Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Quantifizierung der Ähnlichkeit von Daten mit Hilfe neuronaler Netze

Linus Bantel

Studiengang:	Data Science
Prüfer/in:	Prof. Dr.-Ing. Bernhard Mitschang
Betreuer/in:	Dipl.-Inf. Michael Behringer, Manuel Fritz, M.Sc.
Beginn am:	1. Dezember 2020
Beendet am:	1. Juni 2021

Kurzfassung

In den letzten Jahren stieg die Menge, sowie die Heterogenität an Daten immer weiter an. Dieser Trend ist exponentieller Natur und es ist anzunehmen, dass dieser auch in Zukunft weiter steigen wird. Umso wichtiger ist es, Data-Scientisten und Domänenexperten bei der Analyse der Daten zu unterstützen, da es bei steigender Datenmenge für Analysten immer schwieriger wird, eine Überblick über die Daten zu behalten. Dies ist für aussagekräftige Analysen von fundamentaler Bedeutung.

In dieser Arbeit wird daher SDRank vorgestellt, die Daten anhand ihrer semantischen Ähnlichkeit quantifiziert um so ein Ranking für den Nutzer zu erstellen. Ähnlich zu klassischen Suchmaschinen wie bspw. Google soll dies verhindern, dass Anwender alle – und damit auch für ihre Analysen irrelevanten – Daten sichten müssen. Stattdessen soll SDRank ihren Anwendern ein schnelles Überblicken der vorhandenen Daten erlauben um Analysen aufgrund größerer Datengrundlagen statistisch aussagekräfter zu gestalten sowie wertvolle Zeit und Resourcen in nachfolgende Schritte der Analyse investieren zu können.

Um SDRank auf Effektivität und Effizienz zu prüfen, wurde eine prototypische Implementierung erstellt. Die Evaluation zeigt dabei auf, dass mit SDRank neben aussagekräftigen Rankings auch eine Berechnung in Echtzeit möglich ist.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation

Das Volumen der weltweit gespeicherten Daten stieg laut eines insideBIGDATA Reports [1] in den letzten Jahren exponentiell mit einer ungefähren Verdopplung aller zwei Jahre an. Laut einer Studie der IDC aus dem Jahr 2017 im Auftrag von Seagate wird dies bestätigt und ein Datenvolumen von rund 160 Zetabyte für das Jahr 2025 prognostiziert [2] (siehe Abbildung 1.1)

Viele dieser Daten liegen als Freitexte im Web vor, aber viele Daten werden auch in Datenbanken gehalten, sowohl durch Menschen generiert, wie auch im Zuge des publiker werdenden Internet-der-Dinge durch Maschinen generiert [3]. Durch diese vielen Datenquellen ergibt sich eine stark wachsende Heterogenität der Daten – angefangen vom Freitext über semi-strukturierte Textdaten bis hin zu strukturierten numerischen Daten.

Diese angesprochene steigende Heterogenität der Daten sorgt unter anderem auch für eine immer größere Verbreitung und Beliebtheit von sog. NoSQL-Datenbanken [4], sowie Data-Lakes [5], da diese für das Speichern von semi-strukturierten und unstrukturierten Daten aufgrund der schemalosen Architektur geeigneter sind als nach dem dem starren Relationenmodell folgenden SQL-Standard. Auch ist die Flexibilität und Skalierbarket von NoSQL-Systemen und Data-Lakes aufgrund des wegfallenden Schemas klassischer relationaler Datenbanken deutlich höher [6][7].

Beim Einsatz schemaloser Datenhaltung in Unterrnehmen besitzen Domänen-Experten häufig nicht das nötige Know-How um selbstständig Analysen auf diesen Daten durchzuführen. Daher werden als „Self-Service Business Intelligence“ („SSBI“ [8]) Methoden bezeichnet, die es einzelnen Abteilungen erlauben, ohne Hinzuziehung eines Data-Scientists oder der IT-Abteilung des Unternehmens

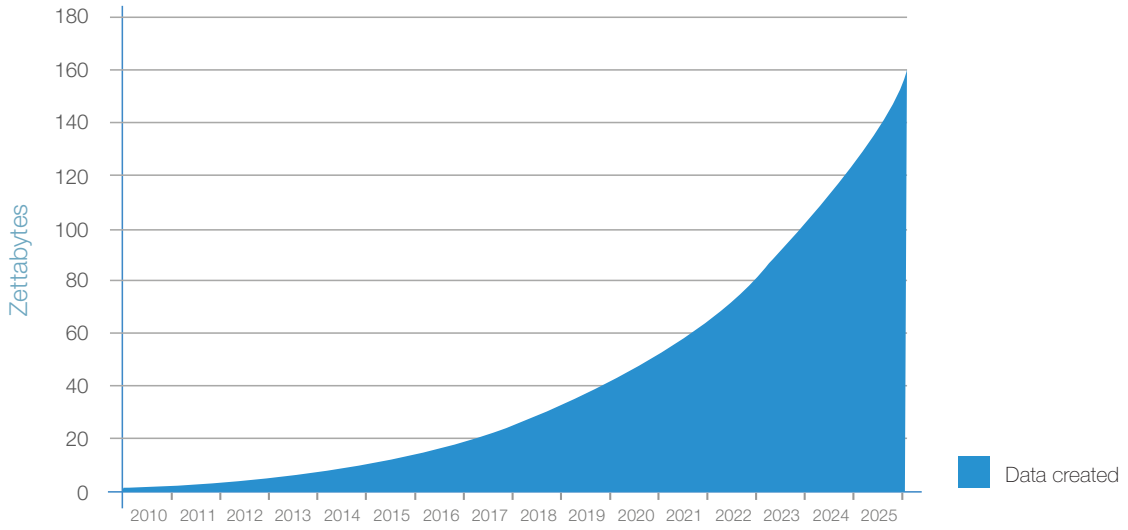


Abbildung 1.1: Wachstum und Verteilung der Daten in Firmen [2]

Analysen auf den Unternehmensdaten durchzuführen . Dies spart den Unternehmen Zeit, Geld und Ressourcen. Klassische Methoden der SSBI sind aufgrund vordefinierter Anwendungsfälle für viele Analysen jedoch ungeeignet, weswegen häufig sog. Data-Mashup-Werkzeuge als flexible Variante verwendet werden [9][10]. Data-Mashup-Werkzeuge bieten dabei häufig eine graphische Benutzeroberfläche, welche die interaktive Analyse der Daten ermöglicht [11]. Mit Data-Mashup-Werkzeugen lässt sich das Know-How-Defizit der Domänen-Experten reduzieren, jedoch nicht das „Know-What“ – das Wissen, welche Daten für die Analyse genutzt werden können. Gerade Domänen-Experten, deren Expertise hauptsächlich in der Domäne und weniger in der Datenbank des Unternehmens liegt, können nicht alle ihnen zur Verfügung stehenden Daten kennen. Wenn nun in einer Analyse die richtigen Daten gefunden werden müssen, kann es schlimmstenfalls dazu kommen, dass der Domänenexperte sämtliche Daten auswählen und sichten muss bis er die für ihn relevanten findet. Bei Datenvolumina wie sie im Bereich des Big-Data-Business üblich sind ist das Überblicken auch für Data-Scientists nicht mehr möglich, sodass auch diesen eine Hilfestellung bei der Auffindung der relevanten Daten geboten werden sollte.

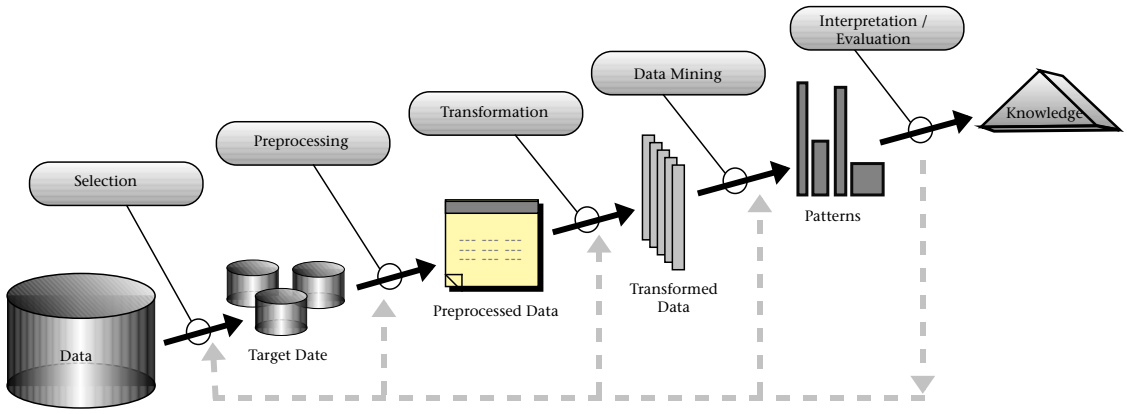


Abbildung 1.2: Knowledge Discovery in Databases [12]

Zusätzlich zu der Datenselektion gibt es in dem in Abbildung 1.2 ersichtlichen KDD-Prozess [12] noch weitere Teilprozesse. Vergleicht man den Zeitaufwand der einzelnen Schritte, fällt die Datenselektion mit rund 20% des gesamten Zeitaufwandes Umfragen zu Folge stark ins Gewicht und ist somit der zweit-zeitintensivste Teilprozess [13]. Ebenso ist nach dieser Umfrage die Datenselektion nach der Datenaufbereitung auf Platz Zwei der unbeliebtesten Aufgaben. Umso wichtiger ist es Domänen-Experten sowie Data-Scientisten bei der Datenselektion bestmöglich zu unterstützen, um die wertvolle Resource Zeit in die anderen Schritte des Prozesses investieren zu können.

Daher wird in dieser Arbeit **SDRank** (**Semantic Data Ranking**) vorgestellt. SDRank soll eine effizientes Erstellen eines Rankings anhand der semantischen Ähnlichkeit von Daten ermöglichen. Ausschlaggebend für das Ranking sollen dabei Daten sein, die der Nutzer schon zu einer Analyse selektiert hat. Durch das Ranking ist es dem Nutzer möglich, schnell auf einen Blick alle für seine Analyse relevanten Daten zu überblicken. Mit SDRank ersparen sich Data-Scientisten das mühselige Selektieren von Daten und Domänen-Experten müssen sich nicht zuerst einen Überblick über die zur Verfügung stehenden Daten verschaffen. Dies erspart im Idealfall einen Schritt des gesamten KDD-Prozesses.

Die Gliederung dieser Arbeit lautet wie folgt: In Kapitel 2 werden die nötigen Grundlagen, die zum Verständnis dieser Arbeit nötig sind, besprochen – insbesondere Ähnlichkeitsmaße und neuronale Netze. Darauf aufbauend wird in Kapitel 3 das Konzept von SDRank vorgestellt – beginnend mit der Gewinnung der Kontextinformation aus den Daten über Konzepte bezüglich des Trainings bis hin zur Auswertung der Vorhersage des Ergebnisses. In Kapitel 4 wird schließlich SDRank evaluiert – bezüglich Effektivität der verschiedenen Netz-Architekturen, Qualität der Ähnlichkeitsmaße bis hin zu Laufzeitmessungen. Zuletzt werden in Kapitel 5 die während des Entstehens der Arbeit gekommenen Verbesserungsmöglichkeiten diskutiert und für weiterführende Arbeiten vorgeschlagen.

2 Grundlagen

In diesem Kapitel werden die zum Verständnis der folgenden Kapitel erklärt. Gegebenenfalls werden zur Vertiefung der Grundlagen Bücher und Publikationen referenziert.

2.1 Information in Daten

Die nach wie vor verbreitetste Technik um Daten zu strukturieren ist die Tabellenstruktur. Eingeführt durch das Codd'sche Relationenmodell [14], welches die Grundlage für den Datenbankstandard SQL ist, werden Tabellen auch in anderen Programmen (Tabellenkalkulation) zur Speicherung von Datenbeständen genutzt. Auch andere Datenbankmodelle (hierarchisch, häufig Basis für NoSQL-Datenbanken) lassen sich letzten Endes wieder in Tabellenform überführen. Mit diesem Hintergrund wird in dieser Arbeit der Begriff der Tabelle stellvertretend für beliebige Speicherstrukturen verwendet.

Tabellen sind in Zeilen und Spalten organisiert. Die Spalten repräsentieren die jeweiligen Eigenschaften der Daten, in den Datenbanken meist verbunden mit einem Datentyp. Es ist von zentraler Wichtigkeit, dass die Reihenfolge der Spalten in der Darstellung der Tabellen ohne Bedeutung ist und sich alle Spaltennamen paarweise unterscheiden müssen. Die Zeilen der Tabellen enthalten die gespeicherten Daten, wobei man jede Zeile als einzelne Instanz betrachten kann. Mehrere Zeilen dürfen (anders als bei den Spalten) den selben Inhalt haben und die Reihenfolge der Zeilen in der Darstellung ist ebenfalls ohne Bedeutung [15]. Typisch für die Tabellendarstellung ist, dass alle Instanzen letzten Endes gleich aufgebaut sind, also die gleichen Eigenschaften haben. Komplexere Datenmodelle aus NoSQL-Datenbanken führen nach einer Transformation zu Tabellen die im Sinne einer Sparse-Matrix dünn besetzt sein können. Am Beispiel von Tabelle 2.1 ist eine solche Instanz z.B. die *Carlynne Sinkins*, mit ihren weiteren Eigenschaften, so z.B. ihre E-Mail-Adresse.

Die vorliegende Arbeit beschäftigt sich mit der Ähnlichkeit von Daten bzw. Datensätzen. Daher ist es wichtig, die Tabellenstrukturen zu kennen. Die trivialste Information, die es über eine Tabelle gibt, ist ihr Name. Eine Tabelle namens „employee“ wird höchstwahrscheinlich eine Form von personenbezogenen Daten über Angestellte einer Firma enthalten und nicht bspw. Klimamesswerte.

id	first_name	last_name	email	gender	...
1	Carlynne	Simkins	csimkins0@utexas.edu	Female	...
2	Clarissa	McAllister	cmcallister1@newsvine.com	Female	...
3	Bord	Beig	bbeig2@census.gov	Female	...

