Hochschule Düsseldorf
Toulouser Allee
40476 Düsseldorf
Landeskrebsregister NRW gGmbH
Gesundheitscampus 10
44801 Bochum

Praxisprojekt

Einstieg in die Thematik und Prototypenbau eines neuronalen Netzes zur Reduktion von manueller Nacharbeit bei der Verknüpfung von Daten zu Patienten

Marvin Sebastian Kirch

Studiengang:	Elektro- Informationstechnik
Prüfer/in:	Prof. Dr. Lux
Betreuer/in:	Dipl.-Inform. Sebastian Bartholomäus Leiter Softwareentwicklung Landeskrebsregister NRW
Beginn am:	29. Juni 2020
Beendet am:	09. Juni 2021

Kurzfassung

Das Landeskrebsregister (LKR) Nordrhein-Westfalen (NRW) erfasst Daten zu Krebserkrankungen von Patienten von Ärzten, Krankenhäusern, Pathologien und Meldeämtern aus ganz NRW um diese auszuwerten. Für die Auswertung müssen Daten, die den gleichen Patienten betreffen, zusammengeführt werden. Zur Verknüpfung gibt es einen automatischen Record-Linkage Algorithmus, der aber bei ca. 5% aller Meldungen eine manuelle Entscheidung erfordert.

Um den Aufwand der manuellen Nachbearbeitung zu reduzieren, soll der Vorgang durch ein künstliches neuronales Netz automatisiert werden. Das künstliches neuronales Netz (KNN) soll mit dem LKR NRW vorliegenden Daten zu manuellen Entscheidungen trainiert werden. Dafür müssen die Daten vor der Verwendung umfangreich aufbereitet werden.

Das Ziel der Arbeit ist die Aufarbeitung und Aktualisierung eines vorliegenden Konzepts und die Implementierung eines Prototypen des Verfahrens. Der Prototyp soll in der Programmiersprache Java umgesetzt werden.

Die folgende Arbeit beginnt mit einer kurzen Einleitung über die Krankheit Krebs, das LKR NRW und KNN. Danach erläutere ich die Methodik, die sich mit und die Implementierung des Themas. In der Methodik erkläre ich was KNN sind und schneide ihre Funktionsweise an. Außerdem erkläre ich wie die Bereitstellung der Daten erfolgt und wie diese für das neuronale Netz verständlich aufbereitet werden. Zur Implementierung werde ich beschreiben was für eine Entwicklungsumgebung verwendet wurde um das Projekt umzusetzen und wie das Programm umgesetzt wurde. Die Erklärung der Umsetzung besteht aus Unified Modeling Language (UML)-Diagrammen für die Datenaufbereitung und Code-Ausschnitte für die Konfiguration und Schnittstellen des KNN.

Das KNN konnte inklusive Datenaufbereitung erstellt werden und klassifiziert 94,4% der Fälle korrekt. Auf Grundlage dieser Arbeit kann das Projekt nachgebaut werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

ANN artificial neuronal network. 19

GB Gigabyte. 29

ICD International Statistical Classification of Diseases and Related Health Problems. 7

KI künstliche Intelligenz. 19

KNN künstliches neuronales Netz. 3

LKR Landeskrebsregister. 3

NRW Nordrhein-Westfalen. 3

PID Person Identification. 23

PM Probability M. 27

POM Project Object Model. 38

PU Probability U. 27

UML Unified Modeling Language. 3

WHO Weltgesundheitsorganisation. 25

1 Einleitung

1.1 Die Krankheit Krebs

Krebs ist eine Krankheit, die die unkontrollierte Vermehrung und das wuchernde Wachstum von Zellen beschreibt. Sie umfasst alle bösartigen Gewebeneubildungen. Kein Organ des menschlichen Körpers kann von einem Befall ausgeschlossen werden, was bedeutet, dass sich Krebs über seine Zellwucherung hinaus auch eine Absiedlung in gesundes Gewebe vornehmen kann. [Wik21b]

Die Zahl der durch Krebs gestorbenen Menschen steigt trotz Forschung kontinuierlich und ist im Moment an einem Punkt, wo weltweit ungefähr jeder sechste Mensch an den Folgen des Krebs stirbt [HRW+20]. Das zeigt, dass noch mehr geforscht werden muss und neue Techniken zur Krebszellenerkennung und Behandlung von hohem Belang sind.

Dies unterstützt die Arbeit des LKR NRW.

1.2 Das Landeskrebsregister NRW

Das LKR NRW ist ein Krebsregister, das vom Land Nordrhein-Westfalen sowohl mit der klinischen wie auch mit der epidemiologischen Krebsregistrierung beauftragt worden ist. Das Landeskrebsregister formuliert abstrakt die eigene Aufgabe auf der eigenen Webseite mit dem Satz: „Ziel der Arbeit des Landeskrebsregisters NRW ist es, einen Beitrag zur Verbesserung der medizinisch-onkologischen Behandlung und Versorgung und damit zur Erhöhung der Überlebenschancen und der Lebensqualität von Krebspatienten zu leisten. “ [Lan]. Genauer definiert sind die Aufgaben des LKR NRW:

I. die kontinuierliche Beobachtung des Krebsgeschehens wie die Häufigkeit von Neuerkrankungen, Sterblichkeit und Prognosen
II. die Dokumentation des Verlaufs von Tumorerkrankungen, einzelner Behandlungsschritte, Nachsorge, Rückfälle, Überleben und Tod
III. die Analyse zeitlicher Verläufe und regionaler Verteilungen
IV. die Unterstützung von Wissenschaft und Forschung
V. die Planung und Bewertung der onkologischen Patientenversorgung
VI. Hilfestellung bei der Entwicklung von erfolgreichen Behandlungsmethoden

[Lan]

Das von mir durchgeführte Projekt fällt vor allem in die beiden zuerst genannten Bereiche.

1.3 Record-Linkage

"Record Linkage bezeichnet die Zusammenführung von Daten über dasselbe Objekt aus verschiedenen Datenbanken. [...] Das Ziel der Zusammenführung besteht also darin, aus mehreren Datensätzen mit Angaben über tatsächlich identische Objekte (meist Personen) einen Datensatz zu erstellen, der alle vorhandenen Informationen über die Schnittmenge der Objekte der Datensätze enthält."[Sch20]

Übertragen auf das LKR NRW wird das Verfahren des Record-Linkages genutzt um zu entscheiden, ob zwei Meldungen zur selben Person gehören. Es wird dazu ein wahrscheinlichkeitsbasiertes Verfahren eingesetzt, bei dem ein Übereinstimmungsgewicht zwischen Meldungspaaren berechnet wird. Einfach gesagt, spricht eine hohe Gewichtung dafür, dass zwei Meldungen zu einer Person gehören und eine niedrige Gewichtung dafür, dass die Meldungen zu zwei unterschiedlichen Personen gehören.

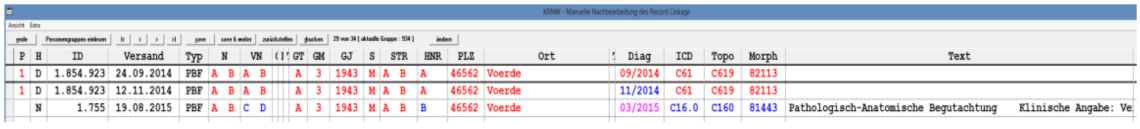
Um eine Entscheidung treffen zu können, braucht man eine Grenze, die entscheidet, ab welchem Grenzwert die Meldungen zu einer Person gehören und welche nicht. Daraus entsteht das Problem, dass die Gewichtungen, die sich nahe um den Grenzwert ansiedeln, Gewichtungen sind, die ein hohes Potenzial zur Falschentscheidung beinhalten.

Um das Risiko der Falschentscheidung zu minimieren, wurde eine Obergrenze und eine Untergrenze eingeführt. Die zwei neuen Grenzwerte ersetzen die einfache Grenze und geben Spielraum für Meldungspaare, die nicht ohne Weiteres zugeordnet werden können. Alle Meldungen, die in diesen Bereich fallen, müssen manuell nachbearbeitet werden. In Abschnitt 1.4 werde ich genauer auf die manuelle Nachbearbeitung eingehen.

1.4 Manuelle Nachbearbeitung

Wie in Abschnitt 1.3 beschrieben, gibt es Meldungspaare, die nicht ausschließlich durch den Record-Linkage-Algorithmus klassifiziert werden können. Diese Meldungspaare werden in die manuelle Nachbearbeitung weitergereicht. Hier werden die Meldungspaare jeweils zuzüglich der eventuell schon in der Vergangenheit verknüpften Meldungen von Mitarbeitern des LKR NRW verglichen. Dazu werden die Meldungsgruppen in einer Tabelle abgebildet, sodass die einzelnen in den Meldungen enthaltenen Informationen miteinander verglichen werden können.

In der Abbildung 1.1 ist ein Beispiel aufgeführt. Das Beispiel zeigt, wie drei Meldungen verglichen werden. Die Zeilen 1 und 2 beinhalten Meldungen, die schon einer Person zugeordnet wurden und nun muss entschieden werden, ob die Meldung aus der dritten Zeile auch dieser Person zugehörig ist.



P	H	ID	Versand	Typ	N	VN	GT	GM	GJ	S	STR	HNR	PLZ	Ort	Diag	ICD	Topo	Morph	Text
1	D	1.854.923	24.09.2014	PBF	A B	A B	A	3	1943	M	A B	A	46562	Voerde	09/2014	C61	C619	82113	
1	D	1.854.923	12.11.2014	PBF	A B	A B	A	3	1943	M	A B	A	46562	Voerde	11/2014	C61	C619	82113	
	N	1.755	19.08.2015	PBF	A B	C D	A	3	1943	M	A B	B	46562	Voerde	03/2015	C16.0	C160	81443	Pathologisch-Anatomische Begutachtung Klinische Angabe: Ve

Abbildung 1.1: Manuelle Nachbearbeitung: Vergleich dreier Meldungen [Sie15]

1.5 Künstliches neuronales Netz

Künstliche neuronale Netze, kurz: KNN (englisch: artificial neuronal network (ANN)), sind ein Teil der immer populärer werdenden künstliche Intelligenz (KI). Die Abbildung 1.2 setzt die Begrifflichkeiten zu diesem Thema in Bezug.

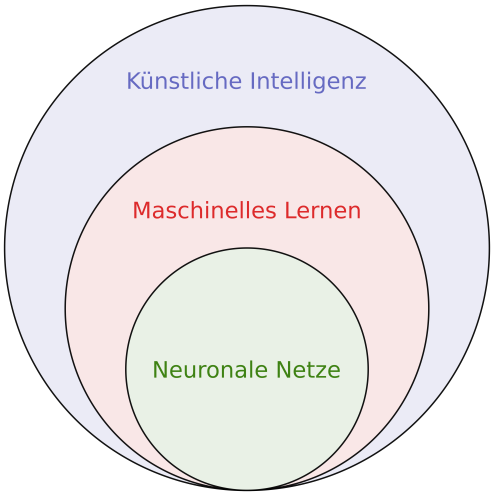


Abbildung 1.2: Begriffseinordnung: Neuronale Netze

Wie der Name impliziert, haben KNN etwas mit Neuronen zu tun, die in einem Netz angeordnet sind. Die Idee dahinter entstammt der biologischen Forschung des Gehirns. Auch hier sind Neuronen durch ein Nervensystem, netzartig, verbunden. Die Leistung des Gehirns ist es hochdifferenziert Sinneswahrnehmungen zu koordinieren und zu verarbeiten. Das Ziel des KNNs ist es diese Fähigkeit nachzuahmen und zu versuchen, diese zu mechanisieren.[Wik21a][Wik21c]

KNN können in ihrem Aufbau unterschiedlich designt sein. Sie differieren in der Anzahl der Neuronen und deren Vernetzung. Dies ist stark abhängig von der Aufgabe und beeinflusst die spätere Erfolgsrate des Netzes. Grundsätzlich wird ein KNN in Schichten aufgeteilt, auf denen die schichtübergreifend verbundenen Neuronen angeordnet sind. Die Schichten werden in drei Schichtbereiche eingeteilt: Eingangsschicht, verdeckten Schichten und Ausgangsschicht. Um von einem KNN sprechen zu können, muss das Modell mindestens eine verdeckte Schicht besitzen, da sonst kein selbstlernender Effekt eintreten kann. Der Anwender interagiert nur mit der Eingangsschicht und der Ausgangsschicht. Die Eingangsschicht nimmt Daten entgegen und die Ausgangsschicht gibt Auskunft über das errechnete Ergebnis.

Ist der Aufbau des KNN festgelegt, muss es trainiert werden. Dazu wird eine Datenbank mit einer Anzahl n Datensätzen (Beispielen) in das KNN eingelesen. Ein Beispiel besteht aus mehreren Input-Parametern, den features (Merkmalen), und einem Output-Parameter, dem label, dass eine Schlussfolgerung auf Grund der feaures beschreibt. Der Trainingsvorgang erfolgt in mehreren Zyklen, wobei zwei verschiedene Arten der Zyklen verschachtelt sind. Der äußere Trainings-Zyklus

wird als Epoche beschrieben, der durch den Durchlauf der gesamten Datenbank definiert ist. Jede Epoche ist wiederum in Unterzyklen, die Batches, unterteilt, die eine definierte Anzahl Beispiele der Datenbank umfassen. Das Training erfolgt nun schrittweise. Ein Batch von Beispielen wird in das KNN geladen und dieses berechnet die Entscheidung. Die berechneten Entscheidungen werden mit den Labels der Beispiele verglichen. Aus der Differenz wird berechnet, ob und wie die Parameter des Modells verändert werden müssen, um im nächsten Durchlauf eine kleinere Differenz zu erhalten. Der Durchlauf wird wiederholt, bis alle Batches einmal geladen wurden. Die Anzahl der Durchläufe kann auf Grund der Batchgröße variieren. Umso größer die Batchgröße ist, desto weniger Durchläufe ergeben sich innerhalb einer Epoche. Um eine noch höhere Genauigkeit des Netzes zu erreichen, kann man die Anzahl der Epochen erhöhen. Die Anzahl der Lernzyklen ergibt sich aus der Multiplikation der Anzahl an Batches innerhalb einer Epoche mit der Anzahl der Epochen.

In jedem Lernzyklus werden, wie schon erwähnt, Parameter des KNN verändert, was als „Lernen“ beschrieben wird. Die Parameteränderungen reichen von der Entwicklung neuer - oder dem Löschen existierender Verbindungen, der Änderung der Gewichtungen, dem Anpassen von Schwellenwerten, dem Hinzufügen oder Löschen von Neuronen bis zur Modifikation der Aktivierungs-, Propagierungs- oder Ausgabefunktion. Nach und nach sollte die Genauigkeit des KNN bei passender Konfiguration zunehmen.

