DEDUPLIZIERUNG DURCH KÜNSTLICHE NEURONALE NETZE

GEORGES ALKHOURI

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

an der

Fakultät Informatik, Mathematik und Naturwissenschaften
Hochschule für Technik, Wirtschaft und Kultur

Betreut von

Prof. Dr. Riechert

Dr. Eric Peukert

Leipzig, den 06. Februar 2017

ZUSAMMENFASSUNG

Eine Deduplizierung versucht Einträge aus einer oder mehrerer Datenquellen, welche das selbe logische Objekt beschreiben, zu finden. Dabei wird die Ähnlichkeit solcher Einträge über Zeichenkettenvergleiche ermittelt. Diese Vergleiche können demnach nur strukturelle Informationen verwerten und beziehen keine semantischen Relationen zwischen den zu vergleichenden Daten ein.
Das Ziel dieser Arbeit ist ein Verfahren vorzustellen, welches den inhaltlichen Kontext der Daten akquiriert und in den Deduplizierungsprozess einfließen lässt. Ermöglicht wird dies durch die Nutzung von künstlichen neuronalen Netzen auf dem Gebiet des maschinellen Lernens und der statistischen Sprachmodellierung.
Getestet wird das Verfahren durch eine Deduplizierung von real existierend Produktdaten.
Die Ergebnisse der Deduplizierung konnten nur teilweise im Vergleich zu anderen Arbeiten überzeugen.
In dieser Arbeit wurde jedoch ein Grundstein für die tiefe Einbettung maschineller Lernverfahren in den Deduplizierungsprozess gelegt.

Machines take me by surprise with great frequency.

— Alan M. Turing [46]

DANKSAGUNG

Bedanken möchte ich mich bei Dr. Eric Peukert für die Unterstützung durch zahlreiche Gespräche und Ansporn. Weiter danke ich Prof. Dr. Riechert, der mir keine Einschränkung und volle Freiheit in der Bearbeitung meines Themas ließ.
Natürlich habe ich auch meiner Freundin, Patricia Dittombée, zu danken, da sie mir oft an langen Tagen mit wenig Zeit für anderes, den Rücken stärkte.
Zu guter Letzt wäre mein ausgiebiges Studium nicht ohne die uneingeschränkte Unterstützung meiner Eltern möglich gewesen.

Danke.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

LISTINGS

ABKÜRZUNGSVERZEICHNIS

AE	Autoencoder
AG-D	Amazon-Google-Datensatz
AM-D	Amazon-Meta-Datensatz
API	Application Programming Interface
BLAS	Basic Linear Algebra Subprogram
BOW	Bag-of-Words
CBOW	Continuous Bag-of-Words
DL4J	Deeplearning4j
DL	Deep Learning
ER	Entity Resolution
KNN	künstlich neuronales Netz
LSTM	Long Short Term Memory
ML	Machine Learning
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NNLM	Neural Net Language Model
NUMA	Non-Uniform Memory Access
OAEI	Ontology Alignment Evaluation Initiative
PCA	Hauptkomponentenanalyse
PV-DBOW	Distributed Bag-of-Words of Paragraph Vectors
PV-DM	Distributed Memory Model of Paragraph Vectors
RNN	Recurrent Neural Network
ReLU	Rectified Linear Units
SAE	Stacked Autoencoder
SVM	Support Vector Machine
ScaDS	Competence Center for Scalable Data Services and Solutions

TF TensorFlow

VAE Variational Autoencoder

VSM Vector Space Model

1 EINLEITUNG

In der heutigen Zeit ist ein stetiger und steigender Bedarf an Daten zu beobachten. Verdeutlicht werden kann dies z.B. durch die Betrachtung der weltweiten verfügbaren Speicherkapazitäten. Zahlen von bis zu 295 Exabytes (10^{18}) werden für das Jahr 2011 genannt [19]. Weiter belaufen sich Schätzungen für die Menge an erstellten Information auf ca. 1000 Exabytes im Jahr 2010 [8].
Durch das Aufkommen von Big Data und einer Vielzahl von Quellen für die Erhebung von Daten entstehen im Jahr 2015 allein 2.5 Exabytes an Daten pro Tag [29]. Damit ist der Trend im Vergleich zu 2010 stark gestiegen.
Die gesammelten Daten werden u.a. in riesigen Open-Data-Sammlungen, wie Data.gov, Open Science Data Cloud oder datahub.io bereitgestellt. Außerdem befinden sich im Internet Millionen von Daten in Tabellenform.

Für Wirtschaft und Forschung besteht immer mehr das Interesse einen einheitlich Zugriff auf heterogene Daten aus möglicherweise verschiedenen Datenquellen zu erlangen. Das Gebiet der Informationsintegration liefert verschiedene Ansätze wie eine physikalische Informationsintegration u.a. mittels Data Warehousing, aber auch virtuelle Integrationen bei denen die Daten in ihrer ursprünglichen Datenquelle verbleiben.

Die wichtigsten Aufgaben einer Datenintegration unterteilen sich in die Datenvorverarbeitung, die Entity Resolution und die Zusammenlegung von Objekten [42].
In dieser Arbeit wird der Versuch unternommen den Entity-Resolution-Prozess mit Hilfe von maschinellen Lernalgorithmen zu Lösung und ggf. zu verbessern. In der Entity Resolution wird versucht Einträge aus einer oder mehrere Datenquellen, welche das selbe logische Objekt beschreiben, zu finden.
Ironischerweise weist der Entity-Resolution-Begriff selbst eine Vielzahl an redundanten Bezeichnungen auf. In der Literatur wird Entity Resolution auch oft als Record Linkage oder wie in diesem Fall als Deduplizierung bezeichnet.

Um den Vorgang der Deduplizierung zu verbessern worden in den letzten Jahren bis dato genutzte regelbasierte Verfahren immer mehr mit Verfahren, welche durch Machine Learning (ML)-Algorithmen gestützt werden, ersetzt.

Überwiegend werden ML-Algorithmen als Klassifizierungsmodelle genutzt, die auf Grundlage von Ähnlichkeitswerten entscheiden, ob Einträge gleich oder verschieden sind. Hinzu kamen in jüngster Vergangenheit auch sogenannte künstlich neuronale Netze (KNNs), um diese Klassifizierungen zu verbessern [47].
Diese der Biologie nachempfundenen und informationsverarbeitenden Modelle fanden seit 2009 ihre Renaissance. Seit dieser Zeit wurden durch die, auch als Deep Learning (DL) bekannte, Technik zahlreiche Wettbewerbe in Bereichen wie Mustererkennung und ML gewonnen.
Im Verlauf der Arbeit soll gezeigt werden, wie ML tief in den Deduplizierungsprozess durch künstlich neuronale Netze eingebettet werden kann.

Die vorliegende Arbeit gliedert sich in sieben Teile.
Im weiteren Verlauf dieser Einleitung wird das Problem der Deduplizierung vorgestellt und die wichtigsten Problemstellungen der Arbeit definiert.

Kapitel 2 vermittelt Grundlagen, zu den in der Arbeit verwendeten Machine-Learning-Begriffen, sowie einen detaillierten Einblick in die Funktionsweise von künstlichen Neuronen und Netzen.

Kapitel 3 stellt verwandte Arbeiten vor und führt einen Vergleich zwischen aktueller Deep Learning-Software druch, mit dem Ziel eine Anwendung, für das Lösen der Problemstellungen, zu finden.

Kapitel 4 und **Kapitel 5** führen ein Konzept ein, welches eine Deduplizierung von Produktdaten ohne Zeichenkettenvergleiche ermöglicht. Anschließend werden detaillierte Schritte zur Umsetzungen erläutert.

Kapitel 6 führt die verwendeten Daten der Evaluierung ein und betrachtet im Anschluss die Resultate.

Kapitel 7 fasst abschließend die Ergebnisse der Arbeit zusammen und diskutiert im Ausblick verschiedene Ideen zur Verbesserung des Konzepts.

1.1 DEDUPLIZIERUNG AM BEISPIEL VON DEDOOP

In diesem Abschnitt soll der Deduplizierungsvorgang und damit die Problemstellung der Arbeit vorgestellt werden. Als Beispiels- und Vergleichsapplikation dient die von von der Universität Leipzig ent-

Titel	Hersteller	Beschreibung	Preis
cook'n vegetarian	dvo enterprises	featuring recipes for tasty and healthful meatless dishes made with fresh ingredients the cook'n veg ...	19.99
cookn vegetarian	—	the diet choice for many world cultures for millennia vegetarian cuisine is widely recognized as inn ...	19.99

Tabelle 1: Zwei gleiche Produkte aus zwei unterschiedlichen Datenquellen.

wickelte Anwendung Dedoop. Anschließend erfolgt eine Problemanalyse für die Verwendung von Textdaten mit einer Klassifizierung der auftretenden Datenverunreinigungen.

Dedoop stellt ein Werkzeug zur Deduplizierung von großen Datenquellen dar [26]. Es basiert auf dem Apache Hadoop Framework, welches die verteilte Verarbeitung von Datenmengen im Big-Data-Kontext ermöglicht.

Die Deduplizierung versucht Tupel als Duplikate oder Nicht-Duplikate zu Identifizieren. Dabei besteht das Problem, dass ein Tupelelement a syntaktisch anders strukturiert ist als ein Tupelelement b, beide jedoch das gleiche Realweltobjekt beschreiben [33, S. 329.]. Der Vorgang der Deduplizierung ist ein Bestandteil der Informationsintegration. Beispielsweise treten oft Duplikate bei der Zusammenführung von unterschiedlichen Systemen auf [33, S. 329.].

Üblicherweise werden die Tupelelemente aus zwei Datenquellen zusammengesetzt. Ein Tupel (a, b) mit $a \in A \wedge b \in B$, wobei A und B unterschiedliche Datenquellen darstellen, setzt sich aus einem Datensatz a und b zusammen.
Dabei ist ein Datensatz $a = (A_0, v_0), \ldots, (A_n, v_n)$ als Menge seiner Attribut-Werte-Paare definiert [25, S. 18.].

Zur Verdeutlichung des Problems sei jewels ein Datensatz aus zwei Produktdatenbanken in Tabelle 1 gegeben. Beide Einträge beschreiben nun das gleiche Produkt, bauen sich strukturell jedoch anders auf.
Das Titelattribut der zwei Datensätze unterscheidet sich nur in einem Zeichen, wohingegen das Beschreibungsattribut völlig unterschiedli-

che Texte umfasst, die sich jedoch inhaltlich aufeinander beziehen.
Zu sehen ist auch, dass nicht zwingend alle Attribute besetzt sein müssen, wie das Herstellerattribut zeigt, welches nur in einem Datensatz einen Wert besitzt.

Da es sich in diesem Fall um den Vergleich von ausschließlich Textdaten handelt wird durch die Anwendung verschiedener Zeichenkettenmetriken, während des Deduplizierungsvorgang versucht ein Ähnlichkeitswert für jedes Attributpaar zu bestimmen. Anhand der ermittelten Ähnlichkeiten wird daraufhin eine Aufteilung der Tupel in Match oder non-Match-Paare vorgenommen.
Ein Match-Paar enthält zwei logisch gleiche Objekte, ein non-Match-Paar dementsprechend keine.

Eine intuitive Vorgehensweise, um nach Möglichkeit alle Match-Paare zu erkennen, ist der Vergleich von allen Tupelkombinationen zweier Datenquellen, um diese in zwei disjunkte Mengen bzw. Kategorien zu unterteilen. Die Umsetzung erfolgt mittels der Bildung des kartesischen Produktes zweier Mengen A und B.
Formal wurde die Aufteilung der beiden Mengen von [13], wie folgt beschrieben:

$$M = \{(a, b); a = b, a \in A, b \in B\}$$
$$U = \{(a, b); a \neq b, a \in A, b \in B\}$$

Dabei definiert M, die Menge aller Match-Paare (engl. matched) und U, die Menge aller non-Match-Paare (engl. unmatched).

Dem kartesischen Produkt geschuldet, entsteht bei dem Vergleich von zwei Datenquellen eine quadratische Laufzeit von $O(|A| * |B|)$ [25, S. 3.].
Die während der Arbeit verwendeten Datenquellen beinhalten lediglich 3 226 und 1 363 Datensätze. Bei einer Deduplizierung dieser Datenquellen müssen 4 401 628 Vergleiche durchgeführt werden, wobei jeder Vergleich rechenintensive Ähnlichkeitsberechnungen einschließt.
In der realen Welt existieren weitaus mehr Objekte in einer Datenquelle. Alleine Amazon führt über 12M Produkte [1].
Somit kann das Verfahren gerade im Kontext von Big Data nicht mit größeren Datenquellen skalieren. [25, S. 20.]

In Abbildung 1 ist der schematische Ablauf einer Deduplizierung mittels Dedoop zusehen. Hierbei wird der Prozess in vier grundsätzliche Schritte unterteilt:

PREPROCESSING Aufgabe der Vorverarbeitungsphase ist es Datensätze zu strukturieren, da meist in großen Datenmengen nicht alle Attribute mit einem Wert belegt sind. Weiterhin ist es bei

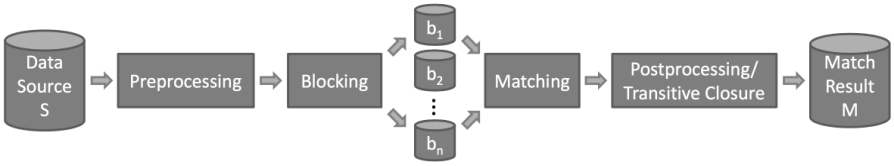


Abbildung 1: Vereinfachter Ablauf eines Entiy-Resolution-Prozesses [25, S. 13.].

der Vorverarbeitung von Textdaten üblich, diese von Stopwörtern zu bereinigen, sowie Satzzeichen oder Whitespaces zu entfernen [25, S. 19 f.]. Weiterhin können Wörter welche keine Eigennamen darstellen auf ihren Wortstamm zurückgeführt werden, damit die Anzahl der verschiedenen Wortarten gering bleibt. Eine genauere Betrachtung folgt in Abschnitt 1.2.1.

BLOCKING Der Blocking-Vorgang versucht große Teile aus der Menge der non-Matches von einer Ähnlichkeitsberechnung auszuschließen, ohne dabei fälschlicherweise Machtes zu entfernen. [25, S. 20.] Weiter ist durch die große Anzahl an zu vergleichenden Elementen eine Aufteilung der Tupel in verschiedene Blöcke zwingend nötig, um eine Deduplizierung in realistischer Zeit durch eine parallele Abarbeitung zu ermöglichen.

MATCHING Nach der Ermittlung einer Menge potentieller Match-Kandidaten C findet der paarweise Vergleich von Attributwerten aller Tupel $(a, b) \in C$ statt.
Die Zeichenkettenähnlichkeit kann unter anderem durch Levenstein, Jaccard oder N-Gram-Ähnlichkeit berechnet werden, um nur einige Verfahren zu nennen. Der daraus resultierende Ähnlichkeitswert wird im Intervall $[0, 1]$ normiert, wobei null absolut nicht ähnlich und eins totale Gleichheit bedeutet.
Als Ergebnis liefert der Matching Schritt für jeden Vergleich ein Triple (a, b, s) zurück, wobei $(a, b) \in C$ und $s \in [0, 1]^n$ ist. s ist also der Ähnlichkeitsvektor von n Ähnlichkeitsmetriken ist [25, S. 19 f.].
Auf Grundlage von s muss dann eine Klassifizierung in Match oder non-Match stattfinden. Hierfür existieren unterschiedliche schwellwertbasierte oder regelbasierte Verfahren [25, S. 27.].

POSTPROCESSING Der Nachbearbeitungsschritt dient der Verbesserung der Match-Ergebnisse. Beispielsweise kann eine transitive Hülle gebildet werden, um logische Widersprüche in den Ergebnismengen M und U zu finden. Eine transitive Relation besagt, dass, wenn die Tupelpaare (a, b) und (b, c) Matches sind, ein Tupel (a, c) ebenfalls ein Match-Paar ist, selbst wenn es nicht in der Menge M existiert. [25, S. 29.]

Neben der oben beschriebenen, allgemeinen Vorgehensweise der Deduplizierung implementiert Dedoop Methodiken, um eine Klassifizierung mittels maschinellen Lernverfahren zu ermöglichen.
Verwendung finden unter anderem eine Support Vector Machine (SVM) oder ein Decision Tree, auf die hier nicht weiter eingegangen werden kann.

1.2 PROBLEMSTELLUNG

P1: Ziel dieser Arbeit ist es einen Ablauf zu entwickeln, welcher die bisherige Deduplizierung unterstützt und nicht auf dem Vergleich von Zeichenketten basiert, sondern die Semantik innerhalb der Produktattribute darstellen kann und diese vergleichbar macht.

Hierfür sollen maschinelle Lernverfahren benutzt werden. Genauer kommen KNNs im Kontext der statistischen Sprachmodellierung und des DLs zum Einsatz, mit dem Ziel Produktdaten in eine Vektorrepräsentation zu transformieren. Diese kann dann mit Ähnlichkeitsmetriken für die Klassifizierung in Match- und non-Match-Paare genutzt werden. Besonders ist hier, dass semantische Bedeutung in den Attributtexten für einen Ähnlichkeitsvergleich verwerten werden. Dadurch könnten auch Matches von Produkten gefunden werden, die syntaktisch anderes aufgebaut sind, aber trotzdem gleich, weil semantisch korreliert sind.

Für das Lösen von *P1* stehen keine besonderen Daten zur Verfügung. Ausnahme bilden die Daten, welche für die Evaluierung der Lösungsstrategie vorgesehen sind und daher nicht direkt für die Lösung von *P1* verwertet werden können.
Daher impliziert *P1* eine weitere Problemstellung.

P2: Die zu benutzenden Algorithmen und Verfahren dürfen kein Vorwissen über ihre verwendeten Daten verlangen. Im Kontext von ML bedeutet dies, von der Nutzung gelabelter Daten, also von Daten welche gezielt ausgesucht und mit Information angereichert wurden, abzusehen.

Weitere Problemstellungen

Da dieser Arbeit ausschließlich Textdaten als Informationsquellen dienen, muss deren Bearbeitung und Manipulation detaillierter betrachtet werden, da sonst die Gefahr besteht die abschließenden Ergebnisse zu verfälschen.

1.2.1 *Linguistische Vorgedanken*

Eine Vorverarbeitung der Textdaten unter linguistischen Aspekten ist unumgänglich, da menschliche Sprache verschiedenste Merkmale besitzt, die eine automatische Verarbeitung erschweren beziehungsweise verzerren können [34, S. 123.]. Folglich sollen Überlegungen über die Behandlung der wichtigsten Merkmale betrachtet werden.
Der Schwerpunkt der Voruntersuchungen liegt hauptsächlich auf dem englischen Sprachgebrauch, jedoch lassen sich Teile auch in anderen Sprachen, wie deutsch verwenden.

***P3**: Formatierung*

Betrachtet wird folgende Fragestellung: Wie sollen Wörter behandelt werden, deren Zeichen identisch sind, sich jedoch in der Groß- und Kleinschreibung des Anfangsbuchstaben unterscheiden? In den meisten Fällen soll zum Beispiel den Worten *the*, *The* oder *THE* eine gleiche Bedeutung zu kommen. Verhindert werden kann dieser Umstand durch die einheitlich Konvertierung aller Buchstaben in Klein -bzw. Großbuchstaben. Trotzdem führt dies zu der Eliminierung von Eigenamen, wie im Beispiel *Richard* ***Brown*** zu der Farbe ***brown***.

Eine einfache Heuristik kann besagen, dass groß geschriebene Wörter an bestimmten Stellungen im Satz (z.B. Satzanfang) oder Überschriften klein geschrieben werden, um somit nur echte Namen mit großen Anfangsbuchstaben zu belassen [34, S. 124.]. Daraus entsteht das Folgeproblem der korrekten Erkennung eines Satzendes, dass hier nicht weiter betrachtet werden soll.

***P4**: Tokenisierung*

Eine weitere Aufgabe der Vorverarbeitung von Textdaten ist es Wörter und anderen Zeichen in einzelne Segmente zu unterteilen. Jedoch kann die einfache Fragestellung, was ein Wort ist, nicht trivial beantwortet werden. Im Bereich der Linguistik sind z.T. verschiedene Wortebenen, wie die phonetische und die syntaktische definiert [34, S. 124.].
Aus Sicht der Informationsverarbeitung könnte man sich auf eine Definition beschränken, die ein Wort als Zeichenkette mit nacheinander auftreten alphanumerischen Zeichen und mit Leerzeichen an Anfang und Ende auffasst. Weiterhin könnte eine solche Wortzeichenkette Apostrophe und Bindestriche, aber keine weitere Punktuation enthalten.
Allerdings existieren zu viele Ausnahmen und Unregelmäßigkeiten für eine optimale Anwendung eben genannter Auffassung von Worten. Als Beispiele seien folgende Zusammensetzungen von Zeichen

gegeben, die weder klassische Worte darstellen aber dennoch Informationsgehalt besitzen und deswegen nicht ignoriert werden sollten.

