Analyse, Design, Entwicklung und Evaluation eines selbstkonfigurierenden Entity Resolution Frameworks für Streaming-Daten

---

Kevin Sapper

17.02.2017

Version: 0.1

Hochschule RheinMain

DCSM - Design Informatik Medien
Informatik (M.Sc.)

Masterarbeit

Analyse, Design, Entwicklung und Evaluation eines selbstkonfigurierenden Entity Resolution Frameworks für Streaming-Daten

Kevin Sapper

Referent Prof. Dr. Adrian Ulges
Hochschule RheinMain
DCSM - Design Informatik Medien

Koreferent Prof. Dr. Reinhold Kröger
Hochschule RheinMain
DCSM - Design Informatik Medien

Betreuer Thomas Strauß
Universum Group

17.02.2017

Kevin Sapper

Analyse, Design, Entwicklung und Evaluation eines selbstkonfigurierenden Entity Resolution Frameworks für Streaming-Daten

Masterarbeit, 17.02.2017

Referenten: Prof. Dr. Adrian Ulges und Prof. Dr. Reinhold Kröger

Betreuer: Thomas Strauß

Hochschule RheinMain

Informatik (M.Sc.)

DCSM - Design Informatik Medien

Kurt-Schumacher-Ring 18

65197 Wiesbaden

1 Einleitung

[TODO Einleitung schreiben! Aktueller Text nur Lorem Ipsum]

Im Rahmen der Thesis soll ein Entity Resolution Framework für Datensatzströme entstehen. Als Basis soll ein (Event) Stream Processing Framework genutzt werden. Das Framework soll eine Reihe von Matchern, sowie Kombinationsfunktionen der Matcher unterstützen. Hauptaugenmerk ist jedoch die Skalierbarkeit. Gelöst werden soll das Data Skew Problem bei verschiedenen Blocking Strategien. Eine weitere Schwierigkeit ist, dass die Datenmenge nicht statisch ist, sondern neue Datensätze jederzeit hinzukommen können. Beim Erweitern des Suchraums soll beachtet werden, dass kein Data Skew auftritt. Dadurch soll vermieden werden, dass der Durchsatz innerhalb des Clusters signifikant sinkt. Idealerweise soll der Durchsatz, sowie die Qualität der Suchergebnisse, mit bereits bekannten Veröffentlichungen verglichen werden. Das Framework soll dabei kein Domainwissen eines bestimmten Entitätstypen berücksichtigen.

2 Grundlagen

2.1 Entity Resolution

Die Methoden zur Duplikatserkennung stammen ursprünglich aus dem Gesundheitsbereich und wurden erstmal 1969 von Felegi & Sunter [1] formal formuliert. Je nach Fachgebiet gibt es unterschiedliche Fachbegriffe. Statistiker und Epidemiologen sprechen von *record* oder *data linkage*, während Informatiker das Problem unter *entity resolution*, *data* oder *field matching*, *duplicate detection*, *object identification* oder *merge/purge* kennen. Identifiziert werden sollen dabei beliebige Entitäten, welche oft in Form von Datensätzen einer Datenbank vorliegen. Die Schwierigkeit dabei ist allerdings, dass Entitäten nicht durch ein einzigartiges Attribut identifiziert werden können, beispielsweise Produkte, bibliografische Einträge oder selbsterfasste Onlineauskünfte. Zudem sind die Datensätze oft fehlerhaft, beispielsweise durch Rechtschreibfehler, welche durch Tippfehler, Hörfehler oder OCR-Fehler enstehen. Eine andere Fehlerquelle sind unterschiedliche Konventionen, beispielsweise bei Endungen von Strassennamen `strasse`, `straße` oder `str`. Auch denkbar sind Fehler aufgrund von Betrug. Die Methoden zur Entity Resolution (ER) vergleichen meist eine oder mehrere Datenbanken, indem Datensatzpaare gebildet werden. Als Ergebnis wird eine Menge von übereinstimmenden Datensatzpaaren, d.h. zwei Datensätze, welche die selbe Entität beschreiben, geliefert. Damit eine Übereinstimmung zwischen zwei oder mehr Entitäten festgestellt werden kann, müssen diese verglichen werden und ein Ähnlichkeitswert (engl. similarity score) bestimmt werden. Dieser Ähnlichkeitswert gibt die Intensität der Übereinstimmung an. Das ER Problem wird Formal von Köpcke & Rahm in [2] folgendermaßen beschrieben. Gegeben sind zwei Mengen von Entitäten $A \in S_A$ und $B \in S_B$ zweier Datenquellen S_A und S_B, welche semantisch dem selben Entitätstypen entsprechen. Das ER Problem ist es, alle Übereinstimmungen in $A \times B$ zu finden, welche derselben Entität entsprechen. Als Spezialfall ist dabei die Suche in einer Datenquelle $A = B, S_A = S_B$ zu betrachten. Eine Übereinstimmung $u = (e_i, e_j, s)$ verknüpft zwei Entitäten $e_i \in S_A$ und $e_j \in S_B$ mit einem Ähnlichkeitwertt $s \in [0, 1]$.

In der klassischen Variante arbeitet Entity Resolution auf statischen Daten, d.h. das während des ER-Prozesses keine neuen Daten hinzukommen. Hierbei werden die zwei Disziplinen Deduplizierung und Entity-Linking unterschieden. Die Deduplizierung wird auf einer Datenquelle durchgeführt und hat den Zweck alle Duplikate in dieser Datenquelle zu finden. Anschließend werden die gefundenen Duplikate automatisch oder manuell

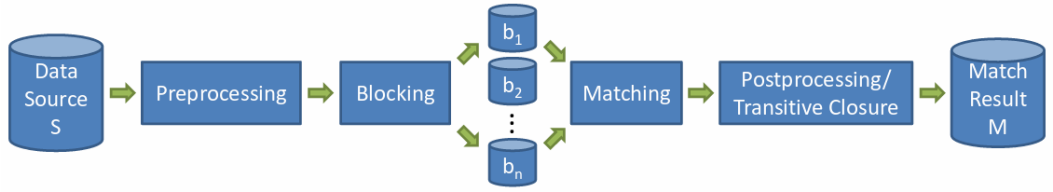

Abbildung 2.1 Vereinfachter Entity Resolution Workflow aus [3]. Die Datenquelle S wird vorverarbeitet und in kleinere Submengen gegliedert. Innerhalb dieser werden Datensatzpaare miteinander verglichen und paarweise bestimmt, ob diese der selben Entität entsprechen. Abschließend werden aus Paaren Gruppen von Duplikaten ermittelt und als Ergenisse M geliefert.

zusammengeführt. Entity Linking hingegen wird auf mindestens zwei verschiedenen Datenquellen durchgeführt. Das Ziel ist es, nicht Duplikate zusammenzuführen, sondern Entitäten zwischen den Datenquellen zu verlinken. Damit die Links eindeutig sind, wird vorausgesetzt, dass die einzelnen Datenquellen dedupliziert sind.

Die Ausführung der Vergleichsmethoden ist enorm teuer, da diese das Kreuzprodukt zweier Mengen bilden müssen. Dies führt zu einer quadratischen Komplexität bei einem Vollvergleich, welcher dafür sorgt, dass bei großen Datenmengen die Ausführungszeit unakzeptabel lang wird. Um die Ausführungszeit zu reduzieren wird versucht den Suchraum auf die wahrscheinlichsten Duplikatsvorkommen zu begrenzen. Diese Vorgehen werden als Blocking oder Indexing bezeichnet.

Abbildung 2.1 zeigt einen vereinfachten typischen Entity Resolution Workflow. Zunächst werden die Datensätze einer Datenquelle S vorverarbeitet, um typische Fehler zu entfernen. Dazu gehört das Korrigieren von Rechtschreibfehler, ignorieren von Groß- bzw. Kleinschreibung, beispielsweise durch Konvertierung in Kleinschreibung, und das Ersetzten von bekannten Abkürzungen. Durch die Vorverarbeitung kann die Qualität des Matchings verbessert, indem verhindert wird, dass offensichtliche Abweichungen den Ähnlichkeitswert beeinflussen. Der nächste Schritt, das Blocking, teilt die Gesamtmenge in Submengen $b_1, b_2, \cdots, b_n$ zur Reduzierung der Gesamtkomplexität, da nur die jeweiligen Blöcke voll verglichen werden. In Abschnitt 2.2 werden detailiert verschiedene Blockingverfahren erläutert. Auf das Blocking folgt das Matching, hierbei werden innerhalb der Submengen von Datensatzpaaren Ähnlichkeitswerte bestimmt. Die Möglichkeiten der Ähnlichkeitsbestimmung werden in Abschnitt 2.3 beschrieben. Anhand der Ähnlichkeitswerte wird anschließend für jedes Datensatzpaar entschieden, ob es sich um ein Match, beide Datensatze beschreiben dieselbe Entität, oder ein Non-Match, die Datensätze beschreiben unterschiedliche Entitäten, handelt. Diese Klassifikation wird genauer in Abschnitt 2.4 erklärt. Abschließend findet noch die Berechnung der transitiven Hülle statt, um beispielsweise aus Paaren von Matches Gruppen zu bilden, welche derselben Entität entsprechen $M = (a, b), (b, c) \implies M = (a, b, c)$.