2 Methodik

2.1 Künstliches neuronales Netz

Der Zweck des KNN in diesem Projekt ist es, die Aufgabe der manuellen Nachbearbeitung sukzessiv zu übernehmen. Das bedeutet, dass die Meldungspaare, die nicht durch den Record-Linkage-Algorithmus zweifelsfrei zugeordnet werden konnten, durch das KNN klassifiziert werden sollen. Da es noch nahezu unmöglich ist, eine sichere hundertprozentige Genauigkeit des Netzes zu erreichen, wird die manuelle Nachbearbeitung zusätzlich zum KNN unentbehrlich sein.

Das Netz kann auf unterschiedlichen Wegen eine Hilfestellung sein. Je nach Erfolg des Projektes kann es eine unterschiedlich große Rolle spielen. Zum Beispiel kann man das KNN als Vorselektierung nutzen. Oder die manuelle Nachbearbeitung kann sich über die Vorselektierung hinaus, ausschließlich einzelne kritische Attribute, hervorheben lassen. Ein anderes Beispiel der Nutzung wäre auch bei der Gewichtung der Entscheidung des Netzes eine Grenze festzulegen. Diese Grenze legt fest, ab welcher Gewichtung man sich sicher sein kann, dass das KNN mit der errechneten Klassifizierung richtig liegt. Alle Ergebnisse, die unter der bestimmten Grenze liegen, müssen in die aus Abschnitt 1.4 bekannte manuelle Nachbearbeitung. Der Rest gilt als vollautomatisch klassifiziert.

Als Basis für unser KNN nutzen wir eine sogenannte, für den hier bearbeiteten Anwendungsfall übliche, FeedForward-Architektur mit drei Schichten. Die Eingangsschicht dient zur Eingabe von Daten in das KNN; die Anzahl der Neuronen in dieser Schicht wird durch den Umfang der Eingangsdaten bestimmt. Der Lerneffekt entsteht hauptsächlich durch die Verwendung von Neuronen auf der verdeckten Schicht. Die Ausgabeschicht dient der Ausgabe der Berechnung des Netzes. In diesem Fall besteht die Ausgabeschicht aus zwei Neuronen, eins für die Wahrscheinlichkeit, dass ein Paar von Meldungen zusammen gehört und eins für das Gegenteil.

Abstrakt ist der Aufbau in Abbildung 2.1 visualisiert.

Für das Training und die Anwendung des KNN müssen zunächst die relevanten Daten ausgewählt und in eine für das KNN verständliche Form gebracht werden.

2.2 Daten

Das KNN kann im vorliegenden Fall nur so intelligent sein, wie die Qualität der Daten und deren Präsentation ist. Darum muss man sich bemühen, eine gute Datenbasis zu schaffen. Um eine gute Basis für die vorliegende Anwendung zu schaffen, betrachtet man die Aufgabe, die das KNN später übernehmen soll. Es soll klassifizieren, ob eine Meldung zu einem schon bestehenden Patienten (Meldungsgruppe) zugeordnet wird. In den folgenden drei Abschnitten werde ich näher drauf eingehen, welche Daten genutzt und wie sie aufbereitet werden.

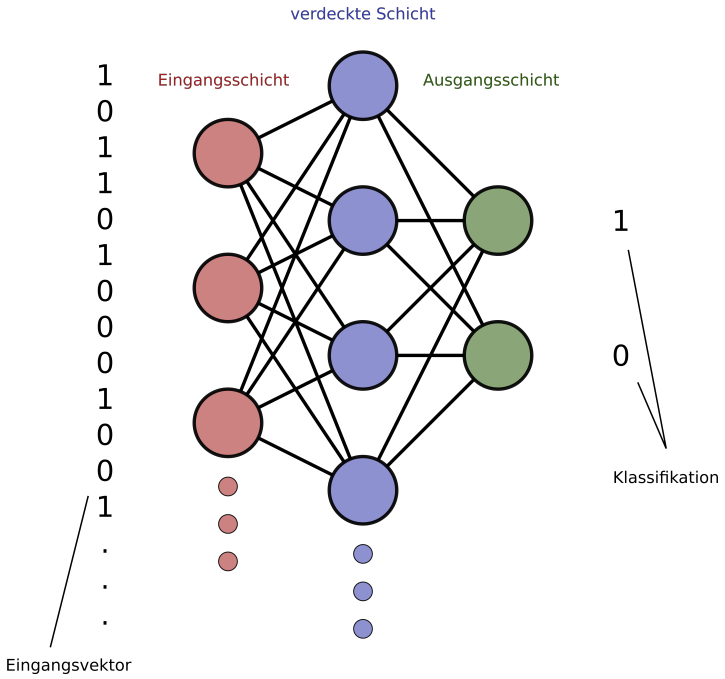


Abbildung 2.1: Abstrakter Aufbau des KNN

2.2.1 Datenauswahl

Das Landeskrebsregister verfügt über eine Datenbank mit Meldungen zu Patienten. Diese enthalten Informationen zu der Person (Stammdaten), der Diagnose und verschiedenen Relationen, um Informationen in bestimmte Beziehungen zu setzen. Für den folgenden Fall sind diese Relationen interessant, die Informationen über die Zuordnung zu Patienten halten. Die in Abschnitt 1.5 beschriebene Aufgabe des KNN ist die Unterstützung der in Abschnitt 1.4 beschriebenen manuellen Nachbearbeitung und somit sollte die Datenbasis auf Grundlage der Daten, die bei der manuellen Nachbearbeitung zur Verfügung stehen, aufbauen. Folgend nutzt man Listen, die eine zu klassifizierende Meldung (neue Meldung) und eine Menge von älteren Meldungen, die den potenziell selben Patienten beschreiben, enthalten.

Die Listen beschreibe ich im Weiteren als Meldungsgruppen. Die Meldungsgruppen werden für das Training des KNN um das Ergebnis der manuellen Nachbearbeitung erweitert. Das Ergebnis kann zwei Zustände annehmen: zugeordnet oder nicht zugeordnet. Die Meldungsgruppen werden dann in zwei Untergruppen unterteilt. Die eine Gruppe umfasst alle „zugeordneten“ Meldungsgruppen, während im Gegensatz dazu die andere Gruppe alle „nicht-zugeordneten“ Meldungsgruppen hält. Die Meldungsgruppen sind allerdings nicht direkt aus den Daten der Datenbanken des LKR NRW zu erkennen, da die Meldungen ja bereits in der Vergangenheit schon verarbeitet und im Zweifel manuell zugeordnet wurden.

Um einen realistischen Trainingsdatensatz von Meldungsgruppen aus der Datenbank aller Meldungen zu extrahieren, wird die Menge aller Meldungen in zwei Teile gesplittet. Dazu wird ein Grenzwert, in Form eines Datums, festgelegt. Nun orientiert man sich an dem Eingangsdatum der einzelnen Meldungen. Ein Teil umfasst alle neuen Meldungen, von denen das Eingangsdatum nach dem Grenzwert liegt. Andersherum liegt bei dem Teil mit den alten Meldungen das Eingangsdatum vor dem Grenzwert.

Bei einem Record-Linkage-Lauf wird nun jede neue Meldung gegen jede alte Meldung vergleichen und die einzelnen Gewichtungen zweier Meldungen betrachtet. Die Meldungspaare, bei denen die Gewichtung, wie in Abschnitt 1.3 beschrieben, über der Zuordnungs-Untergrenze und unter der Zuordnungs-Obergrenze liegt, werden weiter verarbeitet. Die restlichen Meldungspaare sind für die Aufgabe des KNN uninteressant, da sie schon entschieden werden konnten.

Wie die Meldungspaare bei der manuellen Nachbearbeitung eingeordnet wurden, kann man an Hand der Relation zwischen Meldungen erkennen. Die Relation wird durch Setzen des identischen Wertes im Feld Person Identification (PID) gespeichert. Alle Meldungen mit der gleichen PID verweisen auf den gleichen Patienten.

Mit der Information kann man die eine Hälfte der benötigten Daten, die positiven Beispiele, aus der Menge identifizieren. Man iteriert durch alle neuen Meldungen und filtert die, bei denen sich die PID in den alten Meldungen widerspiegelt, heraus. Jede Meldung der konzentrierten Menge an neuen Meldungen fügt man anschließend mit den jeweils alten Meldungen, die die gleiche PID aufweisen, zu einer Meldungsgruppe zusammen. Diese werde im Folgenden als positive Beispiele beschreiben.

Außerdem kann man an diesem Stand bestimmen, welche neuen Meldungen keinem Patienten zugeordnet werden konnten. Das sind alle die neuen Meldungen, die eine neue PID zugewiesen bekommen haben. Sie besitzen eine PID, die nicht in der Menge der alten Meldungen vorkommt. Um herauszufinden, welche potenziellen alten Meldungen, den nicht zugeordneten neuen Meldungen, angeboten wurden, wird ein neuer Record-Linkage-Durchlauf gestartet. Bei diesem Durchlauf werden alle nicht zugeordneten neuen Meldungen mit allen alten Meldungen verglichen. Die daraus resultierenden Meldungspaare werden wieder zu Meldungsgruppen zusammengefügt. Dazu wird jede neue Meldung um die alten Meldungen ergänzt, die im Meldungspaar eine Gewichtung über der Untergrenze und unter der Obergrenze besitzen. Diese Meldungsgruppen sind im Folgenden die negativen Beispiele für das KNN.

2.2.2 Datenaufbereitung

Für das neuronale Netzwerk ist die Ähnlichkeit zweier Meldungen interessant. Um die Ähnlichkeit eines Meldungspaares am besten für das Netz ausdrücken zu können, wird eine Vektorkodierung verwendet. Bei der manuellen Nachbearbeitung gibt es Merkmale der Meldungen, die wichtiger oder weniger wichtig für die Aussage, ob zwei Meldungen zu einem Patienten gehören, sind. Darum filtern wir den Vornamen, den Nachnamen, den Geburtsnamen, den Geburtsmonat, das Geburtsjahr und die Diagnose (ICD-10 Code) heraus und vergleichen diese jeweils. Jeder Vergleich ergibt einen Teilvektor und diese werden dann in Reihe hintereinander zu einem gemeinsamen Vektor von einer Länge mit 72 Dimensionen zusammengefügt.

Beim LKR NRW wird mit vertraulichen Daten gearbeitet, weswegen manche Informationen pseudonymisiert sind und manche nicht. Von den oben genannten gefilterten Informationen sind alle Namen pseudonymisiert und der Geburtsmonat, das Geburtsjahr und der ICD-10 Code im Klartext.

2.2.2.1 Klartextdaten

2.2.2.1.1 Geburtsmonat Der Vergleich zweier Geburtsmonate kann auf den ersten Blick in zwei Zustände klassifiziert werden. Entweder die Monate sind gleich oder nicht. Um eventuelle Fehler der Übertragung oder des Missverstehens mit einzuberechnen, wird eine dritte Möglichkeit hinzugefügt. Diese Variante der Ähnlichkeit tritt ein, wenn sich die zwei Monatsangaben in nur einer Ziffer unterscheiden und eine Differenz gleich eins besitzen. Jede Möglichkeit steht für eine Dimension des Vektors und jede Dimension kann den Zustand einer „0“ oder „1“ annehmen, wobei ausschließlich eine Dimension gleich „1“ sein muss. Damit ergibt sich folgender Vektor:

$$v \in \{0,1\}^3 \text{ mit } \exists! v = 1$$

Die drei Möglichkeiten von Klassifizierungen aufgelistet:

Klassifizierung	Vektor
übereinstimmend	$(1,0,0)$
teils übereinstimmend	$(0,1,0)$
nicht übereinstimmend	$(0,0,1)$

Tabelle 2.1: Dreidimensionaler binärer Vektor für den Geburtsmonat

Beispiele:

Geburtsmonat 1	Geburtsmonat 2	Klassifizierung
07 (Juli)	07 (Juli)	übereinstimmend
11 (November)	12 (Dezember)	teils übereinstimmend
02 (Februar)	11 (November)	nicht übereinstimmend

Tabelle 2.2: Beispiele für die vielleicht resultierende Ähnlichkeit

2.2.2.1.2 Geburtsjahr Ähnlich wie beim Geburtsmonats-Vektor werden die Geburtsjahre verglichen. Jedoch wird dem dreidimensionalen Vektor eine weitere Dimension hinzugefügt. Daraus ergibt sich eine Abstufung über die Übereinstimmung von *übereinstimmend* über *stark übereinstimmend* zu *schwach übereinstimmend* bis zu *gar nicht übereinstimmend*. Wobei *übereinstimmend* wieder voraussetzt, dass beide Geburtsjahre gleich sind. Als *stark übereinstimmend* werden die Vergleiche klassifiziert, die eine Differenz gleich eins oder zwei getauschte Ziffern haben. Zu den als *schwach übereinstimmend* Geltenden zählen solche, die sich in einer Ziffer unterscheiden und eine Differenz größer als eins haben. Die Szenarien, die nicht einem der vorher genannten Abweichungsschemata entsprechen, werden als gar nicht übereinstimmend gewertet. Damit ergibt sich folgender Vektor:

$$v \in \{0,1\}^4 \text{ mit } \exists! v = 1$$

Die vier Möglichkeiten von Klassifizierungen aufgelistet:

Klassifizierung	Vektor
übereinstimmend	$(1,0,0,0)$
stark übereinstimmend	$(0,1,0,0)$
schwach übereinstimmend	$(0,0,1,0)$
gar nicht übereinstimmend	$(0,0,0,1)$

Tabelle 2.3: Vierdimensionaler binärer Vektor für das Geburtsjahr

Beispiele:

Geburtsjahr 1	Geburtsjahr 2	Klassifizierung
1980	1980	übereinstimmend
1980	1890	stark übereinstimmend
1980	1981	stark übereinstimmend
1980	1986	schwach übereinstimmend
1980	1975	gar nicht übereinstimmend

Tabelle 2.4: Beispiele für die vielleicht resultierende Ähnlichkeit

2.2.2.1.3 ICD-10 Code ICD-10 ist die internationale statistische Klassifikation von Krankheiten und verwandter Gesundheitsprobleme. Sie wird von der Weltgesundheitsorganisation (WHO) publiziert. Die Codes beschreiben Krankheiten in unterschiedlichen Genauigkeitsklassen, wodurch sich eine Baumstruktur aus Klassen ergibt. Die Genauigkeitsklassen werden im Folgenden als Ebenen bezeichnet. Darüber hinaus beschreibt eine umso größere Genaugkeitsklasse, eine desto tiefere Ebene in der Baumstruktur. Zum Beispiel kann eine Diagnose, die eine Krankheit im Bereich des Kopfes beschreibt, eine Krankheit, die sich auf das Gehirn bezieht, mit einbeziehen. In dem Fall kann man sagen, dass die Diagnose, die sich auf das Gehirn bezieht, genauer ist. Das heißt umso tiefer die Ebene des Codes liegt, desto genauer ist die Diagnose.

