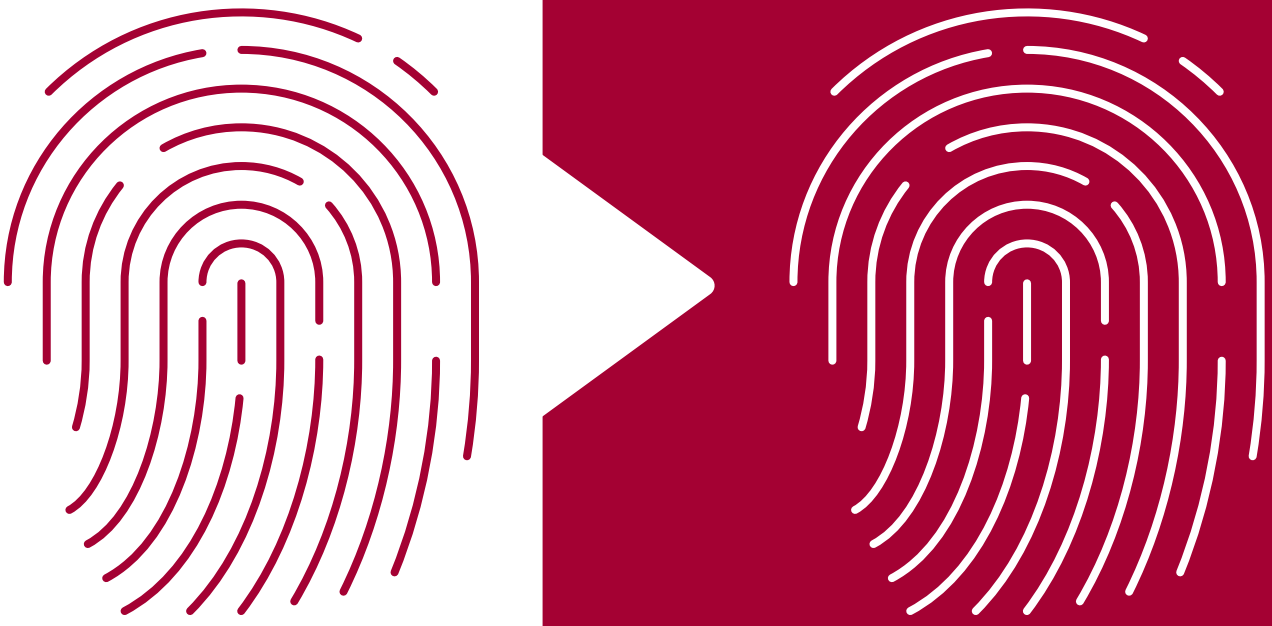
IS IT A MATCH?



Dublettenerkennung in der öffentlichen Verwaltung

| von NEDISLAV NEDYALKOV und DOMINIK MÜLLER

Manchmal erinnert unsere Arbeit an die Aufforderung eines etwas ruppigen Hauptkommissars an seinen Mitarbeiter in einem Vorabendkrimi: „Hier, gleichen Sie mal die Daten der Rentenkassen mit diesen Meldedaten ab!“ Nur landet in unserem Fall kein Aktenstapel auf dem Schreibtisch, sondern der Zugang zu einer äußerst umfangreichen Datenbank – so wie im Projektbeispiel bei der Bundesagentur für Arbeit, das wir in diesem Beitrag vorstellen.

Doch was hat das mit Data *Science* zu tun? Könnten wir nicht einfach jedes mögliche Rentenkassen-Meldedaten-Paar ansehen und prüfen, ob es sich hier um dasselbe Objekt handelt? Leider nicht. Es würde schlichtweg viel zu lange dauern. Im Beispiel der Arbeitsagentur geht es um zehn Millionen Datensätze der Rentenkassen und drei Millionen Datensätze aus Meldedaten. Bei einem vollständigen Abgleich kämen wir auf dreißig Billionen Paarungen. Selbst wenn jeder automatische Abgleich nur eine Millisekunde dauern würde, entspräche das einer Rechenzeit von etwa 950 Jahren – ohne Pause, ohne Ausfälle. So lange möchte niemand warten – weder wir noch die Arbeitsagentur. Gefragt ist also eine weitaus schnellere Lösung. Und sie muss (zumindest fast) so gut sein wie ein vollständiger Abgleich. Damit stehen wir also vor nicht weniger als der Herausforderung, sogenannte *Dubletten* zuverlässig erkennen zu können.