Laut Köpcke & Rahm [2] gibt es keine Methode zur Entity Resolution, welche allen anderen überlegen ist. Vielmehr ist der Erfolg unterschiedlicher Methoden domänabhangig. Deshalb wurde Anfange der 00er Jahre begonnen Frameworks zu entwickeln, welche verschiedene Methoden miteinander kombinieren. Einen Vergleich dieser Frameworks wurde durch Köpcke & Rahm [2] durchgeführt. Ein Framework besteht hierbei aus verschiedenen Matchern. Ein Matcher ist dabei ein Algorithmus, welcher die Ähnlichkeit zweier Datensätze ermittelt. Köpcke & Rahm unterscheiden zwischen attributs- und kontextbasierenden Matchern. Als Kontext bezeichnen Sie die semantische Beziehung bzw. Hierarchie zwischen den Attributen, beispielsweise in Graphstrukturen, welche es erlauben Ähnlichkeitswerte über Kanten zu propagieren. Um die Matcher miteinander zu kombinieren nutzen die Frameworks mindestens eine Matching Strategie. Durch die Match-Strategie werden verglichene Datensatzepaare in die Mengen Matches und Non-Matches klassifiziert.

Ein Großteil der Forschung in Entity Resolution konzentriert sich auf die Qualität der Vergleichsergebnisse. Die von Köpcke & Rahm verglichenen Frameworks konzentrieren sich alle darauf zwei statische Mengen zu miteinander vergleichen. Bei großen Datenmengen kann dies durchaus mehrere Stunden dauern. Daher gibt es in den letzten Jahre einige Ansätze und Frameworks, welche MapReduce Algorithmen zum Skalieren nutzen [4]. Einen Ansatz die Laufzeit für Anwendungen mit Laufzeitanforderungen zu optimieren präsentieren Whang et al. [6]. Anstatt eine Übereinstimmungsmenge nach Abschluss eines Algorithmus zu liefern, zeigen Sie Möglichkeiten partielle Ergebnisse während der Laufzeit des Algorithmus zu erhalten. Dabei modifizieren Sie die Blockingalgorithmen so, dass zunächst die wahrscheinlichsten Kandidaten miteinander verglichen werden. Dabei wird in relativ kurzer Zeit ein Großteil der Duplikate gefunden.

Neben den statischen Verfahren gibt es zunehmend Bedarf an dynamischen Verfahren. Dynamisch bedeutet hier, dass während der Laufzeit neue Datensätze hinzugefügt werden können. Das Finden gleicher Enitäten erfolgt dabei auf Anfrage, weshalb die gesamte Datenmenge vorab nicht bekannt ist. Beispielsweise müssen Kreditauskunfteien auf Anfrage prüfen, ob ein Kunde kreditwürdig ist. Dazu müssen die passenden Entitäten möglichst schnell gefunden werden, um eine Entscheidung treffen zu können. Zudem ist es notwendig eine Historie der unveränderten Anfragen aller Entität vorzuhalten, da diese Beweise über frühere Anfragen liefern. Ramadan et al. [7] formulieren die Problemstellung für dynamische ER-Verfahren folgendermaßen. Für jeden Anfragedatensatz q_j eines Anfragestroms Q sollen alle Datensätze M_{q_j} in R gefunden werden, welche dieselbe Entität wie q_j beschreiben.

$$M_{q_j} = \{r_i | r_i.eid = q_j.eid, r_i \in R\}, M_{q_j} \subseteq R, q_j \in Q,$$

{#eq:dyer} wobei eid ein eindeutiger Identitiver einer Entität ist, welcher so nicht existiert. Die Herausforderung für dynamische ER-Verfahren ist weiter nach Ramadan et al.

Indexing-Verfahren zu entwickeln, welche es erlauben den Index dynamische zu erweitern und eine kleine Zahl qualitativer Ergebnisse in nahe Echtzeit (Subsekundenbereich) zu liefern. Ein dynamisches ER System ist ähnlich einer Suchmaschine, doch anstatt einer gewerteten Liste möglicher Treffer, soll es alle gleichen Entitäten finden, welche zur Anfrage passen. Das bedeutet insbesondere, dass die Anfrage die gleiche Datenstruktur haben muss, wie die zu durchsuchende Datenquelle. Zudem kann eine Anfrage während der Abfrage als neuer Datensatz aufgenommen werden. Erste Ergebnisse Entity Resolution in nahe Echtzeit zu erreichen, präsentieren Christen & Gayler in [8], unter Verwendung von Inverted Indexing Techniken, welche normalerweise bei der Websuche Anwendung finden. Die dynamischen Verfahren werden in Abschnitt 2.2.2 behandelt.

2.2 Blocking

Blocking dient der Reduzierung der quadratischen Komplexität eines ER Verfahrens. Im Folgenden werden Verfahren unterschieden, die entwickelt wurden, um auf statischen oder dynamischen Datenquellen angewandt zu werden.

2.2.1 Statisches Blocking

Für die Duplikatserkennung in zwei Datenquellen A und B sind $|A| \cdot |B|$ Paarvergleiche notwendig. Bei einer einzelnen Datenquelle A müssen $\frac{1}{2} \cdot |A| \cdot (|A| - 1)$ Vergleiche durchgeführt werden. In beiden Fällen ist die Anzahl der Vergleiche quadratisch zur Eingabemenge [3]. In der Studie [9] zeigen Köpcke et al., dass das kartesische Produkt für große Datenmengen nicht skaliert. Aus diesem Grund reduzieren moderne Entity Resolution Frameworks den Suchraum auf die wahrscheinlichsten Kandidaten, die sogenannten Match-Kandidaten. Diese Methoden zur Reduzierung des quadratischen Suchraum werden übergreifend als Blockingmethoden bezeichnet. Neben Blocking werden auch Windowing- und Indexing Verfahren eingesetzt. Während Blockingverfahren die Anzahl der notwendigen Vergleiche drastisch reduzieren, indem Non-Matches ausgeschlossen werden, besteht dennoch die Gefahr, dass fälschlicherweise tatsächliche Matches ausgefiltert werden. Daher ist es notwendig die Güte des Blockingverfahrens zu bestimmen. Dazu werden zwei Kennziffern erhoben. Zum einen die *Reduction Ratio*, welche die Reduzierung der Vergleiche im Gegensatz zum Kartesischen Produkt ausdrückt, sowie die *Pairs Completeness*, welche den Anteil der tatsächlich ausgewählten Duplikate, die sich nach dem Blocking in der Kandidatenmenge befinden, beschreibt. Eine detaillierte Beschreibung der Komplexitätsmaße für Blocking wird in Abschnitt 2.5 vorgenommen.

Prinzipiell erfolgt Blocking entweder durch Gruppierung oder Sortierung. Dadurch sollen sich mögliche Duplikate in der "Nähe" voneinander einfinden. Zur Durchführung der Gruppierung oder Sortierung müssen sog. Block- bzw. Sortierschlüssel für jeden

Datensatz erzeugt werden. Diese Schlüssel werden von den Attributswerten oder einem Teil der Attributswerte abgeleitet und stellen eine Signatur des Datensatzes dar. Eine beliebte Variante für Schlüssel sind etwa phoenitische Enkodierung.

Standard Blocking

Standard Blocking ist eine der ersten und populärsten Blockingmethoden [1]. Die Idee des Verfahrens ist eine Menge von Datensätzen in disjunkte Partitionen (genannt Blöcke) zu teilen. Anschließend werden nur die Datensätze des jeweiligen Blocks miteinander verglichen. Dazu wird jedem Datensatz ein Blockschlüssel zugeordnet. Die Qualität des Blockingverfahrens hängt daher maßgeblich vom gewählten Blockschlüssel ab, da dieser die Anzahl und Größe der Partitionen bestimmt. In einer Menge von Personen ist ein schlechter Blockschlüssel etwa das Geschlecht. Da dieser die Menge lediglich in zwei große Partitionen teilt. Ein besserer Blockschlüssel ist beispielsweise die Postleitzahl oder die ersten Ziffern der Postleitzahl [10]. Abbildung 2.2 zeigt die Ausführung des Blockingverfahrens beispielhaft an einer Datenquelle S. Zunächst wird jedem Datensatz (a - i) ein Blockschlüssel (hier 1, 2, 3) zugeordnet. Anschließend wird anhand dieses Schlüssels gruppiert. Die Größe der einzelnen Blöcke bestimmt die Reduktion Ratio. Diese hängt allerdings immer von der Datenquelle ab und kann daher nicht pauschalisiert werden. Bei der Generierung der Blockschlüssel können fehlerhafte Werte einzelner Attribute dazu führen, dass Duplikate in unterschiedlichen Blöcken landen. Damit diese Duplikate dennoch gefunden werden, kann für jeden Datensatz mehrere Blockschlüssel, anhand unterschiedlicher Attribute, generiert werden. Dieser Ansatz nennt sich Multi-pass Blocking [3]. Im Folgenden werden mit Q-gram Indexing und Suffix Array Indexing zwei Verfahren diskutiert, die ein unscharfes Matching der Schlüssel erlauben und dadurch etwa Tippfehler auflösen können.