Tabelle 2.1: Beispiel einer Tabelle: Es gibt Spalten, die die Attribute „first_name“„last_name“, etc. speichern und es gibt die Zeilen, die die einzelnen Instanzen speichern

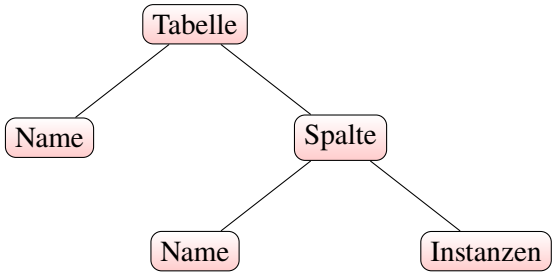


Abbildung 2.1: Tabellen-Struktur entsprechend ihrer Granularität

Allerdings gibt es keine Gewähr, dass der Name Rückschlüsse auf den Tabelleninhalt enthält. Gerade bei durch Konvertierungtools entstandenen Tabellen oder durch den menschlichen Hang zu Abkürzungen / kryptischen Namen können Tabellennamen auch nichtssagend sein (bspw. „table1“). Auch der Spaltenname kann insbesondere bei Vergabe durch Menschen einen semantischen Rückschluss auf Spalten- sowie Tabelleninhalt geben.

Manchmal ist jedoch ein Rückschluss auf die Semantik einer Spalte, auch wenn der Spaltenname nichtssagend ist, aus den gespeicherten Werten möglich. Folgen bspw. alle Einträge einer Spalte dem regulären Ausdruck für eine E-Mail-Adresse, so kann zurecht vermutet werden, dass es sich dabei generell um E-Mail-Adressen handelt. Ausschließlich fünfstellige ganzzahlige Einträge legen die Vermutung nahe, dass es sich dabei um eine Spalte für Postleitzahlen handelt – wenn gleich natürlich keine Sicherheit gegeben ist. Abschließend ist die Granularität einer Tabellenstruktur vom Groben zum Detail im Baum in Abbildung 2.1 dargestellt.

2.2 Ähnlichkeitsmaße

Ähnlichkeitsmaße werden generell benutzt um Objekte verschiedenster Art – ganz gleich ob numerischer, textueller, graphischer,... – vergleichen zu können, bzw. deren Ähnlichkeit angeben zu können. Typischerweise werden Ähnlichkeitsmaße in n-dimensionalen Räumen berechnet, weshalb gegebenenfalls erst eine Vektorisierung der Objekte erfolgen muss. Die einzelnen Eigenschaften können dabei kontinuierlich sein (Gewicht), diskret (Achsenanzahl eines Fahrzeuges) oder als Spezialfall davon auch binär (Vorhandensein einer Automatikschaltung).

Hat man Objekte / Instanzen erfolgreich vektorisiert so liegen sie als Punkte bzw. Vektoren im $\mathbb{R}^n$ vor. Für sie gibt es verschiedene etablierte Ähnlichkeits-/ Distanzmetriken, einige davon sind in exemplarisch in den Formeln 2.1 bis 2.3 dargestellt.

$$\text{Euklid:} \quad d = \sqrt{\sum (a_i - b_i)^2} \tag{2.1}$$

$$\text{Manhattan:} \quad d = \sum |a_i - b_i| \tag{2.2}$$

$$\text{Cosine:} \quad d = \cos\phi = \frac{a * b}{||a|| * ||b||} \tag{2.3}$$

Die Euklid- und Manhattan-Metriken lassen sich als Distanz-Metriken auffassen [16]. Ein Spezialfall ist dabei die Cosinus-Ähnlichkeit die betrachtet werden kann als der Cosinus des Winkels, der sich zwischen den beiden Vektoren vom Ursprung zu den Punkten aufspannt [17].

2.2.1 Wörter

Um Wörter miteinander zu vergleichen existieren sowohl etablierte direkte Vergleichsmöglichkeiten als auch Vektorisierungstechniken.

Trivial ist der direkte Vergleich zweier Wörter mit dem Ergebnis „Gleich“ oder „Ungleich“. Bereits Schreibfehler, Schreibvarianten oder Flexionen führen zum Ergebnis „Ungleich“. Abhilfe können Technologien des Natural Language Processing schaffen (Wortstamm-, oder Grundformbildung) [18]. Weichere und damit besser geeignete Distanzmaße sind die Hamming-Distanz, Levenshtein-Distanz [19] und auch diverse Distanzmaße die auf n-Grammen basieren. Die Hamming-Distanz ist nur für gleich lange Wörter definiert ist, mit der Levenshtein-Distanz können auch Wörter unterschiedlicher Länge gut auf ihre Ähnlichkeit hin getestet werden. Während Hamming- und Levenshtein-Distanz letztenendes eine schreibfehlerbasierte Distanz messen, können mit den n-Gramm Techniken auch ähnliche Wörter oder sogar Wortfolgen entdeckt werden wie z.B. „Bundesgesundheitsminister“ vs. „Bundesminister für Gesundheit“.

Mit den n-Gramm Technologien lassen sich aber nicht nur zwei Wörter direkt vergleichen, sondern auch ideal vektorisieren. Im einfachsten Fall bildet mit $n = 1$ die n-Gramm Abbildung jedes Wort in den $\mathbb{R}^{128}$ ab, jede Dimension gibt dabei die Anzahl des Vorkommens des entsprechenden ASCII-Zeichens an.

Semantische Nähe

Eine andere Technik um die Nähe von Wörtern zu beschreiben ist die Analyse des gemeinsamen (nahe beieinander) Vorkommens in einem Kontext aus vorhandenen Texten. Das Wort „Pferd“ sollte demnach näher an „Reiter“ als an „Informatik“ liegen.

Eine Technik, die im Rahmen der Arbeit verwendet wird, soll hier kurz vorgestellt werden:

Mikolov et al. [20] haben in ihrer Arbeit eine effiziente Technik entwickelt, Wörter unter Berücksichtigung ihrer Semantik in einen hochdimensionalen Vektorraum zu transformieren. Dieser Vektor wird im Fachterminus auch als „Embedding“ oder „Einbettung“ bezeichnet. Die Vektorüberführung ist dabei von solcher Qualität, dass sich die klassische Vektorrechnungen (Addition, Subtraktion, Nächster Nachbar) auf Wörter anwenden lässt. Mit Word2Vec lässt sich (als textuell formuliertes Beispiel) der Differenzvektor von Mann zu König auf das Wort Frau addieren, wobei das ähnlichste Embedding zu diesem Ergebnis, des des Wortes Königin ist.

Word2Vec bietet damit eine ideale Möglichkeit, Wörtern einen Vektor zuzuordnen, der die Semantik abbildet. Wie in Abschnitt 2.3 und Abschnitt 2.1 noch diskutiert wird, ist diese Technik für die vorliegende Arbeit essenziell und wird später auch verwendet werden.

2.3 Neuronale Netze

Neuronale Netze sind eine Form von Berechnungsgraphen [21], die auf Eingaben entsprechende Ausgaben generieren. Diese Ein- und Ausgaben sind Matrizen mit beliebig vielen Dimensionen, auch $n = 1$.

Das Problem der neuronalen Netze ist, dass sie typischerweise als Black-Box verwendet werden, aber hochgradig parametrierbar sind. Die Schwierigkeit bei der Benutzung besteht darin, die Parameter so anzupassen, dass das beste Ergebnis von allen Möglichen generiert wird.

2.3.1 Das einfachste Neuron

Die Grundidee solcher Netze ist, einzelne Neuronen ähnlich wie im menschlichen Gehirn miteinander zu verbinden, um komplexe Probleme lösen zu können. Sie sollen jedoch nicht die biologischen Vorgänge in ihrem Original simulieren, sondern eine mathematische Abstraktion darstellen. Die schematische Darstellung des einfachsten Neurons ist in Abbildung 2.2 zu sehen. Formel 2.4 zeigt die mathematische Funktion zur Berechnung dieses Neurons [22].

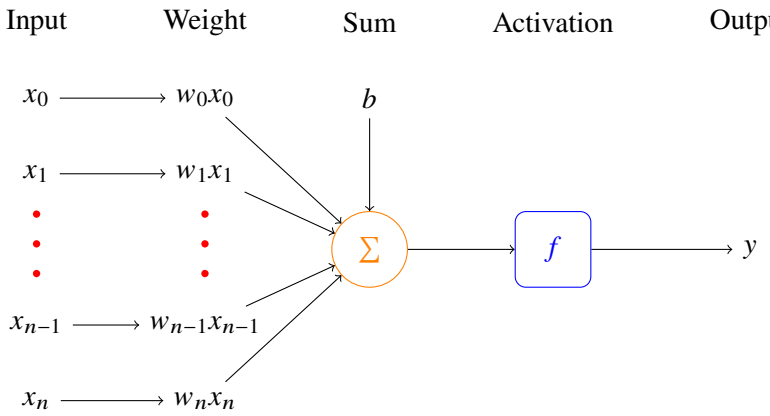


Abbildung 2.2: Schema eines Neurons

$$y = f\left(\sum_{i=0}^{i=n} w_i x_i + b\right) \tag{2.4}$$

Die Aktivierungsfunktion f wird vorrangig dazu benutzt, nicht lineare Probleme besser verarbeiten zu können. In der Praxis werden meist nur eine aus den vier Funktionen Sigmoid, ReLU, Leaky ReLU und Tangens Hyperbolikus verwendet, deren Schaubilder in Abbildung 2.3 dargestellt sind. Beispielsweise bildet man Wahr-Falsch-Entscheidungen auf den Zahlenraum $\{0, 1\}$ ab. In der Praxis sind Entscheidungen jedoch nie zu 100% eindeutig, final muss ein Wert im Intervall $[0, 1]$ ausgegeben werden (Fuzzy-Logik). Hier bietet sich die Sigmoid-Funktion an, die $(-\infty, +\infty) \rightarrow (0, 1)$ abbildet.

In Bibliotheken wie TensorFlow[1] gibt es eine Vielzahl an verschiedenen Neuronen, mit denen eventuelle spezifische Eigenschaften eines Problems besser approximiert werden können. Für SDRank kommt neben dem vorgestellten Neuron auch Filter-Neuronen zum Einsatz.

Ähnlich zu Filtern, wie beispielsweise der Gauß-Filter, der in der Bildbearbeitung genutzt wird, um Bilder zu glätten [23], werden bei sogenannten „Convolutional-Neural-Networks“ ebenfalls Filter eingesetzt. Anders als bei den in der Bildbe- und verarbeitung eingesetzten Filter, haben die

[1] https://www.tensorflow.org/

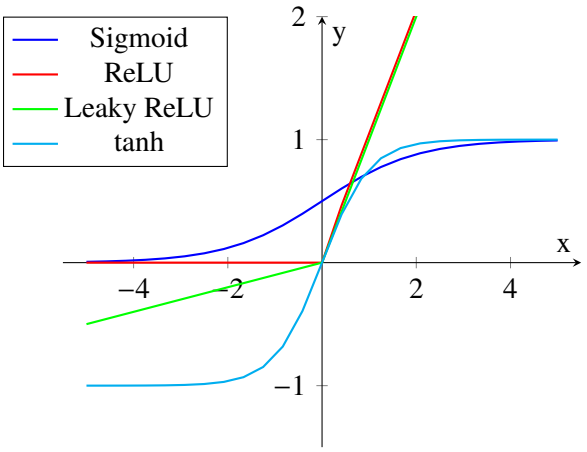


Abbildung 2.3: Verhalten verschiedener beliebter Aktivierungsfunktionen

Filter solcher neuronaler Netze keine vordefinierten, sondern variabel gehaltene Koeffizienten, die beim Trainingsvorgang angepasst werden. Gerade bei neuronalen Netzen, die für Bilderkennung eingesetzt werden, können Filter dabei helfen entscheidende Merkmale in Bildern zu erkennen. Aber auch in anderen Anwendungsbereichen können solche Filter Vorteile bringen.

Ein Filter ist eine n-dimensionale Matrix, deren Koeffizienten komponentenweise mit den zugrundeliegenden Daten multipliziert und schließlich aufsummiert werden. Die Anzahl der Dimensionen des Filters muss identisch zu der der Daten sein. In Abbildung 2.4 ist eine solche Berechnung für 2-dimensionale Daten zu sehen.