- $22.50
- Micro$oft
- C | net (Name einer Webfirma)

In der englischen Sprache ist eine Segmentierung in einzelne Wörter deshalb durch das Auftreten von Leerzeichen bestimmt.
Die Trennung von Worten mittels Leerzeichen ist schwierig, da Wörter auch oft durch Satzzeichen (Kommas, Punkte, etc) getrennt werden. Auch das Entfernen von Punktuation ist kein triviales Problem, da ein Punkt nicht zwingend einen Satz abschließen muss, sondern auch für eine Abkürzung stehen kann. In manchen Fällen kann nur der Punkt einer Abkürzung ein Wort von einem anderen, identisch geschriebenen Wort unterscheiden [34, S. 125.].
Generell ist eine Wortsegmentierung mittels Leerzeichen stark von der betrachteten Sprache abhängig. Im japanischen oder chinesischen ist dies nicht möglich, da hier Wörter nicht per Leerzeichen wie im deutschen oder englischen getrennt werden [34, S. 129.]. Auch bei der Tokenisierung von Texten in denen Produktbezeichnungen oder Herstellernamen vorkommen, ist die Aufteilung der Wörter nicht zwingend Intuitiv. Produktpreise können in den unterschiedlichsten Formatierungen vorkommen. Des Weiteren sind Produktnamen oder Herstellernamen oft Eigennamen, die sich über mehrere Wörter strecken und Nummern bzw. Sonderzeichen enthalten können.

***P5**: Morphologie*

Wörter können auf ihre Lexeme, also ihre sprachliche Bedeutungseinheit zurückgeführt werden, um u.a die statische Signifikanz der Wörter durch die Erhöhung des Vorkommens zu steigern. (singen, singt, singe $\rightarrow$ sing)

Empirische Forschungen im Bereich des Information Retrieval haben gezeigt, dass dieser Vorgang nicht zwingend die Performanz verbessert. Die Variation der Ergebnisse kommt u.a durch die Eliminierung von Informationen bei der Zurückführen eines Wortes auf seinen Wortstamm zustande. Mittels der Tokenisierung wurden Wörter auseinander segmentiert und dadurch separat umgewandelt [34, S. 131f.].
Dementgegen sollten informative Wortgruppen besser als ganzes betrachtet werden.

In Produktattributen (Titel, Beschreibung) sind häufig Neologismen

kombiniert mit Versionierungen enthalten. Diese drücken, wenn als ganze Entität belassen, mehr Informationen über das Produkt aus. Auch können solche Eigennamen schlecht auf einen Wortstamm zurück geführt werden und sollten es auch nicht. Gerade bei nicht trivialen Produktvergleichen, wie z.B. den Vergleich von Produkten der selben Familie mit inkrementierer Version, fallen Eigennamen mit signifikantem Gewicht in den Vergleich.
Als Beispiel zwei Attribute mit Werten aus zwei Produkten:

TITEL iPhone 5s – iPhone 6

HERSTELLER Apple – Apple

Da sich Attribute, wie Hersteller oder Title nur minimal unterscheiden, können beispielsweise semantische Inhalte, die überwiegend im Beschreibungstext enthalten sind, nutzbar gemacht werden und dadurch den Vergleich verbessern.

Die Auswirkung der verschiedenen Vorverarbeitungen ist in einer Evaluation zu überprüfen. In [43] konnte gezeigt werden, dass vorverarbeitende Maßnahmen bei herkömmlichen Deduplizierungen keine signifikanten Verbesserungen erzielen müssen. Eher konnte festgestellt werden, dass anschließende Evaluierungen sich verschlechterten.

1.2.2 *Datenverunreinigung*

Große Teile der für die ML-Algorithmen verwendeten Trainingsdaten stammt aus der Extraktion von Webseiten. Daher ist zu überlegen welche Fehler oder Verunreinigung dabei entstehen können, um diese möglicherweise zu beheben. Hierdurch soll das Verfälschen der Ergebnisse verhindert und der Lernerfolg der Algorithmen maximiert werden.
Im Folgenden wird versucht durch die Taxonomie von [23] diese Verunreinigungen einzuordnen.

***P6**: Fehlende Daten, in denen es keine "Null-not-allowed" Einschränkung gibt*

Dies bedeutet, dass eine Nicht-Existenz von Werten der Produktattribute möglich ist und damit das Attribut nicht im Produkt vorkommt. Dieses Kriterium kann sowohl innerhalb der Trainingsdaten, als auch in den Daten der Evaluierung auftreten.

***P7**: Kodierungsformate*

Geschuldet der Tatsache, dass die Trainingsdaten aus HTML-Code extrahiert wurden, enthalten diese HTML-Kodierungen wie z.B. &#38

für das Et-Zeichen.
Als Beispiel sei der folgende Text genannt:

‘An absolutely gripping, absorbing historical crime thriller …

***P8**: Nutzung von Sonderzeichen*

Hiermit ist das Vorhandensein von CSS-Codefragmenten und anderen HTML-Artefakten in den extrahierten Textdaten zu verstehen. Dies ist der schwerwiegendste Grad der Verunreinigung. Des Weiteren enthalten manche Daten lediglich URL- oder Pfadangaben.
Dazu mehrere Beispiele für mögliche Attributwerte:

- `.` (*Anmerkung*: Pfadangabe)
- `https://www.amazon.de/dp/B01M991Z2P/ref=cm_sw_r_tw_dp_x_DeBKyb4TZJ88F`
- `#jl_box{border:1px solid #ccc;box-shadow:10px 10px 5px #ebebeb;}` ...
- `Your adventurous child..caption{font-family: Verdana} The Kolcraft` ...

Nicht betrachtete Problemstellung

Wie in der Vorgangsbeschreibung der Deduplizierung erwähnt wurde, skaliert die Laufzeit sehr schlecht mit der steigenden Anzahl an Daten. Jedoch liegt der Fokus der Arbeit auf einem neuartigen Vergleich von Produkten und nicht auf der Parallelisierung des Arbeitsablaufs der Deduplizierung. Somit wird dieser Aspekt nicht in einer Problemstellung thematisiert.
In [25] werden effiziente Möglichkeiten für den Ablauf einer Deduplizierung in Dedoop vorgestellt.

2 EINFÜHRUNG: DEEP LEARNING

Ziel dieses Kapitels soll es sein ML und DL in seiner grundsätzlichen Funktionsweise einzuführen, jedoch wird der Fokus auf die verwendeten DL-Konzepte der vorliegenden Arbeit gelegt.

Frühe Anwendungsgebiete künstlicher Intelligenz sahen die Lösung von strukturieren Problemen vor. Maschinen können mathematisch wohldefinierte Aufgaben problemlos lösen, wohingegen Menschen diese mit mehr intellektuellem Aufwand bearbeiten müssen. Es stellte sich heraus das Aufgaben, welche das menschliche Gehirn intuitiv löst, weitaus problematischer für eine maschinelle Verarbeitung sind. Problemstellungen, wie das Verstehen von gesprochenen Wörtern oder der Gesichtserkennung, lassen sich nur schwierig formal beschreiben.

Um solche intuitiven Aufgaben lösen zu können, müssen Computer Verständnis und Erfahrungen in Form von aufeinander aufbauenden Konzepten entwicklen. Die Möglichkeit Konzepte hierarchisch zu gliedern, ermöglicht es Computern komplexere Konzepte aus simpleren zu bilden [18, S. 1 f.].

DL liefert Werkzeuge, um solche tiefen Hierarchien zu modellieren und nutzen zu können und dient somit der Lösung von Problemen im Gebiet der künstlichen Intelligenz. Das Gebiet ist eine Unterform des maschinellen Lernens. Die Stärken der Deep Learning Technologie bestehen in der Erzeugung von Problemmodellen in Form von aufeinander aufbauenden Konzepten und Bildung abstrakter Datenrepräsentationen aus konkreteren Repräsentationen. Die Anwendung solcher Modelle ist für *real world* Probleme geeignet [18, S. 8.]. Beispielsweise kann aus Bildern von menschlichen Gesichtern die generelle From eines Gesichts abstrahiert werden(ein Gesicht besteht aus Mund, Nase, Augen etc.).

2.1 MASCHINELLES LERNEN

Es existieren unterschiedliche Herangehensweisen für maschinelle Lernverfahren. Zwei Lernmethoden sind das überwachte (engl. supervised) und das unüberwachte (engl. unsupervised) Lernen.

Ein überwachter Lernvorgang bildet Eingabe- auf Ausgabewerte ab,

wobei alle korrekten Ausgabewerte bekannt sind. Eine klassische Problemstellung für überwachtes Lernen ist die der Klassifizierung. Anders ausgedrückt bedeutet dies

$$y = g(x; \theta) \text{ mit } x \in X \wedge y \in Y \tag{1}$$

wobei Y die Menge aller Klassen, X die Menge aller Trainingsdaten und θ die Menge aller Parameter darstellt.
Die Eingabedaten x werden auch als markierte (engl. labeled) Daten bezeichnet, da jede Eingabe mit der korrekte Ausgabe gelabelt ist.
Ziel ist es die Funktion $g(\cdot)$ durch ein maschinelles Lernmodell zu approximieren, sodass die Ausgabedaten möglichst korrekt klassifiziert werden. Dieser Vorgang wird als Training bezeichnet, in welchem die Modellparameter θ gelernt werden [2, S. 9ff.].

Im Gegensatz dazu stehen beim unüberwachten Lernen keine bekannten Ausgabewerte zur Verfügung. Intension bei diesem Verfahren ist das Finden von statistischen Regelmäßigkeiten oder Mustern im Eingaberaum X [2, S. 11]. In der Praxis stehen weit mehr nicht gelabelte Daten zur Verfügung.

Im Allgemeinen setzten sich ML-Algorithmen aus den folgenden drei Komponenten zusammen, welche damit den Lösungsvorgang eines maschinellen Lernproblems formen [18, S. 153 f., S.177 f.][2, S. 3.]:

1. MODELL Ein Konstrukt, was eine prädikative oder deskriptive Aussage über Daten treffen soll. Das Modell trifft Entscheidungen auf Grundlage seiner Parameter.

2. KOSTENFUNKTION Die Kostenfunktion (oder auch Verlustfunktion) vergleicht die Eingabe mit der berechneten oder bereitgestellten (gelabelte Daten) Ausgabe eines Modells und liefert einen Fehlerwert.

3. OPTIMIERUNGSVERFAHREN Ein Verfahren, welches den Verlauf bzw. Gradienten der Kostenfunktion auf ein Optimum hinbewegt, indem die Parameter des Modells angepasst werden. Dadurch wird der Fehlerwert minimiert oder ggf. maximieren.

Die Möglichkeit einzelne Verfahren aus diesen Komponenten auszutauschen, erlaubt eine große Variation an anderen ML-Algorithmen.

Weiterhin ist anzumerken, dass ein gelerntes Modell nicht etwa durch Befolgung fest definierter Regeln, etwa eine Klassifizierung von Blütenblättern vornimmt. Sondern das Modell verwertet implizites Wissen aus den Trainingsdaten mit Hilfe statistischer Theorien aus und kann mittels Stichproben Schlussfolgerungen ziehen [2, S. 1ff.]. Dieser Umstand ermöglicht eine Generalisierung, weg von den bereits bekannten Trainingsdaten und zu einem korrekten Umgang mit unbekannten Daten.

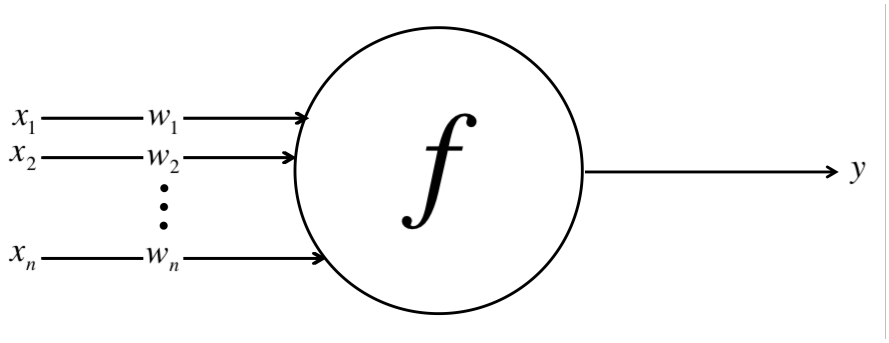


Abbildung 2: Schmatischer Aufbau eines künstl. Neurons [7, S. 12.]. x_n bezeichnet den Eingangswert. Die Verbindungsstärke wird über w_n definiert und die Aktivierungsfunktion f berechnet den Ausgangswert y.

2.2 KÜNSTLICHE NEURONEN

Die Vorlage für die Funktionsweise von Deep Learning lieferte das Gehirn. In den bestehenden neuronalen Netzen des Gehirns werden Informationen hierarchisiert verarbeitet.
Wichtigster Teil dieser Informationsverarbeitung stellt das Neuron dar, welches elektrische Eingangssignale von anderen Neuron empfängt und ab einer gewissen Signalstärke selbst beginnt ein Signal an andere Neuronen zu versenden. Die anliegenden Eingangsverbindungen eines Neurons können unterschiedliche Signalstärken übertragen. Die Stärke der Verbindungen erhöht oder verringert sich basierend auf der Häufigkeit ihrer Nutzung. So ist es dem Gehirn möglich neue Konzepte zu erlernen [6].

In der Abbildung 2 ist ein schematisches Modell eines künstlichen Neurons zu sehen. Der Parameter x_n definiert den übertragenen Wert und w_n modelliert die Verbindungsstärke über die x_n übertragen wird. Die Funktion f berechnet die Stärke der Aktivität und damit die Signifikanz des Ausgangsignals y und wird im Folgenden als Aktivierungsfunktion bezeichnet.

Aktivierungsfunktion

Die Berechnung ab wann ein Neuron aktiv (also ein Signal versendet) wird kann nun, wie folgt formal beschrieben werden [39]:

$$y = \begin{cases} 0 & \text{falls } \sum_n w_n x_n \leqslant \text{ Schwellenwert} \\ 1 & \text{falls } \sum_n w_n x_n > \text{ Schwellenwert} \end{cases} \tag{2}$$

Die Ausgabe des Neurons wird also durch die gewichtete Summe $\sum_n w_n x_n$ bestimmt.

Durch die Festlegung eines Bias $b \equiv -$Schwellenwert kann Gleichung 2 vereinfacht werden, sodass gilt [39]:

$$y = \begin{cases} 0 & \text{falls } \sum_n w_n x_n + b \leqslant 0 \\ 1 & \text{falls } \sum_n w_n x_n + b > 0 \end{cases} \tag{3}$$

Das oben erwähnte Ausgangssignal y wird durch die Aktivierungsfunktion ($f : \mathbb{R} \to \mathbb{R}$) errechnet.
Diese erhält als Eingabe die gewichtete Summe der Eingänge $\sum_n w_n x_n$ addiert mit dem Bias b. Jetzt lautet die endgültige Beschreibung eines künstlichen Neurons [39]:

$$y = f(\sum_n w_n x_n + b) \tag{4}$$

2.3 KÜNSTLICHE NEURONALE NETZE

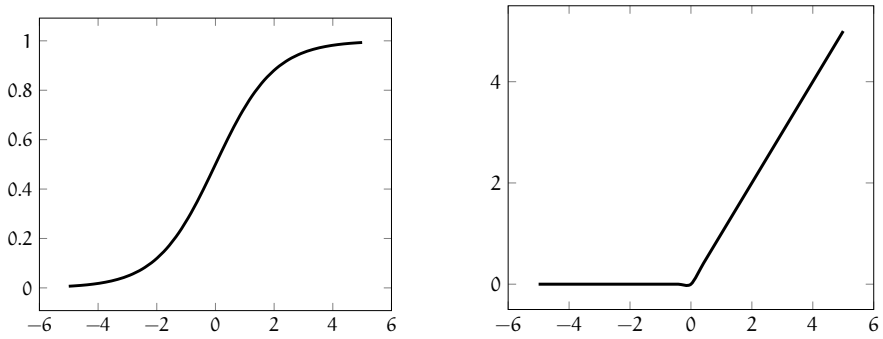
Neuronen sind nicht einzeln organisiert, sondern gliedern sich in ein künstlich neuronales Netz (KNN) ein, welches aus mehreren Schichten von untereinander verbundenen künstlichen Neuronen besteht. Dabei kann jede Schicht eine unterschiedliche Anzahl an künstlichen Neuronen enthalten. Die erste Schicht wird als Eingabeschicht und die letzte als Ausgabeschicht bezeichnet. Alle dazwischen liegenden Schichten sind verdeckte Schichten (engl. hidden layers).

lineare Aktivierungsfunktionen

Um komplexe Probleme in solch einem Netz modellieren zu können, reichen lineare Funktionen wie die Treppenfunktion in Gleichung 3 nicht aus. Im Gegenteil, das Netz selbst bei kleinen Anpassungen der Parameter w_n und b einer Netzschicht seine Ausgangssignale ($y \in [0, 1]$) komplett verändern und damit eine kausale Veränderungskette im gesamten Netz beginnen.
Ein solches unvorhersehbares Verhalten ist jedoch nicht erwünscht, da bei der Optimierung eines KNNs die Lernparameter w_n und b_n nur leicht verändert werden sollen und somit ein kleiner Schritt in Richtung einer optimalen Lösung getan werden kann [39] .

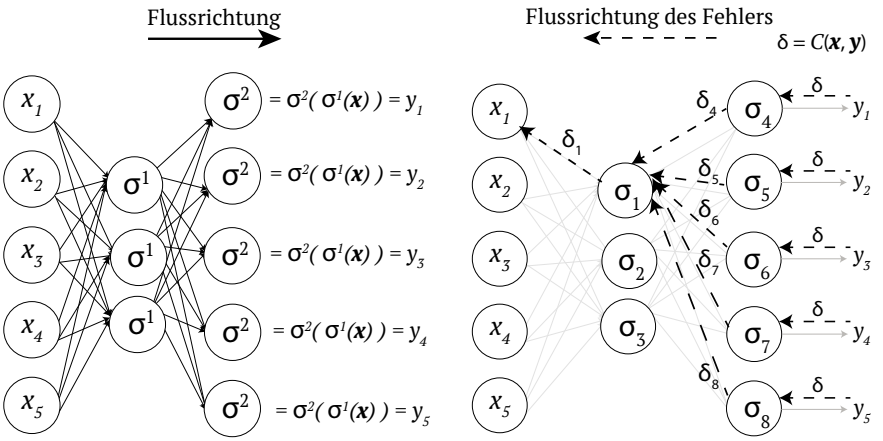
Eine Verwendung von linearen Aktivierungsfunktionen verhindert des Weiteren die tiefe Modellierung von Konzepten, durch das Hinzufügen weiterer Netzschichten. Linearen Aktivierungsfunktionen in den einzelnen Schichten können kein ausdrucksstarkes Netz bilden. Durch die Superpositionseigenschaft können mehrere Linearkombinationen ebenso durch nur eine Linearkombination dargestellt werden [22]. Somit erfolgt keine Steigerung der Komplexität des KNN, durch das Hinzufügen neuer Schichten (mit linearen Aktivierungsfunktionen), da diese durch nur eine Schicht ersetzt werden können.

Nichtlineare Aktivierungsfunktionen ermöglichen das Lösen von nichtlinear separierbaren Problemen. Als Beispiele solcher nichtlinearen



(a) Klassische Sigmoid-Funktion (b) ReLU-Funktion

Abbildung 3: Zwei Bespiele für Aktivierungsfunktionen.



(a) Berechnung der Ausgabeelemente $y_n \in \mathbf{y}$, durch Rekombination mehrerer Aktivierungsfunktion σ^n, wobei $x_n \in \mathbf{x}$ die Eingabeelemente darstellen.

(b) Zurückführung des Fehlers durch Backpropagation. *Anmerkung*: $\mathbf{x}$ und $\mathbf{y}$ symbolisieren Vektoren.

Abbildung 4: Trainingsphasen eines KNNs.

Aktivierungsfunktionen sind in der Abbildung 3 eine klassische Sigmoid-Funktion und eine modernere Rectified Linear Units (ReLU)-Funktion abgebildet. Eine Sigmoid-Aktivierung kann die Lernphase eines Netzes durch das Vanishing Gradient Problem verkomplizieren. Was unter anderem durch die Verwendung einer ReLU behoben werden kann [16]. Auf beides wird hier jedoch nicht weiter eingegangen.