Q-gram Indexing

Das Q-gram Indexing basiert auf der Idee Datensätze unterschiedlicher aber ähnlicher Blockschlüssel miteinander zu vergleichen. Ein Blockschlüssel wird dazu in eine Liste G von q-Grammen überführt. Ein q-Gram ist ein Substring der Länge q des ursprünglichen Blockschlüssels. Beispielsweise erzeugt $q = 2$ angewendet auf den Blockschlüssel **banana** die Liste $G = ba, an, na$. Alle Kombinationen der q-Gram Liste mit einer Mindestlänge $l = max(1, \lfloor \#G \cdot t \rfloor)$ werden konkateniert und dienen als Schlüssel der Blöcke, wobei t ein Schwellwert zwischen 0 und 1 ist. Für $t = 0.9$ werden die Sublisten $(ba, an), (ba, na), (an, na), (ba, an, na)$ der Längen 2 und 3 gebildet. Dabei werden Datensätze mehreren Blöcken zugewiesen. Dieses Verfahren kann als Alternative zum Multi-pass Verfahren beim Standard Blocking genutzt werden. Ist $t = 1$ wird lediglich ein Blockschlüssel erzeugt, was dem Standard Blocking entspricht. Der große Nachteil ist der hohe Aufwand bei der Berechnung aller möglichen Sublisten. Ein Blockschlüssel

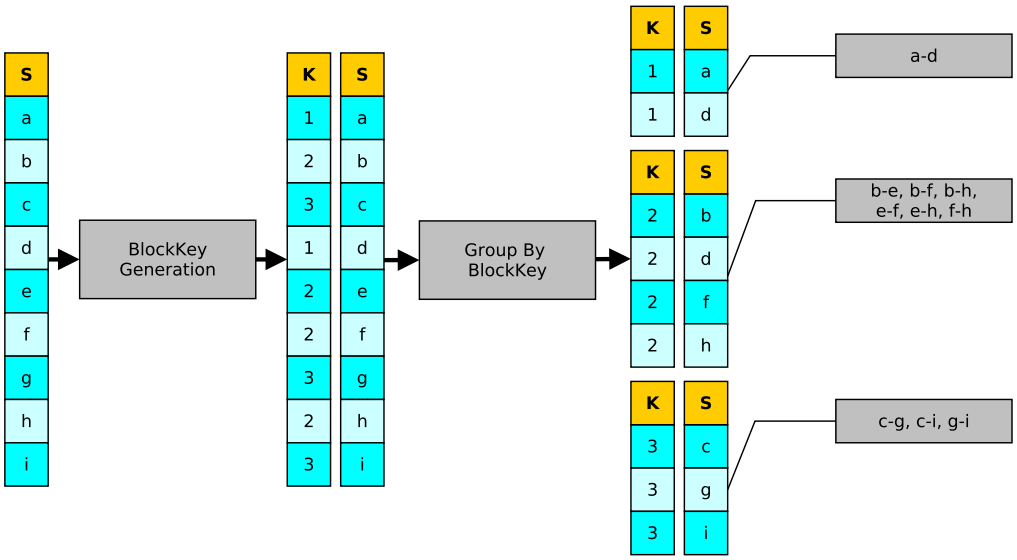

Abbildung 2.2 Beispielhafte Standard Blocking Ausführung nach [3]. Für jeden Datensatz in S wird ein Blockschlüssel K erzeugt. Anhand dessen werden Blöcke erzeugt und innerhalb der Blöcke werden Paare gebildet.

mit n Zeichen muss in $k = n - q + 1$ q-Gramme zerlegt werden. Insgesamt müssen dadurch $\sum_{i=max\{1,\lfloor k \cdot t \rfloor\}}^{k} \binom{k}{i}$ Sublisten berechnet werden [11].

Suffix Array Indexing

Das Suffix Array Indexing [12] leitet, ähnlich wie Q-gram Indexing, mehrere Schlüssel aus einem Blockschlüssel ab. Grundidee ist es alle Suffixe mit einer Mindestlänge von l zu bestimmen. Ein Datensatz mit Blockschlüssellänge n wird in $n-l+1$ Blöcke eingeordnet. Ist $n < l$ wird der Ausgangsschlüssel als einziger Schlüssel verwendet. Durch die größere Menge an Kandidatenpaaren ist i.Allg. die *Pair Completeness* höher (vgl. Multi-pass). Zudem ist der Aufwand der Berechnung der Schlüssel im Gegensatz zu Q-grammen deutlich geringer. Im Gegensatz zum Standard Blocking ist die Menge an Kandidatenpaaren jedoch deutlich höher. Dadurch ist auch die Wahrscheinlichkeit, dass zwei Datensätze unnötigerweise mehrfach miteinander verglichen werden hoch. Deshalb werden aus Blöcken, welche einen bestimmten Schwellwert überschreiten alle Datensätze entfernt, die min. einen weiteren längeren Blockschüssel haben.

Sorted Neighborhood

Das Sorted Neighborhood Verfahren, ist ein Sortierverfahren, welches 1995 von Hernández & Stolfo [13] zur Erkennung von Duplikaten in Datenbanktabellen vorgestellt wurde. Es besteht aus drei Phasen. Zunächst bekommt jeder Datensatz einen Sortierschlüssel

zugewiesen. Dabei muss der Sortierschlüssel nicht einzigartig sein. Um die Berechnung des Schlüssels gering zu halten, soll dieser durch Verkettung von Attributen bzw. Teilen der Attribute bestimmt werden. Attribute die vorne im Schlüssel stehen haben dadurch eine höhere Priorität. In der zweiten Phase werden die Datensätze anhand des Schlüssels sortiert. In der dritten Phase wird ein Fenster (engl. Window) über die sortierten Datensätze geschoben und alle Datensätze innerhalb des Windows werden miteinander verglichen. Dieses Verfahren eignet sich besonders gut zur Erkennung von Duplikaten innerhalb einer Datenquelle. Sollen Duplikate in mehreren Datenquellen gefunden werden, müssen die Einträge beim Sortieren gemischt werden. Dadurch besteht allerdings die Gefahr, dass vorrangig Datensätze einer Datenquelle miteinander verglichen werden. Der Vorteil gegenüber dem Standard Blocking ist, dass die Anzahl der Vergleiche lediglich von der Größe der Datenquelle und der gewählten Fenstergröße abhängen. Ein großer Nachteil ist, dass Datensätze die sich in der ersten Stelle des Schlüssels unterscheiden, weit voneinander entfernt sind und dadurch nicht als Matches identifiziert werden. Um dennoch eine hohe Pairs Completeness zu erreichen, werden mehrere Schlüssel pro Datensatz generiert und ein Fenster mit kleiner Größe über die verschieden sortierten Listen geschoben. Dieses Verfahren entspricht im Grunde dem Multi-pass Verfahren beim Standard Blocking.

Ein großes Problem bei der klassischen und der Multi-pass Variante des Sorted Neighborhood Verfahrens ist, dass die zu wählende Fenstergröße w größer als die Anzahl der Datensätze mit dem am häufigsten vorkommenden Sortierschlüssel sein muss, um eine gute Pair Completeness zu erreichen. Sei n die Menge an Datensätzen mit dem am häufigsten vorkommenden Schlüssel k und m die Menge der Datensätze des darauffolgenden Schlüssels $k + 1$, dann ist $w = n + m$. Nur dadurch kann sichergestellt werden, dass alle Datensätze aus n mit den "nahen" Datensätzen aus m verglichen werden. Da Sortierschlüssel für gewöhnlich nicht gleichverteilt sind, gibt es meist wenige große und viele kleine Mengen an Datensätzen mit dem gleichen Sortierschlüssel. Dadurch werden Datensätze mit seltenen Sortierschlüsseln unnötig oft mit "weit" entfernten Datensätzen verglichen. Zudem dominiert der am häufigsten vorkommenden Schlüssel, genauso wie beim Standard Blocking, die Ausführungszeit des Algorithmus.