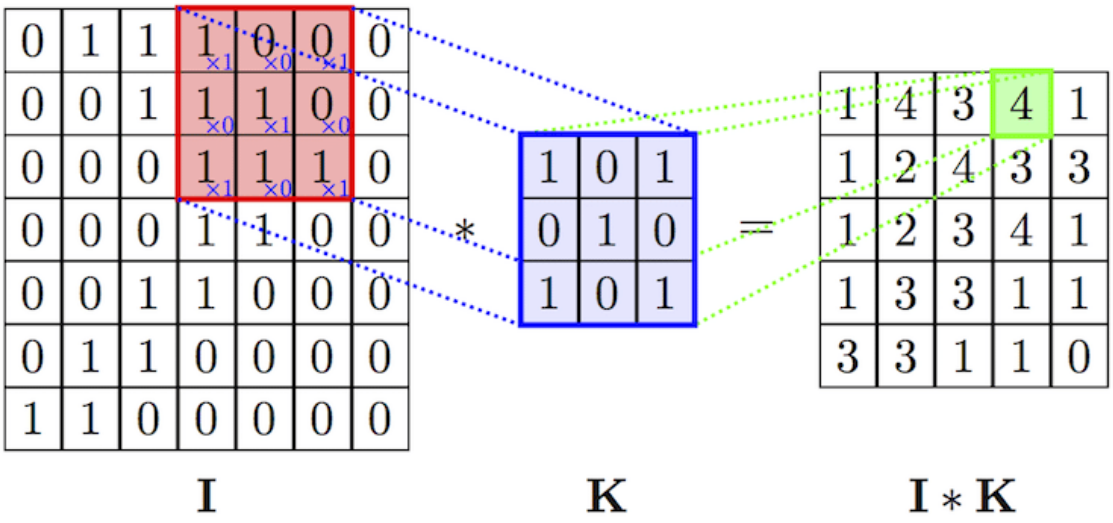


Abbildung 2.4: Berechnung eines Filters auf zwei-dimensionalen Daten [24]

2.3.2 Netze

Um nun ein neuronales Netz zu erhalten, werden viele dieser Neuronen miteinander verbunden. Ausgehend von der Idee, dass Vektoren die Ein- und Ausgabe sind, werden neuronale Netze in sogenannten „Layern“ oder „Ebenen“ organisiert. Dabei sind die Eigenschaften der Neuronen wie bspw. die Wahl der Aktivierungsfunktion innerhalb einer Ebene identisch. Dabei wird in drei verschiedene Ebenenarten unterschieden: Die Eingabeebene, die „versteckten“ Ebenen und die Ausgabeebene. Die Ein- und Ausgabeebene stellen dabei die Schnittstelle des neuronalen Netzes zur Außenwelt dar. Die versteckten Ebenen sind die Ebenen zwischen der Ein- und Ausgabe. In Abbildung 2.5 ist ein einfaches beispielhaftes neuronales Netz mit zwei versteckten Ebenen abgebildet.

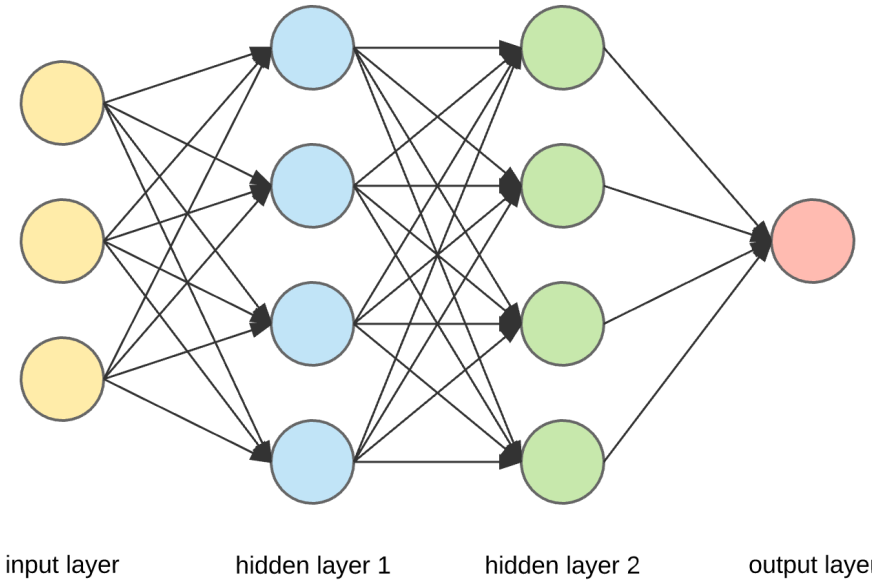


Abbildung 2.5: Aufbau eines neuronalen Netzes mit zwei versteckten Ebenen [25]

2.3.3 Lernvorgang

Neuronale Netze lernen prinzipiell überwacht (supervised-learning [22]). Das bedeutet, dass sehr viele Daten in der Form (x, y) als klassifizierte Menge vorliegen müssen. Diese klassifizierte Menge wird zufällig in eine Trainings- und Testmenge aufgeteilt. Anhand der Daten der Trainingsmenge werden die Gewichte w_i und Konstanten b aus Abschnitt 2.3.1 optimiert, sodass nach dem Lernvorgang für alle Elemente der Trainingsmenge im Idealfall das korrekte Ergebnis generiert wird. Anhand der zur Trainingsmenge disjunkten Testmenge wird das neuronale Netz im Anschluss auf dessen Abstraktionsfähigkeit getestet. Ggf. muss der Data-Scientist das neuronale Netz mit einer anderen Architektur neu aufbauen und Lern- und Validierungsvorgang wiederholen.

In der Praxis treten dabei zwei Probleme häufig auf, die Overfitting und Underfitting genannt werden:

Als Underfitting bezeichnet man das Problem, dass das neuronale Netz nicht in der Lage ist die Trainingsdaten zu lernen. Gründe dafür können eine zu kleine Trainingsmenge im Verhältnis zu der Größe des Netzwerkes sein, sowie eine schlechte Qualität der Trainingsdaten, sowohl auf Seite der Eingabe, als auch der Ausgabe. Underfitting erkennt man daran, wenn selbst auf den Trainigsdaten das neuronale Netz falsche / schlechte Ergebnisse generiert.

Overfitting kann im Gegensatz dazu dann entstehen, wenn die Trainigsdaten zu wenig Rauschen enthalten. Das hat zur Folge, dass das neuronale Netz nicht abstrahieren kann, was bei nicht perfekten Eingaben, wie sie häufig in der Praxis vorkommen, zu schlechten / falschen Ergebnissen führt. Overfitting lässt sich dadurch feststellen, dass auf den Trainingsdaten sehr gute Ergebnisse, aber auf den Test- und Anwendungsdaten sehr schlechte Ergebnisse generiert werden.

Dem Problem des Overfittings wird gerne mit einer Data-Augmentation genannten Technik begegnet: Data-Augmentation, zu Deutsch Daten-Erweiterung, ist eine Technik, Trainigsdaten für neuronale Netze in Bezug auf ihr Volumen zu vergrößern, ohne dabei exakte Kopien der vorhandenen Datensätze zu verwenden. Häufig bei Bilderkennungsaufgaben eingesetzt wird Data-Augmentation auch auf anderen Daten verwendet [26]. Hierzu werden vorhandene Trainingsdaten modifiziert und der Gesamtheit der Trainings- und Testdaten hinzugefügt. Modifikationen für eindimensionale Daten können verschiedene Rausch-Muster wie z.B. gauß'sches Rauschen oder Salz-Pfeffer-Rauschen sein [27]. Häufig wird auch das Auslassen von Daten genutzt, d.h. es werden zufällig zusammenhängende Bereiche der Daten gelöscht bzw. auf Null gesetzt [28].

2.3.4 Zero Padding

Bei den bisher vorgestellten neuronalen Netze müssen die Größe der Ein- und Ausgabeebenen entsprechend der zu verarbeitenden Daten gewählt sein. Damit ist es nicht möglich ein neuronales Netz für eine Tabelle mit zur Entwicklungszeit unbekannter Spaltenanzahl zu entwerfen.

Aus diesem Grund wurden Techniken wie das Zero-Padding entwickelt [29], die sich für diese Arbeit – bei der es um Tabellen beliebiger Größe geht – ideal verwenden lassen.. Zero-Padding beschreibt dabei die Technik, dass Eingabe-Vektoren, die kleiner als die Eingabeebene des neuronalen Netzes sind, mit Nullen aufgefüllt werden. Damit ist durch das neuronale Netz nur noch eine obere Grenze bezüglich der Eingabelänge festgelegt, keine untere mehr.

2.4 Verwandte Arbeiten

Im Rahmen dieser Arbeit wird eine Technik vorgestellt, um anhand des Kontextes von Tabellen ein Ähnlichkeitsmaß zu erzeugen, welches ein Ranking erlaubt um Nutzern von Data-Mashup-Tools die Datenexploration und Datenselektion zu erleichtern. Um einen Vorteile, wie bspw. der geringere Zeitaufwand für gewisse Schritte bei der Arbeit mit Daten und Datenbanken zu erhalten, wurde schon viel Forschungsarbeit in mögliche Techniken investiert. In diesem Abschnitt werden dabei die sowohl für dies Arbeit am relevantesten, als auch ähnlichsten Themen vorgestellt.

2.4.1 Schema Matching

Ein ähnlicher Themenbereich, der sich mit den intertabellarischen Relationen auseinandersetzt ist das Schema Matching.

Eine aktuelle Technik für Machine-Learning gestütztes Schema Matching wird in „LEAPME“[30] vorgestellt. Diese nutzen einen 629-dimensionalen Vektor, kombiniert aus statistischen Feature der Instanzwerte sowie Embeddings, u.a. für den Spaltenname um eine Spalte zu vektorisieren. Das neuronale Netz wertet die Vektoren aus, um zu entscheiden, ob zwei aus unterschiedlichen Tabellen stammenden Spalten einen semantisch identischen Inhalt speichern. Während dieses Paper einen guten Ansatz bezüglich der Vektorrepräsentation von Spalten liefert, ist das Ziel ein anderes.

Schema/Property Matching versucht, aus verschiedenen Tabellen sich entsprechende Spalten zu finden. Konkret bedeutet dies, dass Schema Matching den Ansatz verfolgt, bspw. die Spalte „Postleitzahl“ in mehreren Tabellen einander zuzuordnen, aber nicht herauszufinden, dass die Postleitzahl Teil einer Adresse ist und den Anwender entsprechend auch Straßennamen interessieren könnte.

2.4.2 Ähnlichkeitsmaße

Ein weiteres Themengebiet, mit welchem sich in dieser Arbeit mit auseinandergesetzt wird, ist das Gebiet der Ähnlichkeitsmaße. In diesem Themengebiet wurde schon viel geforscht und es gibt eine Vielzahl an verschiedenen Ähnlichkeitsmaßen. In dieser Arbeit wird z.B. die Cosinus-Ähnlichkeit verwendet um Vektoren miteinander zu vergleichen.

Mit etablierten Verfahren wie der Cosinus-Ähnlichkeit lassen sich zwar Vektoren im Sinne einer Ähnlichkeit vergleichen, für die Zielsetzung der Arbeit ist aber der Vergleich von semantisch zusammenhängenden Vektoren notwendig. Dies erfordert zum einen den Zwischenschritt einer Vektorzusammenfassung als auch die Entwicklung passender Ähnlichkeitsmaße für semantische Ähnlichkeit.

2.4.3 Suchmaschinen

Google hat in „Google Dataset Search“ [31] eine Suchmaschine für Datensätze im gesamten Web vorgestellt. Diese basiert aber auf Metadaten, die den eigentlichen Daten annotiert beiseite gestellt werden müssen. Diese Metadaten liegen in Datenbanken nicht zwingend vor. Daher kann die Google Technik nicht auf das gesetzte Ziel angewendet werden. Statt dessen ist das gesetzte Ziel die Erkennung von semantischen Abhänigkeiten ohne Kenntnis von Metadaten oder Datenschemata. Der Vorteil eines solchen Verfahrens liegt damit darin, dass keinerlei Ansprüche an die zugrunde liegenden Daten gestellt werden.

2.4.4 Kontextgruppen

Ein zentraler Begriff dieser Arbeit ist der der „Kontextgruppe“. Im Zuge der Recherche wurde recht spät die Publikation „Finding Quality in Quantity“ [32] gefunden. Auch in dieser Publikation ist der Begriff „Context-Cluster“ von zentraler Bedeutung. Der Unterschied zur vorliegenden Arbeit ist, dass in [32] Kontextgruppen nicht zur Abbildung semantischer Ähnlichkeit genutzt werden, sondern ähnlich einer Google-Suche Tabellen mit den zu suchenden Informationen gefunden werden sollen. Mit SDRank soll stattdessen erkannt werden, welche Spalten einer Tabelle typischerweise zu einer Kontextgruppe gehören.

3 SDRank – Semantic Data Ranking

Dieses Kapitel stellt das im Rahmen dieser Bachelorarbeit entwickelte Konzept SDRank vor, mit dem anhand von Daten ein Ranking deren bzgl. der Semantik erstellt werden kann. Die Daten liegen dabei in (semi-)strukturierten Formaten vor.

Zuerst wird dabei die Gewinnung von Kontextinformation aus Daten besprochen. Anschließend werden verschiedene Konzepte bezüglich der Ein- und Ausgabe des neuronalen Netzes und der damit verbundenen Generierung der Trainingsdaten erläutert. Nachdem die Vektorrepräsentation der Daten behandelt wird, werden zwei verschiedene für diese Arbeit grundlegende Architekturen neuronaler Netze besprochen. Schließlich wird auf die Auswertung der Vorhersage des neuronalen Netzes eingegangen.

In Abbildung 3.1 ist der gesamte Prozess von SDRank dargestellt. Dieser ist in zwei Teile einzuteilen. Der Trainingsbereich (gelb) beinhaltet den Prozess der sich mit dem Training des neuronalen Netzes befasst. Dazu zählt das Vektorisieren, die Generierung der Trainingstupel und das Training des Netzes selber. Einen Teil der Tupel werden für das Testen während des Trainingprozesses genutzt, um Probleme wie das Overfitting frühzeitig zu erkennen. Der Vorhersagebereich (blau) bildet den Prozess des Berechnen des Ähnlichkeitswertes ab, nachdem das neuronale Netz trainiert wurde. Beginnend mit der Selektion, anhand derer das semantische Ranking erstellt werden soll, wird diese Selektion vektorisiert, der Eingabevektor, der mehrere Spalten enthalten kann, wird erzeugt. Die darauf folgende Vorhersage des neuronalen Netzes wird genutzt, um sie mit den Spalten sämtlicher Tabellen aus den/der zugrundeliegenden Daten(-bank) zu vergleichen. Final kann dann ein Wert berechnet werden, der das Sortieren bezüglich der semantischen Relevanz erlaubt.

Im Verlauf des Kapitels wird mehrfach die Spaltenmenge $\mathbb{S}$ verwendet. Sie soll zur Abstrahierung des Konzepts der Spalte dienen, ohne dabei eine konkrete Darstellungsform wählen zu müssen. Damit ist es u.a. möglich Funktionen, die das neuronale Netz lernen soll, zu definieren, ohne dabei eine konkrete Vektorisierung nutzen zu müssen.