Die Gesamtheit aller Parameter aus einem KNN, d.h. alle Gewichte und Bias aller künstlichen Neuronen, bilden die Matrizen $\mathbf{W}$ und $\mathbf{B}$. Diese sind äquivalent zum Parameter θ aus dem ML-Modell $g(\cdot)$ in Gleichung 1.

2.3.1 *Lernvorgang eines künstlichen neuronalen Netzes*

Abbildung 4 zeigt ein Beispiel für eine Trainingsphase in der alle drei Komponenten des ML-Vorgangs zu sehen sind.

1. Komponente: Das Model $g(\cdot)$ in Abbildung 4a ist ein schematisch illustriertes, künstlich neuronales Netz.
Der Prozess beginnt mit der Eingabe eines Elementes $x \in X$, wobei X die Menge an Trainingselementen ist. Der m-elementige Vektor x fließt durch das Netz und resultiert in einem Ergebnisvektor $y = (y_1, y_2, \ldots, y_m)$.

2. Komponente: Die Qualität der erzielten Ergebnisse wird durch die Kostenfunktion C bestimmt. In dem Falle dieser Arbeit ist dies die euklidische Distanzfunktion

$$\sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

welches die Summe über die Beträge der Distanzen von zwei Vektoren x_i und y_i für alle Elemente der Trainingsmenge X bildet.

3. Komponente: Nach der Ermittlung des Fehlerwertes werden alle Wichtungen w_n und Bias b_n, abhängig von ihrem Einfluss auf den Fehler verändert. Es findet ein Zurückpropagandieren (engl. Backpropagation) des Fehlers σ statt, wie in Abbildung 4b zu sehen ist.
Dies geschieht durch den Einsatz von gradientenbasierten Optimierungsverfahren, welche die Parameter des KNNs verbessern. Das Ziel ist es den Verlauf der Kostenfunktion in Abhängigkeit der Parameter des KNNs in ein globales Minimum hinein zu bewegen, in welchem der Fehlerwert konvergiert.
Der prominenteste Ableger solch eines Verfahrens ist das stochastische Gradientenverfahren, auch genannt das Verfahren des steilsten Abstiegs.[18, S. 294.]

Die Abfolge aller Prozesse aus Abbildung 4 bildet eine Epoche und kann zur Verbesserung des Lernvorgangs beliebig wiederholt werden.

2.3.2 *Netzarten*

Es existieren unterschiedliche Topologien für KNNs. Damit ist im Allgemeinen die Anzahl der Schichten im KNN gemeint und wie die künstlichen Neuronen miteinander verbunden sind. KNNs können als

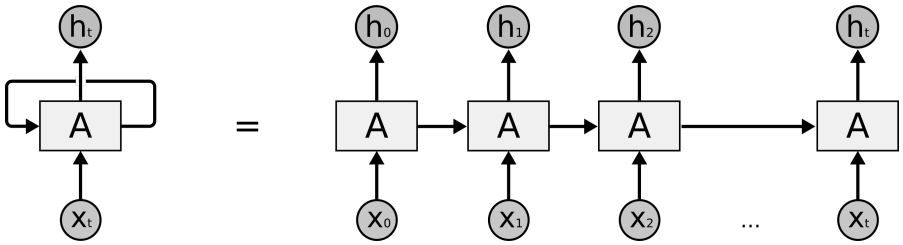


Abbildung 5: RNN mit entfalteter Repräsentation [40]

gerichtete Graphen beschrieben werden. Dabei existieren zwei grundlegende Unterscheidungen.

Ein feedforward KNN ist in Abbildung 4a verdeutlicht, indem der gerichtete Graph keine Kreise beinhaltet und somit die Flussrichtung der Berechnungen stets nach vorne fließt und sich nicht verändert [18, S. 168.]. Weiterhin gelten KNNs als fully connected, wenn jedes künstliche Neuron einer Schicht n mit jedem Neuron seiner Nachfolger Schicht $n+1$ verbunden ist.

Im Gegensatz zu den nach vorne ausgerichteten Feedforward-Netzen existieren in Recurrent Neural Networks (RNNs) auch rückgerichtete Kanten. Ein Beispiel für ein RNN ist in Abbildung 5 zu sehen.
An einem Teil A des KNNs gehen wie gewohnt Ein und -Ausgabekanten ab, mit der Besonderheit, dass eine neue Kante auf A selbst zeigt. Diese Schleife erlaubt das Teilen von Informationen (z.B. Wichtungen) für t Zeitschritte, was mit regulären feedforward KNN nicht möglich ist [18, S. 374.].

Ein RNN unterscheidet sich trotzdem nicht stark von feedforward KNNs. Entfaltet man ein RNN(Abb. 5, rechts) unter der Berücksichtigung von t wird erkennbar, dass ein RNN wie eine Kopie aus sich selbst fungiert, welche Informationen immer an den Nachfolger weiterreicht. Dadurch entsteht eine Art Gedächtnis an Information für eine bestimmte Zeitspanne t. Dabei ist t nicht zwangsläufig als realer Zeitwert zu verstehen, sondern eher für die sequentielle Abhängigkeit von Informationen [18, S. 374.].

Autoencoder

Das in dieser Arbeit genutzte KNN ist der Autoencoder (AE). Der AE gilt als ein sogenanntes feedforward KNN.

Ein AE kann abstrahierte Repräsentationen der Eingabewerte erlernen, indem es Eingabewerte zu Ausgabewerten kopiert. Dazu kodiert eine Funktion $h = enc(x)$ ihre Eingabewerte x in einen latenten Zustand h und dekodiert diesen anschließend mit einer Funktion $r = dec(h)$. Ziel ist es $dec(enc(x)) = x$, so gut es möglich ist, zu ap-

proximieren.
Auf diese Art und Weise eine Identitätsfunktion $i(x) = x$ zu erlernen, würde keinen Effekt erzielen. Um dies zu vermeiden, werden AE mit Restriktionen belegt, damit diese gezwungen sind nur signifikante Trainingsdaten zu kopieren. Durch den Zwang manche Eingabedaten zu priorisieren, erlernen AE oft nützliche Eigenschaften dieser Daten.[18, S. 502.]

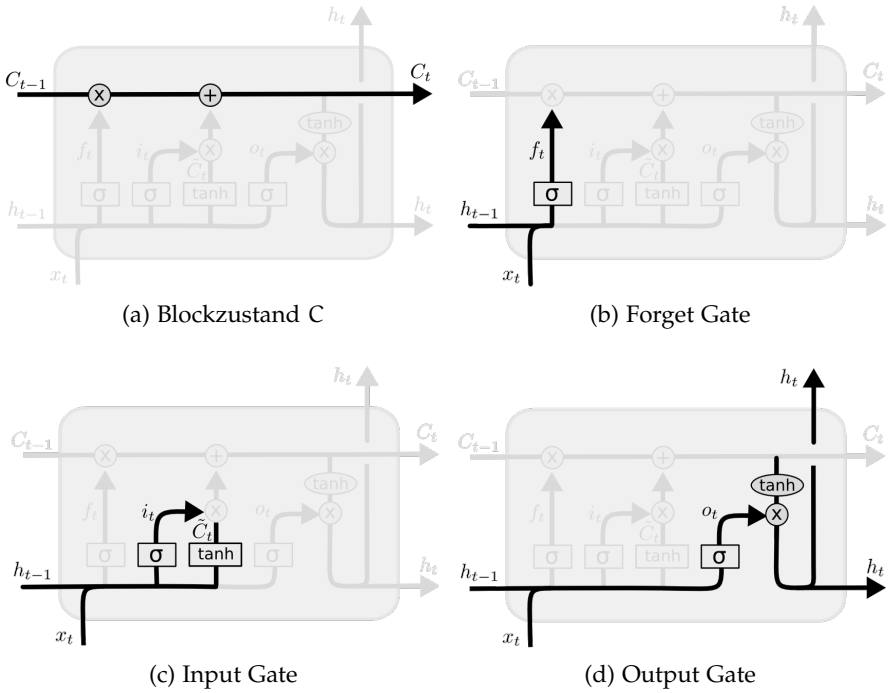
Abbildung 4 veranschaulicht den generellen Aufbau eines AEs. Die minimalste Form eines AEs besteht aus einer Eingabe- und Ausgabeschicht. Dazwischen befindet sich eine verdeckte Schicht.
Die Effizient des Lernvorgangs kann mit der einfachen Minimalisierung einer Kostenfunktion $L(x, dec(enc(x)))$ ausgedrückt werden. Dabei ist L eine Funktion, welche den Abstand zwischen den Vektoren bestraft, wie beispielsweise die Euklidische Abstandsfunktion.[18, S. 503.]

Variational Autoencoder

Der Variational Autoencoder (VAE) weicht von dem Konzept herkömmlicher Autoencoder ab und wird in [24] vorgestellt.
Hierbei wird eine Menge an Trainingsdaten X mithilfe einer unbekannten Wahrscheinlichkeitsverteilung $P_{gt}(X)$ beschrieben. Die Idee ist nun eine Verteilung $P(z|X)$ zu erstellen, wobei z latente Variablen darstellt. Anders formuliert heißt dies, eine Wahrscheinlichkeitsverteilung zu finden, welche die latenten Variablen unter der Bedingung der Trainingsdaten wahrscheinlich macht.
Latenten Variablen stellen somit Eigenschaften bzw. versteckte Zusammenhänge innerhalb der Trainingsdaten dar.

Die Verteilung $P(z|X)$ muss mit Hilfe der Bayesschen Inferenz inferiert werden, da auch diese Verteilung unbekannt ist. In der Praxis wird die Verteilung meist mit einer einfacheren Verteilung wie der Gaußschen Normalverteilung approximiert. Dieser Approximierungsprozess ist als Optimierungsproblem zu verstehen, indem $Q_\phi(z|X)$ durch Optimierung der Netzparameter ϕ an $P(z|X)$ angenähert wird. Da es zwei unterschiedliche Ziele bei dieser Optimierung gibt werden zwei Verlustfunktionen miteinander kombiniert.
Zum einen wird die Abweichung der Verteilungen mit der Kullback-Leibler-Divergenz bestraft und zum anderen werden, wie üblich die Abweichungen des Ergebnisvektors mit dem Eingabevektor bestraft. In dieser Arbeit geschieht dies mittels der Euklidischen Distanz.

Aus der approximierten Verteilung Q können nun auch Werte für, in den Trainingsdaten nicht existierende Punkte, ausgegeben werden. Daher werden VAEs auch als generative Modelle bezeichnet, da sie



(a) Blockzustand C

(b) Forget Gate

(c) Input Gate

(d) Output Gate

Abbildung 6: Funktionsweise eines Memory-Blocks aus einem LSTM-Netz [40].

neue Daten generieren können, die sich stark an den Trainingsdaten orientieren.

Long Short Term Memory

Ein Long Short Term Memory (LSTM)-Netz gehört der Familie der RNNs an. Es wurde bereits 1997 in [20] vorgestellt und behebt das Vanishing-Gradient-Problem für die Gruppe der RNNs. Damit können nun auch RNNs effektiv mit dem Backpropagation-Algorithmus trainiert werden.

In herkömmlichen KNNs beinhalten künstliche Neuronen eine Aktivierungsfunktion, welche den Eingabewert in einen Ausgabewert transformiert.
Grundlegend für eine LSTMs ist die Einführung von meist vier Transformationen die in Blöcken organisiert sind. Die Transformationen werden von sogenannten Gates vollzogen. In Abbildung 6 ist dieser Prozess illustriert.

In den Memory-Block fließt die Eingabe x_t von einer Eingabeschicht. Diese kann entweder die Eingabeschicht des KNNs oder die Ausgabe einer davorliegenden LSTM-Schicht sein. Im Anschluss an die Transformationen fließt die Ausgabe h_t in die endgültige Ausgabeschicht des KNNs oder in eine weitere LSTM-Schicht. Die Kernidee einer LSTM

ist der Blockzustand C_t(Abb. 6a), welcher durch die gesamte Zelle fließt und an den Informationen hinzugefügt, oder auch entfernt werden können.
Anmerkung: Alle horizontalen Ein- und Ausgänge (z.B. C_t, h_{t-1} usw ...) markieren die RNN typischen Schleifen, welche in Abbildung 5 gezeigt wurden.

Sigmoid-Funktion $\equiv \sigma$

Im ersten Schritt entscheidet ein Forget Gate (eine Sigmoid-Funktion) anhand der Betrachtung der gespeicherten Ausgabe h_{t-1} und einer Eingabe x_t, welche Information gehalten und welche Informationen vergessen werden sollen. Das Ergebnis wird anschließend mit dem momentanen Blockzustand C_{t-1} kombiniert (Abb. 6b).

Der nächste Schritt entscheidet, welche Informationen an den Blockzustand hinzugefügt werden sollen. Daran beteiligt ist eine Sigmoid-Funktion und eine Tanh-Funktion, aus deren Kombination eine Aktualisierung des Blockzustands C_t hervorgeht (Abb. 6c).

Am Ende des Vorgangs gilt es zu entscheiden, was an Information ausgegeben werden soll. Eine Sigmoid-Funktion entscheidet, welche Informationen in die Ausgabe gelangen. Nach einer Kombination mit einer Tanh-Funktion werden die Informationen an eine nächste Schicht (vertikales h_t) oder wieder in die selbe LSTM-Schicht (horizontales h_t) eingegeben (Abb. 6d).

3

VERWANDTE ARBEITEN UND STAND DER TECHNIK

Um einen grundlegenden Kenntnisstand im Bereich der Entity Resolution mit Schwerpunkt auf der Anwendung Dedoop zu schaffen, wurde [25] herangezogen.

3.1 RELATED WORK

Damit es während des laufenden Arbeitsprozesses möglich war die ermittelten Ergebnisse der Deduplizierung zu evaluieren, wurden Vergleichswerte aus [28] entnommen. Das Besondere bei der Evaluierung der verschiedenen Verfahren war die Anwendung auf Realdaten aus der Domäne der Bibliographie und des E-Commerce.

In [28] werden maschinelle Lernansätze mit klassischen Verfahren der Deduplizierung verglichen, welche sich auf die Klassifizierung der Match-Paare beschränken. Es konnte gezeigt werden, dass Kombinationen aus Lern- und Nicht-Lernverfahren bei der Deduplizierung von mehreren Attributen bessere Ergebnisse erzielen. Jedoch konnten lernbasierte Methoden bei einem Matching mit nur einem Attribut nicht überzeugen. Attribute stellen hier Merkmale von Produkten, wie Produktname, Hersteller oder Preis dar. Die betrachteten maschinellen Lernverfahren waren eine SVM und der Entscheidungsbaum.

Ziel des Ontology Matching ist es Übereinstimmungen zwischen semantisch in Relation stehenden Entitäten aus verschiedenen Ontologien zu finden.
Eine Übereinstimmung beschränkt sich nach [11], nicht nur auf eine syntaktische Ähnlichkeit, sondern auch auf semantische und externe Charakteristiken. Der semantische Begriff ist in diesem Fall als Semantik einer formellen Sprache zu verstehen.
Extern, bezeichnet die Möglichkeit externe Hilfsressourcen aus einer Domain oder Allgemeinwissen für das Finden einer Übereinstimmung hinzuzuziehen.

Ähnliche Bestrebungen, wie in der vorliegenden Arbeit, die auf die Erstellung von Vektorrepräsentationen aus Textdaten mittels KNNs abzielen, kommen in [48] vor. Es konnte gezeigt werden, dass durch die Verwendung von AE eine abstrakte Repräsentation von Entitäten erlernt werden kann. Entität bedeutet in diesem Fall die Kombination

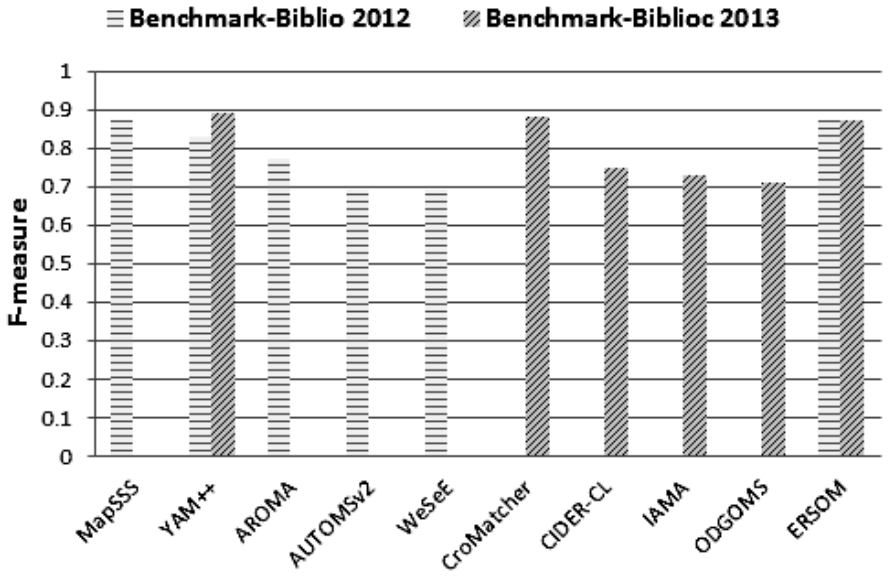


Abbildung 7: ERSOM im Vergleich zu anderen OAEI-Systemen [48].

aus Klassen und Eigenschaften der Ontologie zu einer Entity Description. Die erlernten Repräsentationen des AE ermöglichen einen mathematischen Vektorenvergleich, um somit einer Aussage über Gleichheit von zwei Ontologien zu treffen, ohne dafür domänenspezifische Regeln aufzustellen.

In Abbildung 7 sind die Ergebnisse des Ontologie-Matching-Vorgangs ERSOM von [48] im Vergleich zu anderen Ontologie-Matching-Systemen illustriert. Zu sehen ist, dass ERSOM die meisten, der an der Ontology Alignment Evaluation Initiative (OAEI)-Evaluation beteiligten Systeme übertrifft. Systeme wie YAM++ oder MapSSS konnten nicht übertroffen werden.
Jedoch ist zu erwähnen, dass MapSSS externe Hilfsressourcen nutzt. YAM++ verwendet den Goldstandard eines Benchmark-Datensatzes, welcher in OAEI 2009 veröffentlicht wurde, um einen Entscheidungsbaum zu trainieren. Dieser klassifiziert dann die Elemente der zu vergleichenden Ontologien, in Match oder non Match.
ERSOM erzielt seine Ergebnisse auf Grundlage eines unüberwachten Lernvorgangs und mit nicht markierten Daten, also ohne zusätzliches Wissen.

Durch die günstigen Ergebnisse von [48] mittels AEs kam es zu einer intensiveren Auseinandersetzung mit Encoder- und Decoder-Netzen. Es existieren neben den klassischen AE-Netzen modernere Ansätze, um eine Repräsentation von Eingabedaten zu erlernen. Zu nennen sind hier die Veröffentlichungen von [24] und [44]. In denen spezielle AEs, wie VAE oder LSTM-AE vorgestellt werden.

Model	Error rate
Vector Averaging	10.25%
Bag-of-words	8.10 %
Bag-of-bigrams	7.28 %
Weighted Bag-of-bigrams	5.67%
Paragraph Vector	**3.82%**

Abbildung 8: Performanz von Paragraphvektoren [32].

Für die vorliegende Arbeit weiter von Bedeutung ist die Umwandlung von Textsequenzen unterschiedlicher Größe in Vektoren fester Länge, damit diese anschließend von AEs weiterverarbeitet werden können.
Auf Grund der Einführung von Paragraphvektoren in [32] konnte eine neue State-of-the-Art-Repräsentation von Textabschnitten mit Vektoren erreicht werden. Abbildung 8 zeigt die Verbesserungen von Paragraphvektoren zu herkömmlichen Verfahren der Sprachmodellierung.

Bei der in Abbildung 8 gezeigten Evaluierung musste festgestellt werden, welche Suchergebnisse aus der selben Suchanfrage stammten. Dabei standen drei Suchergebnisse, die aus mehreren Sätzen bestanden, zur Verfügung. Zwei von diesen entstammten aus der selben Suchanfrage, die dritte aus einer anderen, zufällig gewählten.
Somit soll gezeigt werden, dass Paragraphvektoren semantische Relationen besitzen und dies auch im Kontext der Informationsrückgewinnung genutzt werden kann.

3.2 FUNKTIONALER VERGLEICH VON DL-ANWENDUNGEN

Im Folgenden findet eine Betrachtung von unterschiedlichen Toolkits und Frameworks für die Nutzung von KNNs statt. Mit Toolkits sind Bibliotheken, die durch ein Application Programming Interface (API) genutzt werden können, gemeint, wohingegen Frameworks meist in einer speziellen Umgebung integriert sind. Ein Beispiel für ein Framework ist Apache Singa, welches nicht nur DL-Techniken realisiert, sondern auch Berechnungen über Cluster ermöglicht.