In [10] schlagen Draisbach & Naumann eine optimierte Variante des Sorted Neighborhood Verfahrens vor. Dabei zeigen Sie, dass Standard Blocking und Sorted Neighborhood zwei extreme von Überlappungen bei Partitionen sind. Gegeben sind zwei Partitionen P_1 und P_2, dann ist die Überlappung $U_{P_1,P_2} = P_1 \cap P_2$ und $u = |U_{P_1,P_2}|$. Ihre Idee ist es diese Überlappung zu optimieren. Dabei soll die Überlappung groß genug sein, um tatsächliche Matches zu finden, aber gering genug, um die Menge der Vergleiche zu reduzieren. Zunächst wird wie beim klassischen Verfahren sortiert. Danach werden angrenzende Datensätze in disjunkte Partitionen zerlegt und schließlich wird ein Überlappungsfaktor u gewählt. Innerhalb jedes Blockes wird analog zum Standard Blocking jeder Datensatz mit jedem anderen verglichen. Innerhalb des

Overlap-Window $w = u + 1$, wird jeweils das erste Element mit allen anderen verglichen. Ist $w = 0$ entspricht das Verfahren dem Standard Blocking und hat jede Partition nur ein Element entspricht es der Sorted Neighborhood Methode. Um zu vermeiden, dass eine Partition dominiert, können größere Partitionen in Subpartitionen geteilt werden.

Canopy Clustering

Cohen & Richman [14] schlagen ein Clustering-Verfahren zum Blocken, auf Basis von Canopies, vor. Die Idee von Canopy Clustering ist es, Datensätze anhand einer einfachen Vergleichsmetrik in überlappende Cluster (=Canopies) zu partionieren. Zur Generierung wird eine Kandidatenliste gebildet, welche initial als allen Datensätzen besteht. Dann wird zufällig ein Zentroid eines neuen Clusters gewählte und alle Datensätze innerhalb des Mindestabstandes d_1 zugewiesen. Zusätzlich werden alle Datensätze dieses Clusters mit einem weiteren Mindestabstandes $d_2 < d_1$ aus der Kandidatenliste entfernt. Dieser Algorithmus wird wiederholt, bis die Kandidatenliste leer ist. Die *Pair Completeness* hängt hierbei stark der gewählten Abstandsfunktion ab. Anschließend wird werden alle Datensätze eines Cluster miteinander verglichen.

2.2.2 Dynamisches Blocking

Für die Dupliaktserkennung in einer Datenquelle A, sind bei einer Anfrage $|A|$ Vergleiche notwendig. Da dies zu ungewollt hohen Latenzen führen würde, werden auch im dynamischen Fall Blocking Verfahren eingesetzt.

[TODO Was muss anders sein?]

DySimII

DySimII [15] ist die dynamische Variante des Similarity Aware Index von Christen & Gayler [8], welcher es zusätzlich erlaubt den Index während der Laufzeit zu erweitern. Dabei ist die Grundidee die benötigten Ähnlichkeiten vorauszuberechnen, um während der Laufzeit diese nur nachschlagen zu müssen.

Der Index besteht aus drei Teilen. Dem **Record Identifier Index (RI)**, welcher alle Attribute speichert und diese ihren Datensätzen zuordnet, dem **Block Index (BI)**, welcher Attribute anhand einer Enkodierungsfunktion gruppiert und zuletzt dem **Similarity Index (SI)**, welcher dieselben Schlüssel wie der Record Identifier Index verwendet und die Ähnlichkeiten der Attribute im gleichen Block hält. Abbildung 2.3 zeigt ein Beispiel eines DySimII Index. Im RI wurden die Datensatzidentifier von Tony (r1, r3) und Cathrine

Record ID	First name	Double-Metaphone
r1	tony	tn
r2	cathrine	k0rn
r3	tony	tn
r4	kathryn	k0rn
r5	tonya	tn
r6	cathrine	k0rn
r7	linda	lnt
r8	tonia	tn

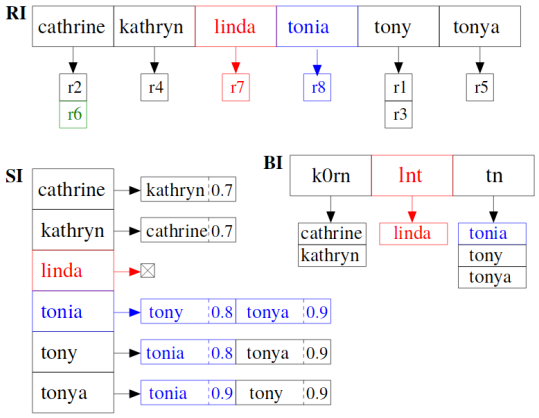

Abbildung 2.3 Ein DySimII-Index, welcher aus der Tabelle links erzeugt worden ist. Die Beispieldatensätze enthalten das Namensattribut und eine Double-Metaphone Enkodierung, welche als Blockingschlüssel genutzt wird. **RI** ist der Record Identifier Index, **BI** der Block Index und **SI** der Similarity Index. Das Beispiel ist aus [15] entnommen.

(r2, r6) als Attributsübereinstimmung gruppiert. Anschließend wurden im BI über die Double-Metaphone Enkodierung, welche einen identischen String für gleich klingende Wörter erzeugt, ähnliche Schreibweisen von Tony und Cathrine zusammengeführt, was dem Standard Blocking entspricht. Im SI wurden die Ähnlichkeiten von (Tony, Tonia und Tonya) bzw. (Cathrine, Kathryn), welche sich in einen gemeinsamen Block befinden, untereinander mit ihren berechneten Ähnlichkeiten verknüpft. Dabei wird für jedes Attribut eines Datensatzes ein eigenes Tripel RI, BI und SI genutzt, um zu verhindern, dass sich Werte unterschiedlicher Attribute vermischen.

Das Verfahren unterscheidet zwei Phasen. Die Bauphase (Build-Phase), in welcher der Index aus einem initialen Datenbestand erzeugt wird und die Anfragephase (Query-Phase), welche Anfragen aus einem Datenstrom beantwortet.

Build Phase. Das Einfügen von Datensätzen läuft nach folgendem Schema ab. Zunächst werden alle Attribute mit Verweis auf den Datensatzidentifier im Record Identifier Index gespeichert. Falls ein Attribut dort schon existiert wird lediglich der Identifier angefügt. Anschließend wird für jedes Attribut eine Enkodierung bestimmt. Anhand dieser Enkodierung werden die Attribute in jeweils einen Block im Block Index eingefügt. Beinhaltet der Block mehr als ein Attribut wird zu allen bereits im Block befindlichen Attributen die Ähnlichkeit bestimmt. Die Ähnlichkeiten gegenüber dem eingefügten Attribute werden unter dem eingefügten Attribute im Similarity Index eingefügt. Gleichzeitig werden die bestimmten Ähnlichkeiten zum eingefügten Attribute auch zu allen Attributen im Block im Similarity Index ergänzt.

Query Phase. Bei einer Anfrage wird zunächst der neue Datensatz dem Index hinzugefügt. Anschließend werden aus dem RI alle Identifier ausgelesen, welche ein gleiches

Attribute besitzen und werden in einen Akkumulator mit dem Ähnlichkeitswert 1 aufgenommen. Bei mehreren gleichen Identifiern werden die Ähnlichkeitswerte addiert. Anschließend werden die Attribute des Anfragedatensatzes im Similarity Index nachgeschlagen und alle Attribute des gleichen Blockes mit ihrer Ähnlichkeit ausgelesen. Zu diesen Attributen werden aus dem RI die Identifier abgefragt und mit ihrer Ähnlichkeit aus dem Similarity Index in Akkumulator aufgenommen.

Im Gegensatz zum Standard Blocking können Anfragen deutlich schneller beantwortet werden, da im Optimalfall keine Ähnlichkeitsberechnung stattfinden muss und lediglich Werte nachgeschlagen werden. Auf der negativen Seite steht hingegen der deutlich erhöhte Speicherbedarf, welcher durch das Halten der Ähnlichkeitswerte zurückzuführen ist.

Similarity-Aware Index with Local Sensitive Hashing (LSH)

Dieses Verfahren ist eine Erweiterung des DySimII durch LSH, welches von Li et al. [16] vorgestellt wurde. Die hier genutzte Variante des Local Sensitive Hashing nutzt das Minhash Verfahren. Minhashing ist eine effiziente Abschätzung der Überlappung zweier Mengen bekannt als Jaccard-Ähnlichkeit. Mittels des Minhash-Algorithmus ist es möglich für jeden Datensatz n Signaturen der Länge k zu generieren. Dazu werden n verschiedene zufällig gewählte Hashfunktionen genutzt. Um die Wahrscheinlichkeit zu erhöhen, dass nur gleiche Paar dieselbe Signatur haben wird eine Technik namens Banding genutzt. Dazu werden l Signaturen zu einem Band zusammengefügt und damit verundet. Mehrere Bänder sind logisch gesehen eine Veroderung. Auch dieses Verfahren teilt sich in Bau- und Anfragephase.