3.1 Gewinnung von Kontextinformation

Wie in Abschnitt 2.3.3 beschrieben, ist sowohl die Eingabe, als auch die Ausgabe eines neuronalen Netzes eine Folge von Werten, die man als Vektor interpretieren kann. Hier soll nun erklärt werden, wie man aus Tabellen zum Eingabevektor des neuronalen Netzes, sowie zum Label für das überwachte Lernen kommt. Bei normaler Textanalyse wird als Kontext typischerweise ein Bereich von wenigen Wörtern zur linken und zur rechten Seite eines zu analysierenden Wortes definiert [20]. Wie unter Abschnitt 2.1 diskutiert wurde, ist aber bei Tabellen ein wichtiges Prinzip, dass die Reihenfolge der Spalten keine Bedeutung hat. Dies widerspricht erst einmal der Idee des Kontextes, es macht keinen Sinn in den Nachbarspalten einer Spalte nach kontextrelevanten

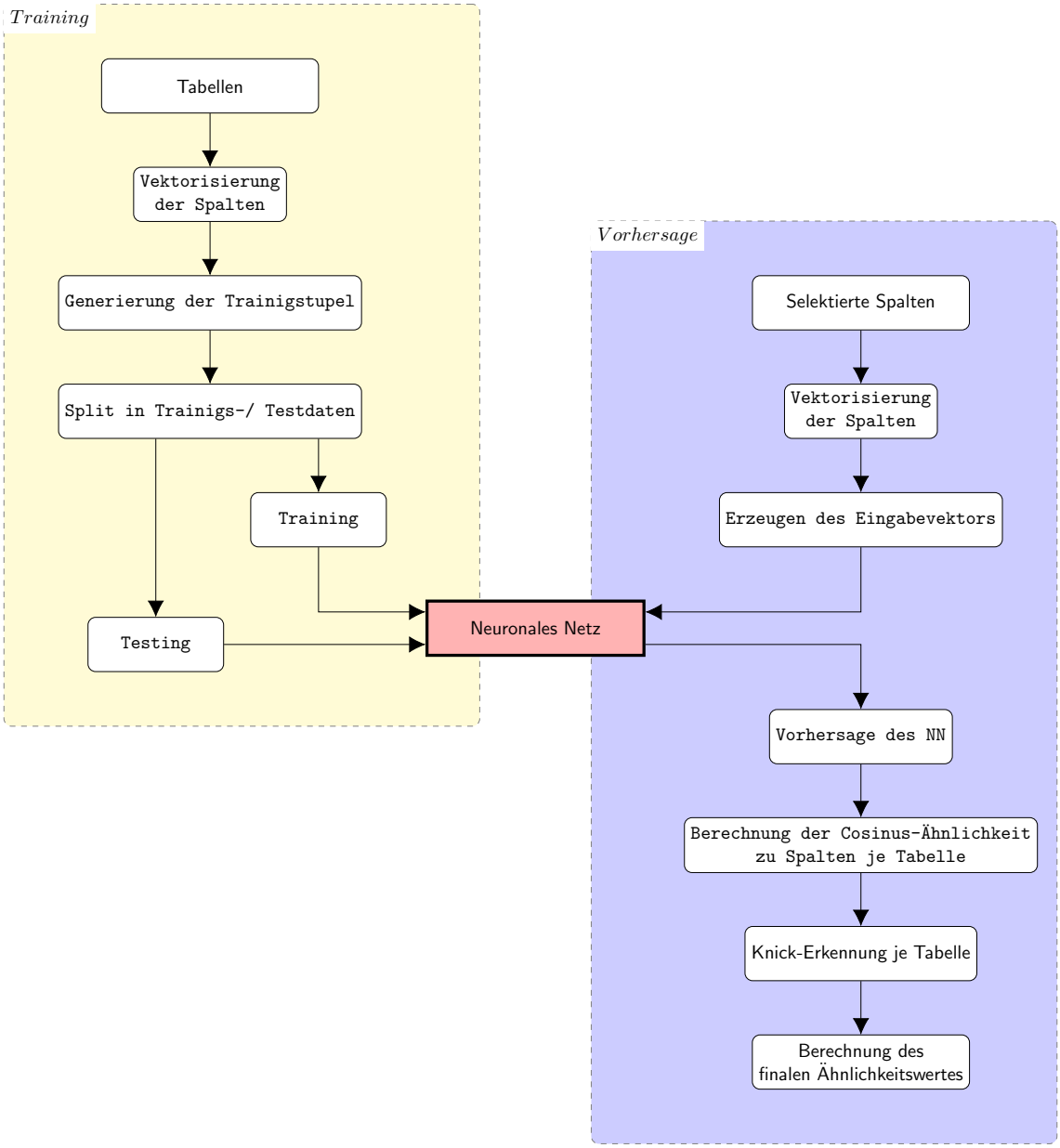


Abbildung 3.1: Schematischer Ablauf von SDRank

Straßenname	Hausnummer	Postleitzahl	Stadt	Bewohner
Universitätsstraße	38	70569	Stuttgart	0
Allmandring	8	70569	Stuttgart	12
⋮	⋮	⋮	⋮	⋮

Tabelle 3.1: Beispielhafte Adresstabelle mit einer „unerwünschten“ Spalte

Informationen zu suchen. Stattdessen muss jede mögliche Kombination der anderen Spalten als mögliche Kontextquelle herangezogen werden. Daraus folgend ist auch eine Information „links“ oder „rechts“ für den Kontext nicht möglich.

Stattdessen ist die zugrunde liegende neue Idee eine a priori beliebige Kombination von Spalten für Ein- und Ausgabe des neuronalen Netzes zu verwenden.

3.1.1 Kontextgruppen

Im vorigen Kapitel wurde erörtert, wie sich die Information über Kontext in Tabellen gewinnen lässt. Die dabei zugrundeliegende Idee ist, dass Spalten die im Kontext zueinander stehen, auch in den selben Tabellen gespeichert werden. Bei Betrachtung vieler Tabellen werden sich dabei immer wieder die selben Gruppen von Spalten finden lassen, wie z.B. die Gruppe Straße, Hausnummer, Stadt, Postleitzahl.

Für solche Gruppen wird an dieser Stelle das Konzept der Kontextgruppe eingeführt. Diese stellt eine Gruppe von semantisch zusammengehörigen Spalten dar.

Diese Idee vereinfacht dadurch auch den Versuch die semantische Ähnlichkeit von Daten zu definieren. Mithilfe von Kontextgruppen kann die Ähnlichkeit darüber bestimmt werden, ob und ggf. wie viele Spalten aus der selben Kontextgruppen in verschiedenen Tabellen vorkommen.

3.1.2 Abstrahierfähigkeit

Mit den bisherigen Überlegungen, dass für das Training Spalten einer Tabelle sowohl für Ein- als auch Ausgaben genutzt werden sollen, lässt zunächst auf ein Problem schließen. Angenommen das Training basiert ausschließlich auf Tabelle 3.1. In diesem Fall würde der Algorithmus die Spalte Bewohner als wichtig erachten und in die Kontextgruppe der Adressen aufnehmen. Werden aber viele verschiedene Tabellen – die zwar alle Straßennamen, Hausnummer, Postleitzahl und Stadt, aber nur wenige Bewohner als Spalte haben – für das Training herangezogen, so wird der Algorithmus ähnlich einer Schnittmenge die vier „wichtigen“ Spalten als Kontextgruppe identifizieren. Im Kontext von Statistik wird bei diesem Phänomen auch häufig vom Gesetz der großen Zahlen gesprochen.

3.2 Training

Dieser Abschnitt befasst sich mit den verschiedenen Konzepten bezüglich der Ein- und Ausgabeanzahl an Spalten des neuronalen Netzes. Dabei wird zusätzlich über die Generierung der Trainigsdaten gesprochen.

Wie in Abschnitt 2.3.3 erklärt, müssen geeignete Tupel (x, y) aus den Tabellen erzeugt werden. Um nicht händisch die y-Werte für Trainings- und Testmenge eingeben zu müssen, wurde in Abschnitt 3.1 erörtert, dass sich die nötige Information zum Kontext aus den Tabellen selber gewinnen lässt. Spaltenwerte werden für das Training als Eingabe in das neuronale Netz gegeben, andere (und / oder die Eingabespalten selber) als Ausgabe verwendet.

Mit diesem Konzept für das Training lässt sich die Anzahl der Spalten für Ein- bzw. Ausgabe verändern um gewünschte Eigenschaften bzw. Möglichkeiten der Technik zu erhalten.

3.2.1 Konzepte

Hier werden die verschiedenen Konzepte erläutert, nach denen Trainingsdaten generiert werden können. Dabei ist die Benennung x-to-y, wobei x und y „One“ oder „Many“ annehmen können um auszudrücken, wie viele Spalten als Ein- und Ausgabe dienen sollen

Konzept 1: One-to-One

Das erste und einfachste Konzept ist das neuronale Netz eine Funktion nach Abschnitt 3.2.1 lernen zu lassen. Auf die Eingabe einer Spalte soll eine Spalte vorhergesagt werden.

$$f : \mathbb{S} \rightarrow \mathbb{S} \tag{3.1}$$

Die Trainingstupel lassen sich dabei leicht als kartesisches Produkt der Spaltenmenge $\mathbb{S}$ generieren, welche dann aus $\mathbb{S}^2$ stammen. In Worten ausgedrückt bedeutet dies, dass auf die Eingabe einer einzelnen Spalte jede Spalte in der selben Tabelle einzeln vorhergesagt werden soll. Dies wird für jede Spalte als Eingabe wiederholt. Die Kardinalität des kartesische Produktes einer Menge ist gleich dem Quadrat der Kardinalität der Menge selber. Daher lassen sich für eine Tabelle mit n Spalten n^2 viele Tupel generieren. Vorteil dieser Technik ist die leichte Generierung der Trainings- und Testdaten. Der Nachteil dieses One-to-One Konzeptes ist, dass nur genau eine Spalte als Ein- und Ausgabe verwendet werden kann. Dies stellt sofort die Frage nach der Sinnhaftigkeit dieses Konzepts, da bei einer Spalte auf der Eingangsseite der Kontext nicht immer eindeutig definiert wird.

Angenommen die Eingabe ist eine Spalte welche Zahlen zwischen 1930 und 2020 beinhaltet und deren Name „year“ ist. Nun stellt sich zurecht die Frage, aus welchem Kontext die Spalte stammt. Möglichkeiten währen z.B. das Geburtsjahr einer Person, aber die Zahlen können auch aus einer Datenbank stammen, in der sämtliche Computer bis in die Neuzeit gelistet sind. Der Kontext ist also nicht zwingend anhand einer einzelnen Spalte zu erkennen. Man erkennt an diesem Beispiel dass der Algorithmus wesentlich eindeutigere Ergebnisse generieren kann, wenn er anhand mehrerer Spalten

definiert und trainiert wird. Ein weiterer Nachteil dieses One-to-One Konzepts ist, dass lediglich eine einzelne Spalte ausgegeben wird. Da das neuronale Netz beim Training auf eine Eingabe jede Spalte vorhersagen soll, wird die Ausgabe eine Art Durchschnitt dieser Spalten erzeugen.

Mit dieser Art des Trainings werden schlechtere Ergebnisse zu erwarten sein.

Konzept 2: One-to-Many

Um das Problem des One-to-One Konzepts, nur eine Spalte als Ausgabe erzeugen zu können, wird in diesem Abschnitt der Ansatz diskutiert, auf die Eingabe einer Spalte mehrere Spalten als Ausgabe zu generieren, welche mit der Eingabespalte im Kontext stehen. Das neuronale Netz sucht also eine Funktion nach Abschnitt 3.2.1.

$$f : \mathbb{S} \rightarrow \underbrace{\mathbb{S} \times \mathbb{S} \times \ldots \times \mathbb{S}}_{\text{k-mal}} \tag{3.2}$$

Entsprechend dieser Formel können die Trainingsinstanzen als Tupel der Form $(x, (y_1, y_2, \ldots, y_k))$ gesehen werden.

Nun hat das neuronale Netz die Möglichkeit Kontextgruppen zu erlernen. Eine Kontextgruppe kann wie schon erwähnt die Gruppe der adressrelevanten Spalten umfassen – wie Straße, Hausnummer, Stadt, Postleitzahl und Land. Das neuronale Netz sollte in diesem Fall auf die Eingabe *Straße* die Ausgabe *(Hausnummer, Stadt, Postleitzahl)* erzeugen. Wichtig ist hierbei nun, dass die Reihenfolge der ausgegebenen Spalten beim Training für eine Kontextgruppe stets die selbe sein muss. Würde die Ausgabe einer Kontextgruppe gemischt sein – also keine strikte Reihenfolge aufweisen – wäre dies gleichbedeutend einem mehrfach angewendeten One-to-One Konzept welches bereits als kritisch diskutiert wurde. Gerade die Reihenfolge macht bei One-to-Many den Unterschied zum (mehrfachen) One-to-One.

Um nun eine Reihenfolge in die Trainingsdaten zu bringen gibt es zwei Möglichkeiten. Die erste ist das Labeln von Hand, d.h. die Trainingsdatensätze müssen von Hand durchgegangen werden und die Kontextgruppen nicht nur markiert werden, sondern dies muss zusätzlich auch noch immer in der selben Reihenfolge passieren. Dies würde aber dem Ziel dieser Arbeit, die Informationen automatisch zu generieren widersprechen, weshalb diese Möglichkeit nicht in Frage kommt.

Die zweite Möglichkeit, die Reihenfolge bei der Ausgabe gleich zu halten wäre einen Schema-Matching-Algorithmus einzusetzen. Dieser kann Gruppen aus gleichen Spalten tabellenübergreifend erkennen. Damit hätte der Algorithmus die notwendigen Informationen, um Spalten immer in der gleiche Reihenfolge in die Trainings- und Testtupel zu schreiben. Allerdings sind die bekannten Schema-Matching-Algorithmen lediglich Unterstützungsalgorithmen für menschliches Korrigieren, vollautomatische Algorithmen sind nicht verfügbar. Daher scheidet auch das One-to-Many Konzept als vielversprechender Ansatz aus.