Um aus der Fülle an DL-Anwendungen, die geeigneten Kandidaten zu wählen, worden Kriterien aufgestellt, die von den jeweiligen Anwendungen zu erfüllen sind. Dies ist ausschlaggebend für die Auswahl und Verwendung in dieser Arbeit.

PROTOTYPING Durch den fest bemessen Zeitraum der Arbeit ist es wichtig keine komplizierten APIs oder Programmiersprachen erlernen zu müssen. Bei der Realisierung der Aufgabenstellung

lag die Priorität auf schnellen, ersten Ergebnissen, um somit ein frühzeitiges Feedback, durch Evaluierungsergebnisse zu erhalten. Ein Toolkit erfüllt diesen Schwerpunkt bspw., wenn es in Python implementiert ist.

AUTOENCODER Die zu nutzende Anwendung sollte möglichst viele Autoencoder-Strukturen, wie Stacked Autoencoder (SAE) oder VAE zur Verfügung stellen, um mit diesen experimentieren zu können oder zumindest explizite Beispiele bereitstellen. Die Anwendungen Dritter, welche das jeweilige Toolkit oder Framework benutzen, sind von dieser Betrachtung ausgeschlossen.

MULTICORE Da große Datenmengen zu verarbeiten sind und Deep Learning rechenintensiv ist, liegt ein weiterer Schwerpunkt auf der Nutzung mehrkerniger CPUs in Bezug auf Parallelisierbarkeit und Skalierung. Dies ist auch relevant, da zur Berechnung der Ergebnisse auch potentere Hardware zur Verfügung steht.

LINUX Das zu nutzende Betriebssystem unterliegt keiner weiteren Anforderung, jedoch ist auf den bereitgestellten Systemen eine Linux-Distribution vorinstalliert. Um Komplikationen bei der Installation oder Ausführung, der noch zu wählenden DL-Anwendung zu vermeiden, wurde eine Linux-Betriebssystem als weiteres Entscheidungskriterium gewählt.

GPU Durch massive Parallelrechner wie eine GPU können KNNs weitaus schneller trainiert werden, als mit herkömmlichen Mehrkernprozessoren. Somit können Änderungen in den Experimenten umgehend evaluiert werden.

Die aufgeführten Hauptschwerpunkte sind nach ihrer Priorität absteigend sortiert.

Unter diesen Kriterien ist die Vergleichstabelle 2 erstellt worden. Außerdem zeigt die Tabelle weitere Eigenschaften von KNNs. Jedoch sind nicht alle möglichen Parameter aufgelistet, sondern es wurde auf eine Relevanz für die vorliegende Arbeit geachtet.

Ein ✓ bedeutet, dass die Anwendung den Schwerpunkt erfüllt, im Gegensatz dazu symbolisiert ein ✗, dass Nicht-Erfüllen des Schwerpunktes. Ein (✓) wird verwendet, wenn der Schwerpunkt nur teilweise verwirklicht oder nicht erfüllt ist, sich aber durch den abstraktionsgrad der API mit wenig Aufwand realisieren lassen kann. Es ist anzumerken, dass Teile der Bewertungen auf subjektiven Einflüssen oder Vorlieben des Autors beruhen.
Die Beobachtungen stammen aus dem Zeitraum Dezember 2016.

Ein grundlegendes Toolkit zu Berechnung von multidimensionalen

	Theano	TF	Keras	Caffe	H_2O	Singa	DL4J
Hauptschwerpunkte							
Prototyping	✓	✓	✓	✓	✓	✗	✗
Autoencoder	(✓)	✓	✓	(✓)	✓	(✓)	(✓)
SAE	(✓)	✗	✓	✗	(✓)	(✓)	(✓)
VAE	✗	✓	(✓)	✗	✗	✗	✗
Multicore	✓	✓	✓	✓	✓	✓	✓
Linux	✓	✓	✓	✓	✓	✓	✓
GPU	✓	✓	✓	✓	✓	✓	✓
Weitere Netzarten							
Recurrent	(✓)	✓	✓	✓	✗	✓	✓
Convolutional	✓	✓	✓	✓	✓	✓	✓
Regularisierung							
Dropout	(✓)	✓	✓	✓	✓	✓	✓
Sparsity	(✓)	✓	✓	✓	✓	✓	✓
Aktivierungen							
Maxout	(✓)	(✓)	✓	(✓)	✓	✗	✗
ReLU	✓	✓	✓	✓	✓	✓	✓
Verlustfunktionen							
Euklid	(✓)	(✓)	(✓)	✓	✗	✓	✗
MSE	(✓)	(✓)	✓	(✓)	✓	✓	✓
Optimierung							
RMSprop	(✓)	✓	✓	✓	✗	✓	✓
Adagrad	(✓)	✓	✓	✓	✗	✓	✓
Version							

Tabelle 2: Vergleich der Hauptschwerpunkte und weiterer Hyperparamter für die Auswahl der DL-Anwendung.

Feldern ist Theano. Dabei abstrahiert Theano mathematische Ausdrücke durch einen Berechnungsgraphen und kann deshalb unnötige oder langsame Berechnungen optimieren [4]. Theano bietet selbst keine komplexe API zur Nutzung von KNNs an, jedoch dient es als Backend-System für andere APIs, die fertige KNN-Schichten zur Nutzung anbieten.
Diese Beobachtungen spiegeln sich auch in der Tabelle 2 wieder, da Theano wenig Schwerpunkte selbst implementiert, jedoch durch den hohen Grad der Abstraktion, dies mit relativ wenig Aufwand ermöglicht.

TensorFlow (TF) wird von Google entwickelt und ähnelt stark Theano. TF nutzen ebenfalls einen gerichteten Berechnungsgraphen [17] und dient für viele Toolkits als Backend-System. Es konnte festgestellt werden, dass TF mehr Schwerpunkte durch Beispiele implementiert, als sein Pendant.

Die Python-Bibliothek Keras kann Theano oder TF als Backend benutzen und stellt viele API für die unterschiedlichsten KNN-Schichten bereit. Weiterhin implementiert Keras fast alle Schwerpunkte oder stellt diese in Beispielen vor.
Dies ist auch der Grund warum Keras als Toolkit für diese Arbeit verwendet wird. Caffe ist ein weiteres Toolkit für DL-Algorithmen und wird an der Berkeley Universität entwickelt. Caffe implementiert viele der betrachteten Schwerpunkte, doch beinhaltet keine AEs und wird daher in dieser Arbeit nicht weiter beachtet.

Die Frameworks H_2O und Deeplearning4j (DL4J) stellen kommerzielle Plattformen dar, welche über Cluster betrieben werden können, aber auch mit einem einzelnen Computer funktionieren. H_2O stellt nur feedforward KNNs zur Verfügung. Beide Anwendungen bieten nicht im ausreichenden Maße AE-Strukturen an.
Singa eine weiteres Framework für DL-Berechnungen stand zum Zeitpunkt der Betrachtung in einer frühen Entwicklungsphase zur Verfügung und konnte nur über Konfigurationsdateien oder C++ Implementierungen angepasst werden. Des Weiteren stellt Singa keine herkömmlichen AE-Strukturen zur Verfügung.

Alle betrachteten Toolkits oder Frameworks können ihre Berechnungen über mehrere CPUs oder GPUs parallelisieren, was die Intensität von DL-Berechnungen zeigt. Anwendungen wie H_2O, DL4J oder SINGA, die auch über einem Cluster ausgeführt werden können, eignen sich zudem auch für verteilte Berechnungen.

4

KONZEPT: DEDUPLIZIERUNG DURCH KÜNSTLICHE NEURONALE NETZE

In diesem Kapitel sollen die theoretischen Konzepte aufgestellt werden, welche eine Deduplizierung unter Beachtung der Problemstellung *P1*, ermöglichen. Für eine Umwandlung der Textdaten in eine vektorielle Form sollen flache und tiefe KNNs genutzt werden.

Anfänglich findet eine Vorverarbeitung der Textdaten unter einem linguistischen Gesichtspunkt statt, welcher die Problemstellungen *P3* bis *P8* erfüllt. Dies soll die verwendeten Daten besser für nachfolgende Schritte nutzbar machen und die statistische Aussagekraft dieser erhöhen. Ausführlich ist dieser Prozess in Kapitel 5.1 der Implementierung beschrieben.

Nach der Vorverarbeitung der Textdaten liegen diese nun strukturiert vor, jedoch eignen sie sich nicht für eine weitere Verarbeitung mittels KNNs. Generell benötigen maschinelle Lernalgorithmen Eingabedaten in einer fixen Form und in einem numerischen Wertebereich [32, S. 1.].
Nachfolgend wird nun diese Transformation von Textdaten betrachtet.

4.1 STATISTISCHE SPRACHMODELLIERUNG

Traditionelle Ansätze im Gebiet des Natural Language Processing (NLP) fassen Wörter als diskrete Werte auf, d.h. das Wort Hund kann durch *ID1* und Katze durch *ID2* repräsentiert werden. Solch eine Kodierung kann keine Relationen, die möglicherweise zwischen den einzelnen Wörtern existieren, modellieren.
Eine Lösungsstrategie ist es an Stelle von diskreten Darstellungen mittels Symbolen (*ID1*, *ID2*) kontinuierliche Repräsentationen, wie Vektoren, zu benutzen.

Für die Umwandlung der Textdaten in Vektoren wurde sich anfänglich für eine Methodik entschieden, welche sich als ungeeignet herausstellte. Diese soll im folgenden erläutert und ihre Nachteile aufgezeigt werden.

Bags \ Wörter	John	likes	to	watch	movies	also	football	games	Mary	too
a′	1	1	1	1	1	0	0	0	1	1
b′	1	1	1	1	0	1	1	1	0	0

Tabelle 3: BOW-Vektoren und die Bedeutung ihrer Indexwerte.

4.1.1 *Bag-of-Words*

Ein klassischer Ansatz im Bereich des NLP ist das Erstellen von Bag-of-Words (BOW) Vektoren.

Dabei können einzelne Wörter oder ganze Textabsätze in Vektoren mit fester Länge $|V|$ überführt werden. Ein Vokabular V ermittelt sich aus der Anzahl aller unterschiedlichen Worte eines Dokuments.
Das folgende Beispiel an einem Dokument D illustriert den Vorgang.

a: John likes to watch movies. Mary likes movies too.

b: John also likes to watch football games.

Das aus D entstehende Vokabular V setzt sich aus den Wörtern *John, likes, to, watch, movies, also, football, games, Mary* und *too* zusammen und hat somit eine Länge von $|V| = 10$.

Die daraus resultierenden BOW-Vektoren sind eine Abbildung eines Indexes auf eine natürliche Zahl, wobei der Index die Postion des Wortes im Vokabular kodiert. Der Wert einer Indexposition beschreibt die Häufigkeit des Vorkommens eines Wortes $x \in V$ im betrachteten Dokument D.
Folglich ergeben sich zwei BOW-Vektoren a′ und b′, die als Eingabedaten für ML-Verfahren dienen können. Tabelle 3 illustriert dies.

Trotz ihrer Popularität bringt die Verwendung dieser Methodik Nachteile mit sich und ist nicht für alle Anwendungsszenarien geeignet. Folgende Probleme können entstehen [32]:

UNGEEIGNETE VEKTORENSTRUKTUR Je nach Größe des verwendeten Dokuments, fällt die Größe von V und damit die Länge der Vektoren unterschiedlich aus. Da sich V nach der Anzahl der verschiedenen Wörter richtet, skaliert die Größe der Vektoren nur sehr schlecht bei großen Datenquellen.
Zudem sind die einzelnen Ergebnisvektoren nur spärlich mit Informationen versehen, da nicht vorkommende Wörter trotzdem mit einer null im Vektor gespeichert werden. So können durchaus Vektoren der folgenden Form auftreten:

$$(0,0,0,0,0,1,0,0,0,1,1,0,\ldots,0,0,1,0,1,0,0,0,0,0,0)$$

VERLUST DER WORTREIHENFOLGE Das Einlesen der Sätze und einzelner Wörter in ein Vokabular geschieht schrittweise und aufeinander folgend. Wie in den transformierten Sätzen a' und b' zusehen ist, erhalten diese nicht die ursprüngliche Reihenfolge der Wörter. Dadurch kann es vorkommen, dass zwei unterschiedliche Sätze durch den selben BOW-Vektor repräsentiert werden.

MISSACHTUNG DER SEMANTIK VON WÖRTERN Die semantische Nähe von Wörtern lässt sich nicht durch das BOW-Verfahren beschreiben. Beispielsweise liegen die beiden Wörter *powerful* und *strong* näher aneinander als zu dem Wort *Paris*. Dies kann allerdings nicht mit dem genannten Verfahren ausgedrückt werden.

Das Lösen der Problemstellung *P1* ist durch oben geschildertes Verfahren nicht möglich. Außerdem steht die Größe der Vektoren in keinem Verhältnis zu den Information, die sie repräsentieren. Aus technischer Sicht eignen sich diese Vektoren weder durch den benötigten Speicherplatz im Arbeitsspeicher noch durch die langen Rechenzeiten.
In der Literatur gibt es verschiedene Ansätze, um den oben genannten Problemen entgegen zu wirken, wie das Bag-of-n-Gram-Verfahren [32]. Probleme mit der Beachtung der Wortreihenfolge lassen sich dadurch leicht verbessern, jedoch nicht die Unfähigkeit semantische Relationen zu beschreiben, vermeiden. Daher erfolgte keine weitere Betrachtung.

4.1.2 *Distributionelle Semantik*

Eine weiteres statistisches Sprachmodell ist das Modell der distributionellen Semantik.
Sogenannte Vector Space Models (VSMs) bilden Text als einen Vektor aus Kennungen ab. Eine wichtige mathematische Relation dieser Vektoren ist die metrische Nähe zueinander im euklidischen Raum, wenn diese sich semantisch ähneln [34, S. 539.].
Das Konzept von Semantik orient sich bei fast allen VSMs an der distributionellen Hypothese (engl. Distributional Hypothesis). Diese ordnet Wörtern, welche immer wieder in einem gleichen Kontext auftreten, eine semantische Bedeutung zu [49]. Somit formt sich die Semantik eines Wortes aus den Wörtern, welche sie umgeben [14].

Distributionelle Hypothese

VSMs werden in zwei Kategorien unterteilt [35]:

KONTEXTZÄHLUNG (ENGL. CONTEXT-COUNTING) Die Berechnung einer Statistik, wie oft bestimmte Wörter zusammen mit ihren benachbarten Wörten (Kontext) auftreten. Für jedes Wort wird anschließend der entsprechende Zählerstand in einen Vektor übertragen.

KONTEXTVORHERSAGE (ENGL. CONTEXT-PREDICTING) Die Vorhersage eines Wortes aus den umschließenden Nachbarwörtern (Kontext).

Eine umfangreicher Vergleich unter verschiedensten Leistungsmerkmalen in [35] zeigt die Überlegenheit von Kontextvorhersagemodellen gegenüber Kontextzählung. Leistungsmerkmale waren unter anderem: Semantische Verwandtschaft, Synonymerkennung und Konzeptkategorisierung (Hubschrauber, Motorrad $\rightarrow$ Fahrzeuge sowie Hund, Elefant $\rightarrow$ Säugetiere).
Diese Ergebnisse führten zu der Auswahl von Vorhersagemodellen für eine Umwandlung von Text in Vektoren.

statistische Schlussfolgerung

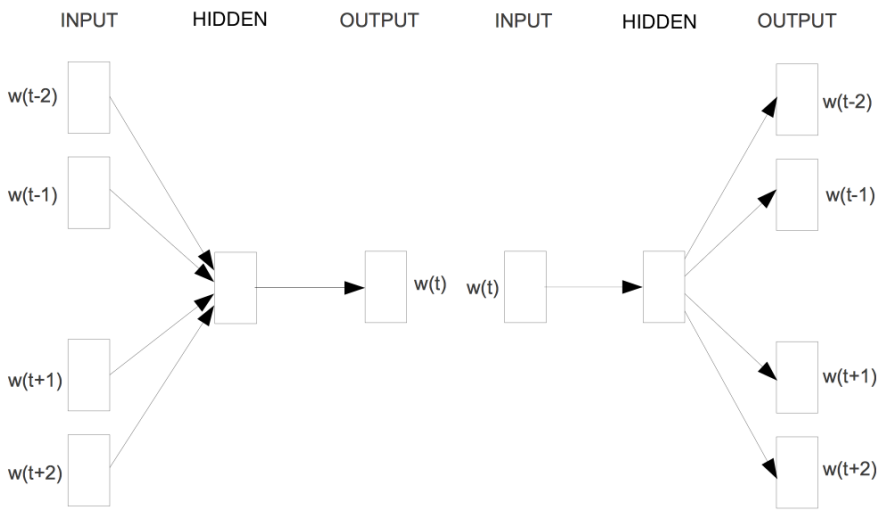
Im Bereich des statistischen NLP wird eine Menge von Textdaten betrachtet, um anschließend statistische Schlussfolgerungen über die Distribution der Daten zuführen.
Konkret wird in diesem Kapitel eine statistisches Sprachmodellierung betrachtet, die versucht ein Wort aus anderen Wörtern zu prognostizieren. Ein prominentes Beispiel für dieses Problem ist das Shannon Game, indem der nächste Buchstabe eines Textes erraten werden soll. Eine Wortvorhersage ist nicht die Problemstellung dieser Arbeit, jedoch hilft diese beim essentiellen Verständnis der Problematik (Auffassung von Semantik) und Übertragung dessen auf andere Probleme(Vergleich von Semantik) [34, S. 191.].
Durch statistische Schlussfolgerungen rückt somit das eigentlich Problem, die Erstellung von semantisch korrelierten Vektoren und damit die State of the Art Repräsentation von Dokumenten bzw. Textdaten, mittels Vektoren näher.

4.2 EINBETTUNG VON WÖRTERN

Grundlage für die Einführung von Modellen, welche ganze Textabsätze in einem Vektor darstellen können, ist die Umformung einzelner Wörter in eine vektorielle Repräsentation. In [38] wird eine neue Architektur für die Vorhersage von Wörtern eingeführt, welche diese Umformung ermöglicht. Die vorgestellte Architektur gehört der Gruppe der neuronalen Sprachmodelle (engl. Neural Net Language Model (NNLM)) an und verkörpert somit ein KNN.

Da es sich um ein maschinelles Lernverfahren handelt, kann der Vorgang anhand des in Kapitel 2.1 eingeführten Modells, wie folgt erläutert werden [32, 37, 38]:

MODELL Es werden zwei Modelle definiert:
Continuous Bag-of-Words (CBOW) versucht ein Wort w_t aus einer Folge von Kontextwörtern, welche w_t umschließen, zu bestimmen (Abbildung 9a). Formal ist dies die bedingte Wahrscheinlichkeit, also



(a) CBOW-Architektur (b) Skip-gram-Architektur

Abbildung 9: Modelle für die Erstellung von Vektorrepräsentationen von Wörtern [38].

$P(w_t|w_{t-k},\ldots,w_{t+k}), k \neq 0$, wobei k die Größe des Kontextes beschreibt. In anderen Worten: Wie wahrscheinlich ist w_t wenn $w_{t-k},\ldots,w_{t+k}$ vorkommen.
Entgegengesetzt dazu versucht die Skip-gram-Architektur aus einem Wort die umschließenden Kontextwörter zu bestimmen (Abbildung 9b), was wie folgt ausgedrückt wird
$P(w_{t-k},\ldots,w_{t+k}|w_t), k \neq 0$.

KOSTENFUNKTION Das Ziel des Modells ist es nun die Durschnittliche log Wahrscheinlichkeit der Vorhersage P mittels $\frac{1}{T}\sum_{t=k}^{T-k} \log P$ zu maximieren.

OPTIMIERUNGSVERFAHREN Das KNN wird über das stochastische Gradientenverfahren trainiert und die Parameter des Netzes anhand von Backpropagation optimiert.

Ein Beispiel: Bevor das ML-Verfahren beginnt, muss aus dem vorliegende Textcorpus ein Trainingsdatensatz aus Wörter und den umschließenden Kontextwörtern erstellt werden. Der Textcorpus sei *the quick brown fox jumped over the lazy dog*.
Angenommen die Größe des Kontextfensters k sei 1, dann hat der erstellte Trainingsdatensatz die Form *([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox)* usw. Die Tupelpaare bestehen aus $(\texttt{Kontext}, \texttt{Ziel})$. Zur Vereinfachung wurden keine Sonderzeichen für Satzanfang und -ende eingefügt.