Build Phase. Beim Erzeugen des Index werden zunächst die Minhash Signaturen erzeugt und zu Bändern verundet. Anschließend werden die Datensatzidenfier wird mit den erzeugten Bändern verknüpft. Dazu wird ein Index erstellt, welcher als Schlüssel zunächst die verschieden Bänder hat. Innerhalb der Bänder gibt es weitere Subindicies, welche als Schlüssel die Minhash Signaturen haben. Den jeweiligen Signaturen innerhalb der Bänder wird der Datensatzidentifier zugewiesen. Dadurch sind gleiche Signaturen durch die Bänder getrennt, was die Wahrscheinlichkeit erhöht, dass unähnliche Datensatze eine gemeinsame Signatur im Index haben. Der LSH Index ersetzt dadurch den Record Index. Die Schritte zum Einfügen in den Block Index bzw. den Similarity Index sind analog zum DySimII.

Query Phase. Für die Beantwortung einer Anfrage werden zunächst für den neuen Datensatz die Minhash Signaturen und Bänder erzeugt und mit Datensatzidentifier in den LSH Index eingefügt. Danach werden die Datensatzidenfier mit gleichen Signaturen in den gleichen Bändern als Kandidatenmenge ausgelesen. Nun müssen die Attribute der

Kandidaten aus einer Datenquelle geladen werden. Mit diesen Attributen können aus dem Similarity Index die Ähnlichkeitswerte jedes Kandidaten bestimmt werden. Die Kandidaten, welche aus dem LSH Index erhalten wurden haben allerdings nicht zwingend Attribute in denselben Blöcken im Block Index wie der Anfragedatensatz. Deshalb können zu einigen Attributen keine vorberechneten Ähnlichkeiten aus dem Similarity Index bezogen werden. Da das Berechnen zur Laufzeit zu lange dauert, werden diese mit dem Ähnlichkeitswert 0 miteinberechnet. Dies mindert zwar die Genauigkeit etwas sorgt dennoch für gute Latenzen.

Im Gegensatz zum DySimII ist die Berechnung des Index aufwendiger, da für jeden Datensatz die Minhash Signaturen und Bänder berechnet werden müssen. Allerdings ist die Kandidatenmenge potentielle deutlich geringer als beim DySimII, wodurch die Anfragen schneller beantwortet werden.

DySNI

Das DySNI Verfahren von Ramadan et al. [7] ist eine dynamische Umsetzung des Sorted Neighborhood Verfahren aus dem statischen Blocking. Anstatt eines Arrays wird eine Baumstruktur verwendet, um Datasätze möglichst effizient zu selektieren. Der gewählte Baum ist ein BraidedTree (BRT), welcher eine Erweiterung eines balancierter binären AVL-Baums ist. Dieser unterscheidet sich, indem zusätzlich innerhalb des Baumes jeder Knoten jeweils einen Verweis auf seinen Vorgänger und seinen Nachfolger hat. Die Sortierung erfolgt alphabetisch nach einem gewählten Sortierschlüssel. Ein Knoten besteht dabei aus einem Sortierschlüsselwert (Sorting Key Value. kurz: SKV) und einer Liste an von Datensatzes mit diesem SKV.

Build Phase. Beim Einfügen eines neuen Datensatzes wird zunächst dessen SKV erzeugt. Wenn der SKV noch nicht im BRT-Baum vorhanden ist, wird ein neuer Knoten erzeugt und der Datensatzidentifier angehängt. Ist der Knoten bereits vorhanden, wird lediglich der Datensatzidentifier zum existierenden Knoten hinzugefügt. Zusätzlich wird der Datensatz in einen Inverted Index D eingefügt, um ihn zum Attributsvergleich mit anderen Datensätzen schnell selektieren zu können.

Query Phase. Zunächst wird der Anfragedatensatz, nach dem Vorgehen aus der Build Phase, eingefügt. Der Knoten in welchen der Anfragedatensatz eingefügt wurde, heißt Anfrageknoten N_q. Ausgehend von N_q wird ein Fenster über die benachbarten Knoten gespannt. Alle Datensätze, welche in Knoten innerhalb des Fensters gespeichert sind, werden als Kandidatenmenge C selektiert. Aus D werden dann für jeden Kandidaten seine Attribute geholt und anschließend in einem Paarvergleiche mit dem Anfragedatensatz die Ähnlichkeit ermittelt. Für die Erzeugung des Fensters werden vier Methoden

vorgestellt, welche sich an Varianten des statischen Sorted Neighborhood Verfahrens orientieren.

- **Fixed Window Size (DySNI-f)** ist das einfachste Verfahren, bei welchem das Fenster um einen festen Wert w in Vorgänger- und Nachfolgerichtung aufgespannt wird.
- **Candidates-Based Adaptive Window (DySNI-c)** erweitert das Fenster abwechselnd in Vorgänger- und Nachfolgerichtung, solange bis eine Mindestanzahl an Kandidaten gefunden wurde.
- **Similarity-Based Adaptive Window (DySNI-s)** nutzt die Ähnlichkeit zwischen SKVs. Dabei wird ein Fenster in eine Richtung solange erweitert bis die Ähnlichkeit zwischen dem SKV von N_q und dem nächsten Vorgänger bzw. Nachfolger eine Mindestähnlichkeit Δ unterschreitet.
- **Duplicate-Based Adaptive Window (DySNI-d)** erweitert das Fenster aus Basis gefundener Matches in beide Richtungen unabhängig. Dabei wird das Fenster um jeweils einen Knoten erweitert und zwischen dem Anfragedatensatz und den Datensätzen des neuen Knoten der Ähnlichkeitswert ermittelt, sowie klassifiziert, ob es sich um ein Match oder Non-Match handelt. Sinkt der Anteil an gefunden Matches unter eine Schranke δ, wird das Fenster in diese Richtung nicht weiter vergrößert.

Damit Ähnlichkeiten zwischen den Datensätzen nicht jedes Mal neu berechnet werden müssen, wird je nach gewählter Fensterberechnung, die Ähnlichkeit der SKVs zu berechnen und in den beteiligten Knoten abzuspeichern. Dadurch wird allerdings die Auswahl an SKVs auf Konkatenation von Attributen beschränkt. Attribute, die nicht im SKV genutzt wurden, müssen bei diesem Verfahren trotzdem jedes Mal neu berechnet werden. Des Weiteren ist auch dieses Verfahren sensitiv auf Fehler am Anfang des SKV. Um dies zu korrigieren wird, ähnlich zum Multi-pass Verfahren des Sorted Neighborhood Verfahren, vorgeschlagen mehrere BRT-Bäume mit unterschiedlichen SKVs zu erstellen.

In ihrer Auswertung zeigen die Ramadan et al., dass das *DySNI-d* Verfahren im BRT-Baum nicht funktioniert, weil ein Großteil der Duplikate im Anfrageknoten N_q landet und damit eine Erweiterung des Fensters nicht zustande kommt. Die besten Recall Ergenisse wurden mit dem *DySNI-s* Verfahren erreicht, da der Baum nach den SKVs sortiert wurde und dieses Fenster sich am besten aufspannt. Die beiden anderen Verfahren *DySNI-f* und *DySNI-c* erzielen, ebenfalls gute Ergebnisse. Zusätzlich haben die Verfahren den Vorteil, dass sich das Fenster und damit die Anzahl der Kandidaten gut kontrollieren lässt und dadurch auch die Latenzen.

[TODO Fazit Dynamisches Blocking]

2.2.3 Blocking Schema

Die Qualität aller Verfahren, ob statisch oder dynamische, hängt maßgeblich von der Auswahl der richtigen Blockschlüssel bzw. Sortierschlüssel ab. Ein Verbund aus mindestens einem Blockschlüssel für einen Entitätentypen nennt man Blocking Schema. Wie genau diese Schemata auszuwählen ist, wird von vielen Blocking Verfahren offen gelassen. Die meisten Verfahren schränken jedoch ein, dass zu einem Datensatz nur ein Blockschlüssel oder Sortierschlüssel erzeugt werden darf. Bei der Verwendung von Multi-pass Ansätzen werden dementsprechend verschiedene Schema gefordert. Ein adäquates Schema zu finden ist oft, auch von Domainexperten, nur durch ausprobieren herauszufinden. Oft genutzt werden phonetische Enkodierung und Konkatenation von Attributen. Weitere Beispiele sind Q-Gramme oder Suffixe aus den vorgestellten statischen Verfahren.

DNF-Blocking Schema

Um die Probleme des manuellen Auswählens von Block- und Sortierschlüsseln zu umgehen schlagen Kejriwal & Miranker [17] ein Verfahren vor, welches ein Blocking Schema in Disjunktiver Normalform erzeugt. Dieses Schema besteht aus den folgenden vier Komponenten.