Konzept 3: Many-to-One

Das Many-to-One versucht das zweite Problem, nämlich die potentielle Mehrdeutigkeit des Kontexts durch nur eine selektierte Spalte, aus dem One-to-One Ansatz zu lösen. Für ein Konzept, das kontextähnliche Daten liefern soll, ist es wünschenswert, wenn für die Suche nicht nur eine Spalte

ausschlaggebend für Suchergebnisse ist. Im One-to-Many Konzept wurde erörtert, warum mehrere Ausgabespalten praktisch nicht umsetzbar sind, nun bleibt nur noch die Option offen, auf Basis mehrerer Spalten eine einzelne Spalte vorherzusagen. Das neuronale Netz soll also eine Funktion nach Abschnitt 3.2.1 lernen.

$$f : \underbrace{\mathbb{S} \times \mathbb{S} \times \ldots \times \mathbb{S}}_{\text{k-mal}} \rightarrow \mathbb{S} \tag{3.3}$$

Die Zahl der Eingabespalten (das k aus Abschnitt 3.2.1) sollte im Sinne eines Kontextes relativ klein sein, es dürfte keinen Sinn machen, bei einer 20 spaltigen Tabelle alle 20 Spalten als Kontext einzusetzen. In der praktischen Evaluation des Konzepts wurde daher die maximale Spaltenanzahl $k = 4$ gesetzt. Damit ist für die Architektur des neuronalen Netzes eine Obergrenze an möglichen Spalten gesetzt. Möchte man weniger Spalten als Eingabe nutzen, kann mit der in Abschnitt 2.3 angesprochenen Technik des Zero-Paddings das neuronale Netz ohne weiteres weiter verwendet werden, unter der Bedingung, dass auch in den Trainingsdaten solche Vektoren mit Zero-Padding vorkommen.

Anders als bei der Ausgabe mit mehreren Spalten, muss bei der Eingabe die Reihenfolge irrelevant sein. Wäre dies nicht gegeben, müsste der Nutzer Wissen über den Trainingsvorgang besitzen, um dann bei der Datenselektion die Spalten in der richtigen Reihenfolge auswählen zu können. Dies muss bei der konkreten Realisierung beachtet werden.

Da alle Eigenschaften diesbezüglich auch in den Trainingsdaten repräsentiert sein müssen, müssen sämtliche Permutationen der Länge $1 \leq i \leq k$ als Eingabe dienen und für jede dieser Permutationen muss jede Spalte als Ausgabe dienen. Aus einer Tabelle mit n Spalten bekommt man mit dieser Methode nach Abschnitt 3.2.1 viele Trainingsinstanzen.

$$l = \sum_{i=1}^{k} \binom{n}{i} * n \tag{3.4}$$

Auswahl des Konzepts

Das One-to-One und das One-to-Many-Konzept scheitern wie diskutiert an der Eindeutigkeit sowie am Fehlen eines perfekten Schema-Matching Algorithmus.

Lediglich Konzept 3 könnte einen Praxistest bestehen. Des Weiteren ist das Many-to-One-Konzept eine Verallgemeinerung des One-to-One-Konzepts und kann damit den unwahrscheinlichen Fall eines trivialen Kontexts ebenfalls erkennen. Problematisch dürfte die Komplexität des Trainings sein, da die Daten genau genommen aus dem Spaltenvektorraum $\mathbb{S}^4$ sind und dies je nach Wahl der Vektorisierung zu komplex sein könnte.

Ein großer Pluspunkt, der für Konzept 3 spricht, ist, dass es beim Einsatz in Mashup-Werkzeugen dem Anwender zusätzlich ermöglicht, Spalten selbst zu selektieren, um im Fall einer großen Tabelle selbst den Kontext des semantischen Rankings bestimmen zu können.

3.3 Vektorrepräsentation

Mit den Vorüberlegungen aus Abschnitt 3.2 wird eine Funktion $\mathbb{S} \rightarrow \mathbb{R}^n$ gesucht, die einer Spalte einen n-dimensionalen Vektor zuordnet wie in Abschnitt 2.3 und Abschnitt 2.2 beschrieben. Dieser Vektor sollte für einen semantischen Typ von Spalte idealerweise eindeutig sein.

In Abschnitt 2.1 wurde über den Informationsgehalt einer Tabelle bzw. Spalte geschrieben. Diese Informationen sind nur in Teilen numerischer Natur. Es benötigt zusätzlich eine Methode, um die Wörter / Strings, die sowohl im Spaltenname enthalten sind, als auch in Instanzwerten enthalten sein können, in eine Vektorrepräsentation zu überführen.

Unter Abschnitt 2.2.1 wurde bereits Word2Vec vorgestellt um semantische Embeddings für Wörter zu erzeugen. Für Spaltennamen bzw. Inhalte die nicht Teil der natürlichen Sprache sind (numerische Werte, Mac-Adressen, Aufzählungswerte, etc.) ist Word2Vec nicht geeignet, da hier weder semantische Beziehungen möglich sind, noch diese nicht in den Traingsdaten von Word2Vec vorkommen können. Für diese Fälle ist die Erstellung eines semantischen Embeddings prinzipiell und damit auch mit Word2Vec nicht möglich. Daher musste für derartige Datentypen ein eigenes Vektorisierungsschema gefunden und implementiert werden.

Die einfachste Idee ist das Bag-of-Characters- / 1-Gramm Modell wie unter Abschnitt 2.2 schon diskutiert.

Ein anderer Ansatz ist wie in „LEAPME: Learning-based Property Matching with Embeddings“ durch Ayalaa et al. in [30] beschrieben: Die Autoren stellen einen Machine-Learning-Property-Matching-Algorithmus vor. Ihre Methode nutzt neuronale Netze um zu entscheiden, ob zwei Spalten unterschiedlicher Tabellen den selben semantischen Typ enthalten (E-Mail-Adressen, Straßennamen, etc.). Das Verwenden eines neuronalen Netzes zwang die Autoren eine Vektorrepräsentation für Spalten zu entwickeln. Sie erzielten – wie der Publikation zu entnehmen ist – mit ihrer Methode gute Ergebnisse. Dabei nutzten sie die Kombination aus mit Word2Vec generierten Embeddings und statistischen Werten um Instanzwerte die aus oben beschriebenen Gründen essenziell sind.

Um einen Freiheitsgrad für die Entwicklung von SDRank zunächst zu eliminieren, wurde die Vektorrepräsentation aus LEAPME als Grundlage in diese Arbeit übernommen. Diese Vektordarstellung ist in Tabelle 3.2 dargestellt.

3.4 Neuronales Netz

In diesem Abschnitt werden die Architektur und Parameter des für SDRank entwickelten neuronalen Netzes besprochen. Zuerst müssen Ein- und Ausgabelänge des Netzes entsprechend den Anforderungen / Daten festgelegt werden. Bei einer Spaltenvektorrepräsentation der Länge n wird ein Ausgabevektor der Länge n und Eingabevektor der Länge $k * n$ benötigt (in Abschnitt 3.2.1 wurde k als die Anzahl an Spalten für die Eingabe auf 4 gesetzt). In Abschnitt 3.3 wurde die Vektorrepräsentation diskutiert und wie in Tabelle 3.2 ersichtlich, ist die Länge der Repräsentation 629. Damit ist die Eingabelänge zunächst festgesetzt.

Wie in Abschnitt 2.3 beschrieben, besitzen alle Ebenen in neuronalen Netzen Aktivierungsfunktionen, so auch die letzte, die die Ausgabeebene ist. Um die y-Werte des neuronalen Netzes an den gewünschten Zahlenbereich anzupassen, kann hier eine entsprechende Aktivierungsfunktion gewählt

Anzahl an Werten	Beschreibung
18	Durchschnitt aller Instanzen: Verhältnis und absolute Anzahl von Buchstaben, Großbuchstaben, Kleinbuchstaben, Punktierungen, Zeichen, Symbole, Trennzeichen, Ziffern und andere
10	Durchschnitt aller Instanzen: Verhältnis und absolute Anzahl von Wörtern, beginnend mit Großbuchstaben, beginnend mit Kleinbuchstaben, Großbuchstabenwörter, numerische Strings
1	Durchschnitt des numerischen Wertes, -1 falls kein numerischer Wert
300	durchschnittlicher Embeddingvektor aller Instanzen, bei Wörter einer einzelnen Instanz wird ebenfalls der Durchschnitt gebildet
300	Embeddingvektor des Spaltenname

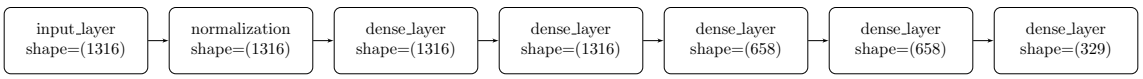
Tabelle 3.2: Vektorrepräsentation einer Spalte in LEAPME

werden. Wie unter Abschnitt 3.3 diskutiert, müssen zwei verschiedene Vektorisierungstechniken gewählt werden, je nachdem ob es sich um Wörter mit einer semantischen Bedeutung oder andere Daten handelt. Der Zahlenraum ergibt sich aus den Eigenschaften der Vektorrepräsentation in Tabelle 3.2 und ist damit der reelle Zahlenraum $\mathbb{R}$. Daher kommen wie in Abbildung 2.3 auf Seite 19 ersichtlich Sigmoid, Tangens Hyperbolicus und ReLU nicht in Frage. Da nicht angenommen werden kann, dass in Tabellen hauptsächlich positive numerische Werte vorkommen, kommt auch nicht die Leaky ReLU in Frage, da diese negative Werte verzerren würde. Die einfachste Funktion, die den Zahlenraum $\mathbb{R}$ abdeckt, ist die identische Abbildung wie in Gleichung (3.5).

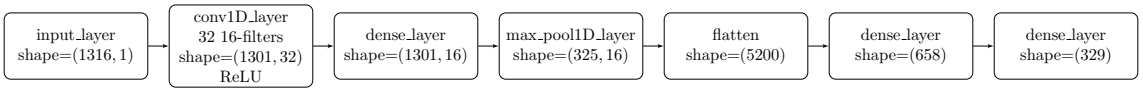
$$f(x) = x \tag{3.5}$$

Für die grundlegende Architekturen können wie in Abschnitt 2.3 beschrieben sowohl klassische neuronale Netze als auch Filter-Netzwerke für SDRank zum Einsatz kommen.

In Abbildung 3.2 sind die beiden Architekturen aufgezeigt.



(a) Architektur des klassischen neuronalen Netzes



(b) Architektur des Filter-Netzes

Abbildung 3.2: Architekturen beider getesteter neuronaler Netze

3.5 Auswertung der Vorhersage des neuronalen Netzes

Mit dem Ergebnis des neuronalen Netzes – das sich als hypotetische Spalte in Vektorrepräsentation auffassen lässt – lässt sich nun die Cosinus-Ähnlichkeit zu jeder beliebigen Spalte jeder beliebigen Tabelle der mit SDRank verarbeiteten Datenbanken berechnen. Für ein anschließendes Ranking nach den semantisch ähnlichsten Tabellen muss nun aber letztendlich ein skalarer Wert pro Tabelle als Rankingwert vorliegen, der aus dem Ergebnisvektor berechnet werden muss.

Da der Vergleich über die Cosinus-Ähnlichkeit der Vektorrepräsentationen spaltenweise erfolgt, jedoch ein einzelner Wert pro Tabelle für das anschließende Ranking gefordert ist, muss eine Möglichkeit gefunden werden, wie mehrere Ähnlichkeitswerte für eine Tabelle in einen einzelnen Wert zusammengeführt werden.

Die einfachste Möglichkeit, einen einzelnen Wert für eine Tabelle zu berechnen ist das arithmetische Mittel der Cosinus-Ähnlichkeit ihrer Spalten zu der Vorhersage des neuronalen Netzes oder andere einfache Maßzahlen wie geometrisches Mittel, Median, etc. Der Nachteil dieser einfachen Maßzahlen ist die Verzerrung des Ergebnisses für lange Tabellen, bzw. Tabellen, bei denen das Verhältnis der Spaltenzahl der Kontextgruppe zur Spaltenzahl der irrelevanten Spalten klein ist. Besser ist eine Kennzahl, die sich nur auf die relevanten Spalten bezieht und die irrelevanten nicht berücksichtigt. Mit Mengenlehre erklärt werden Tabellen gesucht, bei denen die Menge der Spalten einer Kontextgruppe Teilmenge der Tabellenspalten sind.

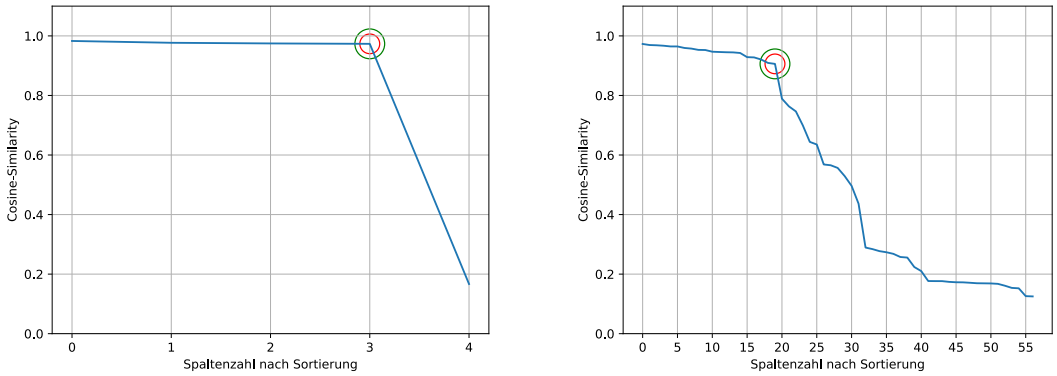
Es ist zu erwarten, dass die Spalten der zu suchenden Kontextgruppen die höchsten Cosinus-Ähnlichkeiten haben. Eine zielführende Möglichkeit die Verzerrung des Ergebnisses durch lange Tabellen zu lösen, ist dass lediglich die top-k Spalten Einfluss auf den Ähnlichkeitswert der Tabelle haben. Dabei sollte das k jedoch nicht konstant gehalten werden, da die Größen der Kontextgruppen a priori nicht bekannt sind. Also muss ein Algorithmus alle top-k Spalten einfließen lassen, wobei das k dabei für jede Tabelle / Kontextgruppe bestimmt werden muss.