Herausforderung angenommen: Im Folgenden stellen wir eine Lösung für dieses Problem vor, ebenso wie einen Prototyp für die Implementierung. Keine Sorge: Auf mathematische Definitionen und Modellbeschreibungen verzichten wir. Stattdessen liefern wir eine anschauliche und praxisnahe Beschreibung. Für diejenigen Leser, die gerne mehr erfahren möchten, haben wir Schlüsselbegriffe *kursiv* gekennzeichnet und die wichtigsten in Infoboxen näher beschrieben.

PIPELINE-ÜBERBLICK

Zum Zweck der Dublettenerkennung wollen wir jeweils zwei Datensätze miteinander vergleichen und damit möglichst schnell und präzise herausfinden, ob sie dasselbe Objekt in der realen Welt beschreiben.

- Zunächst legen wir fest, welche Eigenschaften *(Features)* der Datensätze wir betrachten wollen. Das heißt, wir müssen eine Auswahl der Eigenschaften *(Feature Selection)* vornehmen, um Rechenzeit zu sparen und unser Ergebnis zu verbessern.
- Danach extrahieren wir Kandidatenpaare für die weitere Analyse. Ein Kandidatenpaar liegt vor, wenn mindestens eine bestimmte Anzahl an Features der beiden Datensätze zueinander passen.
- Als Nächstes trainieren wir einen sogenannten *Klassifikator*, der uns später ermöglicht, die Datenpaare in „Dubletten" und „Nicht-Dubletten" einzuteilen. Dafür benötigen wir Testdaten, die von Hand mit der entsprechenden Kategorie annotiert sind.
- Für die tatsächliche Überprüfung der Kandidatenpaare bilden wir aus jedem Paar einen sogenannten *Ähnlichkeitsvektor*. Er zeigt an, wie ähnlich sich die beiden Datensätze sind.
- Zuletzt analysieren wir diese Vektoren mittels des Klassifikators, um endgültig festzustellen, welche Paare dasselbe reale Objekt verkörpern, also echte Dubletten sind.

Im Folgenden unterziehen wir jeden dieser Teilschritte (siehe Abbildung 1) einer genaueren Betrachtung.

FEATURE SELECTION

In diesem Schritt suchen wir jene Eigenschaften unserer Datensätze heraus, die möglichst viele nützliche Informationen für unsere Zwecke enthalten. Das sind Features, die bei unterschiedlichen Objekten in der realen Welt möglichst häufig unterschiedlich sind. Oft ist hier Expertenwissen oder ein genauer Blick auf die Daten gefragt – es gibt jedoch auch statistische Methoden.

Um den Nutzen von Features messen zu können, ist es hilfreich, einen annotierten Testdatenbestand zu haben. Das heißt, jedem Testdatensatz ist eine Kategorie zugeordnet – etwa „Dublette" oder „keine Dublette". So lässt sich herausfinden, welche Features für unser weiteres Vorgehen interessant sind und welche nicht. Zum Beispiel können wir betrachten, welche Features bei Nicht-Dubletten häufig gleich sind. Features wie Geschlecht oder Staatsangehörigkeit helfen verständlicherweise wenig, da sehr viele Personen dieselbe Ausprägung dieser Merkmale aufweisen. Deutlich seltener hingegen tritt bei unterschiedlichen Personen etwa eine identische Kombination von Name und Vorname auf.

Nachdem wir in diesem Schritt unsere Datensätze auf relevante Features getrimmt haben, sind wir bereit, mögliche Kandidatenpaare zu suchen. Da wir jedoch nicht jede mögliche Kombination einzeln analysieren können, nutzen wir *Locality-sensitive Hashing* (LSH).