Indizierungsfunktion. Das kleinste Element eines Blocking Schema ist eine *Indizierungsfunktion* $h_i(x_t)$. Diese akzeptiert einen Attributswert x_t eines Datensatzes und erzeugt eine Menge Y, welche 0 oder mehr Blockschlüssel (engl. *blocking key value*, kurz BKV) beinhaltet. Ein BKV identifiziert einen Block, welchem ein Datensatz zugeordnet wird. Ein Beispiel einer Indizierungsfunktion ist `Tokens`. Mit der Funktion Tokens wird ein Eingabestring, beispielsweise 'Marios Pizza', in eine Menge von Token mittels eines Trennzeichens getrennt, beispielsweise durch eine Leerzeichen in $Y = \{$'Marios', 'Pizza'$\}$.

General Blocking Predicate. Das allgemeine Blockingprädikat (engl. *general blocking predicate*) $p_i(x_{t_1}, x_{t_2})$ nimmt zwei Attribute unterschiedlicher Datensatze t_1 und t_2 und nutzt die i$^{\text{te}}$ Indizierungsfunktion, um zwei Mengen von BKV Y_1 und Y_2 zu erzeugen. Das Prädikat ist wahr, wenn beiden Mengen eine gemeinsame Schnittmenge haben $Y_1 \cup Y_2 \neq$. Angenommen die i$^{\text{te}}$ Indizierungsfunktion ist `Tokens`, dann ist das zugehörige Prädikat `EnthältGemeinsamenToken`, welches Wahr ist, wenn zwei Attribute mindestens einen gemeinsamem Token haben. Beispielsweise $p_{egt}('Marios\ Pizza','Tonys\ Pizza') = Wahr$, weil $Y_1 = \{$'Marios', 'Pizza'$\}$ und $Y_2 = \{$'Tonys', 'Pizza'$\}$ woraus folgt das $Y_1 \cup Y_2 = \{$'Pizza'$\}$ und damit ist das Prädikat erfüllt.

Specific Blocking Predicate. Das spezifische Blockingprädikat (engl. *specific blocking predicate*) ist ein Paar (p_i, f), dass ein allgemeines Blockingprädikat p_i mit einem Attribute f

verbindet. Dazu nimmt das spezifische Blockingprädikat zwei Datensätze t_1 und t_2 und wendet p_i auf die entsprechenden Attribute f_1 und f_2 der Datensätze an. Ein solches Paar ist beispielsweise (`EnthältGemeinsamenToken`, `PLZ`). Dieses spezifische Predikat ist wahr, wenn die Postleitzahl zweier Datensätze einen gemeinsamem Token enthält.

DNF Blocking Schema. Das DNF Blocking Schema f_P ist eine Funktion, welche in der Disjunktiven Normalform ohne Negation durch eine Menge von P Ausdrücken erzeugt wird. Jeder Ausdruck in f_P besteht aus mindestens einem spezifischen Blockingpredikat, beispielsweise (`EnthältGemeinsamenToken`, `Name`) $\vee$ (`ExakteÜbereinstimmung`, `Stadt`). Mehrere spezifische Blockingpredikat in einem Ausdruck werden durch eine Konjunktion verbunden. Das DNF Blocking Schema ist dementsprechend Wahr, wenn einer seiner Ausdrücke Wahr ist. Ist ein Blocking Schema für zwei Datensätze Wahr, haben beide mindestens einen gemeinsamen Blockschlüsselwert. Für den Blockschlüssel entspricht die Konjunktion eines Ausdrucks dem Konkatenieren von Strings. Gegeben sei folgendes Blocking Schema f_P = (`ExakteÜbereinstimmung`, `Name`) $\wedge$ (`ExakteÜbereinstimmung`, `Stadt`) und der Datensatz r = ('Peter', 'Frankfurt'). Daraus erzeugt f_P den Blockschlüssel 'PeterFrankfurt'. Die Disjunktion der Ausdrücke kann bei vielen Verfahren als Multi-pass Ansatz implementiert werden. Des Weiteren ist zu beachten, dass das Blocking Schema potentiell mehrere Schlüssel pro Datensatz erzeugt.

Lernen des DNF-Blocking Schema

Bevor das Verfahren von Kejriwal & Miranker [17] automatisiert ein Blocking Schema bestimmen kann, wird angenommen, dass eine Menge von bekannten Matches und Non-Matches vorhanden ist. Der erste Schritt zum Lernen eines DNF-Blocking Schema ist, eine Liste an spezifischen Blockingprädikaten zu benennen. Beispielsweise `EnthältGemeinsamenToken` und `ExakteÜbereinstimmung` für Strings und `Erste3Ziffern` für nummerische Attribute. Anschließend wird für jedes Match bzw. Non-Match ein boolscher Featurevektor über die spezifischen Blockingprädikate erzeugt, welcher Wahr ist, wenn das Paar das Prädikat erfüllt. Dabei wird die Menge positiver Vektoren mit P_f und negativer Vektoren mit N_f bezeichnet.

Anschließend werden die Ausdrücke des Blocking Schema erzeugt. Da exponentiell viele Ausdrücke erzeugt werden können, muss der Anwender die Tiefe, d.h. Anzahl der Prädikate pro Ausdruck, beschränken. Abbildung 2.4 zeigt wie aus den Einzelausdrücken (`EnthältGemeinsamenToken`, `name`), (`ExakteÜbereinstimmung`, `stadt`) und (`Erste3Ziffern`, `plz`) ein zweistelliger und ein dreistelliger Ausdruck erzeugt werden. Zwei Ausdrücke a_1 und a_2 mit ihren Featurevektoren P_{f,a_1}, N_{f,a_1} und P_{f,a_2}, N_{f,a_2} werden zu einem neuen Ausdruck a_{1-2} konjugiert, indem die Vektoren verundet werden $P_{f,a_1-2} = P_{f,a_1} \wedge P_{f,a_2}$, respektive $N_{f,a_{1-2}} = N_{f,a_1} \wedge N_{f,a_2}$. Dadurch wird vermieden

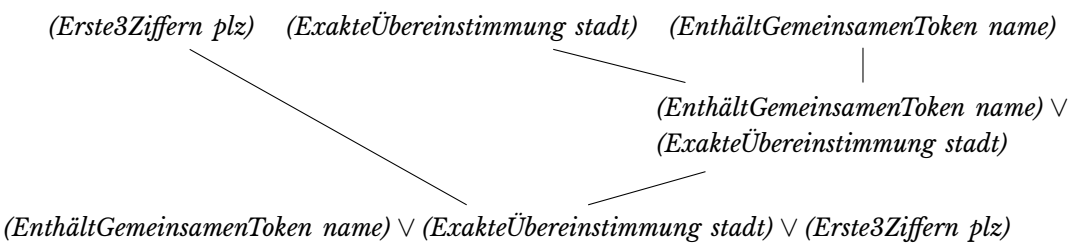

Abbildung 2.4 Konjunktion der drei Ausdrücke (`EnthältGemeinsamenToken`, `name`), (`ExakteÜbereinstimmung`, `stadt`) und (`Erste3Ziffern`, `plz`) zu einem zweistelligen und dreistelligen Ausdruck.

die teuren Prädikatoperationen auf jeden Ausdruck anwendenn zu müssen. Danach wird die Qualität der einzelnen Ausdrücke bewertet. Dazu nutzen Kejriwal & Miranker die Fisher-Score. Die Idee der Fisher-Score nach [18] ist, eine Untermenge von Features (hier: Ausdrücke) zu finden, sodass die Datenpunkte der Klassen (hier: Matches und Non-Matches) möglichst weit voneinander entfernt und gleichzeitig die Datenpunkte innerhalb der Klasse möglichst nahe zusammen sind. Die Formel zur Berechnung der Fisher-Score des i$^{\text{ten}}$ Ausdrucks sieht folgendermaßen aus

$$\rho_i = \frac{|P_{f,i}|(\mu_{p,i} - \mu_i)^2 + |N_{f,i}|(u_{\mu,i} - \mu_i)^2}{|P_{f,i}|\sigma_{p,i}^2 + |N_{f,i}|\sigma_{n,i}^2},$$

dabei ist $\mu_{p,i}$ bzw. $\mu_{n,i}$ der Anteil der wahren Werte in $P_{f,i}$ und $N_{f,i}$. Weiterhin ist μ_i der Anteil wahrer Werte in $P_{f,i}$ und $N_{f,i}$ zusammen und σ ist die positive bzw. negative Varianz.

Anhand der bewerteten Ausdrücke wird das DNF Blocking Schema gebildet. Dazu werden die Ausdrücke nach ihrer Fisher-Score sortiert. Anschließend werden die Ausdrücke zur DNF Blocking hinzugefügt, solange ein Ausdruck mindestens ein weiteres Match erfasst. Dies wird wiederholt, bis ein Minimum an Matches noch nicht erfasst wurden oder alle Ausdrücke verarbeitet sind.