Die Baumstruktur kann genutzt werden, um eine Näherung zweier Codes zu bestimmen. Man berechnet den Abstand beider Codes bis zum letzten gemeinsamen Abzweigpunkt und wählt von den beiden Differenzen (Codetiefe - letzter gleicher Abzweigpunkt), die kleinere Differenz als Näherung.

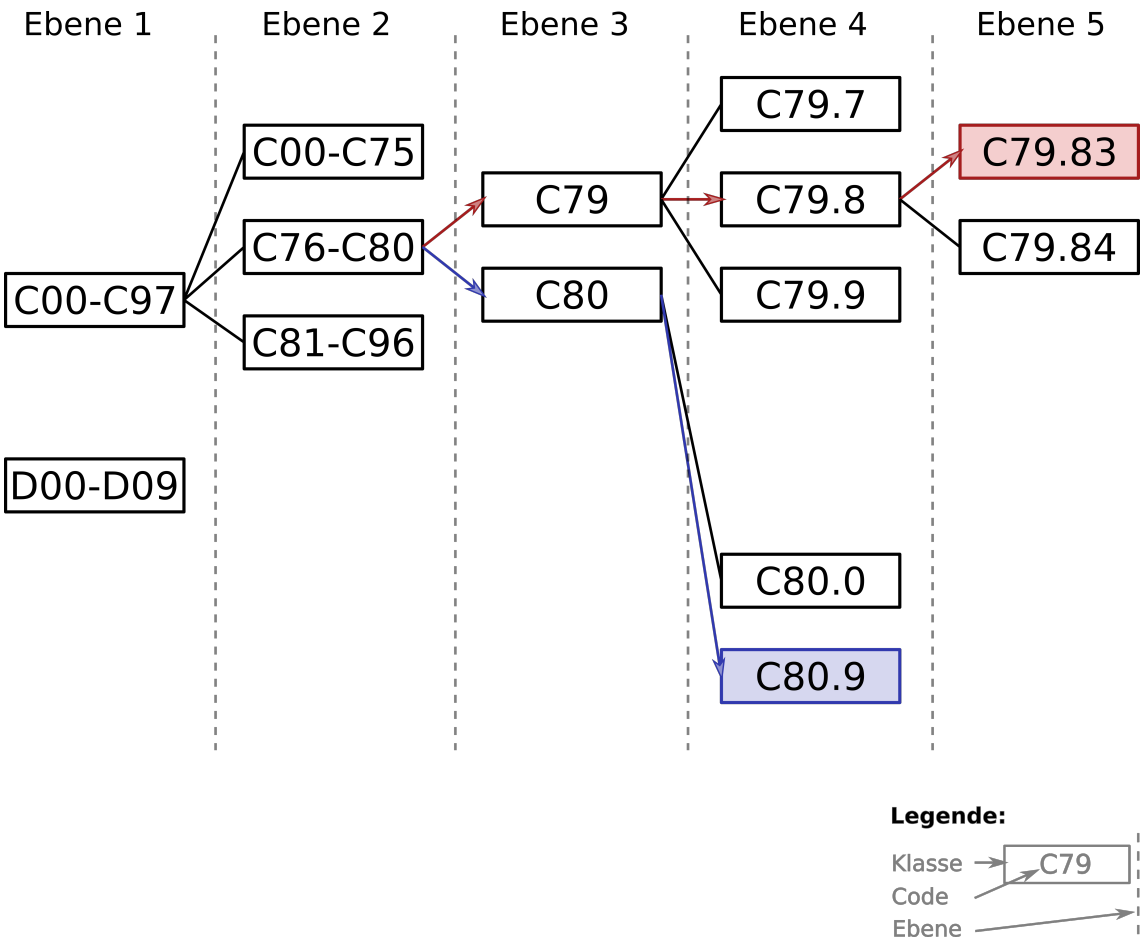


Abbildung 2.2: Baumstruktur der ICD-10-Codes

Wie man in Abbildung 2.2 erkennen kann, hat die Baumstruktur eine maximale Tiefe von fünf Ebenen. Bleibt man bei dem Beispiel aus Abbildung 2.2 mit den gegebenen Beispiel-Codes „C79.83“ und „C80.9“, hat man den Code „C79.83“ auf Ebene 5 und den Code „C80.9“ auf Ebene 4. Der letzte gemeinsame Code „C76-C80“ liegt auf Ebene 2. Daraus ergeben sich zwei Abstände von jeweils der beiden Ebenen der gegebenen Codes bis zu der Ebene des letzten gemeinsamen Codes. Einen Abstand kann man durch die Differenz zwischen der Ebenentiefe eines gegebenen Codes und der Ebenentiefe des letzten gemeinsamen Codes beschreiben. Im Fall des Beispieles ergeben sich die Differenzen 3 aus Ebene 5 und Ebene 2 und 2 aus Ebene 4 und Ebene 2. Die kleinere der beiden Differenzen (Im Beispiel: 2) ergibt die nähere Näherung. Diese Näherung wird nun noch auf einen Vektor projiziert. Dies geschieht indem die Dimension gleich 1 gesetzt wird, die durch den Wert der kleineren Differenz beschrieben wird. Durch setzen der zweiten Dimension ergibt sich der fertige Vektor für das Beispiel: $(0, 0, 1, 0, 0)$

Verallgemeinert ergibt sich folgender Vektor:

$$d_{Differenz_Min} = \min(e_{code_1} - e_{code_{gemeinsam}}, e_{code_2} - e_{code_{gemeinsam}})$$

$$v \in \{0,1\}^5 \text{ mit } \exists! v_{d_{Differenz_Min}} = 1$$

Die vier Möglichkeiten von Klassifizierungen aufgelistet:

Klassifizierung	Vektor
übereinstimmend (null Ebenen)	$(1, 0, 0, 0, 0)$
eine Ebene	$(0, 1, 0, 0, 0)$
zwei Ebenen	$(0, 0, 1, 0, 0)$
drei Ebenen	$(0, 0, 0, 1, 0)$
gar nicht übereinstimmend (vier Ebenen)	$(0, 0, 0, 0, 1)$

Tabelle 2.5: Fünfdimensionaler binärer Vektor für die ICD-10 Codes

2.2.2.2 Pseudonymisierte Daten

2.2.2.2.1 Namen Wie die Merkmale die im Klartext vorliegen, werden die pseudonymisierten Merkmale verglichen und die Näherung vektorisiert dargestellt. Vergleicht man die zwei Zeichenfolgen ohne Hintergrundwissen, kann man ausschließlich bestimmen, ob sie den gleichen Namen oder unterschiedliche Namen widerspiegeln. Das liegt daran, dass jede noch so kleinste Änderung auf Ebene der nicht pseudonymisierten Daten eine komplett unterschiedliche Zeichenfolge auf Ebene der pseudonymisierten Daten ergibt. Um trotzdem eine Abstufung der Ähnlichkeit der beiden Namen feststellen zu können, wird auf weitere Informationen des Landeskrebsregisters zurückgegriffen.

Gleiche Zeichenfolgen im Klartext führen zu immer den gleichen pseudonymisierten Werten. Daraus kann man schließen, wie häufig ein Name vorkommt. Namen, die öfter in der Datenbank vorkommen, kommen mit einer hohen Wahrscheinlichkeit öfter in der Gesellschaft vor. Damit steigt auch die Wahrscheinlichkeit, dass ein Name, der seltener in der Datenbank vorkommt, wahrscheinlicher auf den gleichen Patienten hindeutet, als ein Name, der öfter vorkommt. Daraus ergibt sich eine Wahrscheinlichkeit, die als Probability U (PU)-Wert gekennzeichnet wird.

$$P_u = \frac{Vorkommendes Merkmales in einer Meldung}{Anzahl der Meldungen}$$

Außerdem gibt es eine weitere Wahrscheinlichkeit, die Probability M (PM), die Auskunft darüber gibt, wie häufig bei Meldungen zur selben Person auch die Ausprägungen eines bestimmten Merkmals übereinstimmen.

$$P_m = \frac{Anzahl der Meldungen zur selben Person mit richtiger Auspraegung}{Anzahl der Meldungen zur selben Person}$$

Das LKR NRW führt außerdem Listen von synonymen Vornamen in pseudonymisierter Form, denn im Klartext können unterschiedliche Namen auf den gleichen Namen deuten, wenn zum Beispiel eine Abkürzung des anderen Namen verwendet wird, Umlaute anders dargestellt werden oder vorauszusehende Verständnisprobleme auftreten. So bilden etwa die jeweiligen pseudonymisierten Formen von Rudi und Rudolf eine solche Gruppe.

Der aus dem Vergleich entstehende Vektor besitzt insgesamt fünf Dimensionen, wobei die erste Dimension einer 1 entspricht, wenn die Zeichenketten identisch sind. Die zweite Dimension nimmt den Wert 1 an, wenn die Zeichenketten unterschiedlich sind, jedoch der gleichen Vornamensgruppe angehören. Sind die zwei Zeichenketten nicht gleich und tauchen auch nicht in der selben Vornamensgruppe auf, wird die vierte Dimension des Vektors gleich 1 gesetzt. Für jeden Vektor gilt, dass nur eine der drei genannten Dimensionen den Wert 1 haben darf und muss. Die anderen zwei Dimensionen müssen im selben Vektor den Wert 0 haben. Für die dritte Dimension wird auf den PM-Wert zurückgegriffen. Zuletzt wird die Häufigkeit des Namens, durch Setzen der fünften Dimension, dem PU-Wert, mit eingebunden.

Die Ausprägungen für Dimension 3 (PM-Wert) und 5 (PU-Wert) können Werte größer 0 und kleiner 1 einnehmen, denn die genannten Wahrscheinlichkeiten liefern mehr Spielraum während des Vergleichens.

Damit ergibt sich folgender Vektor:

$$v \in \{a\}^5 \text{ mit } v_{1,2,4} := a \in \{0, 1\} \text{ und } v_{3,5} := a \in \{|\ 0 < a < 1\}$$

3 Implementierung

3.1 Entwicklungsumgebung

Das Projekt wurde auf einem ThinkPad T470p mit einem *64-Bit Operating System* durchgeführt. Die Hardware besteht hauptsächlich aus einem Intel Core i7-7700HQ Prozessor mit einer 2,80GHz Taktfrequenz und einem DDR4 Arbeitsspeichermodul mit einer Kapazität von 16Gigabyte (GB). Für die Berechnung der Daten und ihrer Klassifikation war die Hardware nach eigenem Ermessen ausreichend, aber mindestens notwendig.

Als *integrierte Entwicklungsumgebung* wählte ich IntelliJ IDEA, in welcher ich mit der von Bellsoft zur Verfügung gestellten „Liberica Full JDK 8u282+8 x86 64 Bit for Linux “ arbeitete. Unterstützt wird die Programmierung durch das *Build-Werkzeug* „Apache Maven 3.6.3“.

3.2 Programm-Struktur

Das in Java programmierte Programm ist in drei separat ausführbare Unterprogramme aufgeteilt:

I. **Programm 1**: Daten aufbereiten und analysieren
II. **Programm 2**: Konfigurieren, Trainieren und Testen des KNN
III. **Programm 3**: Nutzen des KNN

In Abbildung 3.1 kann man erkennen, welche Hauptfunktionen der Programme von welchem Anwender ausgeführt, bzw. welchem der drei Programme bearbeitet, werden. Um das gesamte Programm im vollen Umfang zu nutzen, können die einzelnen Unterprogramme in der genannten Reihenfolge ausgeführt werden. Im Folgenden werde ich kurz beschreiben, welche Funktionen in den einzelnen Unterprogrammen umgesetzt sind.

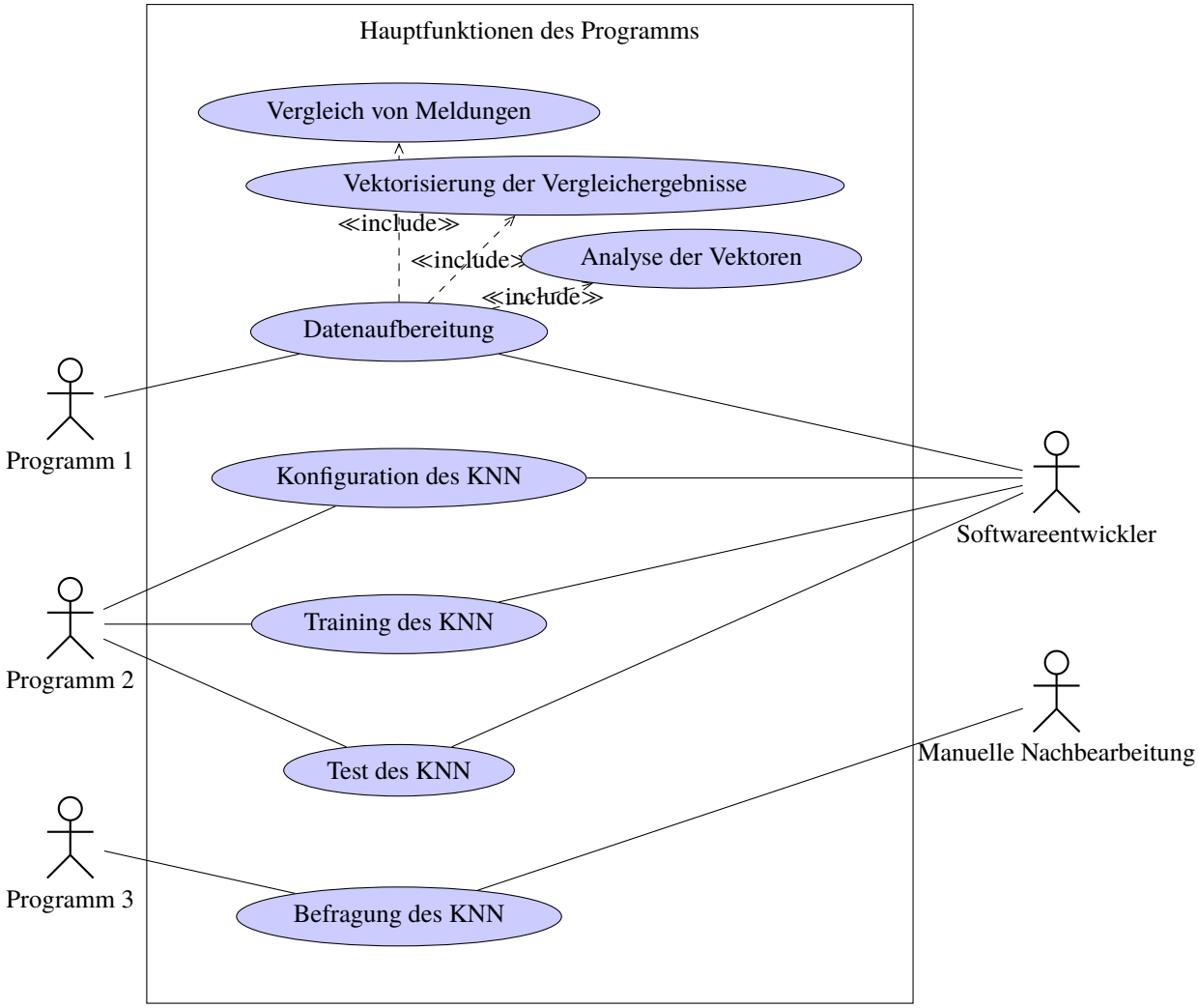


Abbildung 3.1: Use-Case-Diagramm der Hauptaufgaben des gesamten Programms

3.3 Programm 1

Programm 1 umfasst alle Funktionen von der Datenaufbereitung über die Analyse der Daten bis zur Aufteilung dieser in Trainings- und Testdaten. Wobei die Datenaufbereitung die Funktion des Vergleichens und der Vektorisierung einschließt. Die Schritte sind in der Abbildung 3.2 übersichtlich dargestellt.