In „LOG-Means: efficiently estimating the number of clusters in large datasets“ [33] wird eine Technik vorgestellt, um die optimale Anzahl an Cluster für Clusteralgorithmen zu finden. Diese besteht darin, den stärksten Knick im Elbowgraphen zu finden. Sieht man sich die Graphen in Abbildung 3.3 für die Cosinus-Ähnlichkeit der einzelnen Spalten geordnet nach ihrer Größe an, sind in diesen auch eindeutig Knicke (in der dort diskutierten Ellenbogenform) zu erkennen. Also sollte die in [33] auch auf das hier vorliegende Problem der k-opt. Berechnung anwendbar sein.

Allerdings sind hier auch, vor allem bei langen Tabellen, mehrere Knicke zu erwarten, sodass sprachlich gesagt „der erste deutliche Knick“ gefunden werden muss, ohne auf die genaue formale Beschreibung eingehen zu wollen.

Dazu dient der entwickelte Algorithmus 3.1. Damit ist es möglich, das zuvor angesprochene k für jede Spalte einzeln zu berechnen, um so die Anzahl der relevanten Spalten herauszufinden. Die Eingabe in den Algorithmus ist eine absteigend sortierte Liste von Werten – im Fall von SDRank die Cosinus-Ähnlichkeiten der Spalten. Die Ausgabe des Algorithmus ist die Stelle, an der der Knick in der sortierten Liste vorkommt. Ersichtlich an den grünen Kreisen in Abbildung 3.3 ist, dass die Vorhersage dieses Algorithmus mit der menschlichen Knick-Erkennung in rot übereinstimmt.

Um den finalen Ähnlichkeitswert einer Tabelle zum Suchvektor zu berechnen genügt nun, ein einfaches statistisches Maß wie bspw. das arithmetische Mittel, auf den Cosinus-Ähnlichkeiten der relevanten Spalten anzuwenden In Abschnitt 3.2.1 wurde erläutert, dass das neuronale Netz,



(a) Trainingsdatensatz mit Adressen und vier relevanten Spalten

(b) Airbnb Datensatz mit 57 Spalten

Abbildung 3.3: Cosinus-Ähnlichkeit der Vorhersage des neuronalen Netzes zu den Spalten zweier Tabellen absteigend sortiert. Der gewünschte Knick zur Erkennung der relevanten Spalten ist rot markiert, der von Algorithmus 3.1 berechnete Knick grün.

Algorithmus 3.1 Knick-Erkennung

```
procedure GetRelevantColumns(s[N])
    S ← List
    for i ← 0 to N do
        S.append((i, s[i]))
    end for
    dS ← List
    for i ← 1 to N − 1 do
        dS.append((i, abs((S[i + 1][1] + S[i − 1][1]) − S[i][1])))
    end for
    sort(dS,key x : x[1], descending)
    k ← knick(dS)
    l ← N
    for i ← 0 to k do
        if dS[i][0] < l then
            l ← dS[i][0]
        end if
    end for
    return l
```

wenn es nur einen Spaltenvektor als Ausgabe zu generieren hat, dieser als eine Art Durchschnitt von Spaltenvektoren einer Kontextgruppe aufgefasst werden kann. Es bietet sich daher auch an, statt dem arithmetischen Mittel der Cosinus-Ähnlichkeiten das arithmetische Mittel der relevanten Spaltenvektoren zu berechnen, um dann schließlich die Cosinus-Ähnlichkeit zu berechnen. Abstrahiert wird der Ähnlichkeitswert also statt mit $f(g(x_1), g(x_2) \ldots, g(x_n)$ mit $g(f(x_1, x_2, \ldots, _n))$ berechnet.

4 Evaluation

Da die konkrete Implementierung von SDRank nicht allzu umfangreich ist, der Schwerpunkt der Arbeit lag auf der theoretischen Entwicklung, erfolgt die Beschreibung der Implementierung im Evaluationskapitel gefolgt von der eigentlichen Evaluation von SDRank.

Die Evaluation beginnt mit der Datenbeschaffung, sowohl von Trainings- als auch Evaluationsdaten. Im Anschluss werden Training, die Vektorisierung der Spalten, Architekturen der neuronalen Netze sowie das Vorhersageergebnis evaluiert. Abschließend erfolgt die Evaluation der Laufzeit.

4.1 Daten

Um die Evaluation auf eine breite Basis zu stellen wurde mit mehreren unterschiedlichen Datensätzen evaluiert und dabei der Fokus auf unterschiedliche Kontextgruppen – Personen- und Adressdaten – gelegt. Dies ist für eine aussagekräftige Evaluation nötig, da es bei SDRank gerade darum geht, Daten unterschiedlicher Kontextgruppen gegeneinander zu ranken.

Für die Trainingsdaten wurde – um eine gewisse Datenqualität zu sichern – SDRank auf von Mockaroo[1] generierten synthetischen Daten trainiert. Dafür wurden je Kontextgruppe vier Datensätze erzeugt, die jeweils vier der Kontextgruppe zugehörigen Spalten besitzen. Des Weiteren wurden ein bis zwei kontextgruppenfremde Spalten zusätzlich in die Daten aufgenommen um die Knick-Erkennung (siehe Abschnitt 3.5) evaluieren zu können. Zusätzlich lässt sich damit testen, ob das neuronale Netz in der Lage ist, die den Kontextgruppen zugehörigen Spalten von den nicht zugehörigen zu trennen. Die Anzahl der Instanzen beträgt je Tabelle 100. In Tabelle 4.1 sind Ausschnitte aus Tabellen beider Kontextgruppen zu sehen.

Um die Evaluation aussagekräftig gestalten zu können wurden hierfür Echtwelt-Datensätze, die jeweils einer der beiden Kontextgruppen zuzuordnen sind, ausgewählt. In Tabelle 4.2 sind die Datensätze als Evaluationsgrundlage mit ihrer händischen Zuordung zur korrekten Kontextgruppe aufgelistet. Die Evaluationsdaten sind Echtdaten, welche wahllos durch Recherche im Internet gefunden und heruntergeladen wurden. Die Tabellen enthalten zwischen 10 und 60 Spalten mit jeweils deutlich über 100 Instanzen. Damit ist dies auch ein Proof-of-Concept, ob man mit synthetischen Daten trainieren kann um echte Daten auszuwerten.

[1] https://www.mockaroo.com/

id	street_address	city	country	state	...
1	79892 Dottie Trail	Metz	France	Lorraine	...
2	685 Vermont Way	Périgny	France	Poitou-Charentes	...
3	039 Stuart Center	Nice	France	Provence-Alpes-Côte d'Azur	...

(a) Adressdaten

id	first_name	last_name	email	gender	...
1	Carlynne	Simkins	csimkins0@utexas.edu	Female	...
2	Clarissa	McAllister	cmcallister1@newsvine.com	Female	...
3	Bord	Beig	bbeig2@census.gov	Female	...

(b) Personendaten

Tabelle 4.1: Beispielhafter Auszug aus den Trainingsdaten

Personen	Adressen
shootings	chipotle_stores
voter	open_pubs
	toiletmap

airbnb

Tabelle 4.2: Evaluationsdatensätze und Soll-Zuordnung

4.2 Implementierung

Die Implementierung von SDRank ist vollständig in Python 3.8 geschrieben. Module, die dazu verwendet wurden, sind Pandas 1.1.5[2], Numpy 1.19.5[3], Gensim 3.8.3[4] und Tensorflow 1.14.0 [5].

Die Implementierung wird an Abbildung 3.1 angelehnt diskutiert:

Mithilfe von Pandas werden die Daten, die als CSV vorliegen eingelesen. Darauf folgt die Vektorisierung der Spalten in die Vektordarstellung. Dabei wird über die Instanzen iteriert, um die statistischen Größen für den Vektor zu erfassen (siehe Abschnitt 3.3 Tabelle 3.2). Da je nach Größe der Datensätze diese mehrere Millionen Zeilen haben können führt das Berechnen des Vektors für alle Instanzen einer Spalte zu einem sehr hohen Rechenaufwand. Um Rechenzeit zu sparen, wurde mit einem Auszug aus den Tabellen gearbeitet, selbstverständlich mit einer statistischen Auswahl der Zeilen um für evtl. sortierte Dateien Artefakte zu vermeiden. Um Vergleichbare Laufzeiten zu bekommen, wurden aus allen Datensätzen ein gleichmächtige Auszüge der Größe 100 erstellt. Die Vektorisierung der Wörter erfolgt wie bereits unter Abschnitt 3.3 beschrieben durch mit Word2Vec[20] generierte Embeddings. Das Python-Modul Gensim enthält eine Implementierung für Word2Vec.

[2]https://pandas.pydata.org/
[3]https://numpy.org/
[4]https://pypi.org/project/gensim/
[5]https://www.tensorflow.org/

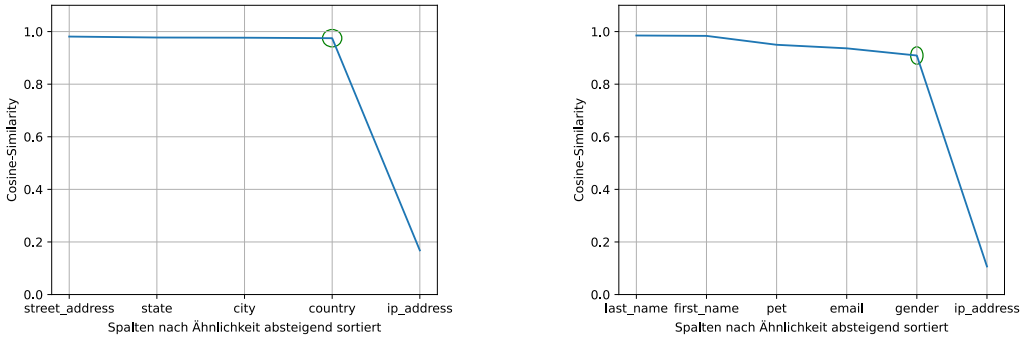
Im nächsten Schritt wurden alle für das Many-to-One-Konzept (siehe Abschnitt 3.2.1) erforderlichen Tupel aus den Trainingsdaten generiert. Mit diesem Schritt ist die Erstellung der Trainingsdaten für das neuronale Netz abgeschlossen.

Die neuronalen Netze wurden wie schon erwähnt auf dem Tensorflow-Modul aufbauend unter Nutzung der Keras-API konstruiert. Ihre Ablaufdiagramme sind in Abbildung 3.2 dargestellt. Als „loss function“ wurde die Cosinus-Ähnlichkeit gewählt, da die Auswertung der Vorhersage des neuronalen Netzes ebenfalls auf der Cosinus-Ähnlichkeit beruht. Als Optimierer wurde der als de-facto-Standard geltende „Adam-Optimizer“ verwendet. Die Asymptote der Loss-Funktion wurde nach etwa 15 Epochen erreicht, weswegen über 20 Epochen trainiert wurde um auf der sicheren Seite zu sein.

4.3 Erkennung der Kontextgruppen

In Abbildung 4.1 sind zwei Graphen zu sehen, die die absteigende Sortierung der Cosinus-Ähnlichkeit der Spalten zur Vorhersage des neuronalen Netzes zeigen. Es ist der typische Ellenbogengraph zu erkennen wie er in Abschnitt 3.5 angesprochen wurde. Dieser Graph kann als Zwischenergebnis für die abschließende Kontextgruppenbestimmung gesehen werden. Dieses Zwischenergebnis ist wiederum die Eingabe für den letzten Schritt, die Kontextgruppenbestimmung.

Der kreisförmig hervorgehobene Punkt ist wiederum das Endergebnis der Knick-Erkennung. Damit können anhand der Abbildung sowohl Zwischen- als auch das Endergebnis diskutiert werden. Angemerkt sei, dass für das absolute Endergebnis noch ein Ranking erfolgen muss, dieses ist jedoch von untergeordneter Komplexität.



(a) Adressdaten **(b)** Personendaten

Abbildung 4.1: Spaltenweise Cosinus-Ähnlichkeit absteigend sortiert mit dem erkannten Knick

4.4 Vektorrepräsentation

In Abschnitt 3.3 wurden bereits die Grundlagen der Vektorrepräsentation diskutiert und dabei auf „LEAPME“ [30] verwiesen. Zu evaluieren ist nun, wie sich die Vektoren idealerweise repräsentieren lassen, sei es als Ein- oder Ausgabe des neuronalen Netzes.

Im Zuge der Implementierung wurde das Problem deutlich, dass in vielen Vektoren der Bereich, der das Embedding der Instanzen hält, mit Nullen gefüllt ist. Dies lässt darauf schließen, dass viele Eingabewörter für das Word2Vec-Modell unbekannt sind. Die Ursache ist wohl darin zu suchen, dass die Inhalte der Tabellen bezüglich der einzelnen Wörter, keine allzu große Deckung zu den Wörten aus den Google-News, auf denen Word2Vec trainiert wurde, besitzen.

Darum muss nun diskutiert werden, ob die Nutzung von Embeddings für Instanzen überhaupt das Ergebnis positiv beeinflussen oder auch ob der positive Einfluss die verlängerte Rechenzeit rechtfertigt.