2.3 Ähnlichkeitsmaße

In Abschnitt 2.2 wurde beschrieben wie Kandidaten für einen Vergleich gruppiert und selektiert werden. Über Ähnlichkeitsmaße (engl. similarity measures) wird die Ähnlichkeit zweier Datensätze bestimmt. Genauer wird die Ähnlichkeit der einzelnen Attribute bestimmt, aus welcher sich die Gesamtähnlichkeit der Datensätze bestimmen lässt. Die meisten Fehler, welche zu unterschiedlichen Datensätzen führen sind typographische Variationen von Strings. Weshalb sich entsprechend viele Ansätze für den Vergleich von Stringattributen finden. Attributsähnlichkeiten werden nach Elmagarmid et al. [19] in vier Kategorien geordnet:

- zeichenbasierend
- tokenbasierend
- phonetisch
- nummerisch

zusätzlich werden noch die kernelbasierend Methoden betrachtet.

Zeichenbasierende Ähnlichkeit

Wie sich die Ähnlichkeit von Strings bestimmen lässt, wird seit den 60er Jahren [20] intensiv erforscht. Die Stärke von zeichenbasierten Ähnlichkeiten sind das Erkennen von typografischen Fehlern.

Der älteste und wohl auch bekannteste Algorithmus ist die Levenshtein Distanz [20]. Die Levenshtein Distanz ist eine **Editierdistanzen**, welche die minimalen Schritte berechnet, die benötigt werden um einen String σ_1 in einen anderen σ_2 umzuwandeln. Diese Schritte beinhalten das Einfügen, das Löschen, das Ersetzen und mit der Modifikation von Damerau [21] auch das Vertauschen von Zeichen. Je weniger Schritte für die Transformation notwendig sind, desto ähnlicher sind zwei Strings. Da potentiell alle Zeichen beider Strings miteinander vergleichen werden ist die Komplexität $O(|\sigma_1| \cdot |\sigma_2|)$. Needleman und Wunsch [22] erweitern die originale Editierdistanz dahingehend, dass bestimmte Operationen anders gewichtet werden. Dazu können die Kosten für die einzelnen Schritte, welche in der einfachsten Form 1 sind, auf einen beliebigen Gleitkommawert angepasst werden. Beispielsweise können Transpositionen, welche auf einen Tippfehler zurückgeführt werden können, niedriger gewichtet werden. In dieser Variante entspricht das Lösen der Editierdistanz dem Traveling Salesmen Problem und ist daher NP-schwer.

Eine weitere Modifikation der Editierdistanz ist es Lücken zu erkennen [23], beispielsweise wenn ein Wort abgekürzt wurde, etwa *John R. Smith* statt *Johnathan Richard Smith*. Dementsprechend können Kosten für das Öffnen und das Erweitern einer Lücke festgelegt werden. Eine andere ist Fehler am Anfang oder am Ende des String mit geringeren Kosten zu versehen [24].

Eine weitere gängige Alternative ist die Jaro-Distanz, welche die Anzahl der gemeinsamen Zeichen m berechnet, wobei eine Verschiebung von $\frac{1}{2} \cdot min(|\sigma_1|, |\sigma_2|)$ zugelassen wird. Von den gemeinsamen Zeichen werden anschließend die Transpositionen t berechnet, d.h. wie viele gemeinsame Zeichen nicht in der gleiche Reihenfolge sind. Daraus berechnet sich die Jaro-Distanz $d_j = \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right)$.

Tokenbasierende Ähnlichkeit

Die tokenbasierende Ähnlichkeit bietet, im Gegensatz zu den zeichenbasierenden, den Vorteil, dass Vertauschungen von Wörtern erkannt werden. Beispielsweise bei Vorname und Nachname.

Monge und Elkan [25] schlagen einen Algorithmus auf Basis atomarer Strings vor. Für zwei Strings σ_1, σ_2 werden alle Token aus $\sigma_1 = (\sigma_{1_{t_1}}, \ldots, \sigma_{1_{t_i}})$ mit allen Token aus $\sigma_2 = (\sigma_{2_{t_1}}, \ldots, \sigma_{2_{t_j}})$ vergleichen. Zum Vergleich wird eine beliebige zeichenbasierende Ähnlichkeit sim gewählt werden. Dadurch kann dieser Algorithmus auch typographische Fehler erkennen. Anschließend wird für jeden Token t in σ_1 die Ähnlichkeit mit $s_t = max(sim(\sigma_{1_t}, \sigma_{2_{t_1}}), \ldots, sim(\sigma_{1_t}, \sigma_{2_{t_j}}))$ berechnet. Die Gesamtähnlichkeit ist der Durchschnitt der Tokenähnlichkeiten $m = avg(s_{t_1}, \ldots, s_{t_i})$. Das Problem dieses Algorithmus ist seine Komplexität, welche quadratisch zur Tokenmenge und damit zu sim ist.

Eine weitere Möglichkeit Token zu bilden sind Q-Gramme. Diese erlauben es ebenfalls neben Vertauschungen von Wörtern auch typographische Fehler zu erkennen. Über Q-Gramme wird ein String σ in k überlappende Token der Länge n zerlegt. Dazu wird ein Fenster der Länge n von Position 1 bis $|\sigma| - (n-1)$ geschoben. Um aus Q-Grammen eine Ähnlichkeit zu bestimmen gibt es viele Möglichkeiten.

Eine deutlich einfachere und schnellere Möglichkeit die Ähnlichekit von Token zu berechnen, ist der *Jarcard-Koeffizienten*. Dieser gibt die Ähnlichkeit zweier Mengen A, B mit $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ an. Ein ähnliches Maß bietet der *Simpson-Koeffizienten*, welcher die Überlappung zweier Mengen $S(A,B) = \frac{|A \cap B|}{\min(|A|,|B|)}$ bestimmt.

Ein weiteres Vorgehen auf Basis atomarer Strings ist WHIRL von Cohen [26]. Es kombiniert die Kosinus-Ähnlichkeit mit dem TF/IDF Gewichtungsschema. Dabei ist TF die Vorkommenshäufigkeit (engl. term frequency) und gibt an wie häufig ein Token in einem String vorkommt. IDF ist die Inverse Dokumenthäufigkeit (engl. inverse document frequency) und gibt an wie oft ein Token in den bekannten Strings eines Vokabulars D vorkommnt. Für alle atomaren Strings w wird ein Gewicht berechnet

$$\nu_\sigma(w) = log(tf_w + 1) \cdot log(idf_w).$$

Die Kosinus-Ähnlichkeit zweier String σ_1, σ_2 ist dementsprechend definiert als

$$sim(\sigma_1, \sigma_2) = \frac{\sum_{i=1}^{|D|} \nu_{\sigma_1}(j) \cdot \nu_{\sigma_2}(j)}{\|\nu_{\sigma_1}\| \|\nu_{\sigma_2}\|}$$

Mit WHIRL ist es möglich vertauschungssicher die Ähnlichkeit von Strings zu bestimmen. Ein großer Vorteil dieses Algorithmus ist, dass nach erheben des TF/IDF Index, die

Ähnlichkeit in $O(1)$ berechnet werden kann, da lediglich Werte nachgeschlagen werden müssen. Dem entgegen steht, dass typographische Fehler nicht erkannt werden. Deshalb haben Gravano et. al [27] WHIRL erweitert und nutzen statt atomare Strings Q-Gramme. Dadurch lassen sich bei gleicher Komplexität auch Rechtschreibfehler erkennen, da ein Großteil der Q-Gramme gleich ist. Allerdings geht zu zugunsten von Speicherkosten, da der TF/IDF Index dementsprechend größer wird.

Phonetische Ähnlichkeit

Die phonetische Ähnlichkeit ist sowohl zeichen- als auch tokenbasiert. Es werden jedoch nicht die Zeichen oder Token miteinander verglichen, sondern die Sprechlaute von Wörtern. So ist es möglich gleich gesprochene Wörter mit unterschiedlicher Schreibweise zu finden. Dies ist vor allen Dingen bei Namen sehr nützlich, da es hier eine besonders hohe Dichte an gleich klingenden Worten mit kleinen Variationen in der Schreibweise gibt. Phonetische Enkodierungsschemata funktionieren allerdings meist nur für eine Sprache oder einen Akzent. Bekannt Beispiele für englisch Sprache sind Soundex und NYSIIS, sowie Metaphone und Double Metaphone, welche sich zum Teil auch auf nicht englische Sprachen anwenden lassen. Ein Methode für die deutsche Sprache ist die Kölner Phonetik.

Nummerische Ähnlichkeit

Während es für Strings eine Vielzahl ein Vergleichmöglichkeiten gibt ist die Anzahl bei den nummerischen Überschaubar. Die offensichtlichste Methode ist eine Nummer als String zu behandeln und entsprechende Algorithmen zu verwenden. Andere eher primitive Vorgehensweisen sind etwa, die ersten n oder letzten m Ziffern miteinander zu vergleichen, beispielsweise bei einer Postleitzahl. Für Mengenangaben zwischen zwei Werten n_1 und n_2 kann die maximale absolute Differenz genutzt werden $sim_{d_{max}} = 1.0 - \left(\frac{|n_1 - n_2|}{d_{max}}\right)$.