LOCALITY-SENSITIVE HASHING

Locality-sensitive Hashing (LSH) benutzt Hashfunktionen, um die Datensätze einzuteilen. Eine Hashfunktion verwandelt große oder lange Eingabewerte in kleinere Zielwerte, mit denen man leichter arbeiten kann. LSH benutzt hierbei Hashfunktionen, die für ähnliche Ausgangswerte wieder ähnliche Zielwerte ergeben.

Auf jedes Feature werden mehrere solcher Hashfunktionen angewendet. Die Ergebnisse werden in sogenannte bins (Eimer) eingeteilt. In jedem bin sind die Werte zueinander ähnlich, ein Wert kann aber auch in mehreren bins vorkommen. Zwei Datensätze werden dann als Kandidatenpaar angesehen, wenn ihre Features sich eine Mindestanzahl an bins teilen. Diese Mindestanzahl ist abhängig von der Gestalt der Daten und muss durch Tests auf annotierten Testdaten bestimmt und optimiert werden.

Abbildung 1: Schematische Darstellung eines möglichen Arbeitsablaufs bei der Dublettenerkennung

KANDIDATENPAARE BESTIMMEN

In diesem Schritt reduzieren wir drastisch die Menge an tatsächlichen Vergleichen, die wir explizit durchführen müssen. Wir versuchen also, aus allen möglichen Paarungen der Daten diejenigen zu finden, die am wahrscheinlichsten eine Dublette bilden.

Das LSH ordnet jeden Datensatz auf Basis seiner Features sogenannten *bins* (Eimern) zu.

Bis zu diesem Punkt haben wir unsere Features ausgewählt und eine stark reduzierte Anzahl an Kandidatenpaaren erzeugt. Diese müssen wir jetzt genauer untersuchen.

KLASSIFIKATOR TRAINIEREN UND ÄHNLICHKEITSVEKTOREN ERMITTELN

In diesem Abschnitt betrachten wir den letzten Schritt – die genaue Analyse der Kandidatenpaare. Dazu bauen wir für jedes Paar zunächst einen Ähnlichkeitsvektor. Er gibt an, wie ähnlich sich die beiden Datensätze sind. Danach füttern wir einen Klassifikator mit unseren Ähnlichkeitsvektoren. So finden wir schlussendlich heraus, bei welchen der Paare es sich tatsächlich um Dubletten handelt.

Die Ähnlichkeit wird einzeln für jedes Feature berechnet und unterscheidet sich je nach Datentyp. So bietet sich etwa für Zahlen der Abstand voneinander als Ähnlichkeitskriterium an und für Features mit eingeschränkter Wertemenge (zum Beispiel vier bis zehn mögliche Ausprägungen beim Familienstand je nach Anwendung) schlicht deren Übereinstimmung. Für Zeichenketten (etwa bei Namen) verwenden wir hingegen die sogenannte *Cosine Similarity*.

Diese Ähnlichkeitsbestimmung führen wir für alle Features unserer Kandidatenpaare durch und bekommen so eine Menge Ähnlichkeitsvektoren. Im letzten Schritt nutzen wir nun diese Vektoren, um herauszufinden, welche Paare wirklich Dubletten darstellen.

KLASSIFIKATION DER KANDIDATENPAARE DURCHFÜHREN

Wir haben jetzt für jedes Kandidatenpaar einen sogenannten „Ähnlichkeitsvektor". Nun werden die Vektoren noch in „gleich" und „nicht gleich" eingeteilt. Dafür bedienen wir uns eines sogenannten „Klassifikators", hier einer *Support Vector Machine* (SVM).

Ein Klassifikator ist ein Konzept, mit dem man Daten in Kategorien einteilen, also klassifizieren kann. Das Vorgehen unterteilt sich hierbei allgemein in zwei Schritte: Training und Anwendung.

- Beim **Training** eines Klassifikators werden die Parameter festgelegt, anhand deren der Klassifikator später die Klassenzugehörigkeit der Datensätze bestimmt. Dies geschieht üblicherweise auf Basis von Trainingsdaten, also bereits vorher annotierter Daten, für die die korrekte Klassenzugehörigkeit bekannt ist.