Ein Verfahren kodiert anschließend alle Worttypen in eine Vektorrepräsentation. Für das Beispiel geschieht dies mit der in Kapitel 4.1.1

eingeführten BOW-Kodierung. Somit entsteht die folgende Repräsentation:

$the \rightarrow (1,0,0,0,0,0,0,0,0)$

$brown \rightarrow (0,1,0,0,0,0,0,0,0)$

$quick \rightarrow (0,0,1,0,0,0,0,0,0)$

USW. ...

Diese Vektoren fließen in das KNN ein. Dabei nimmt ein Wort zwei Repräsentationen ein von der Eingabeschicht zur versteckten Schicht v_w und von der versteckten Schicht zur Ausgabeschicht v'_w. Diese Repräsentationen werden von den Parametern (Gewichten und Bias) des Netzes beeinflusst.
Da bei dem CBOW-Verfahren mehrere Wörter (Kontextwörter *the* und *brown*) in das KNN einfließen, entsteht der Eingabevektor v_w durch Konkatenation oder Bildung des Durchschnitts der Vektorrepräsentationen.
Anschließend wird die Vorhersage durch eine Softmax-Klassifizierung getroffen. Vereinfacht dargestellt gibt diese Funktion einen Vektor aus normierten Wahrscheinlichkeiten (Summe aller Wahrscheinlichkeiten $= 1.0$) zurück.
Zum Beispiel $(0.3,0,0.5,0,0,0,0,0.1,0.1)$, wobei die Indexposition die Klasse, also den Worttyp identifiziert. In diesem Fall wurde aus den Wörter *the* und *brown* das Zielwort *quick* mit einer Wahrscheinlichkeit von 0.5 korrekt bestimmt.

Der Fehlerwert wird pro Epoche bestimmt und anschließend die Parameter des KNNs optimiert, sodass in der nächsten Iteration eine verbesserte Vorhersage zu beobachten ist. Analog dazu verläuft das Skip-Gram-Verfahren, wobei hier die Richtung der Vorhersage invertiert ist.

Die Berechnung der Softmax-Klassifizierung stellt sich als zu rechenaufwendig heraus, da für die Normierung der Wahrscheinlichkeiten alle Wörter des Vokabulars betrachtet werden müssen. Die Anzahl der Worttypen bestimmt somit die Effizienz des Verfahrens. Da ein Wortkorpus, wie z.B. der Google News Corpus, mit ca. 6 Mrd. Worttypen[38] sehr groß ist, ist die Effizienz ein maßgebliches Kriterium, um das Verfahren praxistauglich zu machen und damit auch die Anwendung des Verfahrens in dieser Arbeit zu ermöglichen.

Daher ist eine Besonderheit an dem beschriebenen Vorgehen, die Effizienz der Vorhersageberechnung von P durch die hierarchische Softmax-Klassifizierung. Diese nutz die Strukturierung des Huffmannbaums aus.

Huffmannbaum

Die unterschiedlichen Wörter eines Vokabulars werden durch den binären Huffmannbaum [21] kodiert. Die Idee ist häufig vorkommende Wörter in möglichst kurze Gegenstücke zu kodieren. Dabei ist das Codealphabet die binäre Menge $0, 1$.
Nach der Huffmannkodierung kann jedes Wort des Vokabular mit einem binären Code repräsentiert werden. Durch das Ausnutzen der Baumstruktur bei der Kostenfunktion müssen für die Berechnungen der Vorhersagewahrscheinlichkeit nicht mehr W sonder nur $\log(W)$ Betrachtungen durchgeführt werden, wobei W die Größe des Vokabulars ist.
Jetzt kann das Verfahren auch bei einem großen Textkorpus angewendet werden.

Die Vektoren, die sich für jedes Wort des Vokabular aus der Eingangscodierung und den Parametern der verdeckten Schicht ergeben, werden als eingebettete Wortvektoren (engl. Embedded Vectors) bezeichnet. Diese erfassen semantische Relationen, als indirektes Ergebnis der Wortvorhersage[32], was in [38] gezeigt wurde.
Weiter kann die semantische Nähe der Vektoren durch Ähnlichkeitesmetriken gemessen werden. Das Verfahren kann daher die Problemstellung *P1* lösen.

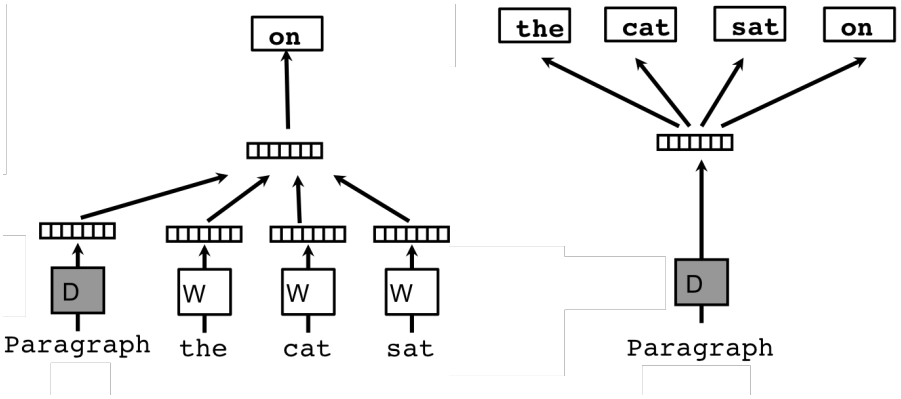
Da die Worteinbettung keine zusätzlichen Information an den Trainingsdaten und somit keine gelabelten Daten benötigt, kann die Problemstellung *P2* erfüllt werden.

Produkte bestehen jedoch nicht nur aus einzelnen Wörtern. Daher müssen die Wortvektoren eines Textabschnitts (z.B. Produtkbeschreibung) kombiniert werden, um einen Vektor für den gesamten Textabschnitt zu erhalten. Um semantische Relationen nicht nur zwischen den Wörtern zu erfassen, sondern auch zwischen ganzen Textabschnitten, wird das folgende Verfahren vorgestellt.

4.3 EINBETTUNG VON DOKUMENTEN

Für die Erstellung von Dokumenteinbettungsvektoren wird die oben beschriebene Methodik um einen Paragraphenvektor ergänzt. Dieser stellt nicht wie im oberen Bespiel ein einzelnes Wort dar, sondern den gesamten Textabschnitt.

Der Paragraphenvektor soll zur Vorhersage des nächsten Wortes beitragen. Die Verfahrensweise gleicht sich mit dem oben geschilderten Ablauf für die Erstellung von Worteinbettungen, nur das zusätzlich ein Vektor für die Vorhersage des nächsten Wortes hinzugefügt wird. In Abbildung 10a ist die Hinzunahme eines Paragraphenvektors abgebildet, der wie ein zusätzlichen Wort, welches den Kontext des ge-



(a) PV-DM-Architektur (b) PV-DBOW-Architektur

Abbildung 10: Modelle für die Erstellung von Paragraphvektoren [32].

samten Textabschnitts speichert, gesehen werden kann [32]. Daher trägt das Modell den Namen Distributed Memory (PV-DM). Weiterhin berücksichtigt das Modell die Wortreihenfolge.

Neben der Wortvorhersage aus einem Kontext und einem Paragraphenvektor stellt [32] einen Ansatz vor, indem der Kontext ignoriert wird und das Modell Wörter, die im Textabschnitt vorkommen aus dem Paragraphenvektor vorhersagen muss (Abbildung 10b).
Die Berechnung dieses Modells stellt sich im Gegensatz zum oberen Verfahren als ressourcenschonend heraus und ist mit dem Skip-gram-Model aus [37] verwandt. Jedoch berücksichtigt das Verfahren Distributed Bag-of-Words of Paragraph Vectors (PV-DBOW) nicht die Wortreihenfolge (daher die Teilbezeichnung Bag-of-Words) in der Erstellung der Dokumenteinbettungsvekotren [32].

Das Verfahren der Dokumenteinbettung erfüllt Teile der Problemstellungen *P1* und *P2*. Es erstellt geeignetere Vektorrepräsentationen aus Textabschnitten als das Verfahren der Worteinbettung. Somit wird dieses Modell, genauer das Distributed Memory Model of Paragraph Vectors (PV-DM)-Modell für die Transformation von Produktdaten in Vektoren genutzt.

4.4 TIEFE NEURONALE NETZE

Nach der Erstellung von Vektoren fester Länge wird nun ihre weitere Verwendung dargestellt. Ziel ist es Muster aus den erstellten Einbettungsvektoren mittels tiefen KNNs zu extrahieren. Die daraus extrahierten Vektorrepräsentationen sollen den Ähnlichkeitsvergleich der in *P1* gefordert ist, verbessern. Hierzu sollen Autoencoder zum Einsatz kommen. Als Eingabedaten dienen die Paragraphvektoren, welche aus den Texten der Produktattribute erstellt wurden.

Aus der Einführung in Kapitel 2.3.2 geht hervor, dass sich eine bei AEs die gewünschte Ausgabe aus der Eingabe ergibt. Dadurch können die verwendeten Daten ungelabelt sein und der ganze Vorgang kann als unüberwachter Lernvorgang bezeichnet werden [18, S. 505.].

Durch diesen Umstand kommt es zu einer näheren Betrachtung von Autoencoder. Bei ihrer Verwendung ist allerdings zu beachten, dass ausreichend Trainingsdaten zur Verfügung stehen sollten, da Autoencoder versuchen alle Variationen der Eingabedaten festzuhalten und bei nur wenigen Trainingsdaten keine nützlichen Repräsentationen erlernen können [3, S. 47.]. Folgend sollen mögliche AE-Konzepte besprochen werden.

4.4.1 *Stacked Autoencoder*

Wie in Kapitel 2 beschrieben, ermöglichen KNNs eine hierarchisierte Modellierung von Konzepten. Dies ist auch mit AEs möglich. Durch das Addieren von weiteren Kodier- und Dekodierschichten erhöht sich die Abstraktion des latenten Verständnisses.
Der SAE baut sich wie folgt auf:

$$E \to h_{1,enc} \to h_{2,enc} \to \cdots \to h_{n,enc} \to h_{n,dec} \to \cdots \to h_{2,dec} \to h_{1,dec} \to A$$

E, A bezeichnet die Ein- und Ausgabeschicht. Alle Schichten, welche für die Kodierung zuständig sind, werden mit enc hinter ihrem Index dargestellt. Im Gegensatz dazu werden alle Dekodierschichten mit einem dec gekennzeichnet.
Nach dem Trainingsprozess kann die gewünschte Vektorrepräsentation aus den Eingabedaten kodiert werden. Dabei setzt sich der Encoder aus den Schichten $E, h_{1,enc}, h_{2,enc}, \ldots, h_{n,enc}$, zusammen.

4.4.2 *Dimensionalitätsreduktion*

Unter dem Begriff Dimensionalitätsreduktion ist die Verringerung von zufälligen Variablen zu verstehen. Genauer bedeutet es in dieser Arbeit die Komprimierung von Eingabevektoren, um besten falls deren Aussagekraft zu erhöhen.
Bei der Verwendung von AEs ist daher nicht die Ausgabe r wesentlich, sondern viel mehr die erlernte Repräsentation der Kodierungsfunktion h.

$$Enc = h = f(x)$$
$$Dec = r = g(h)$$

Eine Möglichkeit, um aussagekräftige Repräsentationen zu erlernen ist die Dimensionalität[1] der verdeckten Schicht merklich zu verkleinern. Durch das Forcieren dieser Unvollständigkeit der verdeckten Schicht wird der AE dazu gezwungen die am stärksten hervorstechen-

1 MIt Dimensionalität ist hier die Anzahl der künstlichen Neuronen gemeint.

den Merkmale der Eingabedaten zu erfassen [3].

Eine Dimensionalitätsreduktion ist eine wichtige Restriktion für AEs jedoch werden noch anderen Verfahren genutzt um die kodierte Repräsentation zu verbessern. Im Gegenteil ist eine Reduktion der Dimensionen für manche Anwendungsfälle ungünstig, da stark komprimierte Daten hochgradig miteinander verstrickt sind und somit keine Weiterverarbeitung von Teilmengen dieser möglich ist [3, S. 45f.].

4.4.3 *Sparsamkeitsbedingungen*

Ein anderer Ansatz für die Nutzung von AEs ist es die Größe der versteckten Schicht nicht oder nur leicht zu komprimieren und diese mit ähnlicher Größe, wie die der Eingabe- und Ausgabeschicht, zu belassen. Stattdessen wird die Anzahl der hindurchfließenden Informationen beschränkt [3, S. 45.] .

Realisiert werden kann dies durch das Hinzufügen eines Strafterms an die Verlustfunktion $L(x, g(f(x))) + \boldsymbol{\Omega}(\mathbf{h})$ [18, S. 505f.]. Das Resultat dieser Beschränkung ist die verminderte Aktivität der künstlichen Neuronen.
In der Praxis existieren auch Kombinationen aus einer verringerten Reduktion der Dimension verbunden mit der Anwendung von Sparsamkeitsbedingungen.

4.4.4 *Long Short Term Memory*

Recurrent Neural Networks können Abhängigkeiten zwischen Informationen über eine gewisse Zeitspanne modellieren. Kurze Abhängigkeiten zwischen den Informationen können dargestellt werden, jedoch entstehen Schwierigkeiten, wenn zwischen Informationen langlebige Abhängigkeiten bestehen [18, S. 401f.].

Zum Beispiel kann ein Sprachmodell alle nötigen Informationen für die Wortvorhersage, „Eine Kuh macht **Muh**“ aus einem kurzweiligen Kontext beziehen. Betrachtet man aber folgendes Beispiel:
„Ich wuchs als in Kind in Italien auf.“ . . . „Ich spreche gut **italienisch**“ steht der benötigte Kontext, um das richtige Wort zu finden, weiter entfernt. Durch die Funktionsweise eines LSTM-Netzes (siehe Kapitel 2.3.2) können kurzweilige, aber auch an langlebige Abhängigkeiten[2], modelliert werden. LSTMs haben sich in der Vergangenheit bei der Verarbeitung von Textdaten bewährt und ihre erlernte Textrepräsentationen sollen nun auch in From eines LSTM-Autoencoders extrahiert und genutzt werden.

2 Daher der Begriff long short term

4.5 VERZAHNUNG DES KONZEPTS

In den oberen Abschnitten wurden statistische Sprachmodellierungen und Methoden, um Muster aus Zahlenvektoren zu abstrahieren vorgestellt. Die Idee ist es nun Textdaten mittels eines Sprachmodels zu transformieren. Die resultierenden Repräsentationen sind reelle Zahlenvektoren oder anders ausgedrückt, Worteinbettungen.

Diese Vektoren drücken bereits semantisches Verständnis aus. Weiterhin ist es notwendig, dass aus den unterschiedlich großen Textabschnitten verschiedenster Längen, Ausgaben mit einer fixen Größe geschaffen werden, damit diese geeignet sind, um von AE-Netzen weiterverarbeitet werden zu können. Auf Grund dieser, möglicherweise tiefen KNNs, kann jetzt eine Repräsentationsschicht erlernt werden, in denen ein Vektor entsteht, der auf Ähnlichkeit in einem semantischen Kontext geprüft werden kann.

Ein solcher Vergleich kann im Grunde mit einfachen Vektormetriken durchgeführt werden. Ein Beispiel für solch eine Metrik ist die Kosinus-Distanz [48].

$$\cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|}$$

Dabei ist $\cdot$ in diesem Fall das Skalarprodukt. Die Kosinus-Distanz von zwei Vektoren a und b, ist der Kosinus des eingeschlossenen Winkels θ. Der Wertebereich von $\cos(\theta)$ liegt zwischen -1 und 1.
Eine 1 bedeutet hier die Gleichheit zwischen den beiden Vektoren und -1 beschreibt genau entgegengerichtete Vektoren. Wenn $\cos(\theta) = 0$ ist, dann sind die beiden Vektoren unabhängig voneinander und damit ungleich.

Ziel ist es, die oben genanten Verfahren so einzusetzen, dass bei dem Vergleich der Match-Kandidaten signifikante Unterschiede zwischen M und U, bezüglich der ermittelten Wert der Kosinus-Distanz auftreten.

5

IMPLEMENTIERUNG: DEDUPLIZIERUNG DURCH KÜNSTLICHE NEURONALE NETZE

In folgendem Kapitel werden die Eigenschaften der Trainingsdaten untersucht. Diese Daten repräsentieren Produkte, welche bestimmte Attribute, wie z.B. Titel, Hersteller oder Beschreibung besitzen. Eine genauere Einführung der Produktdaten erfolgt im nächsten Kapitel 6.

5.1 VORVERARBEITUNG

Die Phase der Vorverarbeitung dient der Strukturierung und Bereinigung der rohen Produktdaten. Dieser Vorgang ist notwendig, da die anschließende Umwandlung in Einbettungsvektoren eine Sequenz von Wörter verlangt und keine zusammenhängende Zeichenkette. Weiter wird durch diese Phase die Steigerung der statistischen Aussagekraft von Wörtern ermöglicht, indem unterschiedliche Ausprägungen eines Wortes auf ihren Wortstamm reduziert werden.
Im bestmöglichen Falle verbessert die Vorverarbeitung die Evaluierungsergebnisse. Ein Codeausschnitt der Vorverarbeitung ist in Listing 1 abgebildet.

5.1.1 *Datenverunreinigungen*

Vor der eigentlichen Vorverarbeitung werden Daten unter den Gesichtspunkten der Problemstellungen *P6* bis P8 betrachtet.
Unvollständige Produktdaten (siehe *P6*) sollen durch eine Filterung reduziert werden, um somit die Trainingsphase der ML-Algorithmen zu verbessern. Als unvollständig gelten Produktdaten bei denen nicht alle verlangten Attribute definiert und durch Werte bestimmt sind.

Da die Trainingsdaten überdies mit HTML-Kodierungen verschmutzt sind (siehe *P7*) und diese Kodierungen keine nutzbaren Informationen enthalten, erfolgt der Versucht die Kodierungen in ASCII-Text zu transformieren (Zeile 10 in Listing 1). Falls dies nicht gelingt, werden die entsprechenden Daten ignoriert.
So kann das Eingangs erwähnte Beispiel wie folgt umgewandelt werden.

‘An absolutely gripping, absorbing historical crime thriller …

↓

'An absolutely gripping, absorbing historical crime thriller ...

```
stemmer = EnglishStemmer()
tokenizer = RegexpTokenizer('[a–zo–9]+')
p_tokenizer = SExprTokenizer(parens='{}')
english_stopwords = stopwords.words('english')

def clean(raw_text):
    assert match(raw_text, rule='URI_reference') == None

    text = BeautifulSoup(raw_text,'lxml').get_text()
    text = unicodedata.normalize('nfkd', text) \
        .encode('ascii', 'ignore').decode('ascii')

    assert len(p_tokenizer.tokenize(text)) == 1

    text = text.lower()
    text = tokenizer.tokenize(text)
    text = [w for w in text if not w in english_stopwords]
    text = [stemmer.stem(w) for w in text]
    text = [w for w in text if len(w) > 1]
    text = [w for w in text if not is_number(w)]
    return text
```

Listing 1: Vorverarbeitung einer Zeichenkette

Solche Daten die Sonderzeichen in Form von Inline CSS enthalten, stellen die gröbste Art der Datenverschmutzung dar. Um diese Daten von der weiteren Verarbeitungen auszuschließen, werden Klammerkonstrukte, welche mit { beginnen und mit } enden, ignoriert. Es erfolgt außerdem keine weitere Betrachtung von Daten, die nur aus URL- oder Pfadangaben bestehen, da diese keinen Informationsgehalt besitzen.

Ergänzend zu den oben geschilderten Maßnahmen werden Daten, welche nur aus einer Zeichenkette der Länge eins oder aus einer alleinstehenden Zahl bestehen, ebenfalls ausgeschossen.

5.1.2 *Umgang mit Groß- und Kleinschreibung*

In Kapitel 1.2 wurde eingangs diskutiert die Groß- und Kleinschreibung von Zeichenketten entweder zu missachten oder beizubehalten. Um eine bessere Entscheidung treffen zu können, folgt eine Betrachtung der fünf häufigsten Schreibweisen des Eigennamens iPhone, sortiert nach der Häufigkeit der Vorkommen in den verwendeten Trainingsdaten.

1. iPhone – 143 799
2. iphone – 13878

3. Iphone – 7 506
4. iPhones – 2 798
5. IPHONE – 1 487

15% der betrachteten Schreibweisen weichen von der Standardschreibweise des Eigennamens ab. Bei Beachtung der Groß- und Kleinschreibung würde somit 15% der semantischen Kontexte, die das Wort iPhone[1] enthalten, nicht in die Steigerung des semantischen Gehalts einfließen, da das Wort jedesmal anders geschrieben wird.
Andererseits verliert ein Eigenname seinen informativen Einfluss auch, wenn dieser kein Neologismus, sondern ein normales Wort ist (bspw. die Marke Sharp zu dem englischen Wort sharp).
Dieses Problem wird im Ausblick, Kapitel 7.2, hingegen versucht zu lösen. Für die Lösung der Problemstellung *P3* werden jedoch alle Textsequenzen in kleingeschriebene Buchstaben umgewandelt, damit die statistische Signifikanz der Trainingsdaten steigt.