Tabelle 4.3 zeigt wie sich die Ergebnisse mit und ohne Instanz-Embedding verhalten. Die unterste Zeile der Tabelle – der Score – wird später unter Abschnitt 4.5 erläutert.

	ohne Instanz-embedding		mit Instanz-embedding	
Rank	Adresse	Person	Adresse	Person
1	address1	person3	address3	address1
2	address2	person2	address0	address3
3	address0	person1	address1	person2
4	address3	person0	address2	address2
5	toiletmap	address3	open_pubs	address0
6	chipotle_stores	shootings	voter	person1
7	open_pubs	voter	person1	chipotle_stores
8	shootings	toiletmap	chipotle_stores	shootings
9	person1	address1	person2	open_pubs
10	person0	address2	person0	person3
11	person3	address0	person3	person0
12	person2	open_pubs	shootings	toiletmap
13	voter	chipotle_stores	toiletmap	voter
Score	0	2	23	51

Tabelle 4.3: Vergleich der Vektorrepräsentationen

Es lässt sich leicht erkennen, dass die Qualität des Rankings bei den Spaltenvektoren ohne das Instanz-Embedding de-facto um ein vielfaches besser ist. Die Idee aus Abschnitt 3.3 muss daher im Zuge der Evaluation als nur teilweise zielführend eingestuft werden.

Tabelle 4.4 zeigt nun nochmals die Originaltabelle von Tabelle 3.2 (Seite 32), erweitert um die Erkenntnisse aus der Evaluation, welche Nutzung welcher Option sinnvoll ist. Alle weiteren Analysen / Evaluationen basieren auf diesen Erkenntnissen.

Nutzung	Anzahl an Werten	Beschreibung
✔	18	Durchschnitt aller Instanzen: Verhältnis und absolute Anzahl von Buchstaben, Großbuchstaben, Kleinbuchstaben, Punktierungen, Zeichen, Symbole, Trennzeichen, Ziffern und andere
✔	10	Durchschnitt aller Instanzen: Verhältnis und absolute Anzahl von Wörtern, beginnend mit Großbuchstaben, beginnend mit Kleinbuchstaben, Großbuchstabenwörter, numerische Strings
✔	1	Durchschnitt des numerischen Wertes, -1 falls kein numerischer Wert
✘	300	durchschnittlicher Embeddingvektor aller Instanzen, bei Wörter einer einzelnen Instanz wird ebenfalls der Durchschnitt gebildet
✔	300	Embeddingvektor des Spaltenname

Tabelle 4.4: Anpassung der Vektorrepräsentation

4.5 Evaluation des Rankings

Um Rankings miteinander zu vergleichen, wird eine Möglichkeit benötigt, die Präzision des Rankings numerisch zu beschreiben. Anders als bei Information-Retrieval-Systemen, die nur die relevanten Daten ausgeben sollen, erzeugt die in dieser Arbeit vorgestellte Technik ein kontinuierliches Ranking. Daher lassen sich Maße wie F-Measures, die aus Precision und Recall bestehen, nicht sinnvoll berechnen.

Das Problem für SDRank ist nicht essenziell, für das Ranking ist es aber gut eine einfache Maßzahl zur Verfügung zu haben. Diese Maßzahl, deren Erzeugung im folgenden Vorgestellt wird, war in Tabelle 4.4 bereits als „Score“ enthalten:

Da die Evaluation nur anhand zweier Kontextgruppen stattfindet, kann als Gütemaß die Ranking-Distanz genutzt werden. Dazu wird versuchsweise die Distanz zwischen dem letzten relevanten und den irrelevanten Datensätzen (die vor dem letzten relevanten Datensatz eingeordnet wurden) aufsummiert. In Abbildung 4.2 ist die Idee schematisch dargestellt anhand eines Rankings bezüglich Adressen für das ein Wert von drei berechnet werden würde. Der Wert der Funktion ist gesprochen die Summe aller Distanzen falsch eingeordneter Datensätze.

Diese Maßzahl liefert den Wert Null für ideale Rankings, d.h. alle Datensätze wurden in der richtigen Reihenfolge bezüglich der Kontextgruppen eingeordnet. Je größer der Wert, desto schlechter das Ranking.

Mit Sicherheit ist dieser Ansatz diskussionswürdig, da es aber letzten Endes nur darum geht, eine streng monotone Funktion zum Vergleichen zu bekommen, die konkret numerischen Werte aber irrelevant sind, wird dieses Maß zur Evaluation eingesetzt.

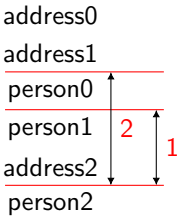


Abbildung 4.2: Evaluation des Rankings

4.6 Neuronale-Netze-Architekturen

Das Ergebnis des Rankings ist natürlich von dem Zwischenergebnis – der Ausgabe des neuronalen Netzes – abhängig. Neben einem aufwendigen Fine-Tuning des neuronalen Netzes – das letzten Endes keine allzu großen Änderungen der Qualität erwarten lässt – kann der prinzipielle Aufbau eines neuronalen Netzes von der grundlegenden Architektur gravierenden Einfluss auf das Gesamtergebnis haben. Daher wurde SDRank wie in Abschnitt 3.4 beschrieben mit zwei verschiedenen Architekturen implementiert und die Ergebnisse evaluiert.

Die erste Architektur ist ein klassisches neuronales Netz, die zweite ein Filter-Netz. In Tabelle 4.5 sind die Ergebnisse des Rankings beider Architekturen im Vergleich dargestellt. Es ist zu erkennen, dass die Qualität des Ergebnisses des Filter-Netzes sowohl bei Adress- als auch bei Personendaten besser ist, bei den Adressdaten ist der Vorsprung geringer, gravierend ist er bei den Personendaten. (Wie in der Evalutation des Rankings diskutiert ist der absolute Wert des Scores nicht entscheidend, man erkennt jedoch Verhältnisse / Distanzen).

Im Zuge der Evaluation kann damit das Filter-Netz als klarer Favorit bezüglich der Architektur empfohlen werden.

	Convolution-Netzwerk		klassische neuronale Netz	
	Adresse	Person	Adresse	Person
Score	0	2	1	30

Tabelle 4.5: Score-Werte der verschiedenen Netz-Architekturen

4.7 Erkennung der kontextrelevanten Spalten

Ein weiterer entscheidender Faktor für ein zuverlässiges Ranking ist der Umgang mit der Vorhersage des neuronalen Netzes beim Berechnen des Ähnlichkeitswertes. In Abschnitt 3.5 wurde der Algorithmus vorgestellt, der es erlaubt, die Anzahl der relevanten Spalten einer Tabelle zu bestimmen. Ein zweiter Ansatz, der dort ebenfalls diskutiert wurde, ist immer die Top-k Spalten für ein festes k einfließen zu lassen. Wie dort erläutert kann bei diesem Ansatz aufgrund der Größe der Kontextgruppen das Probleme auftreten, dass zu viele oder zu wenige relevante Spalten Einfluss auf das Ergebnis bekommen.

Beide Kontextgruppen in den Trainingsdaten haben die Größe 4, daher ergibt es Sinn, $k = 4$ zu testen. Zusätzlich wird auch noch $k = 1$ getestet, um einen Anhaltspunkt zu bekommen, in welchen Größenordnung der Score für die ähnlichste Spalte in einer Tabelle liegt. Das Ergebnis ist in Tabelle 4.6 zu sehen.

	$k = 1$	$k = 4$	k mit Knick-Erkennung
Durchschnittlicher Spalten Score	16,5	26	1

Tabelle 4.6: Evaluation der Knick-Erkennung

Man erkennt dass der auf [33] aufbauende Algorithmus (siehe Algorithmus 3.1) wesentlich bessere Ergebnisse liefert als die einfachere Top-k Methode.

4.8 Evaluation der Mittelwertbildung

In Abschnitt 3.5 wurde bereits diskutiert, dass die zwei Schritte „Bildung der Cosinus-Ähnlichkeit“ und „Mittelwertbildung über die Spalten“ prinzipiell in der Reihenfolge vertauscht werden können. Im Zuge der Evaluation wurden beiden Reihenfolgen implementiert und die Ergebnisse miteinander verglichen. Das Ergebnis zeigt Abbildung 4.3. Die orange dargestellte Kurve zeigt die Ähnlichkeitsmaße der Tabellen, wenn zuerst der Durchschnitt der Vektoren der relevanten Spalten und danach die Cosinus-Ähnlichkeit berechnet wird, die blau dargestellte Kurve entsprechend mit umgekehrter Reihenfolge. Man sieht deutlich, dass in allen Fällen die erste Berechnungsmöglichkeit – also zuerst der Durchschnitt, dann die Cosinus-Ähnlichkeit – deutlich höhere Ähnlichkeits-Kennzahlen ergibt. Dieses Ergebnis deckt sich mit der unter Abschnitt 3.2.1 postulierten Vermutung, dass das neuronale Netz bei einer einzelnen Ausgabespalte einen Art „Durchschnitt“ der Spalten lernt.

Das relative Ranking wurde durch die Wahl einer der beiden Möglichkeiten jedoch nur marginal beeinflusst, ohne Auswirkungen, die sich numerisch im Score bemerkbar machen. Bei der Anwendung von SDRank auf großen heterogenen Datenmengen könnte allerdings ein messbarer Unterschied auftreten.

4.9 Laufzeitanalyse

Zuletzt soll eine Laufzeitanalyse empirisch zeigen, ob SDRank praxisnah eingesetzt werden kann. Dabei wurde gerade wegen der Praxisnähe ist die Laufzeit des Trainingsvorgangs von untergeordneter Bedeutung. Viel wichtiger ist ob der Anwender in subjektiver Echtzeit durch SDRank Vorschläge bei der Datenselektion vorgestellt bekommt.

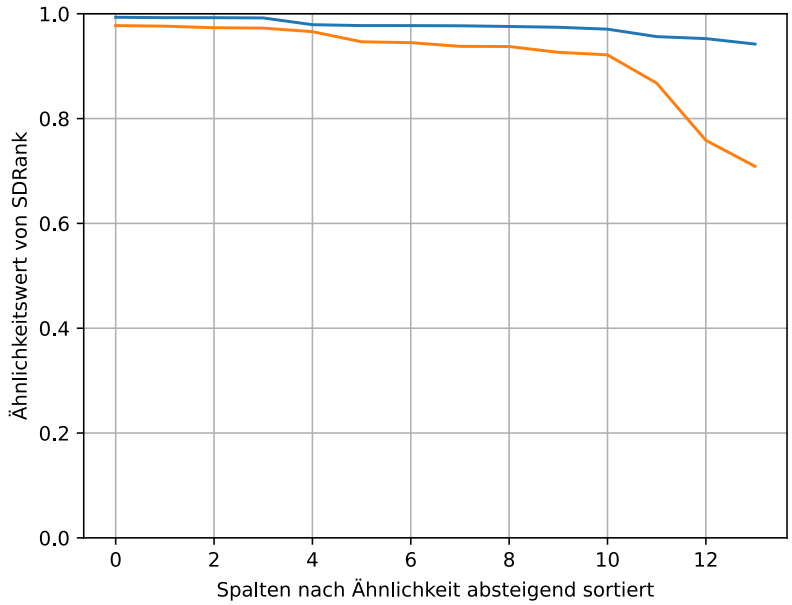


Abbildung 4.3: Ähnlichkeitswerte im Vergleich bezüglich der Reihenfolge von Cosinus-Ähnlichkeit und Mittelwertbildung
(orange: zuerst Mittelwert, dann Cosinus-Ähnlichkeit, blau: zuerst Cosinus-Ähnlichkeit, dann Mittelwert)

Für die Laufzeitanalyse wurde der Algorithmus in sechs Abschnitte unterteilt, deren Laufzeiten in für unterschiedliche Datensätze gemessen wurde :

1. Erzeugung des Eingabevektors
2. Vektorisierung der Tabellen
3. Vorhersage des neuronalen Netzes
4. Berechnung der Cosinus-Ähnlichkeiten
5. Bestimmung der relevanten Spalten (Knick-Erkennung)
6. Berechung der finalen Ähnlichkeit

In Abbildung 4.4 sind diese dargestellt, wobei die logarithmische Skalierung der y-Achse zu beachten ist. Der airbnb-Datensatz fällt bei der Vektorisierung mit ungewöhnlich langen Laufzeit auf, aber alle Teilprozesse sind in der Größenordnung von wenigen Millisekunden. Selbst die Summe aller Teilprozesse ergibt eine Laufzeit von deutlich unter einer Sekunde (mit Ausnahme des airbnb-Datensatzes).

Die lange Laufzeit der Vektorisierung des airbnb-Datensatzes ist darauf zurückzuführen, dass dieser eine große Menge von Fließtext enthält.