COSINE SIMILARITY

Die Cosine Similarity (Kosinus-Ähnlichkeit) vergleicht zwei Vektoren basierend auf ihrem Winkel zueinander. Um unsere Wörter aber in Vektorform zu bringen, zerlegen wir sie zunächst in 3-Gramme. Das sind Teile eines Worts (umliegende Leerzeichen oder Ähnliches eingeschlossen), die drei Zeichen lang sind. Diese Zerlegungen werden in Vektorform gebracht und dann verglichen, um eine Ähnlichkeit der Wörter zu ermitteln.

Das bedeutet, wir konstruieren aus allen möglichen 3-Grammen in den Wörtern unseres Features eine einzelne Menge, quasi eine sehr lange Liste. Diese Liste eines Wortes ist genau so lang, wie das Wort 3-Gramme enthält, besteht aber nur aus „0" und „1". Eine „0" steht an einer Stelle, wo das entsprechende 3-Gramm nicht im Wort auftaucht. Wenn das 3-Gramm Teil des Wortes ist, so steht an dieser Stelle eine „1".

Diese Vektorrepräsentationen werden nun anhand des Winkels zwischen den beiden Vektoren verglichen.[1]

Vergleichen wir zum Beispiel die Wörter „Ende" und „Ente". Die 3-Gramme der beiden Wörter werden in Tabelle 1 gezeigt. Nun transformieren wir diese in Vektoren. Die Grundmenge der Einträge ist die Anzahl aller unterschiedlichen 3-Gramme, also genau die Menge M = {"_En", "End", "nde", "de_", "Ent", "nte", "te_"}. Die Vektorrepräsentationen sind ebenfalls in Tabelle 1 eingetragen.

Wort	3-Gramme	Vektor auf {"_En", "End", "nde", "de_", "Ent", "nte", "te_"}
Ende	"_En", "End", "nde", "de_"	(1, 1, 1, 1, 0, 0, 0)
Ente	"_En", "Ent", "nte", "te_"	(1, 0, 0, 0, 1, 1, 1)

Tabelle 1: Aufteilung von Wörtern in 3-Gramme und Vektorrepräsentation

SUPPORT VECTOR MACHINE

Eine Support Vector Machine (SVM) unterteilt Datensätze in zwei Kategorien. Dies geschieht mittels einer sogenannten „Hyperebene", die den gesamten Wertebereich der Datensätze aufspannt. Datensätze, die auf der einen Seite der Hyperebene liegen, bekommen dann die Klasse A (in diesem Fall Dubletten) zugewiesen. Die Datensätze auf der anderen Seite der Hyperebene sind Teil der Klasse B (in diesem Fall Nicht-Dubletten).

Die Lage der Hyperebene wird durch die Trainingsdaten bestimmt. Aus der Postionierung der annotierten Trainingsdaten ergibt sich eine optimale Hyperebene. Optimal heißt: Die Klassifikation der Testdaten in die Klassen A und B ist möglichst korrekt (das heißt, die Testdaten liegen auf der richtigen Seite der Hypereben), und sie ist möglichst eindeutig (das heißt, der Abstand zwischen den beiden Klassen A und B ist möglichst groß).

In der trainierten Anwendung werden neue Datensätze dann einfach durch Positionierung im Hyperraum auf Basis ihrer Kriterien den Klassen A (Dubletten) oder B (Nicht-Dubletten) zugeordnet.

- Bei der **Anwendung** wird eine Funktion auf die Datensätze angewendet und so entschieden, welcher Klasse jeder Datensatz angehört.

In unserem Fall möchten wir unsere Ähnlichkeitsvektoren danach einteilen, ob sie eine Dublette darstellen oder nicht. Unser Klassifikator soll bei der Anwendung also genau das entscheiden.

BEISPIELHAFTE IMPLEMENTIERUNG

Bis hierher haben wir ein Vorgehen zur Dublettenerkennung beschrieben, das so auch beim Projekt der Bundesagentur für Arbeit zum Einsatz kam. Im Folgenden stellen wir eine prototypische Implementierung vor.