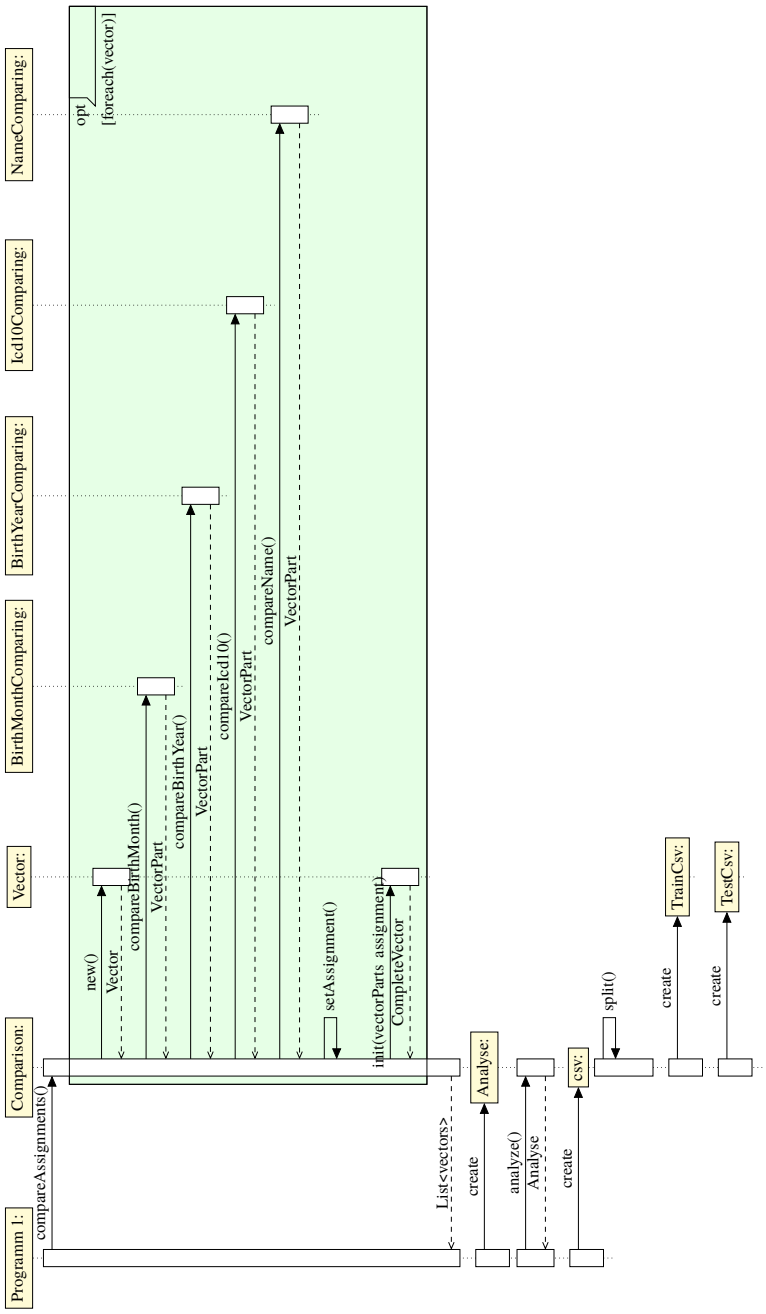


Abbildung 3.2: Sequence-Diagramm: Vergleichen und Vektorisieren von Meldungen, Analysieren der Vektoren und Aufteilen dieser in Trainings- und Testvektoren

Wie am Ende von Abschnitt 2.2.1 beschrieben, benötigen wir Meldungsgruppen, die aus einer neuen Meldung, mehreren alten Meldungen und einer Information über ihre Zuordnung, bestehen. Die Meldungsgruppen liegen uns in Form einer XML-Datei vor. Ein Beispiel einer gekürzten, beispielhaften XML-Datei ist in Listing 5.1 gegeben. Die Abbildung 3.3 beschreibt, wie die Meldungsgruppen aus der XML-Datei auf die Struktur der Entity-Klassen abgebildet werden.

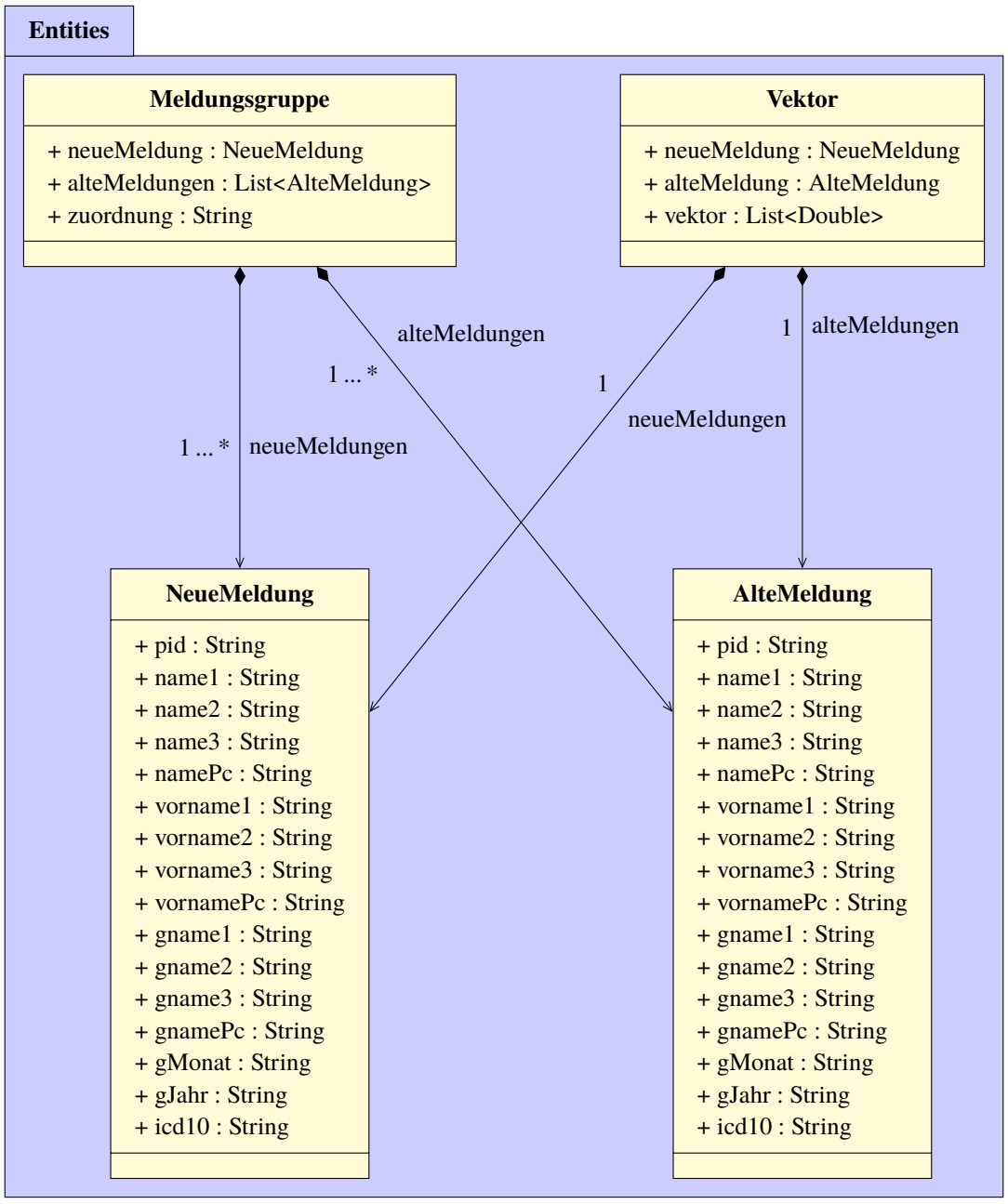


Abbildung 3.3: Entity Struktur für Meldungen und Vektoren

Nun iterieren wir durch die einzelnen Meldungsgruppen und vergleichen innerhalb der Meldungsgruppe die neue Meldung mit jeder vorliegenden alten Meldung. Die Meldungspaare werden, nach dem in Abschnitt 2.2.2 beschriebenen Vorgehen, verglichen und vektorisiert. Anschließend werden die Vektoren noch einer Analyse unterzogen, um implausible Ergebnisse zu vermeiden.

Der Prozess des Vergleichens und der Erstellung der Vektoren ist sehr rechenintensiv, weswegen wir alle Vektoren in einer CSV-Datei speichern, um diese immer wieder verwenden zu können.

3.3.1 Datenaufbereitung

Der Schritt „Datenaufbereitung“ ist mindestens genauso wichtig und entscheidend wie die Konfiguration des KNN selbst. In Abschnitt 2.2.2 habe ich schon die Theorie erklärt, nach welchem Schema die einzelnen Daten zweier Meldungen verglichen werden sollen und anschließend die Ergebnisse der Vergleiche in Vektoren umgewandelt werden. Nun gehe ich auf die Umsetzung von der Theorie in die Praxis, anhand von Aktivitätsdiagrammen, ein.

Beim Vergleich zweier Meldungen werden jeweils die gleichen Merkmale verglichen. Ein Teilvektor bezieht sich auf ein Merkmal. Die Informationen aus den Merkmalen werden in Form von Zeichenfolgen vom Typ *String* instanziiert. Für die Vektor-Instanzen der Klartextdaten verwenden wir *Integer-Arrays* und für die pseudonymisierten Daten *Float-Arrays*. Die Länge der Arrays entspricht der höchsten Dimension des Vektors. Eine mögliche Art der Implementierung zur Erstellung der Teilvektoren kann man folgenden Aktivitätsdiagrammen entnehmen:

I. **Abbildung 3.4** Vergleich zweier Geburtsmonate
II. **Abbildung 3.5** Vergleich zweier Geburtsjahre
III. **Abbildung 3.6** Vergleich zweier Diagnosen in Form von ICD-10 Codes
IV. **Abbildung 3.7** Vergleich zweier Namen

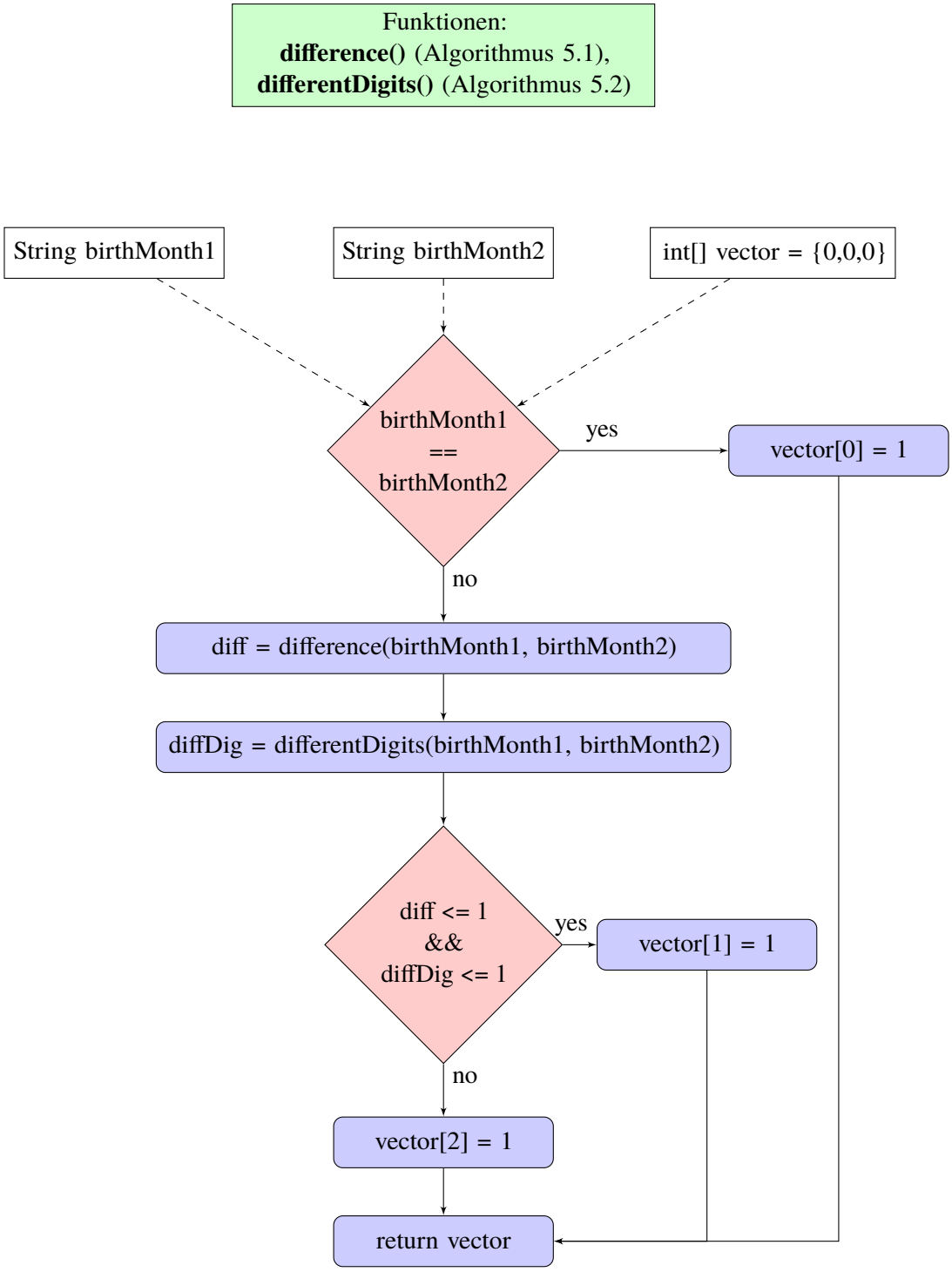


Abbildung 3.4: Aktivitäts-Diagramm: Validierung zweier Geburtsmonate. Eine inhaltliche Beschreibung befindet sich zu diesem Ablauf in Abschnitt 2.2.2.1.1.

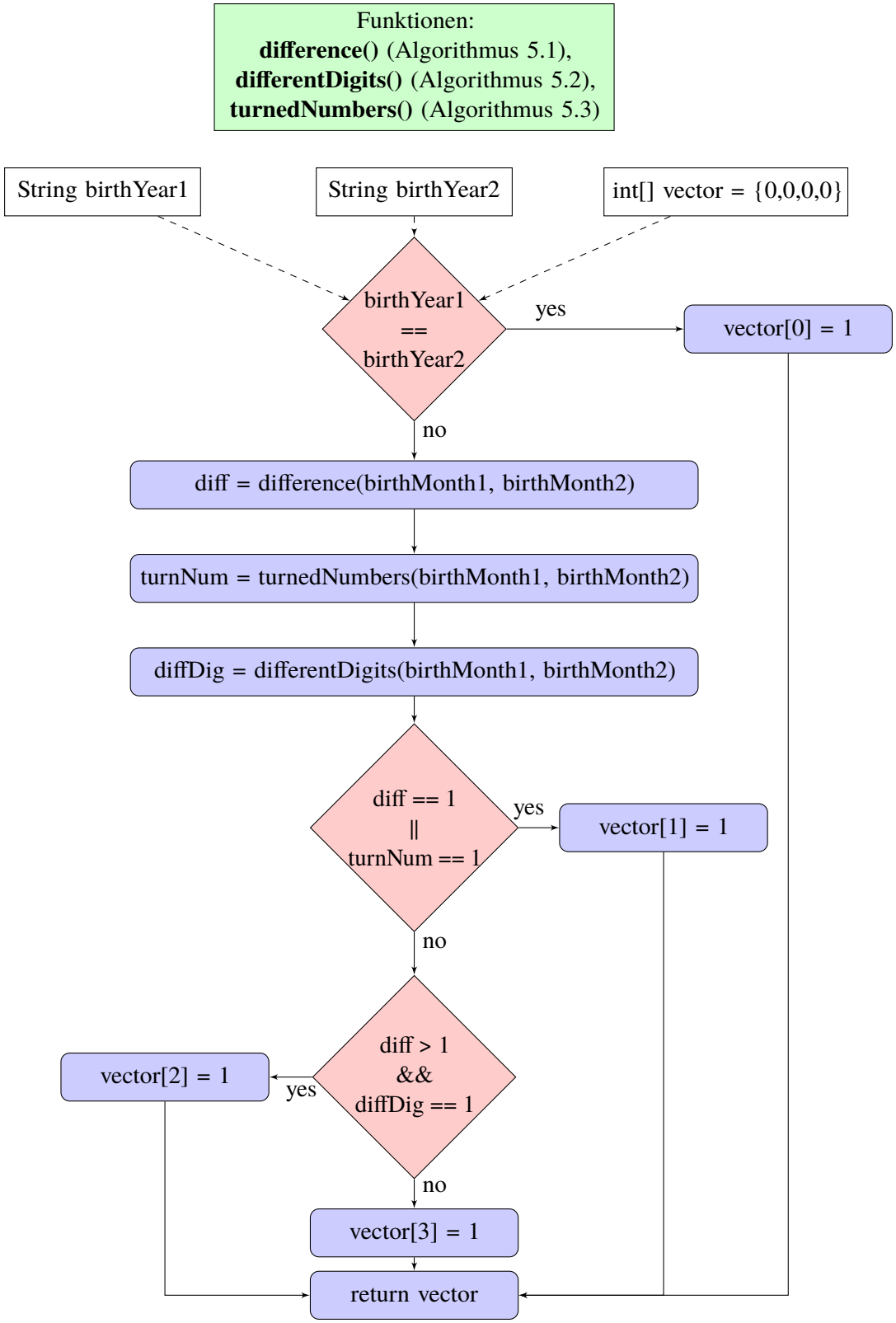


Abbildung 3.5: Aktivitäts-Diagramm: Validierung zweier Geburtsjahre. Eine inhaltliche Beschreibung befindet sich zu diesem Ablauf in Abschnitt 2.2.2.1.2.

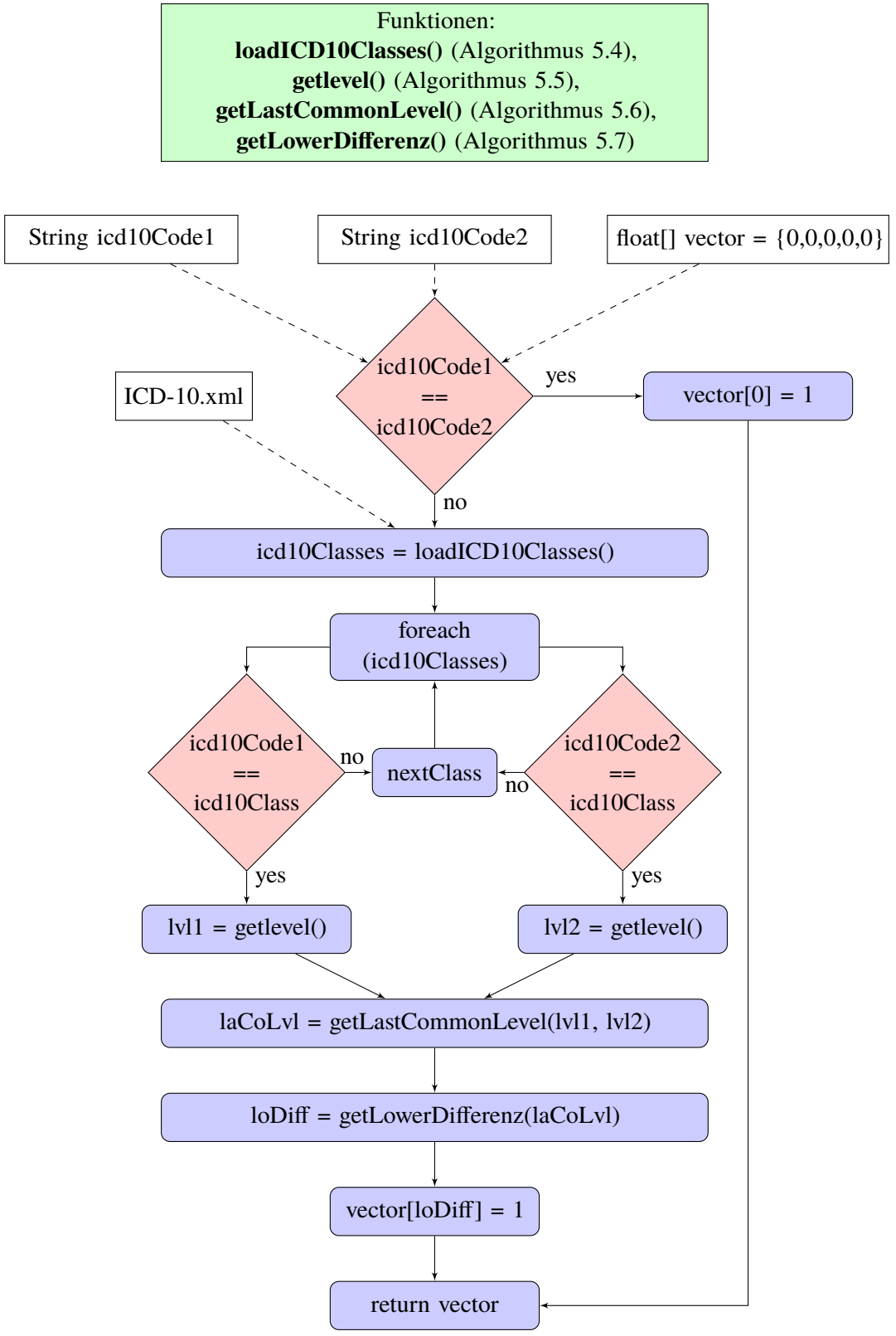


Abbildung 3.6: Aktivitäts-Diagramm: Validierung zweier ICD-10 Codes. Eine inhaltliche Beschreibung befindet sich zu diesem Ablauf in Abschnitt 2.2.2.1.3.

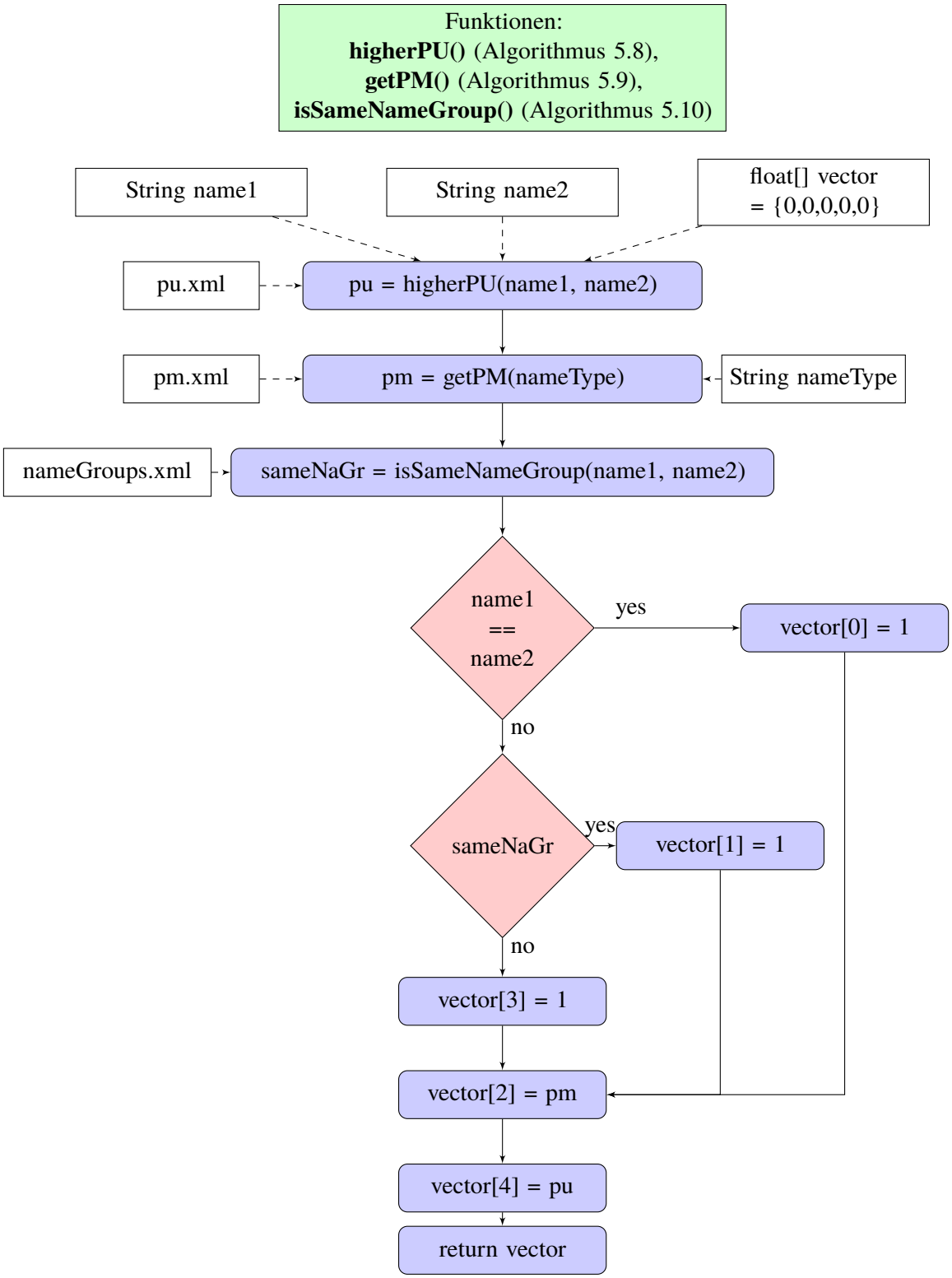


Abbildung 3.7: Aktivitäts-Diagramm: Validierung zweier Namen. Eine inhaltliche Beschreibung befindet sich zu diesem Ablauf in Abschnitt 2.2.2.2.1.

Nachdem alle Teilvektoren ermittelt worden sind, werden sie in Reihe hintereinander gereiht, sodass sich ein Vektor ergibt, der so viele Dimensionen, wie die Summe der Dimensionen aller Teilvektoren, besitzt. Dieser Vektor wird noch um die Dimension der Zuordnung erweitert. Die Dimension kann den Wert „1 “ oder „0 “ annehmen. Ist die „1 “ gesetzt, gehören die verglichenen Meldungen zu einem Patienten. Beim Wert „0 “ gehören die Meldungen zu unterschiedlichen Patienten. Bei der späteren Anwendung des KNN ist die Dimension der Zuordnung nicht gesetzt, da das Ergebnis des Vergleichs noch nicht bekannt ist.

3.3.2 Datenanalyse

Um uns zu vergewissern, dass die Ergebnisse der Datenaufbereitung in den vorgesehenen Wertebereichen liegen, analysieren wir mittels der Bibliothek „Datavec“ unsere erstellten Vektoren. Die Auswertung doppelt sich in zwei Ausprägungen. Die eine Ausprägung liegt in Form einer Tabelle, wo jede Zeile eine Dimension des Eingabevektors beschreibt. Die andere Ausprägung liegt in mehreren Balkendiagrammen vor. Auch hier nimmt jedes Balkendiagramm Bezug auf eine Dimension des Eingabevektors.

Für die Umsetzung fügen wir die „datavec-api ,“ -Dependency, wie in Listing 3.1 unserer Project Object Model (POM)-Datei hinzu und implementieren den Code aus Listing 3.2.

In Listing 3.2 erstellen wir ein Schema, die Struktur der vorliegenden Daten, für die anschließende Analyse unserer Eingabevektoren. Zuerst müssen wir die Daten erst in ein Objekt vom Typ *FileSplit* laden. Dies geschieht während der Instanziierung mit Übergabe des Dateipfades zu den Datenvektoren und des Objektes *Random*, dass sicherstellt, dass die Vektoren in zufälliger Reihenfolge geladen werden. Anschließend initialisieren wir ein neues Objekt vom Typ *CSVRecordReader* über die Methode *initialize()* mit unserem Objekt *Filesplit*. Nun erzeugen wir eine Instanz vom Objekt *Schema* über das Objekt *Schema.Builder()*. Bei unseren Datenvektoren liegt jede Dimension in einem *Double*-Wert vor, abgesehen vom Label, dass einen Integer-Wert vorweist. Deswegen fügen wir unserem Schema mit der Methode *addColumnDouble()*, 72 mal eine Spalte vom Typ *Double* hinzu. Ähnlich fügen wir für unser Label, die Zuordnung (Information, ob die zwei Meldungen die selbe Person beschreiben), eine Spalte vom Typ *Integer* mit dem Methodenaufruf *addColumnCategorial()* zu unserem Schema hinzu. Bei beiden Methoden-Aufrufen übergeben wir jeweils eine Bezeichnung für den jeweiligen Wert, bei uns einfachheitshalber die Dimension des Vektors. Zusätzlich übergeben wir der *addColumnCategorial()* noch die möglichen Werte 0 und 1. Mit den Instanzen von den Objekten *Schema* und *CSVRecordReader* können wir über die Methode *analyze()* auf dem Objekt *AnalyzeLocal* eine Instanz vom Typ *DataAnalysis* generieren, dass nun die Informationen über das *Schema* und eine Analyse der Daten zu jeder Spalte hält.

Listing 3.1 Maven: DataVec Api

```
<!-- https://mvnrepository.com/artifact/org.datavec/datavec-api -->
<dependency>
<groupId>org.datavec</groupId>
<artifactId>datavec-api</artifactId>
<version>0.9.1</version>
</dependency>
```

Listing 3.2 Datenanalyse mit DataVec

```
Random random = new Random();
random.setSeed(0xC0FFEE);
File file = new File(inputVectors);
FileSplit inputSplit = new FileSplit(file, random);
CSVRecordReader recordReader = new CSVRecordReader();

recordReader.initialize(inputSplit);

Schema schema = new Schema.Builder()
.addColumnDouble("0")
.addColumnDouble("1")
.addColumnDouble("2")
.addColumnDouble("3")
[...]
.addColumnDouble("69")
.addColumnDouble("70")
.addColumnDouble("71")
.addColumnCategorical("72", "0", "1")
.build();

DataAnalysis analysis = AnalyzeLocal.analyze(schema, recordReader);

try {
HtmlAnalysis.createHtmlAnalysisFile(analysis, new File(analyseOutput));
} catch (Exception e) {
e.printStackTrace();
}

Analyse analyse = new Analyse();
analyse.setAnalysis(analysis);
analyse.setSchema(schema);
```

Zur Ausgabe der Analyse rufen wir die Methode *createHtmlAnalysisFile()* auf dem Objekt *HtmlAnalysis* mit den Übergabeparametern, der Instanz des Objektes *DataAnalysis* und dem Dateipfad zur Ausgabe der Analyse, auf. Zum Schluss speichern wir noch die Objekte *DataAnalysis* und *Schema* im Objekt *Analyse* ab, da wir die Informationen später noch einmal benötigen.

Die Abbildungen: Abbildung 3.8, Abbildung 3.9, Abbildung 3.10 enthalten Beispiele auf Grundlage meiner Datenanalysen.

Summary Column Analysis

Column Name	Column Type	Column Analysis
0	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.6790037567830822, sampleStDev=0.4668918353652986, sampleVariance=0.21798798593077712, countZero=2307, countNegative=0, countPositive=7187, countMinValue=2307, countMaxValue=4880, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 1.0, 0.9 : 1.0, 0.99 : 1.0, 0.999 : 1.0])
1	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.2166411576457486, sampleStDev=0.41198468770531005, sampleVariance=0.16973138290364184, countZero=5630, countNegative=0, countPositive=7187, countMinValue=5630, countMaxValue=1557, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.0, 0.9 : 1.0, 0.99 : 1.0, 0.999 : 1.0])
2	Double	DoubleAnalysis(min=0.0, max=0.98247962, mean=0.9775583362487815, sampleStDev=0.0693650591457811 7, sampleVariance=0.004811511430297719, countZero=36, countNegative=0, countPositive=7187, countMinValue=36, countMaxValue=7151, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.98247962, 0.1 : 0.98247962, 0.5 : 0.98247962, 0.9 : 0.98247962, 0.99 : 0.98247962, 0.999 : 0.98247962])
3	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.09934604146375399, sampleStDev=0.29914688198109207, sampleVariance=0.08948885699900944, countZero=6473, countNegative=0, countPositive=7187, countMinValue=6473, countMaxValue=714, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.0, 0.9 : 0.5023971377459772, 0.99 : 1.0, 0.999 : 1.0])
4	Double	DoubleAnalysis(min=1.0E-7, max=1.0, mean=0.010861517031688847, sampleStDev=0.07048981561669133, sampleVariance=0.004968814105675 1425, countZero=0, countNegative=0, countPositive=7187, countMinValue=3, countMaxValue=36, count=7187, quantiles=[0.001 : 1.68054E-7, 0.01 : 1.68054E-7, 0.1 : 3.1091006999560 36E-5, 0.5 : 0.0035283579929918907, 0.9 : 0.01710086255025886, 0.99 : 0.023483789867, 0.999 : 1.0])
5	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.8181438708779748, sampleStDev=0.3857527474857536, sampleVariance=0.1488051821928 0758, countZero=1307, countNegative=0, countPositive=7187, countMinValue=1307, countMaxValue=5880, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 1.0, 0.9 : 1.0, 0.99 : 1.0, 0.999 : 1.0])
6	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.014331431751774046, sampleStDev=0.11886129558567815, sampleVariance=0.014128007588305952, countZero=7084, countNegative=0, countPositive=7187, countMinValue=7084, countMaxValue=103, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.0, 0.9 : 0.0, 0.99 : 1.0, 0.999 : 1.0])

Abbildung 3.8: Datenanalyse mittels DataVec: Tabelle (Anfang)

67	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.49450396549325215, sampleStDev=0.5000045792494145, sampleVariance=0.25000457927038405, countZero=3633, countNegative=0, countPositive=7187, countMinValue=3633, countMaxValue=3554, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.1922061928219563, 0.9 : 1.0, 0.99 : 1.0, 0.999 : 1.0])
68	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.028802003617642964, sampleStDev=0.1672612950939583, sampleVariance=0.027976340836508195, countZero=6980, countNegative=0, countPositive=7187, countMinValue=6980, countMaxValue=207, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.0, 0.9 : 0.0, 0.99 : 1.0, 0.999 : 1.0])
69	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.18853485459858 12, sampleStDev=0.3911658895324615, sampleVariance=0.1530107531337219, countZero=5832, countNegative=0, countPositive=7187, countMinValue=5832, countMaxValue=1355, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.0, 0.9 : 1.0, 0.99 : 1.0, 0.999 : 1.0])
70	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.05342980381243917, sampleStDev=0.2249046417392775, sampleVariance=0.050582097875872756, countZero=6803, countNegative=0, countPositive=7187, countMinValue=6803, countMaxValue=384, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.0, 0.9 : 0.0, 0.99 : 1.0, 0.999 : 1.0])
71	Double	DoubleAnalysis(min=0.0, max=1.0, mean=0.23472937247808484, sampleStDev=0.4238590468526872, sampleVariance=0.17965649159886848, countZero=5500, countNegative=0, countPositive=7187, countMinValue=5500, countMaxValue=1687, count=7187, quantiles=[0.001 : 0.0, 0.01 : 0.0, 0.1 : 0.0, 0.5 : 0.0, 0.9 : 1.0, 0.99 : 1.0, 0.999 : 1.0])
72	Categorical	CategoricalAnalysis(CategoryCounts={1=4117, 0=3070})

Abbildung 3.9: Datenanalyse mittels DataVec: Tabelle (Ende)

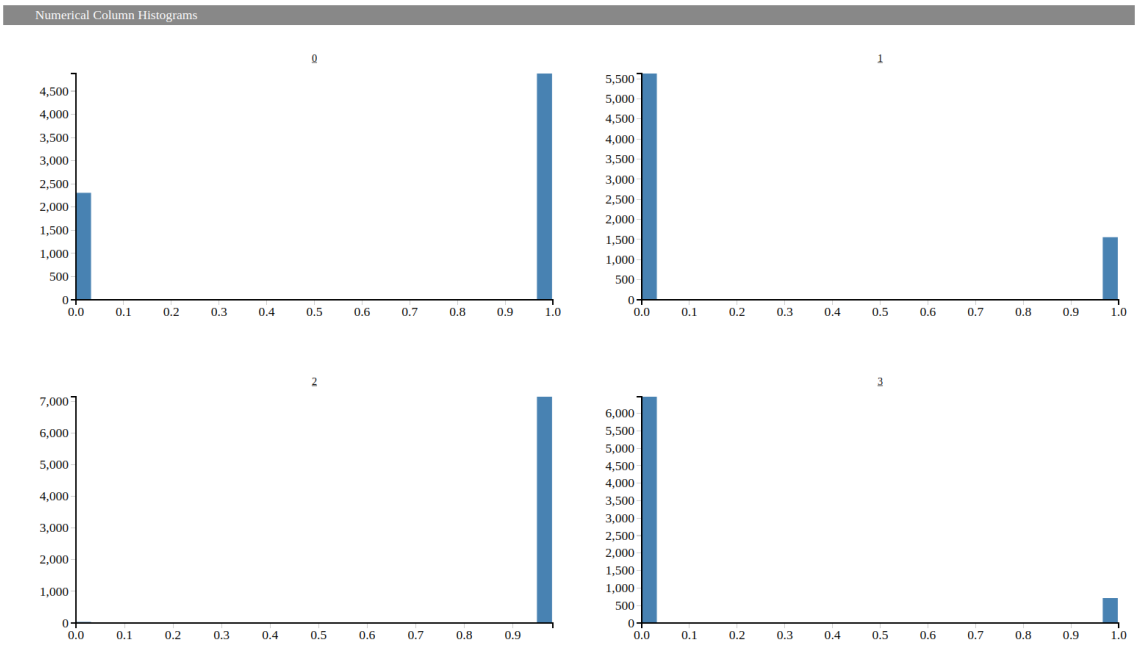


Abbildung 3.10: Datenanalyse mittels DataVec: Visuell in Diagrammen

3.4 Programm 2

Im zweiten Unterprogramm fahren wir mit der Implementation des KNN fort. Dazu gehört die Konfiguration des KNNs, das Training, sowie die Evaluation des Modells. Zusätzlich überwachen wir das Training und lassen uns während des Trainings und nach der Evaluierung ausgeben, wie der Lernprozess des Modells voranschreitet. Da die Lernfortschritte bei Abbruch des Programms nicht verfallen sollen, speichern wir diese. Die Details der Implementierung ist jeweils in eigenen Unterkapiteln nachgestellt (Abschnitt 3.4.1, Abschnitt 3.4.2, Abschnitt 3.4.3). Nach Unterprogramm 2 ist das KNN einsetzbar. Es fehlt nur noch die Implementierung zur Eingabe der unbekannten Daten und der Ausgabe der Ergebnisse.

Für die Implementation des KNN nutzen wir eine Bibliothek von „Eclipse “ namens „Eclipse Deeplearning4j “. Um diese nutzen zu können, müssen die in Listing 3.3 aufgeführten Dependencies unserer POM-Datei hinzufügen.

3.4.1 Kofiguration

Wie sich aus der Kodierung der Eingangsdaten ergibt, muss die Eingangsschicht 72 Neuronen besitzen. Jedes der 72 Neuronen präsentiert eine Dimension unseres Datenvektors ohne Label. Auf der verdeckten Schicht platzieren wir 144 Neuronen, da unter meinen Probeläufen so das beste Ergebnis erzielt werden konnte. Abschließend umfasst die Ausgangsschicht zwei Neuronen, eines für jede mögliche Klassifizierung.

Nachdem wir den Grundaufbau des Netzes festgelegt haben, nehmen wir uns nun der Instanziierung der Konfiguration an. Wir instanziieren ein Objekt *MultiLayerConfiguration* und weisen ihm folgende Parameter zu. Zuerst setzen wir einen Random Seed, da wir den Gewichtungen im Modell Zufallswerte zuweisen. Der Random Seed ermöglicht es Zufallswerte zu reproduzieren. Die Zufallswerte folgen dem „Xavier“-Schema. Um einen Lerneffekt des Modells zu erreichen, konfigurieren wir einen „Updater“. In unserem Modell wird der „Stochastic Gradient Descent“ (stochastische Gradientenabstieg) Lernalgorithmus verwendet, der vom „Adam-Updater “ zur

Listing 3.3 Maven: deeplearning4j-core - und nd4j-native-platform Dependency

```
<!-- https://mvnrepository.com/artifact/org.deeplearning4j/deeplearning4j-core -->
<dependency>
  <groupId>org.deeplearning4j</groupId>
  <artifactId>deeplearning4j-core</artifactId>
  <version>1.0.0-beta7</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.nd4j/nd4j-native-platform -->
<dependency>
  <groupId>org.nd4j</groupId>
  <artifactId>nd4j-native-platform</artifactId>
  <version>1.0.0-beta7</version>
</dependency>
```

Listing 3.4 Instanziierung der MultiLayerConfiguration und des MultiLayerNetworks

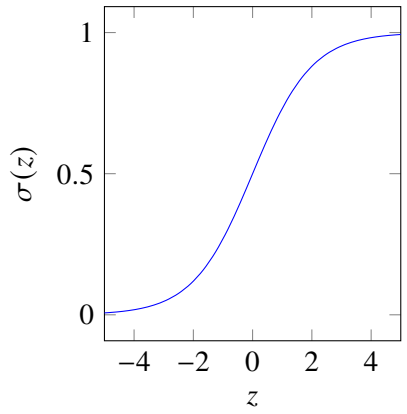
```
MultiLayerConfiguration config = new NeuralNetConfiguration.Builder()
  .seed(0xC0FFEE)
  .weightInit(WeightInit.XAVIER)
  .activation(Activation.SIGMOID)
  .updater(new Adam.Builder().learningRate(0.005).build())
  .l2(0.0000316)
  .list(
    new DenseLayer.Builder().nOut(144).build(),
    new OutputLayer.Builder(new LossMCXENT()).nOut(2).activation(Activation.SOFTMAX).build()
  )
  .setInputType(InputType.feedForward(finalSchema.numColumns() - 1))
  .build();

MultiLayerNetwork model = new MultiLayerNetwork(config);
model.init();
```

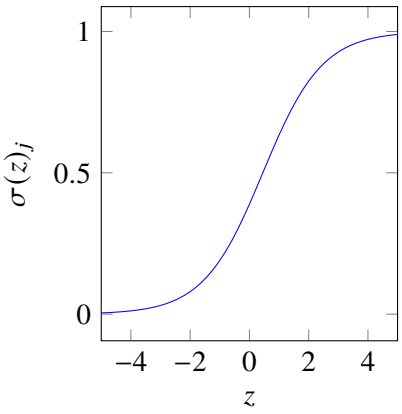
Verfügung gestellt wird. Den Updater initialisieren wir mit einer „learning rate “ (Lernrate) von 0.005. Durch die Lernrate kann festgelegt werden, wie stark die Anpassungen des Modells nach jedem Batch-Durchlauf sind. Um so höher die Lernrate, desto stärker sind die Anpassungen. Zuletzt müssen noch die einzelnen Schichten konfiguriert werden. Die verdeckte Schicht und Ausgangsschicht werden als Liste zweier unterschiedlicher Objekte erstellt. Das eine Objekt, die verdeckte Schicht, ist vom Typ *DenseLayer*. Dem *Dense-Layer* weisen wir 144 Neuronen zu und legen fest, dass diese durch die „Sigmoid “-Aktivierungsfunktion aktiviert werden sollen. Die andere Schicht ist die Ausgangsschicht und wird als *OutputLayer*-Objekt festgelegt. Die Ausgangsschicht wird mit zwei Neuronen bestückt. Diesen weisen wir nicht einfach eine Aktivierungsfunktion zu, sondern verschachteln eine *Softmax*-Aktivierungsfunktion in einer *Loss*-Funktion. Die Softmax-Funktion eignet sich, da sie eine Wahrscheinlichkeitsverteilung darstellt, in der die Werte zwischen „0 “ und „1 “ liegen und aufsummiert „1 “ ergeben. Die *Loss*-Funktion berechnet aus dem Ergebnis der *Softmax*-Funktion eine Differenz zum jeweiligen Label und diese wird dann an den schon beschriebenen *Updater* weitergegeben. Die Aktivierungsfunktionen sind in Abbildung 3.11 zur besseren Veranschaulichung verbildlicht. Die Eingangsschicht (Inputlayer) wird nicht mit den anderen in der Liste übergeben, sondern getrennt über die Methode *setInputType()* gesetzt. Der Methode übergeben wir die Modell-Architektur und die Anzahl der Neuronen auf der Eingangsschicht mittels des Objekts *InputType*. Die erstellte Konfiguration weisen wir letztendlich der Instanz des Typ *MultiLayerNetwork* zu und führen die Methode *init()* auf ihr aus. Der Code zur Erstellung des Modells ist in Listing 3.4 dargestellt.

3.4.2 Training

Für das Training des im Abschnitt 3.4.1 erzeugten Modells benötigen wir eine Instanz vom Objekt *FileSplit*. Die Instanz muss auf den Dateipfad zeigen, wo sich die Trainings-Vektoren befinden. Außerdem müssen die Daten im *Random*-Modus eingelesen werden, damit sich das Trainingsergebnis nicht auf Grundlage der Daten erst nur in die eine Richtung und dann nur in die andere Richtung bewegt.



(a) Sigmoid Aktivierungsfunktion



(b) Softmax Aktivierungsfunktion

Abbildung 3.11: Sigmoid und Softmax Aktivierungsfunktionen

Über das Objekt *TransformProcess* können die Spalten unserer Daten nochmal in andere Formate transformiert werden. Dazu greifen wir auf das in Abschnitt 3.3.2 erstellte *Schema* zurück, um die einzelnen Spalten anzusprechen. Da unsere Datenaufbereitung gründlich ist, brauchen wir dies nicht und transformieren das Schema, grundlegend gleichbleibend, in das andere Objekt.

Anschließend instanziieren wir die Größen der Batches und die Anzahl der Epochendurchläufe.

Um die Trainingsdaten an das Modell zu übergeben, benötigen wir ein Objekt *RecordReaderDataSetIterator*. Dieses referenziert das Objekt *TransformProcessRecordReader*, dass mit den zufällig angeordneten Trainingsdaten und der Batchgröße initialisiert wird. Außerdem legen wir im *RecordReaderDataSetIterator* die Klassifikation, die Label (Spalte) der Trainingsdaten und die Anzahl der möglichen Ergebnisse fest.

Vor dem Trainingbeginn rufen wir noch eine Methode *addListener()* mit dem Parameterwert 50 auf das Modell auf, um im Abstand von 50 Batch-Durchläufen den Durchschnittswert der Differenz zwischen den Labeln und der errechneten Ergebnisse in die Terminal-Ausgabe geloggt zu bekommen. An der Differenz kann man einen Trainingsfortschritt erkennen, wenn die Differenz abnimmt. Die Differenz wird kleiner, wenn die Ergebnisse öfter näher an den Labeln liegen und dadurch immer kleinere Anpassungen der Parameter des Modells vorgenommen werden müssen.

Nun starten wir mittels Aufruf der Methode *fit()* auf das Objekt des Modells das Training.

Der Code zum Training des Modells ist in Listing 3.5 gezeigt.

3.4.3 Evaluierung

Die Evaluierung umfasst den Vorgang des Testens. Zum Testen nutzen wir die restlichen 20% der in Abschnitt 3.3.1 erstellten Datenvektoren.

Zum Einlesen der Vektoren brauchen wir eine neue Instanz vom Objekt *TransformProcessRecordReader*. Diese wird wieder mit einem Objekt *FileSplit* initialisiert, das diesmal auf das Dateiverzeichnis der Testdaten verweist. Diesmal brauchen wir die Daten nicht in einer zufälligen Reihenfolge, da

Listing 3.5 Training des Modells

```
  Random random = new Random();
  random.setSeed(0xC0FFEE);
  File file = new File(trainVectors);
  FileSplit inputSplit = new FileSplit(file, random);

  TransformProcess transformProcess = new TransformProcess.Builder(analyse.getSchema())
    .build();

  int batchSize = 64;
  int epochCount = 500;

  TransformProcessRecordReader trainRecordReader = new TransformProcessRecordReader(new
CSVRecordReader(), transformProcess);
  trainRecordReader.initialize(inputSplit);

  RecordReaderDataSetIterator trainIterator = new RecordReaderDataSetIterator.Builder(
trainRecordReader, batchSize)
    .classification(finalSchema.getIndexOfColumn(columns[72]), 2)
    .build();

  model.addListeners(new ScoreIterationListener(50));
  model.fit(trainIterator, epochCount);
```

die Evaluation keinen Einfluss auf die Parameter des Modells hat. Wir legen die Batchgröße fest (Batchgröße der Trainingsdaten) und definieren wieder die Klassifikation mit unseren Labeln auf dem Objekt *RecordReaderDataSetIterator*.

Anchließend rufen wir die Methode *evaluate()* mit unserer Instanz vom Typ *RecordReaderDataSetIterator* auf und erhalten als Rückgabeparameter ein Objekt vom Typ *Evaluation*. Das Objekt *Evaluation* enthält nun Angaben über das Ergebnis aus dem Zusammenspiel des Trainings und der Evaluation. Eine Standardausgabe mit einer Grundübersicht über das erzielte Trainingsergebnis lassen wir uns über das Objekt *Evaluation* in der Terminal-Ausgabe ausgeben. Das Ergebnis meiner Evaluation ist in Listing 3.7 dargestellt.

Nach der Evaluierung speichern wir unseren Fortschritt ab, da das Training und die Evaluation ziemlich zeitintensiv ist und wir nicht vor jeder Nutzung das Modell neu trainieren wollen. Außerdem kann das Training durch Zwischenspeichern des Fortschritts unterbrochen und zu einem späteren Zeitpunkt fortgeführt werden. Die Datei wird im angegebenen Pfad als Binäre-Datei gespeichert und mit „model.bin“ benannt.

Den Code zur Evaluierung und Speicherung des Modells sieht man in Listing 3.6.

Listing 3.6 Evaluation und Sicherung des Modells

```
  TransformProcessRecordReader testRecordReader = new TransformProcessRecordReader(new
CSVRecordReader(), transformProcess);
  testRecordReader.initialize( new FileSplit(new File(testVectors)));

  String label = columns[72];
  RecordReaderDataSetIterator testIterator = new RecordReaderDataSetIterator.Builder(
testRecordReader, batchSize)
    .classification(finalSchema.getIndexOfColumn(label), 2)
    .build();

  Evaluation evaluate = model.evaluate(testIterator);
  System.out.println(evaluate.stats());
  System.out.println("MCC: "+evaluate.matthewsCorrelation(EvaluationAveraging.Macro));

  File modelSave = new File("util/train/vectors/model.bin");
  model.save(modelSave);
  ModelSerializer.addObjectToFile(modelSave, "dataanalysis", analyse.getAnalysis().toJson());
  ModelSerializer.addObjectToFile(modelSave, "schema", finalSchema.toJson());
```

Listing 3.7 Ergebnis der Evaluation

```
  ========================Evaluation Metrics========================
  # of classes:    2
  Accuracy:        0.9440
  Precision:       0.9429
  Recall:          0.9627
  F1 Score:        0.9527
  Precision, recall & F1: reported for positive class (class 1 - "1") only


  =========================Confusion Matrix=========================
  0    1
  -----------
  680   61 | 0 = 0
  39 1007 | 1 = 1

  Confusion matrix format: Actual (rowClass) predicted as (columnClass) N times
  ==================================================================
  MCC: 0.8845082910462817
```

3.5 Programm 3

In diesem Programm wenden wir das trainierte Modell an, indem wir es mit unbekannten Daten konfrontieren und uns das errechnete Ergebnis ausgeben lassen. Dieser Teil ist der Kürzeste, da dieser nur zu Demonstrationszwecken erarbeitet wurde. Die Grundprinzipien sind jedoch erkennbar und lassen sich erweitern. Die Implementierung folgt in Abschnitt 3.5.1.

3.5.1 Verwendung

Nun können wir unser trainiertes Modell, die Analyse und das Datenschema aus unserer „model.bin“-Datei laden. Anschließend definieren wir ein neues Schema und instanziieren auf Grundlage dessen ein neues Objekt *TransformProcess*, damit das neu erstellte Schema dem, in Abschnitt 3.3.2 erstellten Schema, entspricht. Dadurch ist sichergestellt, dass die Spalten der Beispiele in der gleichen Reihenfolge angeordnet sind. Letztendlich müssen wir in dem alten Schema nur die letzte Spalte, das Label, entfernen, da das Merkmal unbekannt ist und errechnet werden soll.

Der Code ist in Listing 3.8 zu sehen.

Um nun eine Klassifizierung zu erhalten, brauchen wir komplett neue Daten. Ein neuer Vektor wird nach Abschnitt 3.3.1 erstellt, jedoch ohne die letzte Dimension, die sich auf die Zusammengehörigkeit bezieht. Wir instanziieren diesen vom Typ *ArrayList<Double>* und transformieren ihn anschließend mittels des Objekts *TransformProcess*, sodass er für das Modell verständlich ist. Anschließend „befragen “ wir das Modell mit der *predict()*-Methode, die uns einen Rückgabewert in Form eines *Integer-Arrays* zurückliefert. Da wir nur einen Vektor abfragen, liefert uns die Methode auch nur

Listing 3.8 Vorbereitung

```
  File modelSave = new File(modelPath);
  MultiLayerNetwork model = ModelSerializer.restoreMultiLayerNetwork(modelSave);
  DataAnalysis analysis = DataAnalysis.fromJson(ModelSerializer.getObjectFromFile(modelSave, "
dataanalysis"));
  Schema targetSchema = Schema.fromJson(ModelSerializer.getObjectFromFile(modelSave, "schema")
);

  Schema schema = new Schema.Builder()
      .addColumnDouble(columns[0])
      .addColumnDouble(columns[1])
      .addColumnDouble(columns[2])
      [...]
      .addColumnDouble(columns[71])
      .build();

  String[] newOrder = targetSchema.getColumnNames().stream().filter(it -> !it.equals(label).
toArray(String[]::new);
  TransformProcess transformProcess = new TransformProcess.Builder(schema)
      .reorderColumns(newOrder)
      .build();
```

Listing 3.9 Eingabe und Ausgabe

```
  List<Writable> record = RecordConverter.toRecord(schema, Arrays.asList
(1.0,0.0,0.1,0.0,[...],0.0,1.0,1.0));
  List<Writable> transformed = transformProcess.execute(record);
  INDArray data = RecordConverter.toArray(transformed);

  int labelIndex = model.predict(data)[0];
  INDArray output = model.output(data, false);
  double[] result = model.output(data, false).toDoubleVector();
```

ein Array mit einer Länge von 1 zurück. Dies beinhaltet die Werte, die uns das Modell berechnet hat. In unserem Fall klassifiziert es zwischen 0 und 1. Da wir durch die *Softmax*-Funktion eine Normalverteilung als Klassifikation berechnen lassen, lassen wir uns zusätzlich die Regression ausgeben, woraus wir erkennen können, wo sich das Ergebnis zwischen 0 und 1 einordnet. Daraus lässt sich ableiten, mit welcher Wahrscheinlichkeit das Ergebnis richtig ist. Der Code für die Eingabe des Vektors und die Ausgabe des Ergebnisses ist in Listing 3.9 zu lesen.

4 Schluss

Mit der Fertigstellung des Praxisprojekts wurde ein voll funktionsfähiges KNN, inklusive der Datenaufbereitung, entworfen. Das KNN kann bei der Reduktion von manueller Nachbearbeitung eingesetzt werden. Aufgrund der hohen Validität des KNN mit Testdaten liegt das Modell in 1687 von 1787 Fällen richtig. Das entspricht einer Richtig-Klassifizierungsrate von 94,4%. Projiziert man dieses Ergebnis auf unseren Anwendungsfall, kann man sagen, dass das KNN bei der Fragestellung, ob Daten zu einem Patienten gehören, in ca. 94% der Fälle richtig entscheidet.