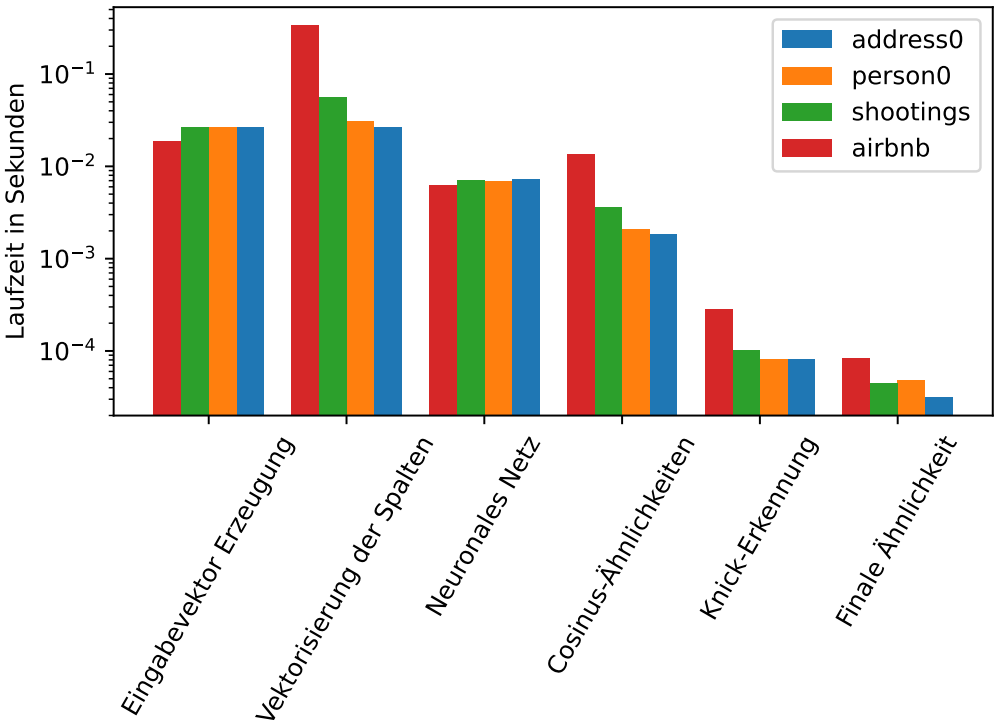


Abbildung 4.4: Laufzeitanalyse für vier verschiedene Datensätze

Als Verbesserungsmöglichkeit bei der Implementierung kann man nun vorschlagen, die Vektorisierung der Spalten – die unabhängig von der konkreten Suchanfrage sind – einmalig in einem Vorberechnungsschritt durchzuführen und zusammen mit dem Datensatz abzuspeichern (die konkrete Umsetzung ist dabei irrelevant). Damit entfällt sowohl der Vektorisierungsvorgang bei der Erzeugung des Eingabevektors, sowie der gesamte Schritt Zwei der Verarbeitung. Mit dieser Verbesserung ist es auch möglich, ohne Laufzeit-Einbußen bei der Anwendung, mehr als nur 100 Instanzen in die Vektorisierung der Spalten einfließen zu lassen (vgl. Abschnitt 4.2). Fallen also die ersten beiden Schritte weg, so ist eine Berechnung des Ähnlichkeitsmaßes für eine Tabelle im Millisekundenbereich möglich – selbst für große Datenmengen.

5 Zusammenfassung und Ausblick

Die Datenmengen in der heutigen Zeit wachsen immer schneller. Dabei steigt nicht nur das Volumen exponentiell an, sondern auch die Heterogenität der Daten. Dadurch wird es für Data-Scientisten und Domänenexperten immer schwieriger einen Gesamtüberblick über alle zur Verfügung stehenden Daten zu haben und in Zukunft auch zu behalten. Umso wichtiger ist es, effiziente Tools für sowohl Data-Scientisten, als auch Domänenexperten zu bieten, die den Nutzer bei der Datenselektion entlasten.

Um diesem Problem entgegenzuwirken, wurde in dieser Arbeit SDRank entwickelt. SDRank bietet dabei Data-Scientisten und Domänenexperten Unterstützung bei der Datenselektion, indem es Daten anhand der semantischen Ähnlichkeit einordnet. Um SDRank auf seine Effektivität und Effizienz zu überprüfen wurde dafür ein Prototyp implementiert.

Zuerst wurden dafür die Grundlagen erarbeitet, welche Möglichkeiten es gibt, beliebige Daten in Vektorrepräsentation zu überführen und wie es möglich ist an Kontextinformationen bezüglich der Daten zu gelangen.

SDRank macht sich dabei die Kontextinformation zunutze, dass Spalten ein und derselben Tabelle im Kontext zueinander stehen. Mit diesem Wissen können Trainingsdaten für ein neuronales Netz generiert werden, welches später für die Vorhersage ähnlicher Spalten genutzt wird. Mit dieser Vorhersage wird schließlich zusammen mit den Spalten anderer Tabellen einen Ähnlichkeitswert für Tabellen berechnet. Dieser ermöglicht ein Ranking, um dem Nutzer die relevantesten Daten zuerst zu präsentieren.

Die Evaluation hat gezeigt, dass mit SDRank zuverlässige Rankings erstellt werden können.

Vorteile dieser Technik sind unter anderem die schnelle online Berechnungsmöglichkeit, sowie das Offline-Training des neuronalen Netzes. Um SDRank zu evaluieren wurden in dieser Arbeit Trainingsdaten generiert. Es spricht auch nichts dagegen, bei großen Data-Warehouses aus den dort gespeicherten Daten die Trainingsdaten zu generieren. Dann steht ein großer Pool an Trainingsdaten zur Verfügung und beinhaltet dementsprechend auch alle Kontextgruppen, die in diesem Warehouse vorkommen.

Ausblick

Das Konzept der Datensuche vollständig in der Daten-Domäne ist neu und somit gibt es in Zukunft viele Möglichkeiten daran weiter zu forschen und darauf aufbauende Konzepte zu entwickeln. Neben einer Vielzahl anderer Architekturen für neuronale Netze ist auch die Technik entscheidend, die die Anzahl relevanter Spalten herausfinden soll.

Weitere Verbesserungen können wie schon in Kapitel 4 angesprochen die Architektur des neuronalen Netzes liefern. Dies muss auf deutlich größeren Datenmengen getestet werden. Des Weiteren ist die Erkennung der relevanten Spalten ein entscheidender Faktor für die Qualität des Rankings. Während die Knick-Erkennung eine gute Grundlage bietet ist auch dort sicherlich noch Raum zur Verbesserung. Auch die Vektorrepräsentation der Spalten lässt sich sicherlich noch anhand der Erkenntnisse aus Abschnitt 4.4 verbessern. Idealerweise so, dass die Instanzen einen größeren Einfluss auf den Vektor bekommen, als der Spaltename – der wie in Abschnitt 2.1 besprochen willkürlicher Natur sein kann.

Ein schlüssiger nächster Schritt nach weiteren Optimierungen ist das Trainieren und Evaluieren von SDRank auf einer großen heterogenen Datenmenge, sowie die Einbindung in ein Data-Mashup-Tool. Darauf kann dann eine Nutzerstudie erfolgen, um SDRank ausgiebig in der Praxis zu testen.

Literaturverzeichnis

[1] *The Exponential Growth of Data*. online. Feb. 2017. URL: https://insidebigdata.com/2017/02/16/the-exponential-growth-of-data/ (zitiert auf S. 11).

[2] D. Reinsel, J. Gantz, J. Rydning. *The Digitization of the World*. online. Nov. 2018. URL: https://www.import.io/wp-content/uploads/2017/04/Seagate-WP-DataAge2025-March-2017.pdf (zitiert auf S. 11).

[3] T. Insights. *Global IoT Market Will Grow to 24.1 Billion Devices in 2030, Generating $1.5 Trillion Annual Revenue*. online. Mai 2020. URL: https://www.prnewswire.com/news-releases/global-iot-market-will-grow-to-24-1-billion-devices-in-2030--generating-1-5-trillion-annual-revenue-301061873.html (zitiert auf S. 11).

[4] D. Correa. *NoSQL Market Estimated to Reach $22.08 Billion by 2026, Says Allied Market Research*. online. Sep. 2020. URL: https://www.globenewswire.com/news-release/2020/09/28/2099954/0/en/NoSQL-Market-Estimated-to-Reach-22-08-Billion-by-2026-Says-Allied-Market-Research.html (zitiert auf S. 11).

[5] L. Wood. *Data Lakes Market Report 2021 - Global Forecast to 2026 - ResearchAndMarkets.com*. online. Feb. 2021. URL: https://www.businesswire.com/news/home/20210210005664/en/Data-Lakes-Market-Report-2021---Global-Forecast-to-2026---ResearchAndMarkets.com (zitiert auf S. 11).

[6] *NoSQL Databases Explained*. online. URL: https://riak.com/resources/nosql-databases/index.html?p=9937.html (zitiert auf S. 11).

[7] *What is a data lake?* online. URL: https://aws.amazon.com/de/big-data/datalakes-and-analytics/what-is-a-data-lake/ (zitiert auf S. 11).

[8] D.-I. (S. Luber, N. Litzel. *Was ist Self-Service BI (Business Intelligence)?* online. Nov. 2017. URL: https://www.bigdata-insider.de/was-ist-self-service-bi-business-intelligence-a-659722/ (zitiert auf S. 11).

[9] M. Behringer., P. Hirmer., B. Mitschang. „Towards Interactive Data Processing and Analytics - Putting the Human in the Center of the Loop“. In: *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 1: ICEIS,* INSTICC. SciTePress, 2017, S. 87–96. ISBN: 978-989-758-249-3. DOI: 10.5220/0006326300870096 (zitiert auf S. 12).

[10] P. Hirmer, B. Mitschang. „FlexMash – Flexible Data Mashups Based on Pattern-Based Model Transformation“. In: *Rapid Mashup Development Tools*. Hrsg. von F. Daniel, C. Pautasso. Cham: Springer International Publishing, 2016, S. 12–30. ISBN: 978-3-319-28727-0 (zitiert auf S. 12).

[11] F. Daniel, M. Matera. „Mashup Components“. In: *Mashups: Concepts, Models and Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 97–136. DOI: 10.1007/978-3-642-55049-2_5. URL: https://doi.org/10.1007/978-3-642-55049-2_5 (zitiert auf S. 12).

[12] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. „From Data Mining to Knowledge Discovery in Databases“. In: *AI Magazine* 17.3 (März 1996), S. 37. DOI: 10.1609/aimag.v17i3.1230. URL: https://ojs.aaai.org/index.php/aimagazine/article/view/1230 (zitiert auf S. 12).

[13] G. Press. *Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says*. online. März 2016. URL: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/ (zitiert auf S. 12).

[14] E. F. Codd. „A Relational Model of Data for Large Shared Data Banks“. In: *Commun. ACM* 26.1 (Jan. 1983), S. 64–69. ISSN: 0001-0782. DOI: 10.1145/357980.358007. URL: https://doi.org/10.1145/357980.358007 (zitiert auf S. 15).

[15] E. Erkec. *An introduction to SQL tables*. online. Juli 2020. URL: https://www.sqlshack.com/an-introduction-to-sql-tables/ (zitiert auf S. 15).

[16] C. C. Aggarwal, A. Hinneburg, D. A. Keim. „On the Surprising Behavior of Distance Metrics in High Dimensional Space“. In: *Database Theory — ICDT 2001*. Hrsg. von J. Van den Bussche, V. Vianu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, S. 420–434. ISBN: 978-3-540-44503-6 (zitiert auf S. 16).

[17] R. Alake. *Understanding Cosine Similarity And Its Application*. online. Sep. 2020. URL: https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a (zitiert auf S. 16).

[18] A. Jivani. „A Comparative Study of Stemming Algorithms“. In: *Int. J. Comp. Tech. Appl.* 2 (Nov. 2011), S. 1930–1938 (zitiert auf S. 17).

[19] M. Mayank. *String similarity — the basic know your algorithms guide!* online. Feb. 2019. URL: https://itnext.io/string-similarity-the-basic-know-your-algorithms-guide-3de3d7346227 (zitiert auf S. 17).

[20] T. Mikolov, K. Chen, G. Corrado, J. Dean. „Efficient Estimation of Word Representations in Vector Space“. In: (2013) (zitiert auf S. 17, 25, 36).

[21] T. E. Bettilyon. *Deep Neural Networks As Computational Graphs*. online. Juni 2018. URL: https://medium.com/tebs-lab/deep-neural-networks-as-computational-graphs-867fcaa56c9 (zitiert auf S. 17).

[22] A. Arnx. *First neural network for beginners explained*. online. Jan. 2019. URL: https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf (zitiert auf S. 18, 20).

[23] *Gaussian Filtering*. online. URL: https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf (zitiert auf S. 18).

[24] I. S. Mohamed. „Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques“. Diss. Sep. 2017. DOI: 10.13140/RG.2.2.30795.69926 (zitiert auf S. 19).

[25] G. Ognjanovski. *Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun*. online. Jan. 2019. URL: https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a (zitiert auf S. 20).

[26] F. J. Moreno-Barea, F. Strazzera, J. M. Jerez, D. Urda, L. Franco. „Forward Noise Adjustment Scheme for Data Augmentation“. In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2018, S. 728–734. DOI: 10.1109/SSCI.2018.8628917 (zitiert auf S. 21).

[27] J. Brownlee. *Train Neural Networks With Noise to Reduce Overfitting*. online. Dez. 2018. URL: https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/ (zitiert auf S. 21).

[28] Z. Zhong, L. Zheng, G. Kang, S. Li, Y. Yang. „Random Erasing Data Augmentation“. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (Apr. 2020), S. 13001–13008. DOI: 10.1609/aaai.v34i07.7000. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7000 (zitiert auf S. 21).

[29] Chris. *What is padding in a neural network?* online. Feb. 2020. URL: https://www.machinecurve.com/index.php/2020/02/07/what-is-padding-in-a-neural-network/ (zitiert auf S. 21).

[30] D. Ayalaa, I. H. Ìandeza, D. Ruiza, E. Rahm. „LEAPME: Learning-based Property Matching with Embeddings“. In: (2020) (zitiert auf S. 22, 31, 38).

[31] N. Noy, M. Burgess, D. Brickley. „Google Dataset Search: Building a search engine for datasets in an open Web ecosystem“. In: *28th Web Conference (WebConf 2019)*. 2019 (zitiert auf S. 22).

[32] T. Rekatsinas, X. Dong, L. Getoor, D. Srivastava. „Finding Quality in Quantity: The Challenge of Discovering Valuable Sources for Integration“. In: *CIDR*. 2015 (zitiert auf S. 23).

[33] M. Fritz, M. Behringer, H. Schwarz. „LOG-Means: Efficiently Estimating the Number of Clusters in Large Datasets“. In: *Proc. VLDB Endow.* 13.12 (Juli 2020), S. 2118–2131. ISSN: 2150-8097. DOI: 10.14778/3407790.3407813. URL: https://doi.org/10.14778/3407790.3407813 (zitiert auf S. 33, 41).

Alle URLs wurden zuletzt am 31. 05. 2021 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift