataset, Lizenz: CC0: Public Domain.
- *List of Colors (wikipedia)* https://github.com/codebrainz/color-names, Lizenz: GPL v2.0.

Weitere Ressourcen

- WordNet (https://wordnet.princeton.edu/), Lizenzbestimmungen: WordNet 3.0 license, https://wordnet.princeton.edu/license-and-commercial-use.
- wikidata (https://www.wikidata.org/wiki/Wikidata:Main_Page), Lizenz: Creative Commons CC0.
- BERT (https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html), Lizenz: Apache License Version 2.0.
- fastText (https://fasttext.cc/), MIT License.
- GLove, https://nlp.stanford.edu/projects/glove/, Lizenz: Apache License Version 2.0.
- Word2Vec, wurde über die *gensim*-API bezogen, Lizenz: https://github.com/RaRe-Technologies/gensim-data/blob/master/LICENSE.

B. Ordnerstruktur des USB-Sticks bzw. Github

Alle relevanten Dateien sind auf dem beigelegten USB-Stick sowie über die GitHub über Github-Hauptordner erreichbar. Dabei sind jedem Ordner *readMe.txt*-Dateien mit Informationen über Abhängigkeiten, Installation und Ausführung beigefügt.

Ordnerstruktur

- Mehrklassen-Klassifizierung (Abschnitt 4.2, Abschnitt 5.1)
  - category_model
  - Text-Vorverarbeitung
- Neo4J-Skripte (Abschnitt 4.3)
  - 1 - Startknoten
  - 2 - Wordnet (Hypernomie,Hyponymie,Meronymie, POS-Tag)
  - 3 - Wikidata (Alias, Beschreibung)
  - 4 - Farben
  - 5 - Fashionpedia
  - 6 - GLove
  - 7 - Word2Vec
  - 8 - FastText
  - 9 - Statische Modelle
  - 10 - BertMaskFlask + IDF
  - CSV_raw
  - Datenbank_Dump
- Verworfene Ansätze (Abschnitt 4.5)
  - DeepFashion2 - ImageRecommendation-Tool
  - Wikidata-Hierarchie / Klassifizierung von Kleidungsstücken
- Programm zur Detektion von Kleidungsstücken (Abschnitt 4.4, Abschnitt 5.2)

C. Funktionsbeschreibungen aus dem Programm zur Detektion von Kleidungsstücken

Funktion: *get_nodeData(token)*

Aufgabe: Fragt zu einem Knotennamen alle Eigenschaften des Knotens ab.

Rückgabewert: Eigenschaften des Knoten.

Parameter: Knoten.

CYPHER-Statement:

```
MATCH (n) WHERE n.name = token RETURN n
```

Funktion: *get_aliases(token)*

Aufgabe: Fragt zu einem Knotennamen alle Alias-Bezeichner ab.

Rückgabewert: Liste von Alias-Bezeichnern.

Parameter: Token.

CYPHER-Statement:

```
MATCH (n)-[]-(a:ALIASES) WHERE n.name = token RETURN
↪  n.name AS source, a.name AS target
```

Funktion: *get_nodeLabel(token)*

Aufgabe: Gibt zu einem Token-Knoten das entsprechend hintelegte *Label* (Klasse) aus.

Rückgabewert: *Label*-Name.

Parameter: Token.

CYPHER-Statement:

```
MATCH (n {name: token}) RETURN labels(n)
```

Funktion: *get_shortestPathToClothes(source)*

Aufgabe: Berechnet den kürzesten Weg zwischen einem *source*-Knoten und einem Knoten mit dem *Label:clothing*.

Rückgabewert: Pfad-Objekt.

Parameter: Startknoten.

CYPHER-Statement:

```
MATCH p=shortestPath((selectedNode
↪ {name:source})-[*]-(y:clothing)) RETURN length(p)
↪ AS l, collect(y.name) as targets ORDER BY l LIMIT 1
```

Funktion: *get_shortestPathToClothesWithoutBERT(source)*

Aufgabe: Berechnet den kürzesten Weg zwischen einem Startknoten und einem Knoten mit dem *Label:clothing*, wobei der Pfad über keine Relationen gehen darf, die mit Hilfe des BERT-Modells berechnet wurden.

Rückgabewert: Pfad-Objekt.

Parameter: Startknoten.

CYPHER-Statement:

```
MATCH p=shortestPath((selectedNode
↪ {name:source})-[*]-(y:clothing)) WHERE NONE (r in
↪ relationships(p)  WHERE type(r)=BMF_IDF) RETURN
↪ length(p) AS l, collect(y.name) as targets, [n in
↪ nodes(p) | n.name] as arrayOfName, [r in
↪ relationships(p) | type(r)] as arraOfRels ORDER BY l
↪ LIMIT 1
```

Funktion: *get_shortestPathToClothesOnlyBERT(source)*

Aufgabe: Berechnet den kürzesten Weg zwischen einem Startknoten und einem Knoten mit dem *Label:clothing*, wobei der Pfad über keine Relationen gehen darf, die durch statische Modelle berechnet wurde.

Rückgabewert: Pfad-Objekt.

Parameter: Startknoten.

CYPHER-Statement:

```
MATCH p=shortestPath((selectedNode
↪ {name:source})-[*]-(y:clothing)) WHERE NONE (r in
↪ relationships(p)  WHERE type(r)=GLove_IS_SIMILAR_TO
↪ OR type(r)=FastText_IS_SIMILAR_TO OR
↪ type(r)=W2C_IS_SIMILAR_TO OR
↪ type(r)=FIVE_NEAREST_vector_based) RETURN length(p)
↪ AS l, collect(y.name) as targets, [n in nodes(p) |
↪ n.name] as NodePath, [r in relationships(p) |
↪ type(r)] as RelationshipPath ORDER BY l LIMIT 1
```

Funktion: *get_clothingDetails(token)*

Aufgabe: Gibt zu einem Token-Knoten Informationen über Altersgruppen- und Geschlechtszuordnung sowie zu Haupt- und Subkategorie aus.

Rückgabewert: Altersgruppen- und Geschlechtszuordnung, Haupt- und Subkategorie.

Parameter: Token.

CYPHER-Statement:

```
MATCH (n {name:token}) RETURN n.name AS Clothing,
↪ n.masterCategory AS Category, n.subCategory AS
↪ SubCategory, n.ageGroup_detailed AS AgeGroup,
↪ n.gender_detailed AS Gender
```