Die Dublettenerkennung bei Personendaten aus unterschiedlichen Datenbanken im Behördenumfeld ist ein konkretes und regelmäßig wiederkehrendes Matching-Problem. Deshalb haben wir auf dieser Grundlage einen Proof-of-Concept erstellt – also einen Software-Prototyp, mit dem die prinzipielle Durchführbarkeit des KI-basierten Vorgehens belegt werden kann.[2]

PROGRAMMIERSPRACHE UND BIBLIOTHEKEN

Bei unserer prototypischen Implementierung haben wir auf die in KI-Kreisen gängige Programmiersprache *python* gesetzt. Neben python selbst haben wir uns zusätzlich die freie Software-Bibliothek scikit-learn und das Python Record Linkage Toolkit (recordlinkage) zum maschinellen Lernen für die Programmiersprache python zunutze gemacht.

DIE DATEN

Jedes KI-Projekt braucht eine große Menge Daten. Nur mit ihnen können die Algorithmen lernen, und nur mit ihnen können wir im Anschluss die Güte des antrainierten Algorithmus bestimmen.
Im konkreten Fall haben wir Daten aus der Bibliothek *Febrl* (Freely Extensible Biomedical Record Linkage) genutzt. Dabei handelt es sich um Personendaten, die für KI-Zwecke frei verfügbar sind und beispielsweise Merkmale wie Vor- und Nachname aufweisen.

SOFTWARE-PROTOTYP

Basierend auf den bisher besprochenen theoretischen Grundlagen und Software-Tools haben wir die Implementierung in vier Schritten durchgeführt (siehe Abbildung 2).

Schritt 1 – Auswahl der Kandidatenpaare
Zuerst haben wir die zu vergleichenden Paare von Objekten aus der Menge aller möglichen Paare gesammelt. Für die Bestimmung dieser Dubletten-Kandidatenpaare haben wir das sogenannte Index-Konstrukt aus der Bibliothek recordlinkage ausgewählt. Dieses ermöglicht es vergleichsweise schnell, Paare zu identifizieren, die mit einer recht hohen Wahrscheinlichkeit Dubletten sein könnten. Somit haben wir die Anzahl der zu vergleichenden Paare erheblich reduziert, um die Menge an zu vergleichenden Objekten mit vertretbarem Aufwand bewältigen zu können. Im vorliegenden Fall konnten wir auf das Konzept der Feature Selection verzichten, weil die vorliegenden Trainingsdaten eine überschaubare Anzahl an Merkmalen (Features) aufweisen und eine weitere Auswahl somit nicht notwendig ist.

Schritt 2 – Vergleichsregeln definieren
Im zweiten Schritt werden für die vorher bestimmten Dubletten-Kandidatenpaare mit aufwendigeren und viel präziseren Zeichenkettenvergleichen Vergleichsregeln definiert. Beispielsweise werden die Vor- und Nachnamen auf Ähnlichkeit geprüft. Für diesen Schritt bietet sich unter anderem das Verfahren aus dem Abschnitt „Kandidatenpaare bestimmen"an.

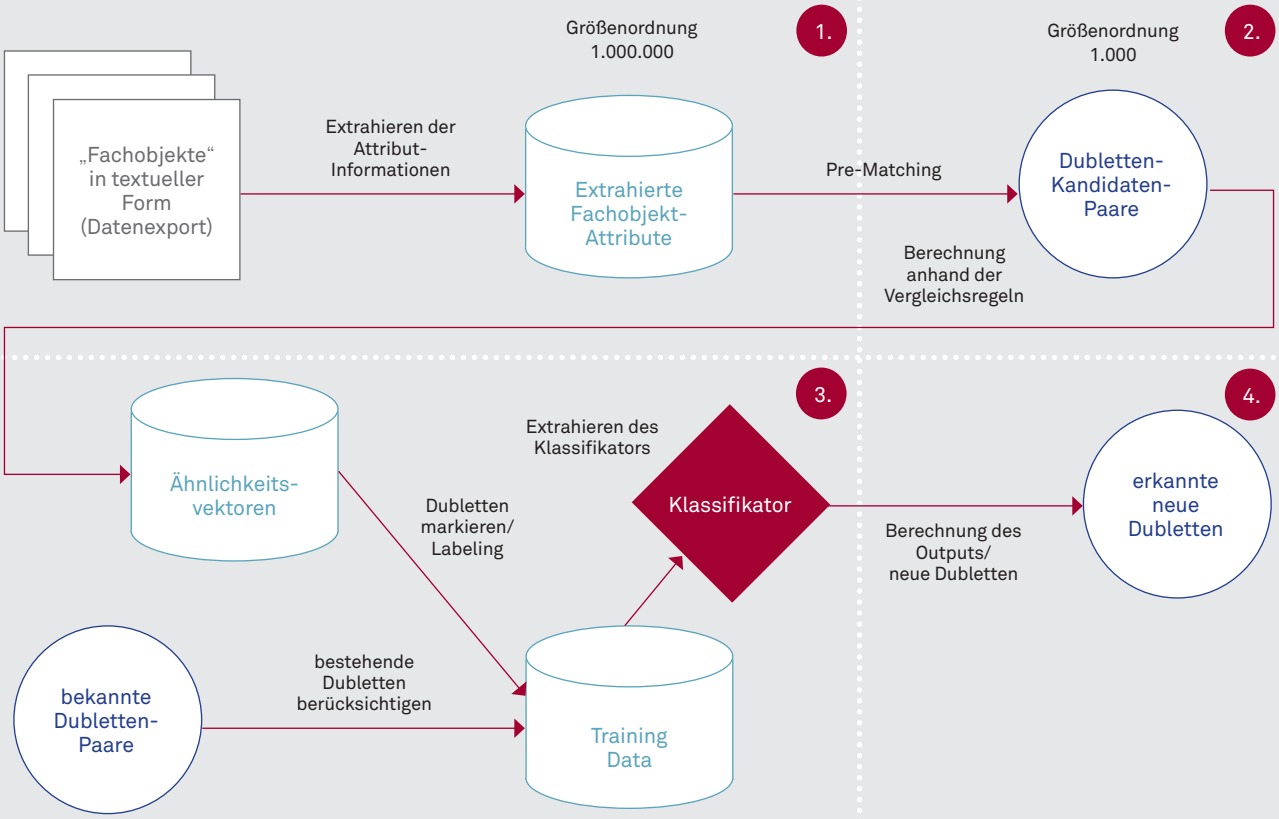


Abbildung 2: Der Prozess der Dublettenerkennung durch KI

Mit den einzelnen Ergebnissen dieser Vergleiche werden die sogenannten Ähnlichkeitsvektoren aufgestellt, die dann der Input für den nächsten Schritt sind.

Schritt 3 – Wahl des Klassifikators und Lernen

Für den Kern der Lösung wählen wir einen Klassifikator und trainieren ihn anhand der Trainingsdaten. Es wird das sogenannte überwachte Lernen angewendet, bei dem bereits klassifizierte Daten die Grundlage bilden. Für unseren Algorithmus haben wir die *Logistic Regression* ausgewählt, ein etabliertes mathematisches Verfahren in der Statistik. Mit dieser Methode kann die Wahrscheinlichkeit bestimmt werden, mit der Fragen etwa mit *Ja* oder *Nein*, *Gewinn* oder *Verlust, Gesund* oder *Krank* beantwortet werden. Die Parameter unseres mathematischen Models wurden anhand bekannter Daten bestimmt.

Schritt 4 – Dublettenerkennung anwenden

Erst jetzt, nach dem Trainieren der KI, können wir die gelernten Verfahren auf neue, unbekannte Daten anwenden, um die Datenpaare korrekt in *Dubletten* oder *Nicht-Dubletten* zu unterteilen. In diesem Schritt wenden wir deshalb die antrainierte Logistic Regression auf neue Datensätze an. Dabei sind wir in zwei Durchläufen folgendermaßen vorgegangen:

Im ersten Durchlauf haben wir die Logistic Regression mit dem in Schritt 3 antrainierten Parametern durchgeführt. Gefunden wurden 490 Dubletten bei tatsächlich 500 vorhandenen, was ein sehr gutes Ergebnis ist.