Für die Zukunft ist wichtig zu wissen, dass ein KNN mit unterschiedlich starken Wahrscheinlichkeiten klassifiziert. Das heißt, dass das KNN in manchen Fällen klarere Entscheidungen trifft, als in anderen Fällen. Daraus kann man schließen, dass in Fällen mit geringerer Klarheit der Klassifizierung das Potenzial der Falschklassifizierung steigt.

Für den Fall des produktiven Einsatzes gibt es mehrere mögliche Szenarien. Da die Entscheidung, ob Daten zu einem Patienten gehören, tiefgreifende Folgen für das weitere Vorgehen, wie zum Beispiel Kosten für Patienten und Krankenversicherung oder Forschung im Themenbereich Krebs, trägt, muss auf die Klassifizierung des KNN mindestens so viel Verlass sein wie auf den Record-Linkage-Algorithmus oder die Entscheidung eines Menschen.

Um das Kriterium zu erfüllen, können wir folgende Verwendungszwecke in der Zukunft in Betracht ziehen:

I. Jede Klassifizierung wird durch einen Menschen bestätigt oder korrigiert.
   Vorteil: Möglichkeit der Absicherung
II. Klassifizierungen mit einem geringen Risiko der falschen Klassifizierung (Grenzwert) werden als richtig klassifiziert angesehen. Die restlichen Klassifizierungen werden in die bekannte manuelle Nachbearbeitung weitergereicht.
   Vorteil: Weniger Aufwand, dadurch schnellere Bearbeitung
III. Kombination aus vorangestelltem Verwendungszweck und Überprüfung einzelner Attribute mit hoher Abweichung, wenn das KNN in kritischen Fällen ein hohes Risiko der falschen Klassifizierung aufweist.
   Vorteil: Minimaler Aufwand, dadurch schnellere Bearbeitung

Aufgrund der genannten Vorteile lohnt es sich auf jeden Fall, das Projekt weiter zu verfolgen und zu verbessern. Es gibt noch viele Anhaltspunkte zu Verbesserung des KNN:

I. Datenauswahl: Weiter mit mehr Meldungen trainieren
II. Datenauswahl: Weitere Merkmale der Meldungen vergleichen
III. Datenaufbereitung: Weitere Vergleichs-Algorithmen verwenden (z.B. Levenshtein-Distanz)
IV. KNN: Komplexere Struktur verwenden
V. KNN: Konfiguration des Modells genauer auf den Anwendungsfall anpassen

5 Anhang

Algorithmus 5.1 Berechnung der Differenz zweier Zahlen

```
function DIFFERENCE(int number1, int number2)
    diff = |number1 − number2|
    return diff
end function
```

Algorithmus 5.2 Berechnung wie viele Zeichen sich zwischen den zwei Zahlen unterscheiden

```
function DIFFRENTDIGITS(char[] number1, char[] number2)
    count ← 0
    for i ← 0; i < number1.length; i++ do
        if !(number1[i] == number2[i]) then
            count ← count + 1
        end if
    end for
    return count
end function
```

Algorithmus 5.3 Berechnung wie viele Zeichendreher sich zwischen den zwei gegebenen Zahlen befinden

```
function TURNEDDIGITS(char[] number1, char[] number2)
    count ← 0
    skipNext ← false
    for i ← 0; i < number1.length; i++ do
        if skipNext then
            skipNext ← false
        end if
        if !(i == number1.length) then
            if !(number1[i] == number2[i]) then
                if (number1[i] == number2[i + 1])&&(number1[i + 1] == number2[i]) then
                    count ← count + 1
                    skipNext ← true
                end if
            end if
        end if
    end for
    return count
end function
```

Algorithmus 5.4 Einbindung der XML-Datei mit enthaltenen ICD-10-Codes

```
function loadICD10Classes(ICD − 10 − Codes.xml)
    classes
    for all class in ICD − 10 − Codes.xml do
        if (number1[i] == number2[i + 1])&&(number1[i + 1] == number2[i]) then
            count ← count + 1
            skipNext ← true
        end if
    end for
    return classes
end function
```

Algorithmus 5.5 Zuordnung des ICD-10-Codes zu ICD-10-Klasse und Rückgabe der Tiefe des Codes in der Baumstruktur

```
function getLevel(classes, icd-10-code)
    for all class in classes do
        if class.code == icd-10-code then
            return List class.getParentClasses
        end if
    end for
end function
```

Algorithmus 5.6 Rückgabe der letzten gemeinsamen ICD-10-Klasse anhand von zwei Listen mit ICD-10-Code Eltern Elementen

```
function getLastCommonLevel(icd-10-code2.parentClasses, icd-10-code1.parentClasses)
    for i ← 0; i < icd-10-code1.parentClasses.length; i + + do
        for j ← 0; j < icd-10-code2.parentClasses.length; j + + do
            if icd-10-code1.parentClasses[i] == icd-10-code2.parentClasses[j] then
                lastCommonClass ← icd-10-code1.parentClasses[i]
                return lastCommonClass
            end if
        end for
    end for
end function
```

Algorithmus 5.7 Berechnung der kleineren Differenz von zwei ICD-10-Codes zum letzten gemeinsamen ICD-10-Code

```
function getLowerDiffrence(lastCommonClass, icd-10-code1, icd-10-code2)
    diff1 ← lastCommonClass.getLevel() − icd-10-code1.getLevel()
    diff2 ← lastCommonClass.getLevel() − icd-10-code2.getLevel()
    lowerDiff ← min(diff1,diff2)
    return lowerDiff
end function
```

Algorithmus 5.8 Ermittlung des PU-Wertes des selteneren Namens

```
function getHigherPu(namePu1, namePu2)
    if namePu1 < namePu2 then
        return namePu2
    elsenamePu1 > namePu2
        return namePu1
    end if
end function
```

Algorithmus 5.9 Ermittlung des PM-Wertes auf Grund des Attributes (Vorname, Nachname, Geburtsname)

```
function getPm(nameType, pm.xml)
    for all entry in pm.getEntries do
        if entry.nameType == nameType then
            return entry.value
        end if
    end for
end function
```

Algorithmus 5.10 Ermittlung, ob zwei Namen in der selben Namensgruppe sind

```
function isSameNameGroup(name1, name2, nameGroups.xml)
    for all group in nameGroups.xml do
        for all name in group do
            if name1 == name then
                foundName1 ← true
            end if
        end for
        for all name in group do
            if name2 == name then
                foundName2 ← true
            end if
        end for
        if foundName1&&foundName2 then
            return true
        end if
        foundName1 ← false
        foundName2 ← false
    end for
    return false
end function
```

Listing 5.1 Meldungsgruppen XML-Datei

```
<Meldungsgruppe>
  <NeueMeldung>
    <pid>1500899</pid>
    <psn_name1>293e472e839384a2c39839387f3527c39512f352729453d7136</psn_name1>
    <psn_namepc>703d4f583983938734a2c353754a2c398393877294d5b5t3136</psn_namepc>
    <psn_vorname1>4a2c3983938734a2c32e44543c286a56606f3a4f5614783136</psn_vorname1>
    <psn_vornamepc>504b3544a2c398393872f3527294b3a632uihc515c5f783136</psn_vornamepc>
    <gmonat>5</gmonat>
    <gjahr>1948</gjahr>
    <icd>C34.9</icd>
    [...]
  </NeueMeldung>
  <AlteMeldungenliste>
    <AlteMeldung>
      <pid>1500899</pid>
      <psn_name1>283938734a2c3394a2c398394a2c39839387c39839387d76</psn_name1>
      <psn_namepc>703d4f583938734a2c3983938734a4a2c398393872783136</psn_namepc>
      <psn_vorname1>342f365e4942483938734a24a2c4a2c3983938752729136</psn_vorname1>
      <psn_vornamepc>504b354d83938734a2c32e6f5038738735272945f783136</psn_vornamepc>
      <gmonat>5</gmonat>
      <gjahr>1948</gjahr>
      <icd>C34.8</icd>
      [...]
    </AlteMeldung>
    <AlteMeldung>
      <pid>1500899</pid>
      <psn_name1>293e472e335d373c39512f38394a4a2c39839387372353d783136</psn_name1>
      <psn_namepc>703d4f583938734a2c3953754257c834a2c398393879452783136</psn_namepc>
      <psn_vorname1>342f365e4983938734a2c344a2c39839387512f3527294f83136</psn_vorname1>
      <psn_vornamepc>504b354d56312c583938734a2c3a5b4a2c398393875272783136</psn_vornamepc>
      <gmonat>5</gmonat>
      <gjahr>1948</gjahr>
      <icd>C34.8</icd>
      [...]
    </AlteMeldung>
    <AlteMeldung>
      <pid>1500899</pid>
      <psn_name1>293e472e335d373c394a2c3984a2393875284d353d783136</psn_name1>
      <psn_namepc>703d4f58393874a2c3983938744a2c398393875b52783136</psn_namepc>
      <psn_vorname1>342f34a2c3983938764a2c39839387a56606f3a4f783136</psn_vorname1>
      <psn_vornamepc>504b354d56314a2c39839387a5b3a6324a2c39839387136</psn_vornamepc>
      <gmonat>5</gmonat>
      <gjahr>1948</gjahr>
      <icd>C34</icd>
      [...]
    </AlteMeldung>
  </AlteMeldungsliste>
  <Zuordnung>=</Zuordnung>
</Meldungsgruppe>
```

Listing 5.2 Probability-U XML-Datei

```
<probu>
  <probMap>
    <entry>
      <key>e765da7342253f88c89484a2c39839387146svbs24136</key>
      <value>7.51693E-7</value>
    </entry>
    <entry>
      <key>253f88c89642253f88c896b7b78a87154a2c398393876</key>
      <value>1.87923E-7</value>
    </entry>
    <entry>
      <key>232e02s4thv3463zub3m2434a2c39839387b7b78a8763</key>
      <value>6.01355E-6</value>
    </entry>
    [...]
  </probMap>
</probu>
```

Listing 5.3 Probability-M XML-Datei

```
<probm>
  <prob>0.99439781</prob>
  <varId>1</varId>
  <varName>NAME</varName>
</probm>
<probm>
  <prob>0.98247962</prob>
  <varId>2</varId>
  <varName>VORNAME</varName>
</probm>
<probm>
  <prob>0.98772576</prob>
  <varId>3</varId>
  <varName>GEBURTSNAME</varName>
</probm>
<probm>
  <prob>0.99949391</prob>
  <varId>4</varId>
  <varName>GEBURTSTAG</varName>
</probm>
<probm>
  <prob>0.9995371</prob>
  <varId>5</varId>
  <varName>GEBURTSMONAT</varName>
</probm>
<probm>
  <prob>0.99971322</prob>
  <varId>6</varId>
  <varName>GEBURTSJAHR</varName>
</probm>
<probm>
  <prob>0.99999304</prob>
  <varId>7</varId>
  <varName>GESCHLECHT</varName>
</probm>
<probm>
  <prob>0.9915867</prob>
  <varId>8</varId>
  <varName>GEMEINDEKENNZIFFER</varName>
</probm>
```

Listing 5.4 Vornamensgruppen XML-Datei

```
<gruppe>
  <vorname>4e5756395c4575e4a2c398393870c2673136</vorname>
  <vorname>553362c64a2c39839387e54a2c323e353e40</vorname>
</gruppe>
<gruppe>
  <vorname>554e5039354bc4a2c3983938723e40287778</vorname>
  <vorname>4f5d44426c6867c4574a2c39839317787a70</vorname>
</gruppe>
<gruppe>
  <vorname>265c3e4c4554e50393543e404434a9839387</vorname>
  <vorname>70622c4575e443523e353e4082f617473136</vorname>
  <vorname>2c2a704b4e40554e50394a2693874e783136</vorname>
  <vorname>24564a2c39839387c4575523e353e4087136</vorname>
  <vorname>342d5cc4575e454e5034a2c3983938744736</vorname>
</gruppe>
<gruppe>
  <vorname>356024a2c3983938745e4435287243535357</vorname>
  <vorname>3a4143c4575e444a2c39839874639878136</vorname>
</gruppe>
[...]
```

Literaturverzeichnis

[HRW+20] S. M. S. Hasan, D. Rivera, X. .-C. Wu, E. B. Durbin, J. B. Christian, G. Tourassi. „Knowledge Graph-Enabled Cancer Data Analytics“. In: *IEEE Journal of Biomedical and Health Informatics* 24.7 (2020), S. 1952–1967. DOI: 10.1109/JBHI.2020.2990797 (zitiert auf S. 17).

[Lan] Landeskrebsregister NRW. *Unser Auftrag*. URL: https://www.landeskrebsregister.nrw/das-lkr-nrw/unser-auftrag/. accessed: 16.03.2021 (zitiert auf S. 17).

[Sch20] R. Schnell. „Record Linkage als zentraler Baustein der Forschung mit Registern und Big Data-Nutzungen“. In: *Qualität bei zusammengeführten Daten: Befragungsdaten, administrative Daten, neue digitale Daten: miteinander besser?* Hrsg. von B. Klumpe, J. Schröder, M. Zwick. Wiesbaden: Springer Fachmedien Wiesbaden, 2020, S. 151. ISBN: 978-3-658-31009-7. DOI: 10.1007/978-3-658-31009-7_11. URL: https://doi.org/10.1007/978-3-658-31009-7_11 (zitiert auf S. 18).

[Sie15] Y. Siegert. „Combining Record Linkage with Machine Learning Methods (Matching cancer records to their patients for epidemiological cancer registries)“. Magisterarb. The address of the publisher: WESTFÄLISCHE WILHELMS-UNIVERSITÄT MÜNSTER, Sep. 2015 (zitiert auf S. 18).

[Wik21a] Wikipedia. *Gehirn — Wikipedia, Die freie Enzyklopädie*. Online; Stand 21. März 2021. 2021. URL: https://de.wikipedia.org/w/index.php?title=Gehirn&oldid=209804079 (zitiert auf S. 19).

[Wik21b] Wikipedia. *Krebs (Medizin) — Wikipedia, Die freie Enzyklopädie*. Online; Stand 6. April 2021. 2021. URL: https://de.wikipedia.org/w/index.php?title=Krebs_(Medizin)&oldid=210472655 (zitiert auf S. 17).

[Wik21c] Wikipedia. *Künstliche Intelligenz — Wikipedia, Die freie Enzyklopädie*. Online; Stand 21. März 2021. 2021. URL: https://de.wikipedia.org/w/index.php?title=K%C3%BCnstliche_Intelligenz&oldid=209931832 (zitiert auf S. 19).

Alle URLs wurden zuletzt am 06.06.2021 geprüft.