5.1.3 *Tokenisierung*

In dieser Phase werden Textabschnitte auf ihrer Wortebene sequenziert. Die Implementierung erfolgt mit Hilfe des Natural Language Toolkit (NLTK) [5]. Das Toolkit umfasst verschiedene Funktionen (sogenannte Tokenizer), welche anhand von unterschiedlichen Kriterien zusammenhängende Text zerteilen.

Da es sich bei einigen Produktattributen, um ausformulierte Beschreibungen handelt, ist ein wichtiges Kriterium an den Tokenizer, das Entfernen von Punktationen und anderen Sonderzeichen.
Mittels dem regulären Ausdruck $[a-z0-9]+$, werden belieb lange alphanumerische Zeichen extrahiert. Der Nachteil dieser Methode ist, dass nun Wörter die Sonderzeichen enthalten (bspw. Eigenamen) sequenziert werden. Zur Veranschaulichung folgen zwei Beispiele:

1. RICHTIG „'Hard to put down !." → hard, to, put, down
2. FALSCH „Micro$oft is King!" → micro, oft, is, king

Im zweiten Beispiel ist zusehen, dass fälschlicherweise der Eigenname getrennt wird. Um eine Aussage über den Einfluss dieses Defizits treffen zu können, werden die am häufigsten vorkommenden Wörter des Titelattributes betrachtet. Hier ist die Wahrscheinlichkeit am größten auf Eigenamen in Form von Produkttiteln zu treffen.
Damit Eigennamen nicht wie in Beispiel 2 verloren gehen wurde die Tokenisierung in Zeile 16 im Listing 1 entfernt und durch Leerzeichentrennung ersetzt.

1 In diesem Fall ist das Wort jedoch anders geschrieben.

In den 100 häufigsten Wörtern, der Produkttitel, kommt auf Platz 74 die Bezeichnung „t-shirt“ vor, welche der einzige Eigenname mit Sonderzeichen ist. Im Kapitel 7.2 werden nichtsdestotrotz mögliche Verbesserungsansätze besprochen.

Da der Informationsverlust durch das falsche Auftrennen von Eigennamen mit Sonderzeichen nicht schwerwiegend ist, wird der Problemschwerpunkt *P4* durch den vorgestellten Tokenizer berücksichtigt.

5.1.4 *Entfernung von Stoppwörtern*

Nach der Zerteilung eines Textabschnitts wird versucht, Wörter mit geringen Informationsgehalt zu entfernen. Sogenannte Stoppwörter, die u.a von Adverbien, Pronomen oder Artikeln dargestellt werden, besitzen wenig bis gar keinen Informationsgehalt und erfüllen nur eine grammatikalische Aufgabe [15, S. 203.]. Mit Hilfe des NLTK werden die einzelnen Wörter einer Sequenz mit einer englischen Stoppwortliste verglichen und bei vorhanden sein entfernt (Siehe Zeile 17 in Listing 1).

Die statistische Signifikanz von Wörtern, welche keine Stoppwörter sind, wird in diesem Fall erhöht, indem Stoppwörter entfernt werden.

5.1.5 *Stemming*

Eine weitere Maßnahme, um die Signifikanz eines Wortes durch vermehrtes Auftreten zu steigern, ist das Stemming. Für das Zurückführen der verschiedenen morphologischen Ausprägungen eines Wortes auf seinen Wortstamm wurde ein English-Stemmer gewählt. Genauer handelt es sich bei diesem Stemmer-Algorithmus, um eine verbesserte Version des Porter-Stemmer-Algorithmus. Dieser entfernt, basierend auf sprachspezifischen Verkürzungsregeln, solange die Suffixe eines Wortes bis eine Minimallänge erreicht ist [41].

Ausschlaggebend für die Nutzung dieses Stemmers (Zeile 18 in Listing 1) waren die ausschließlich in englischer Sprache vorliegenden Trainingsdaten. Hierdurch soll auf die Problemstellung *P5* eingegangen werden. Anzumerken ist, dass auch hier Eigennamen verloren gehen, da die Entfernung der Suffixe, ohne Ausnahmen bei allen Wörtern erfolgt.

5.2 ANORDNUNG DER TRAININGSDATEN

Der Ausgang eines ML-Verfahrens ist stark mit der Auswahl der Trainingsdaten verbunden. Im Fall dieser Arbeit spielt die Anordnung der Wörter innerhalb der Trainingsdaten eine weitere Rolle. Grund hierfür ist das Konzept der distributionellen Hypothese (siehe Kapitel 4.1.2), welches die semantische Bedeutung definiert.
Unterschiedliche Anordnungen werden durch die Veränderung von Reihenfolge und Position verschiedener Produktattribute erreicht. Im Laufe dieser Arbeit wurden drei Datenanordnungen definiert.

Die erste Anordnung $\mathcal{O}_1$ besteht lediglich aus einem Beschreibungs-Attribut pro Zeile.
Die zweite Anordnung $\mathcal{O}_2$ gliedert jedes Attribut eines Produktes hintereinander an. Somit ergibt jedes Produkt P mit Titel t, Hersteller h und Beschreibung b eine Zeile $z_0 = t, h, b$ mit $t, h, b \in P$.
Die letzte Anordnung $\mathcal{O}_3$ legt pro Attribut eine Zeile in den Trainingsdaten mit $z_0 = t$, $z_1 = h$ und $z_2 = b$, wobei gilt $t, h, b \in P$, an.

Ziel ist es, das Modell nicht einzig und allein auf semantische Relationen zwischen den Wörtern innerhalb der Produktattribute zu trainieren, sondern ggf. auch Beziehungen von Wörtern zwischen verschiedenen Produktattributen zu fördern. Somit fließen Attribute nicht mehr unabhängig voneinander in den Vergleich von Produkten ein, sondern sind in deren Vektorrepräsentation korreliert.

5.3 PARAGRAPHVEKTOREN

Die bereinigten und geordneten Wortsequenzen sollen nun, mit Hilfe des in [32] vorgestellten Verfahrens, in eingebettete Paragraphvektoren umgewandelt werden. Dabei finden die in Kapitel 4.3 eingeführten Architekturen, DM-PV und D-BOW, ihren Einsatz.
Das Gensim Framework ermöglicht das unüberwachte Lernen von semantischen Modellen und wurde in Python realisiert. Weiter implementiert das Framework oben aufgeführten Verfahren auf Grundlage der Google Implementation von [38], aber erweitert diese, um die Möglichkeit ganze Textabschnitte (Paragraphen) umzuwandeln, auf Grundlage von [32]. Das Listing 2 zeigt das Erstellen des Doc2Vec Sprachmodells mit seinen Hyperparametereinstellung. Diese und deren Werte sollen im Folgenden erläutert werden.

Das Sprachmodell erfordert als Eingabe die bereitgestellten Trainingsdaten in Form von Wortsequenzen (Textabschnitt bzw. Paragprah) und einem ID-Tag. Der ID-Tag referenziert den Paragraphen. Somit kann später in der Evaluierung jeder Produktvektor identifiziert werden.

```
model = Doc2Vec(
        documents=training_data,
        iter=20, # Anzahl der Trainingsepochen
        workers=8, # Anzahl der Threads
        size=300, # Größe der Paragraphvektoren
        min_count=1 # Kleinste Wortfrequenz,
        window=8, # Größe des Kontextfensters
        sample=0 # Keine Sampling nutzen ,
        hs=1 # sondern Hierarchische Softmax
        )
```

Listing 2: Training des Doc2Vec Sprachmodells

```
input = Input(shape=(300,), name='input')
encode = Dense(300, activation='relu')
encode = Dropout(0.5, name='Dropout_1')(encode)
decode = Dense(300, activation='tanh')(encode)
```

Listing 3: Modell des Stacked Autoencoders

Die Größe der Ausgabevektoren wurde auf dem von [32] empfohlenen Wert belassen. Die Kontextfenstergröße k gibt an, wie groß der umgebende Kontext des Zielwortes ist. Der Wert für $k = 8$ ist aus [32] übernommen und dient als Ausgangswert bei der Evaluierung.

Da eine intensive Vorverarbeitung vorherging, wurde der min_count Parameter bei $f = 1$ belassen. Dieser ignoriert alle Wörter mit einer niedrigeren Frequenz als f im Vokabular. Dementsprechend werden keine Wörter ignoriert. Ebenfalls wird das Subsampling (Zeile 8), bei dem die Bedeutung von hoch frequentierten Wörten verwaschen wird, nicht verwendet.
sae = Model(input=input, output=decode)

5.4 AUTOENCODER

Nach der Erstellung der Paragraphvektoren soll ihre Produktrepräsentation weiter verbessert werden, indem die tiefste, Encoder-Schicht eines AEs extrahiert wird. Dazu gilt es diesen zuvor mit eben jenen Paragraphvektoren anzulernen.

Alle drei verwendeten AE-Arten (SAE, VAE, LSTM-AE) wurden mit Keras implementiert.
Keras ist eine in Python geschriebene KNN-Bibliothek, welche für die schnelle Entwicklung von Prototypen geeignet ist.
Die Implementierungen der AE nutzen keine Dimensionsreduktion

sondern verwenden Regulierungen, um den Neuronenfluss zu begrenzen. Ein Codebeispiel des SAE ist in Listing 3 zu sehen.
Dieser nutzt eine Dropout-Regulierung. Die Idee ist, durch das Zufällige deaktivieren von Verbindungen zwischen den künstlichen Neuronen eine Überanpassung des KNNs, an die Trainingsdaten, zu verhindern [45]. Die Dropout-Regulierung wird in diesem Fall (Zeile 3 in Listing 3) mit einer Wahrscheinlichkeit von 0.5 eine Neuronenverbindung deaktivieren.

5.5 HARDWARE

Im Verlaufe der Arbeit kristallisierten sich zwei Schwerpunkte an die zu verwendende Hardware heraus.

H1: Die Lernverfahren für die Erstellung der Paragraphvektoren und AEs benötigen eine große Menge an Daten, um ausreichende Schlussfolgerungen über diese treffen zu können. Gerade das Modell für die Erstellung von Paragraphvektoren muss das gesamte Vokabular der Daten, also alle Worttypen speichern. Damit stellt der Arbeitsspeicher eine Obergrenze dar.

Die AE-Implementierung mittels Keras stellt eine Generatorfunktion zur Verfügung, die Daten stapelweise aus dem Dateisystem lädt und somit nicht den gesamten Datensatz im Arbeitsspeicher halten muss. Dies ist möglich, da nach jeder Iteration die Parameter des KNNs, unabhängig von allen vorhergehenden und nachfolgenden Daten, berechnet werden.

H2: Ein KNN benötigt eine gewisse Anzahl an Trainingsepochen, um seine Parameter korrekt anzupassen. So konvergiert der Fehlerwert mancher KNN erst ab 1000, 5000 oder 10000 Iterationen. Damit dieser Prozess in nachvollziehbarer Zeit realisiert werden kann, ist eine angemessen Anzahl von Recheneinheiten notwendig.

Teile der Berechnungen für diese Arbeit wurden von einem Multiprozessorsystem durchgeführt, welches vom Competence Center for Scalable Data Services and Solutions (ScaDS) in Leipzig bereitgestellt wurde. Das System umfasst 128 Intel Xeon E7-8860 CPUs und ist mit über 6TB Arbeitsspeicher ausgestattet.
Besonders an diesem System ist die Non-Uniform Memory Access (NUMA) Speicherarchitektur. Prozessoren verfügen bei dieser Architektur über einen eigenen lokalen Speicher, können aber dennoch über einen verteilten, gemeinsamen Speicher auf den Adressraum anderer Prozessoren zugreifen. Dadurch kann theoretisch jede Anwendung, welche für eine herkömmliche Computerarchitektur programmiert wurde, ohne weitere Umstände auf das Multiprozessorsysteme

portiert werden.

Die ausgeführten Python-Programme wurden nicht explizit für Multiprozessorsysteme in der obigen Größenordnung optimiert, verfügen jedoch über hochgradig optimierte Basic Linear Algebra Subprogram (BLAS) Implementierungen. BLAS sind hardwarenahe Implementierungen für lineare Algebraoperationen wie Vektor- oder Matrizenmultiplikationen.

Weiter kann bei den verwendeten Anwendungen angeben werden, wieviele Arbeitsprozesse für eine parallele Abarbeitung genutzt werden sollen. Es wurden verschiedene Angaben ausprobiert, um zu beobachten wie sich eine Kommunikationslatenz, bedingt durch den verteilten Arbeitsspeicher ‚auf die Rechenzeit auswirkt.

6

EVALUATION

Die Qualität der Evaluierungsergebnisse wird durch die Effektivitätsmaße Recall (Vollständigkeit) und Precision (Genauigkeit) gemessen. Der Recall einer Abfrage gibt an, wie vollständig, d. h. wie fähig, ein System relevante Dokumente aus der Menge aller Dokumente auswählen kann. Die alleinige Verwendung des Recalls reicht jedoch nicht aus, da keine Angaben über nicht relevante Dokumente, sog. Ballastdokumente in der Rückgabemenge gemacht werden können. Die Einführung des Percision-Maßes ermöglicht die Angabe, in welchem Verhältnis Ballastdokumente in der Rückgabemenge enthalten sind [30, S.399f.]. Dabei werden beide Maße gleichgewichtet.

Dieses Verfahren ist im Allgemeinen auch als F_1-Maß bekannt mit $F_1 = 2 * (percision * recall)/(percision + recall)$ und einem Wertebereich von 0.0 bis 1.0.

6.1 AUSWAHL DER VERGLEICHSDATEN

Für die Evaluierung von lern- und nicht lernbasierten Match-Verfahren wurden in [28] verschiedene Datensätze aus zwei Domänen verwendet. Betrachtet wurden drei Datensätze aus dem Gebiet der Bibliographie (DBLP[1], ACM[2] und Scholar[3]) und vier Datensätze aus dem Gebiet des E-Commerce (Abt[4], Buy[5], Amazon und Google Search für Produkte[6]). Hierdurch kam es zu folgenden F_1-Werten, wobei nur die besten Werte aus der Evaluierung ausgewählt wurden [28]:

DBLP – ACM: 97.6

DBLP – SCHOLAR: 89.4

ABT – BUY: 71.3

AMAZON – GOOGLE: 62.2

Für die Bewertung der Ergebnisse dieser Arbeit wurden die Produktdaten aus der Deduplizierung von AMAZON – GOOGLE verwendet. Wie oben gezeigt, erzielten diese im Vergleich zu den anderen Datensätze die schlechtesten Ergebnisse und sind somit die anspruchsvollsten Daten für eine Deduplizierung.

1 Digital Bibliography & Library Project
2 The ACM Digital Library
3 Google Scholar
4 `http://www.abt.com`, Online-Händler
5 `http://www.buy.com`, Online-Händler
6 Google Search API

6.2 VERWENDETE DATEN

Produktdaten

In der vorliegenden Aufgabenstellung werden Produkte als die Summe ihrer Attribute definiert. Ein Produkt besteht aus Titel, Hersteller, Beschreibung und Preis. Zusätzlich ist jedes Produkt mit einer eindeutigen ID versehen, die jedoch nichts über dieses aussagt. Daher wird die ID nicht zum Vergleich von Produkten benutzt. Für alle Attribute die Sprache beinhalten, gilt, dass diese Englisch ist.

Ferner wurde sich dafür entschieden, dass das Preisattribut keine weitere Betrachtung in der Erstellung von Produktvektoren erhält. In Kapitel 4.1.2 ist gezeigt worden, dass Wörter, welche sich oft in einem selben Kontext befinden semantisch korreliert sind. Dabei wird ein Preisattribut als Zeichenkette aufgefasst und nicht etwa als Zahl. Dadurch geht jede Information einer numerischen Angabe (bspw. die Nähe zu einem anderen Produktpreis) verloren. Auch kommen die Preisangaben der Produkte in zu hoher Variation (19.99€; €19.95; 19.91) vor, als das ein häufiges Vorkommen eines immer selben Preises auftreten würde. Somit führt der Produktpreis zu keiner Verbesserung der Produktrepräsentationen. Diese worden im übrigen auch nicht in [28] verwendet.

Die vorliegenden Produktdaten setzten sich aus 1 363 Amazon-Produkten A und 3 226 Google Produktdaten G zusammen. Das kartesische Produkt aller Produkttupel (A, G) enthält somit insgesamt 4 397 038 Vergleichskandidaten, die aus 4 395 738 non Matches und 1 300 Matches bestehen.
Beide Datenquellen umfassen die selbe Attributmenge mit den folgenden Elementen:

id Eine Kennzeichnung, welche das Produkt beim Anbieter eindeutig referenziert.

title / name Der Name des Produkts.

manufacturer Der Hersteller des Produkts.

price Der Preis des Produkts.

Außerdem besitzen alle Attribute den gleichen Wertebereich, bestehend aus Zeichenketten. Eine Ausnahme stellt das Attribut des Preises dar, welches in der Datenquelle A ausschließlich aus reellen Zahlen besteht, wohingegen sich der Preis in Datenquelle G aus einer Kombinationen von Zahlen und Zeichenketten (*13.1 gbp*) zusammensetzt.

Da die Anzahl an Produkten mit 4 589 nicht ausreichend für das Lernen von Produktrepräsentationen ist, musste eine neue Datenquelle

	Amazon-Produkte	Google-Produkte	Amazon Meta
id	1363	3226	9430088
title	1363	3226	**7997369**
description	**1248**	**3035**	**5701344**
manufacturer	1363	**232**	**1808999**
price	1363	3226	**6063208**

Tabelle 4: Anzahl der Attributvorkommen in den verwendeten Datenquellen.

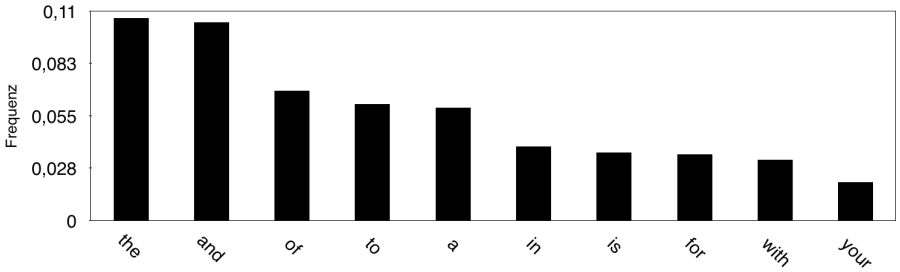
hinzugezogen werden. Diese beinhaltet keine gelabelten Daten, wobei das für die verwendeten Methodiken in Kapitel 4.1 bzw. Kapitel 4.4 nicht von Bedeutung ist und durch *P2* beachtet wurde.
Die von Julian McAuley[7] und seinem Team extrahierten Daten umfassen ca. 9.4M Metadaten aus Amazon-Produkten. Für die Verwendung wurden die selben Attribute, wie aus den Datenquellen A und G, ausgewählt.

Die Anzahl der Attribute, welche nutzbare Daten enthalten, sind in Tabelle 4 festgehalten. Hervorgehobene Zahlen kennzeichnen unvollständige Daten. Nicht berücksichtigt wurden Daten, die zwar enthalten sind, jedoch keinen Informationsgehalt besitzen wie beispielsweise eine leere Zeichenkette oder ein Preis von Null[8].
Zu sehen ist, dass die Attribute des Amazon-Produktdatensatzes fast vollständig vorhanden sind. Im Gegensatz dazu enthalten nur 7% des Google-Produktdatensatze brauchbare Werte. Der Datensatz der Amazon-Produktmetadaten belegt ebenfalls nur 19% der Herstellerattribute mit einem Wert.
Im weiteren Verlauf werden die Datenquellen A und G zusammengefasst und als Amazon-Google-Datensatz (AG-D) bezeichnet. Die Amazon-Metadaten werden in Amazon-Meta-Datensatz (AM-D) umbenannt.

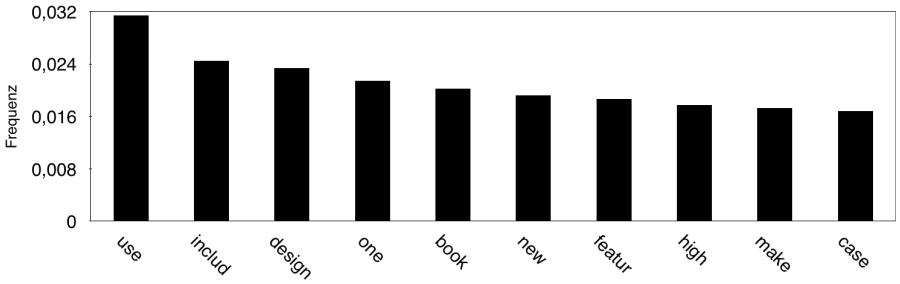
Datenstruktur

Die vorgestellten Produktdaten liegen in unterschiedlichen Formaten vor. Der AG-D, der Universität Leipzig, ist in strukturierten CSV-Daten verkörpert.
Im Gegensatz dazu liegt der AM-D in einer nicht validen JSON-Datei vor. In der Datei selbst ist kein vollständiges JSON-Objekt, in dessen Datenstruktur sich alle weiteren Produkte befinden, serialisiert. Ein Produkt wird pro Zeile als JSON-Objekt dargestellt und auch durch eine neue Zeile in der Datei getrennt.
Der AM-D ist durch die Extraktion von Amazon-Webseiten entstanden [36], was den weitaus höheren Grad der Datenverunreinigung, im Vergleich zum AG-D, erklärt.

7 `http://jmcauley.ucsd.edu/data/amazon/links.html` (Stand 08.12.2016)
8 Gemeint ist die Zahl Null.



(a) Ohne Vorverarbeitung



(b) Mit Vorverarbeitung

Abbildung 11: Frequenz der häufigsten Wörter aus den Produktbeschreibungen des AM-D.

	M1	M2	M3	M4
Epochen	20	20	1000	20
Trainings-daten	1 534 497	4 472 433	4 600 179	5 501 743
Anordnungen	$z_0 = t + b + d$	$z_0 = d$	$z_0 = t$ $z_1 = b$ $z_2 = d$	$z_0 = d$
Vorverarbeitet	Ja	Ja	Ja	Nein
Subsample	0	0	0	1e−6
Wort-typen	976 618	2 131 383	976 618	10 349 661
Wort-anzahl	83 122 819	229 964 452	83 122 819	131 057 318

Tabelle 5: NNLM-Modelle und ihre Hyperparameter für die Erstellung der Paragraphvektoren.

6.3 EVALUIERUNGSVERLAUF

6.3.1 *Vorverarbeitung*

In Abbildung 11 werden die Frequenzen der häufigsten Wörter der Produktbeschreibung, ohne und nach der gesamten Vorverarbeitung, gegenübergestellt. Die abgebildeten Wortfrequenzen wurden über die 100 am häufigsten vorkommenden Wörter normiert.
Nach einer Vorverarbeitung treten offensichtlich keine Stoppwörter mehr auf. An ihrer Stelle kommen nun Wörter mit produktbeschreibendem Charakter vor, wie die Wörter new, featur oder high zeigen.

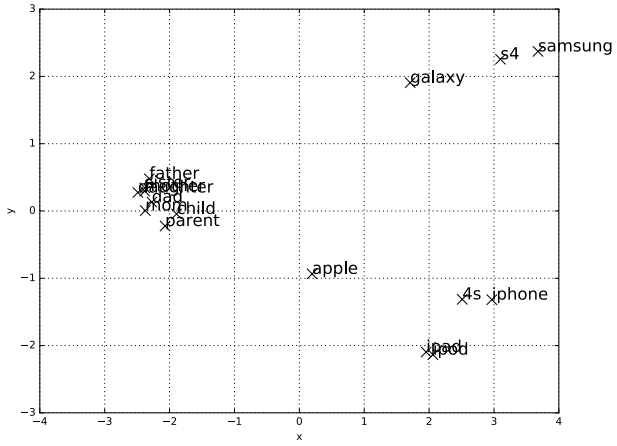
6.3.2 *Paragraphvektoren*

Als Grundlage der Evaluierung wurden mehrere Modelle für die Erstellung von Paragraphvektoren, mit unterschiedlichen Hyperparametern formuliert. Tabelle 5 zeigt alle erstellten Modelle mit ihren Metadaten und Parametern.
Die bezeichnendsten Unterschiede zwischen den Modellen stellen die verschiedenen Anordnungen der Trainingsdaten dar.
Weiter wurde das Modell M3 über 1000 Trainingsepochen trainiert, um möglicherweise Unterschiede der F_1-Werte, im Vergleich zu den anderen Modellen festzustellen.
Damit der Effekt der Vorverarbeitung beobachtet werden kann, wurde das letzte Modell M4 mit unbearbeiteten Trainingsdaten angelernt. Um den Einfluss von unbedeutenden Stop- und Füllwörter zu mindern wurde in diesem Modell das Subsampling, mit einem aus [31] vorgeschlagenen Schwellenwert trainiert.
Alle Modelle nutzen das PV-DM-Verfahren sowie die hierarchische Softmax-Klassifizerung, um die Paragraphvektoren zu erstellen.

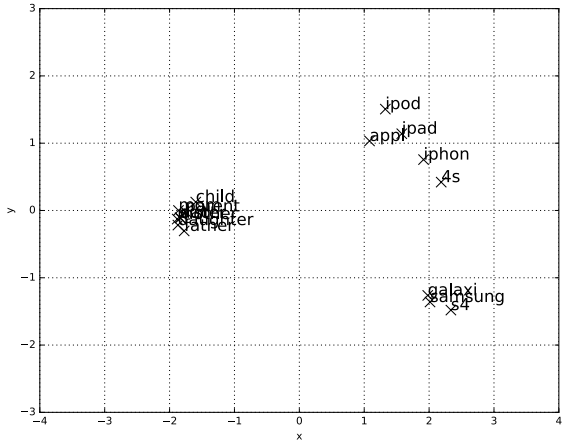
Hauptkomponentenanalyse

Im Anschluss soll der semantisch wertvolle Charakter der Modelle veranschaulicht werden. Hierbei werden die Paragraphvektoren durch eine Hauptkomponentenanalyse (PCA) auf zwei-dimensionale Vektoren reduziert. Eine PCA verringert eine Menge an untereinander korrelierende Variablen, auf einige wenige [12, S. 661.].
Dies ermöglicht die Visualisierung der Vektoren im zweidimensionales Raum. Verglichen werden die Modelle M4 (Abb. 12a) und M2 (Abb. 12b). Die Daten in Modell M4 wurden keiner Vorverarbeitung unterzogen, im Gegensatz zu den Daten des Modells M2. Dabei kamen für diese Gegenüberstellung zum einen produktnahe Begriffe, wie iphone, apple, 4s, s4 und Begriffe mit keinem Produktbezug, wie parent, child oder mother, zur Auswahl. Der Vergleich soll erste Tendenzen über die Auswirkung einer Vorverarbeitung aufzeigen.

Es ist bei beiden Abbildungen zu beobachten, dass eine Vorverar-



(a) Ohne Vorverarbeitung



(b) Mit Vorverarbeitung

Abbildung 12: Visualisierung der eingebetteten Vektoren mittels PCA.

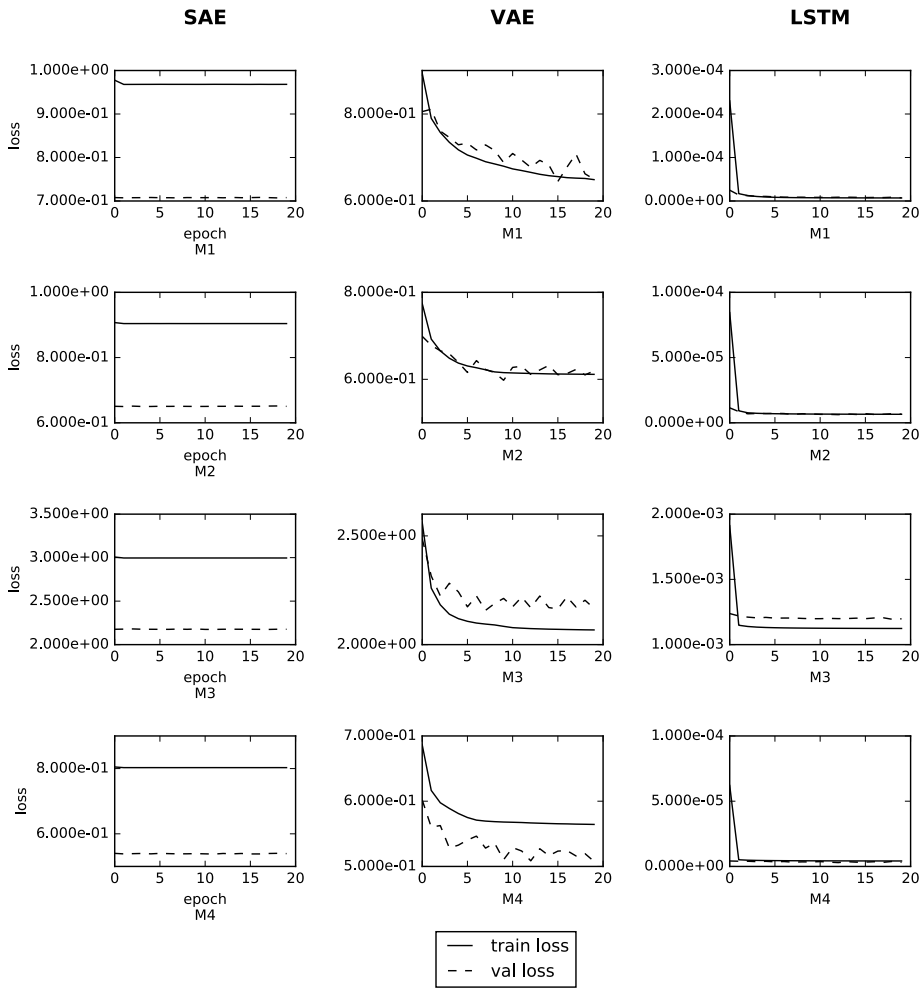


Abbildung 13: Verlauf der Verlustwerte

beitung auf die Clusterung von semantisch Nahen begriffen keine Auswirkung hat. Beide Modelle tun dies korrekt. Lediglich die Nähe der Begriffe innerhalb der Cluster, ist bei den Daten, welche vorverarbeitet wurden enger, was deren Zugehörigkeit verstärkt. Weiter ist bemerkenswert, dass selbst filigrane Unterschiede in der Bedeutung wie bei 4s (Smartphone von Apple) und s4 (Smartphone von Samsung), die Produkte in das richtige Cluster eingeordnet werden.

6.3.3 *Autoencoder*

Anschließend dienen die Paragraphvektoren der Modelle M1 bis M4 als Eingabedaten für drei unterschiedliche Autoencoder, einen SAE, VAE und einen LSTM-AE.
In Abbildung 13 ist der Verlauf der Trainingsepochen für die unterschiedlichen AEs zu sehen. Die Diagramme zeigen den Verlustwert zu der jeweiligen Trainingsepoche an.

Der SAE und der VAE nutzen die euklidische Verlustfunktion, um die Nähe der Ausgabevektoren zu den Eingabevektoren zu messen. Der LSTM-AE nutzt die klassische mittlere quadratische Abweichung, um den Fehlerwert zu berechnen. Daher entstehen die unterschiedlich großen Wertebereiche in Abbildung 13. Betrachtet wurde das Verhalten der ersten 20 Trainingsepochen.

Auffällig ist, dass die Verlustwerte von SAE und LSTM schon nach wenigen Epochen konvergieren. Beim Variational Autoencoder ist noch am ehesten eine Lernkurve zu erkennen, jedoch pegelt sich diese schnell ein.
Gut zuerkennen sind auch die Auswirkungen der Regulierung des SAE. Alle Validierungsverluste des SAEs sind kleiner als die Trainingsverluste, da nur während des Trainings die Regulierungsbeschränkungen aktiv sind.
Des weiteren sind die Verlustwerte bei den Modellen M1, M2 und M4 sehr klein. Während bei der Nutzung des Modells M3, die Fehlerwerte, bei SAE und VAE, höher liegen. Die sehr kleinen und schnell konvergierenden Verlustwerte deuten stark darauf hin, dass die AE-Modelle keine abstrakteren Repräsentationen, der Paragraphenvektoren, erlernen. Jedoch ist auch keine Überanpassung zu beobachten, da die Fehler der Validierungen nicht stark von denen der Trainingsverluste abweichen. Durch die hohe Anzahl an Trainingsdaten ist eine Überanpassung unwahrscheinlich.

6.3.4 *Deduplizierung*

Blocking

In [28] wurde die Anzahl der nötigen Tupelvergleiche durch ein Blocking mit Trigram-Ähnlichkeit und niedrigem Schwellenwert verringert. Bei dem AG-D konnte somit die Anzahl der Vergleiche von ca. 4.4M auf 342 761 reduziert werden.
In dieser Arbeit wurde ein Blocking durchgeführt, indem die Paragraphvektoren aus den Modellen M1, M2, M3, M4 mit einem niedrigen Schwellenwert von 0.15 verglichen wurden, sodass eindeutige non-Match-Paare aus der Menge entfernt werden konnte. Dadurch verringerte sich die Anzahl der möglichen Match-Paare von ca. 4.4M auf 248 636, was eine relative Verbesserung von 27%, zu [28], darstellt.

Bei dem Vorgang gingen 4% der echten Match-Paare verloren. Dies kann durch einen niedrigeren Schwellenwert verhindert werden. Es wurde beobachtet, dass die Änderung des Schwellenwerts keinen explosionsartigen Anstieg der non-Match-Paare verursacht und somit auch bei Anpassung des Schwellenwerts nach unten, die Menge der

	M1	M2	M3	M4
Blocking Ergebnisse	293 952	**248 636**	501 843	1 469 750
Match-Verlust	3.5%	4.0%	3.0%	4.2%

Tabelle 6: Ergebnisse des Blockings durch den Ähnlichkeitsvergleich der Produktrepräsentationen.

verbleibenden non-Match-Paare trotzdem gering bleibt.
Grund hierfür ist die relativ gute Stabilität der ermittelten Ähnlichkeitswerte der Paragraphvektoren. Der Durschnittswert der Ähnlichkeit während des Blockings lag bei 0.32 für alle Match-Paare, mit einer Standardabweichung von 0.13. Der Durchschnitt der non-Match-Paare lag bei 0.09, mit einer Standardabweichung von 0.06. Die hohe Diskrepanz der Werte führt zu der Stabilität des Verfahrens.
In Tabelle 6 sind die Ergebnisse des Blockings aller Modellen abgebildet.

Ähnlichkeitsmaße

Nach der Reduzierung der zu vergleichenden Tupel erfolgte die Berechnung der Ähnlichkeiten. Dazu wird aus dem jeweiligen Modell M_x, der Paragraphvektor in die jeweilige Repräsentation des Autoencoders kodiert. Die Ermittlung der Ähnlichkeit erfolgt anschließend durch die Kosinus-Distanz.
Da die Nutzung von nur einer Ähnlichkeit eine unstabile Klassifizierung durch eine SVM zur Folge hat, werden noch weiter Ähnlichkeiten hinzugefügt.
Es werden die Zeichenkettenähnlichkeit von Titel und Beschreibung, durch Kosinus-Distanz und Trigram-Ähnlichkeit, hinzugefügt. Die Auswahl der Ähnlichkeitsmetriken für Zeichenketten erfolgte anhand der in [28] getätigten Auswahl.

Klassifizierung

Nun muss eine Entscheidung, bezüglich der Klassifizierung in Match oder non Match, getroffen werden.

Im Laufe dieser Arbeit wurde sich für eine lernbasierte Methode zur Klassifizierung, mittels SVM entschieden. Da eine SVM ein beobachtetes Lernverfahren ist hängt die Leistung des Lerners stark von den verwendeten Trainingsdaten ab. Diese sollten eine ausgewogen Menge an Match und non-Match-Beispielen enthalten und repräsentativ für alle Paare sein. In den Trainingsdaten sollten sich viele der möglichen Fehler, oder Ausnahmen in der Verteilung der gesamten Daten, wiederspiegeln können [27].

Zur Auswahl der Daten wurde das in [27] vorgeschlagene Threshold-Equal-Verfahren genutzt. In diesem werden $n/2$ Match- und non-Ma-

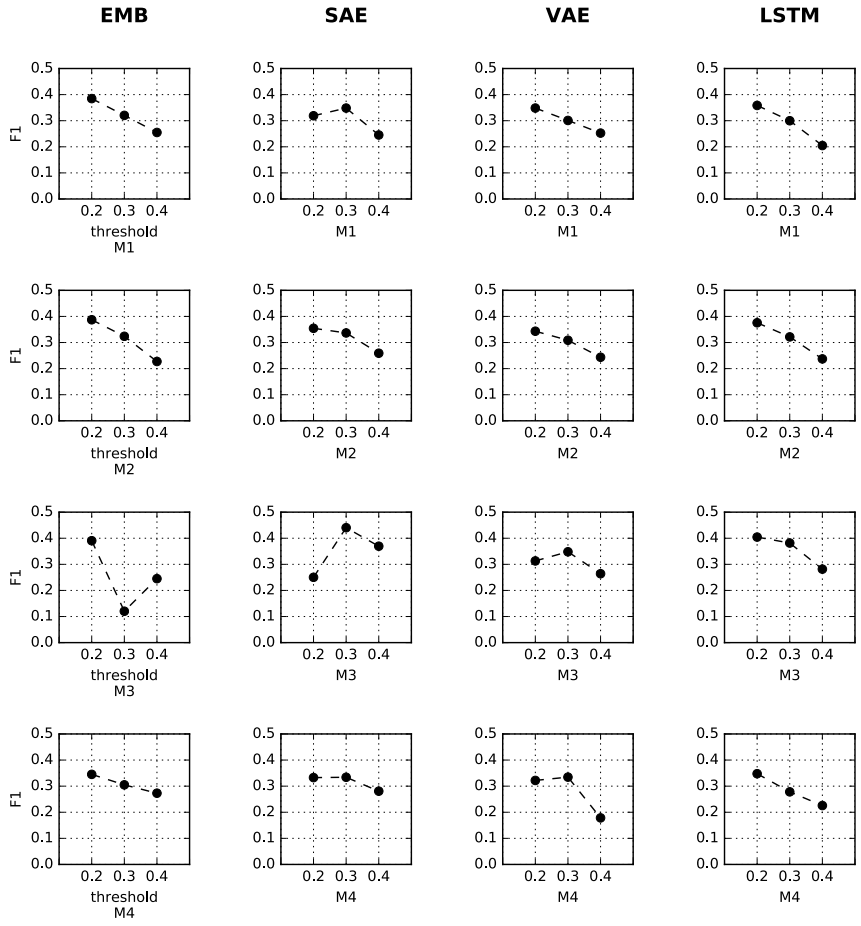


Abbildung 14: Evaluierungsergebnisse der Deduplizierung.

tch-Paare ausgewählt, indem mit einem Ähnlichkeitsmaß eine Vorauswahl von Paaren getroffen wird. Anschließend werden diese manuell in die entsprechende Klasse eingeordnet. Die Entscheidung zur Vorauswahl wird durch die Erfüllung eine Schwellenwertes entschieden. Die genutzten Schwellenwerte in dieser Arbeit sind 0.2, 0.3 und 0.4
Da die Perfekten Match-Ergebnisse des AG-D bekannt sind, musste keine manuelle Klasseneinteilung der ausgewählten Trainingsdaten erfolgen.

Die Ergebnisse der Deduplizierung sind in Abb.14 zusehen. Zu beachten ist, dass 0.5 und nicht 1.0 der höchste F_1-Wert in den Diagrammen ist. Das beste Ergebnis erzielte der SAE mit den Eingabevektoren aus Modell M3. Dieser erreichte eine F_1-Wert von 0.44 bei einem Schwellenwert von 0.3. Damit konnte der in [28] erreichte Wert, von 0.62, nicht verbessert werden.

Die Ergebnisse, welche nur durch die Repräsentation der Einbettungsmodelle (EMB) M1 bis M4 entstanden, stellen oft eine Obergrenze für die Ergebnisse der Autoencoder dar. Dies bestätigt die Vermutung aus Abschnitt 6.3.3, dass die gelernten AE-Repräsentationen keine Verbesserungen, zu den Repräsentationen der Modelle M1 bis M4 darstellen. Generell treten keine großen Abweichungen, im Vergleich zu den Ergebnissen der Einbettungsmodelle, auf.
Interessanterweise hat auch der Vorverarbeitungsprozess keinen merklichen Einfluss auf die Endergebnisse, welche unter M4 in Abbildung 14 zusehen sind. Die relativ besten F_1-Werte werden durch den Schwellenwert 0.2 erreicht, was die sehr niedrige gesamt Ähnlichkeit der Repräsentationen aufzeigt.

6.3.5 *Technische Anmerkungen*

Abschließend sollen noch Anmerkungen zu Performanz der benutzen Hardware erfolgen.