Um eine mögliche Veränderung der Ergebnisse zu analysieren, haben wir in einem zweiten Durchgang die Parameter

Anzahl Testdatensätze	1.000
Alle möglichen Paare	ca. 1.000.000
Anzahl Dubletten-Kandidatenpaare nach der Bestimmung der Dubletten-Kandidatenpaare (Schritt 1)	3.636
Anzahl als Dubletten erkannte Kandidaten (Schritt 4)	490 (Durchlauf 1) 415 (Durchlauf 2)
Tatsächliche Dubletten	500

Tabelle 2: Ergebnisse der Dublettenprüfung des Prototyps

für die Bestimmung der Dubletten-Kandidatenpaare geändert. Das Ergebnis: Ein Wegfall des Vergleichs der Sozialversicherungsnummer führt zu einer Verschlechterung der Ergebnisse (siehe Tabelle 2). Hier finden wir nur noch 415 Dubletten bei 500 tatsächlich vorhandenen. Dementsprechend haben wir nun zusätzlich belegt, dass die Sozialversicherungsnummer eine zentrale Bedeutung für die Qualität der Lösung hat.

Durch diese schrittwese Verfeinerung können die Ergebnisse besser verstanden und optimiert werden, bis die Ergebnisse für die reale Nutzung geeignet sind.

FAZIT

In diesem Artikel haben wir das reale und regelmäßig wiederkehrende Problem der Dublettenerkennung vorgestellt, Lösungsansätze skizziert und einen Proof-of-Concept beschrieben, der mit realen Testdaten funktioniert. Dies sind wichtige Grundlagen für die Orientierung und den Einstieg in den Themenkomplex.

In der Wirklichkeit existieren zusätzliche Hürden. Eine davon ist die Qualität der zugrunde liegenden Daten: Datenbestände werden zumeist von Menschen gepflegt, und Menschen machen Fehler. Zum Beispiel Tippfehler oder unterschiedliche Konvertierung von Sonderzeichen. Im Projekt der Bundesagentur für Arbeit war zum Beispiel nicht immer ersichtlich, welches Feld eines Datensatzes den Vornamen und welches den Nachnamen enthielt. Die gute Nachricht: Auch dafür gibt es Lösungen. Im konkreten Fall wurden beide Features als eine einzige Zeichenkette betrachtet.

Das Beispiel zeigt, wie die Arbeit von Data Scientists, Data Analysts und KI-Experten für die öffentliche Verwaltung immer wichtiger wird. Denn die öffentliche Verwaltung steht in nahezu allen Bereichen vor demselben Problem: Schnell wachsende Datenbestände machen eine manuelle Verarbeitung schon heute so gut wie unmöglich. Die Lösung für solche Probleme kann bereits gut durch automatisierte KI-Verfahren unterstützt werden. ●

1 Für die genaue Formel verweisen wir auf https://arxiv.org/pdf/1910.09129.pdf (abgerufen am 04.02.2021).

2 Der Prototyp basiert auf den konzeptionellen und technologischen Erkenntnissen und Lösungen aus:
Shu Rong, Xing Niu, Evan Wei Xiang, Haofen Wang, Qiang Yang and Yong Yu, A Machine Learning Approach for Instance Matching Based on Similarity Metrics (https://link.springer.com/content/pdf/10.1007%2F978-3-642-35176-1_29.pdf, abgerufen am 30.03.2021)
sowie Bilenko et al., Learnable Similarity Functions and their application to data linkage and clustering (https://www.cs.utexas.edu/~ml/papers/marlin-aaaidc-04.pdf, abgerufen am 30.03.2021).
Für die technologische Auswahl siehe: https://www.python.org (abgerufen am 01.02.2021),
https://sourceforge.net/projects/febrl/ (abgerufen am 01.02.2021),
http://recordlinkage.readthedocs.io/en/latest/ref-datasets.html (abgerufen am 01.02.2021),
https://scikit-learn.org/stable/index.html (abgerufen am 01.02.2021),
http://recordlinkage.readthedocs.io/en/latest/notebooks/data_deduplication.html (abgerufen am 01.02.2021).
https://scikit-learn.org/stable/ (abgerufen am 29.03.2021).