Da der Amazon-Meta-Datensatz über 9.4M Produkte führt und die Vorverarbeitung aus teils aufwendigen Umformungen, Listenvergleichen etc. beruht, wurde sich für die Nutzung von Dask, einer Bibliothek für parallele, analytische Datenberechnungen [10] entschieden.
Grund für die Entscheidung war auch die Möglichkeit, Produkte aus dem Dateisystem einzulesen, parallel zu verarbeiten und anschließend wieder in das Dateisystem zu schreiben. Nötig war dies, da sonst die Kapazität des Arbeitsspeicher überschritten worden wäre.
Die Rechenzeit der Vorverarbeitung, mit über 9.4M Einträgen, betrug 2, 3 Stunden. Dabei kam ein Vierkernprozessor mit Hyper-Threading und ohne SSD zum Einsatz.

Für die Berechnungen der Paragraphvektoren wurde das, in Kapitel 5.5 vorgestellte, Multiprozessorsystem verwendet. Als Metrik für die Leistung des Systems wurde die Anzahl, der pro Sekunde verarbeiteten Wörter (w/s) gewählt und nicht die benötigte Zeit der Berechnungen, da Durchläufe auch frühzeitig abgebrochen werden mussten und das Programm somit nicht korrekt terminierte.
Es konnte festgestellt werden, dass w/s stark abhängig von der Anzahl der verwendeten Prozessoren ist. Beispielsweise war ein Durchlauf mit allen 128 Prozessoren (ca. 50000 w/s) langsamer, als jener, mit nur 16 (ca. 72000 w/s) Prozessoren. Dies ist möglicherweise auf den Kommunikationsoverhead zurückzuführen.
Bei jeder Durchlaufskonfiguration wurde genau die angegebene Anzahl an Kernen genutzt und die Lasten gleichmäßig verteilt. Die Nutzung des Arbeitsspeichers belief sich teilweise auf über 90GB.

Die Implementierung der AEs konnte nicht über das Multiprozessor-

system berechnet werden, da hier scheinbar der Kommunikationsoverhead zu hoch war, sodass es zu keinen vernünftigen Laufzeiten kam. Das Training der AEs konnte durch eine RAM schonende Arbeitsweise, auf einem herkömmlichen Vierkernprozessor, durchgeführt werden.
Besonders war der Einsatz eins Massiv-parallel Rechners, in Form einer GPU. Diese verringerte die Trainingszeiten einer Epoche teilweise um 88% (von 722s auf 88s) Prozent.

7

ZUSAMMENFASSUNG UND AUSBLICK

7.1 ZUSAMMENFASSUNG

In der vorliegenden Arbeite wurde ein Lösungsweg für das Problem der Deduplizierung mit künstlichen neuronalen Netzen beschrieben. Es entstand ein Arbeitsablauf, der in Abbildung 15 zu sehen ist.

Dieser beinhaltet die Erstellung eines neuronalen Sprachmodells (b) aus nicht gelabelten Produktdaten, welche einer Vorverarbeitung (a) unterzogen wurden.
Anschließend konnten die so erzeugten Vektoren, als Eingabedaten für Deep-Learning-Algorithmen dienen (c).
Die AE-Netze lernten erneut eine Repräsentation der Produktdaten.
Nach diesem Schritt wurden die erlernten Repräsentationen durch eine Vektormetrik verglichen und ihre Ähnlichkeit ermittelt. Für die

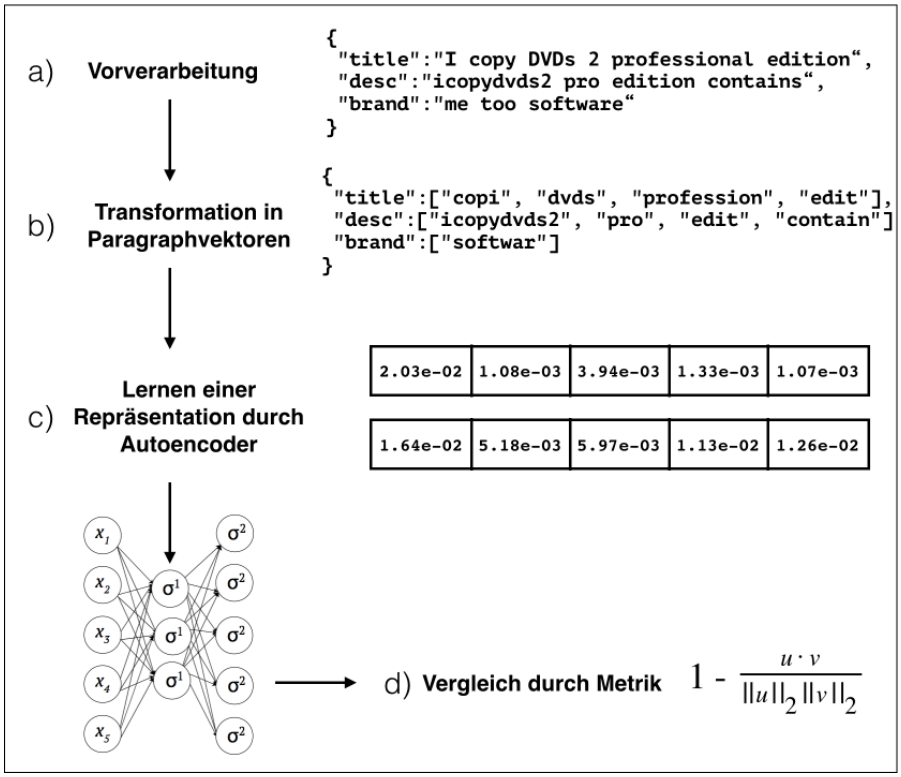


Abbildung 15: Arbeitsablauf

Einteilung in Matches oder non Matches wurde eine SVM, ein weiterer maschineller Lerner trainiert. Dieser teilte die Menge an Match-Kanditaten, welche nach einem Blocking-Schritt übrig blieben, end-

gültig in Matches oder non Matches auf.

Vor der eigentlichen Deduplizierung musste einer Vorverarbeitung der Daten unter verschiedenen Problemschwerpunkten (*P3* bis *P8*) betrachtet werden. Es konnte kein Einfluss der Vorverarbeitung auf die Resultate der Evaluierung festgestellt werden. Jedoch stellte sich die Verarbeitung von bereinigten Daten als performanter heraus, da diese keine unnötigen Informationen mehr beinhalteten.

In dieser Arbeit konnten KNNs in einem Deduplizierungprozess angewendet werden. Das große Vorkommen von nicht gelabelten Produktdaten konnten durch den Einsatz von unüberwachten Lernverfahren verwertet werden. Somit ist die Problemstellung *P2* erfüllt worden.

Bei der Erstellung der Paragraphvektoren konnte eine Vielzahl von Einstellungsmöglichkeiten der Hyperparameter für nachfolgende Betrachtungen ausgeschloßen werden.
Zudem wurde der Blocking-Schritt um 27% im Vergleich zu [28] durch ein Blocking mit Paragraphvektoren, verbessert.

Der gesamte Deduplizierungprozess konnte in der endgültigen Evaluation nicht überzeugen, da keine Verbesserung der F_1-Werte aus [28] stattfand.
Außerdem sind die ermittelten Ähnlichkeiten mit einem Durchschnitt von 0.32 sehr niedrig, die Diskrepanz zwischen Match- und non-Match-Tupeln jedoch sehr groß. Daher eignen sich die erstellten Produktrepräsentationen gut für den Blocking-Schritt, weisen aber für ein durchgehend korrektes Matching zu viele Ausreißer nach unten, in Bezug auf den Durchschnitt der Match-Paare auf.
Zurückblickend konnte jedoch eine Verfahren vorgestellt werden, welches den Vergleich von Produkten unter Einbeziehung der Semantik von Produktattributen ermöglicht. Somit ist die Problemstellung *P1* zum Teil erfüllt worden.

Die vorliegende Arbeit gibt eine Einführung in die Thematik der Deduplizierung, mit neuronalen Modellen. Die vorgestellten Ansätze knüpfen gut, an das stark mit Big Data verwobene Gebieten an und können die massiven Datenmengen verwerten. Dies ist auch in den neuesten Frameworks (z.B. Singa) verdeutlicht, welche sich auf verteiltes maschinelles Lernen mit Clustern konzentrieren, um die Menge an Daten in vertretbarer Zeit bearbeiten zu können.

Im Folgenden werden mögliche Verbesserungsansätze vorgestellt, die zur Erhöhung der Ähnlichkeit zwischen zwei Match-Paaren führen sollen.

7.2 AUSBLICK

Dieser Abschnitt stellt Ideen zur Verbesserung der vorgestellten Implementierung, aber auch Ansätze für die Veränderung der Architektur von [32] vor. Die Ideen können als Anknüpfungspunkte für nachfolgende Betrachtungen und Arbeiten verstanden werden.

7.2.1 *Verbesserung der Implementation*

Hyperparameter

Bei den Versuchen die Evaluierung durch andere Hyperparametereinstellungen zu verbessern konnte beobachtet werden, dass diese nur selten die Evaluierungsergebnisse signifikant veränderten.

Ungeachtet dessen sollte die Größe der Einbettungsvektoren bei nächsten Versuchen wesentlich erhöht werden. In [38] wurde angemerkt, dass zwar die Größe der Trainingsdaten stark anwuchs, jedoch die Größe der Vektoren, welche diese Daten verkörpern, nicht angepasst worden.
Die in dieser Arbeit verwendete Vektorgröße liegt bei 300. Vorschläge für neue Größen können aus [9] entnommen werden, dort kommen Größen von 10 000 zum Einsatz. Jedoch muss bei derartigen Versuchen entsprechende Hardware, wie z.B genügend Arbeitsspeicher, vorhanden sein.

Vortrainierte Modelle

Da der semantische Gehalt der Paragraphvektoren ausschließlich durch die vorhandenen Produktdaten geformt wurde, liegt es nahe, als Ausgangspunkt nicht diese zur verwenden, sondern ein vortrainiertes Modell zu benutzen. Dieses könnte bspw. aus dem gesamten englischen Wikipedia-Korpus angelernt werden. Anschließend wird das Modell mit den spezifischen Domaindaten (hier Produktdaten) verbessert.
Dies bedeutet das keine zufällige Wortvektorinitialisierung [32] für das lernen von Paragraphvektoren mehr erfolgt. Wegen der Verwendung des vortrainierten Modells fließen die Wortvektoren, als semantische Grundlage, in die Erstellung der Paragraphvektoren mit ein und erhöhen dadurch deren Aussagekraft.

Zusätzlich kann eine Kombination der unterschiedlichen Verfahren für die Erstellung der Paragraphvektoren Verwendung finden [32]. Dies bedeutet, dass jeweils ein Vektor mit PV-DBOW und ein Vektor mit PV-DM erstellt und dann durch Konkatenation oder Durchschnittsbildung vereint wird.

Bildmerkmale

Ein bisher ungenutztes Merkmale beim Vergleich von Produkten, ist das zum jeweiligen Produkt gehörende Bild. In den letzten Jahren stellten sich DL-Verfahren mit Bilddaten als sehr effektiv heraus und

sind das bisher erfolgreichste Anwendungsfeld von Deep Learning. Mit Hilfe eines Autoencoders kann eine Vektorrepräsentation der Bilder erlernt werden und dann diese mit einem Ähnlichkeitsmaß verglichen werden. Diese Vorgang erweitert somit die Deduplizierung um einen weiteren Ähnlichkeitswert.

Domainspezifischer Einfluss

Die Entscheidung während der Vorverarbeitung alle vorkommenden Wörter klein zu schreiben oder Wörter an der falschen Position zu sequenzieren, führt möglicherweise zu dem Verlust von Eigennamen und damit zum Verlust domainspezifischer Informationen.
Um dem entgegenzuwirken kann von einem Domainexperten eine Liste mit Eigennamen angefertigt werden, welche bei der Umwandlung in Kleinbuchstaben oder der Sequenzierung ignoriert werden.

7.2.2 *Anpassungen DM-PV*

Grundlegend lernt die PV-DM-Architektur bisher durch die Möglichkeit, Wortfolgen aus einzelnen oder mehreren Wörtern vorherzusagen. Während des Lernprozesses sammeln sich somit implizite Informationen, über semantische Korrelationen von Wörter im KNN an.
Nachfolgend werden zwei Anpassungen vorgeschlagen, um die Ausrichtung des Lernens auf die spezifische Charakteristik von Produkten besser einzustellen. Mit spezifische Charakteristik sind die versteckten Relationen zwischen den einzelnen Produktattributen gemeint.

Kombination von Wortvektoren mit Attributvektoren

Im ursprünglichen Ansatz kombiniert [32] Wortvektoren aus dem Kontextfenster eines Paragraphen mit einem Vektor, der den gesamten Paragraphen repräsentiert, aus dem die Wörter entstammen. Hierdurch werden Wörter mit ihrem Paragraphen in Relation gesetzt.

Die Idee ist nun, Wörter aus einem Produktattribut mit einem anderem Attribut des Produktes in Verbindung zu bringen. Beispielsweise kann so ein Titelvektor mit den Wortvektoren der Produktbeschreibung zusammengefügt werden.

Vorhersage von Attributvektoren

Die Wortvorhersage aus Wort- und Paragraphkombinationen wird so verändert, dass nun daraus keine Wörter mehr prognostiziert werden, sondern ein anderes Produktattribut. Dieses Produktattribut muss Teil des selben Produkts sein, wie die Wörter, die aus einem anderen Attribut, des gleichen Produktes verwendet wurden. Bspw. wird aus dem Paragraphen und Titelattribut, das Beschreibungsattribut, des selben Produkts vorhergesagt.

Das Verfahren kann jedoch keine vernünftigen Wortrepräsentationen mehr erlernen, da nun keine Beziehungen zwischen den Wörtern betrachtet werden. Daher ist die Nutzungen von vortrainierten Wortvektoren zu empfehlen.

[1] 360pi. *How Many Products Does Amazon Actually Carry?* Letzter Zugriff: 30.01.2017. 2016.

[2] Ethem Alpaydin. *Maschinelles Lernen*. Oldenbourg Wissenschaftsverlag, 2008.

[3] Yoshua Bengio. „Learning Deep Architectures for AI". In: *Found. Trends Mach. Learn.* (2009), S. 1–127.

[4] Yoshua Bengio u. a. „Theano: A CPU and GPU Math Compiler in Python". In: Python for Scientific Computing Conference.

[5] Steven Bird, Ewan Klein und Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 2009.

[6] Nicolas Brunel, Vincent Hakim und Magnus Richardson. „Single neuron dynamics and computation". In: *Current opinion in neurobiology* (2014), S. 149–155.

[7] Nikhil Buduma. *Fundamentals of Deep Learning*. O'Reilly Media, Inc., 2017.

[8] Kenneth Cukier. „Data, Data Everywhere: A Special Report on Managing Information". In: *Economist Newspaper* (2010). URL: `http://www.economist.com/node/15557443`.

[9] Andrew M. Dai, Christopher Olah und Quoc V. Le. „Document Embedding with Paragraph Vectors". In: *CoRR* (2015).

[10] Dask Development Team. *Dask: Library for dynamic task scheduling*. 2016. URL: `http://dask.pydata.org`.

[11] Jérôme Euzenat und Pavel Shvaiko. *Ontology matching*. Springer-Verlag, 2013.

[12] Ludwig Fahrmeir, Alfred Hamerle und Walter Häussler. *Multivariate statistische Verfahren*. De Gruyter, 1996.

[13] Ivan P. Fellegi und Alan B. Sunter. „A Theory for Record Linkage". In: *Journal of the American Statistical Association* (1969), S. 1183–1210.

[14] John R. Firth. „A synopsis of linguistic theory 1930-1955". In: *Studies in Linguistic Analysis* (1957), S. 1–32.

[15] Mario Fischer. *Website Boosting 2.0: Suchmaschinen-Optimierung, Usability, Online-Marketing*. Mitp-Verlag, 2009.

[16] Xavier Glorot, Antoine Bordes und Yoshua Bengio. „Deep sparse rectifier neural networks". In: Journal of Machine Learning Research, 2011.

[17] Ian J. Goodfellow u. a. „TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems“. In: *CoRR* (2016).

[18] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. MIT Press, 2016.

[19] Martin Hilbert und Priscila López. „The World's Technological Capacity to Store, Communicate, and Compute Information“. In: *Science* (2011), S. 60–65.

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

[21] David A. Huffman. „A Method for the Construction of Minimum-Redundancy Codes“. In: *Proceedings of the Institute of Radio Engineers* (1952), S. 1098–1101.

[22] Valerie Illingworth. *The Penguin Dictionary of Physics*. Penguin Books, 2011.

[23] Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim und Doheon Lee. „A Taxonomy of Dirty Data“. In: *Data Mining and Knowledge Discovery* (2003), S. 81–99.

[24] Diederik Kingma und Max Welling. „Auto-Encoding Variational Bayes“. In: The International Conference on Learning Representations, 2014.

[25] Lars Kolb. „Effiziente MapReduce-Parallelisierung von Entity Resolution-Workflows“. Diss. Universität Leipzig, 2014.

[26] Lars Kolb, Andreas Thor und Erhard Rahm. „Dedoop: Efficient Deduplication with Hadoop“. In: *PVLDB* (2012), S. 1878–1881.

[27] Hanna Köpcke und Erhard Rahm. „Training selection for tuning entity matching“. In: International Workshop on Quality in Databases und Management of Uncertain Data, 2008, S. 3–12.

[28] Hanna Köpcke, Andreas Thor und Erhard Rahm. „Evaluation of entity resolution approaches on real-world match problems“. In: *PVLDB* (2010), S. 484–493.

[29] Peter Korsten u. a. *IBM Global Chief Marketing Officer Study*. Techn. Ber. IBM, 2011.

[30] Rainer Kuhlen, Wolfgang Semar und Dietmar Strauch. *Grundlagen der praktischen Information und Dokumentation. Handbuch zur Einführung in die Informationswissenschaft und -praxis*. De Gruyter, 2013.

[31] Jey Han Lau und Timothy Baldwin. „An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation“. In: Association for Computational Linguistics, 2016.

[32] Quoc V. Le und Tomas Mikolov. „Distributed Representations of Sentences and Documents“. In: International Conference on Machine Learning, 2014.

[33] Ulf Leser und Felix Naumann. *Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen*. Dpunkt Verlag, 2007.

[34] Christopher D. Manning und Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[35] Germán Kruszewski Marco Baroni, Georgiana Dinu. „Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors". In: Association for Computational Linguistics, 2014, S. 238–247.

[36] Julian McAuley, Rahul Pandey und Jure Leskovec. „Inferring Networks of Substitutable and Complementary Products". In: Association for Computing Machinery, 2015, S. 785–794.

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado und Jeff Dean. „Distributed Representations of Words and Phrases and their Compositionality". In: Neural Information Processing Systems Conference, 2013.

[38] Tomas Mikolov, Kai Chen, Greg Corrado und Jeffrey Dean. „Efficient Estimation of Word Representations in Vector Space". In: International Conference on Learning Representations, 2013.

[39] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[40] Christopher Olah. *Understanding LSTM Networks*. Letzter Zugriff: 30.01.2017. 2015. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`.

[41] Martin F. Porter. „An algorithm for suffix stripping". In: *Program* (1980), S. 130–137.

[42] Erhard Rahm. „The case for holistic data integration". In: East European Conference on Advances in Databases und Information Systems, 2016, S. 11–27.

[43] Sean M. Randall, Anna M. Ferrante, James H. Boyd und James B. Semmens. „The effect of data cleaning on record linkage quality". In: *BMC Medical Informatics and Decision Making* (2013), S. 64.

[44] Nitish Srivastava, Elman Mansimov und Ruslan Salakhutdinov. „Unsupervised Learning of Video Representations using LSTMs". In: Journal of Machine Learning Research, 2015.

[45] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever und Ruslan Salakhutdinov. „Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* (2014), S. 1929–1958.

[46] Alan. M. Turing. „Computing Machinery and Intelligence". In: *Mind* (1950), S. 433–460.

[47] D. Randall Wilson. „Beyond Probabilistic Record Linkage: Using Neural Networks and Complex Features to Improve Genealogical Record Linkage“. In: International Joint Conference on Neural Networks, 2011.

[48] Chuncheng Xiang, Tingsong Jiang, Baobao Chang und Zhifang Sui. „ERSOM: A Structural Ontology Matching Approach Using Automatically Learned Entity Representation“. In: Conference on Empirical Methods in Natural Language Processing, 2015.

[49] Harris S. Zellig. „Distributional Structure“. In: WORD (1954), S. 146–162.

EIGENSTÄNDIGKEITSERKLÄRUNG

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

Leipzig, 06. Februar 2017

Georges Alkhouri