Kernel Ähnlichkeit

[TODO H]inweis Attribute nicht Datensätze

[TODO S]VM weglassen Verweis auf später. Nur Hinweis, dass Kernels dort auch benutzt werden.

Anhand zweier gegebener Strings gibt es keine offensichtliche Antwort auf die Frage: Wie ähnlich sind σ_1 und σ_2? Im Gegensatz dazu kann dies für Vektoren in $\mathbb{R}^d$ eindeutig, beispielsweise über die Kosinus-Ähnlichkeit $\frac{\sigma_1 \cdot \sigma_2}{\|\sigma_1\|\|\sigma_2\|}$ berechnet werden [28]. Kernelfunktionen werden unter anderem in Support Vector Machines (SVM) eingesetzt. Die-

se Kernel weisen eine Reihe statistischer interessanter Eigenschaften auf, beispielsweise dass ihre Performanz unabhängig von der Dimensionalität ist, auf welcher die Berechnung stattfindet. Dadurch ist es möglich in hohen Dimensionalitäten zu arbeiten ohne Überanzupassen [29].

Der einfachste und meist genutzte String Kernel ist der Bag-of-Words Kernel. Dabei werden die Anzahl der vorkommenden Worte in σ gezählt und in einem dünnbesetzten Vektor, über die Menge aller bekannten Worte aller bekannten Strings, erzeugt. Wie bereits bekannt sind atomare String anfällig für typographische Fehler. Aus diesem Grund gibt es auch Variationen des Kernels, welcher Q-Gramme nutzen, um dieses Problem zu umgehen.

Lodhi et al. stellen eine Kernelfunktion vor, um Strings im Feature Space miteinander zu vergleichen, ohne diese vorher in Vektoren zu zerlegen. Der sogenannte String Subsequence Kernel (SSK) vergleicht Strings, indem er Stringvektoren erzeugt, welche einen bestimmten Substring beinhalten oder nicht. Dabei wird jedes Vorkommen eines Substrings anhand der Übereinstimmung gewichtet. Die Übereinstimmung erlaubt beispielsweise auch Lücken, sodass der Substring 'c-a-r' in den beiden Wörtern '**car**d' und 'cust**ar**d' mit unterschiedlicher Gewichtung vorkommt.

2.4 Klassifikatoren

Die Aufgabe von Klassifikatoren oder Matching-Strategien (vgl. Köpcke & Rahm [2]) ist es, Datensatzpaare in zwei Mengen Matches und Non-Matches kategorisieren. Im Gegensatz zu den Attributesähnlichkeitsmaßen bewerten diese einen kompletten Datensatz, welcher im Normalfall aus mehreren Attributen besteht. Klassifikatoren können nach Elmagarmid et al. [19] in zwei Kategorien einordnen werden.

- Vorgehen die *Trainingsdaten* benötigen, um zu Lernen welche Datensätze übereinstimmen. Hierzu gehören überwachte, semi-überwachte, aktive und unüberwachte Lernstrategien.
- Vorgehen die *Domänenwissen* oder *generische Distanzmaße* nutzen, um Übereinstimmungen zu finden.

2.4.1 Distanzbasierende Verfahren

Nachdem die Ähnlichkeit zwischen den Attributen der Kandidatenpaaren berechnet wurden, gibt es für jedes Kandidatenpaar (t_1, t_2) einen Ähnlichkeitsvektor $(sim(t_{1,a_1}, t_{2,a_1}), \ldots, sim(t_{1,a_n}, t_{2,a_n}))$ über die Attribute $a_1, \ldots, a_n$ beider Daten-

sätze. Dieser Vektor wird von den nachfolgenden Verfahren genutzt, um eine Match-Entscheidung zu treffen.

Schwellenwertbasierend

Die naivste Art und Weise zu klassifizieren sind nach Christen [30, Kap. 6] Schwellenwerte. Dazu werden die Ähnlichkeitsvektoren zu einer Gesamtähnlichkeit g aufsummiert. Anschließend werden je nach Ausprägung bis zu zwei Schwellen festgelegt. In der Variante mit einer Schwelle t, werden die Kandidatenpaare in zwei Klassen mit $g >> t$ als Matches und $g < t$ als Non-Matches klassifiziert. Werden zwei Schranken $t1$ und $t2$ genutzt, wird in drei Klassen gegliedert, Matches mit $g \geq t1$, Non-Matches mit $g \leq t2$ und zusätzlich gibt es noch den Bereich $t2 < g < t1$, welcher ein potentielles Match bedeutet und manuelle klassifiziert werden muss. Der Nachteil dieser Methodik ist, dass beim Mitteln des Vektors alle Attribute mit gleichem Gewicht zum endgültigen Wert beitragen. Dadurch wird die Wichtigkeit der unterschiedlichen Attribute und ihre Wertstellung innerhalb des Datensatzes verworfen. Um dem entgegenzuwirken, können für jedes Attribut Gewichte vergeben werden, mit welchen die Wertstellung der Attribute angegeben werden kann. Dennoch gehen beim Aufsummieren von Ähnlichkeiten detaillierte Informationen über die einzelnen Ähnlichkeiten verloren.

Regelbasierend

Christen [30, Kap. 6] beschreibt die regelbasierende Klassifikation als Anwendung der Prädikatenlogik erster Stufe (PL1). Dabei wird ein Klassifikationspredikat in konjunktiver Normalfrom mit disjunktiven Ausdrücken geschrieben.

$$((sim(name)[r_i, r_j] \geq 0.9) \wedge (sim(stadt)[r_i, r_j] = 1.0)) \\ \vee (sim(plz)[r_i, r_j] \geq 0.7) \implies [r_i, r_j] \rightarrow Match \tag{2.1}$$

Formel 2.1 zeigt eine beispielhafte Regel, wobei $sim(\cdot)$ einer Attributsähnlichkeit entspricht und r_i, r_j zwei Datensätze sind. Diese Regel klassifiziert ein Paar als Match, wenn entweder der Ähnlichkeitswert für Name größer 0.9 und die Stadt gleich ist, oder der Ähnlichkeitswert der Postleitzahl größer 0.7 ist. Der Vorteil gegenüber den schwellenbasierenden Verfahren ist, dass Ausdrücke auf Attribute angewendet werden und dadurch die Informationen der einzelnen Attributsähnlichkeiten nicht verloren gehen. Mit der regelbasierten Klassifikation kann in beliebig viele Klassen kategorisiert werden. Typischerweise werden entweder zwei Klassen Match und Non-Match oder zusätzlich potentielles Match klassifiziert. Bei Match und Non-Match ist nur ein Prädikat P_m notwendig, da alle wahren Paare als Matches und alle falschen Paare als Non-Matches klassifiziert werden. Für potentielle Matches wird ein weiteres Prädikat P_{pm} benötigt, dementsprechend sind Attribute Non-Matches, wenn sowohl P_m als auch P_pm falsch ist. Für die Be-

stimmung der Prädikate gibt es zwei Möglichkeiten. Die erste ist einen Domänexperten das Prädikat festlegen zu lassen. Dies ist allerdings ein sehr zeitintensiver Prozess, welcher bei aller Expertise in den meisten Fällen durch ausprobieren gelöst werden muss. Die Alternative ist ein Prädikat zu Lernen, was ähnlich zum Lernen eines Blocking Schema (vgl. Abschnitt 2.2.3) funktioniert.

2.4.2 Überwachtes bzw. semi-überwachtes Lernen

Die Verfahren für überwachtes und semi-überwachtes Lernen benötigen eine Menge von klassifizierten Daten in der Form von Matches und Non-Matches. Anhand dieser Trainingsdaten kann ein Klassifikationsmodell erstellt werden. Soll das Model Datensätze nur in die zwei Klassen Matches und Non-Matchses ordnen, wird ein binärer Klassifikatoren gesucht.

Decision Trees

Decision Trees sind als Klassifikatoren sehr beliebt, da ihre Funktionsweise anschaulich ist. Zudem kann ein Modell übersichtlich visualisiert werden, sodass es intuitiv, auch von Laien, interpretiert werden kann. Ähnlich zu dem regelbasierten Verfahren prüft auch der Decision Tree den Ähnlichkeitswert eines bestimmten Attributes, welches einem Wert im Vektor entspricht. Dementsprechend kann ein Model eines Decision Tree direkt in einem Prädikat formuliert werden.

(Beispiel?)

Support Vector Machines

Support Vector Machines wurden von Boser et al. [31] eingeführt. In ihrer einfachsten Form lernen SVMs eine separierende Hyperebene zwischen einer Menge von Punkten, welche den Abstand zwischen der Hyperebene und den nähesten Punkten der jeweiligen Klassen maximiert. Eine Kernelfunktion berechnet das innere Produkt zwischen Punkten im Hyperraum (Feature Space) [29].

Bilenko & Mooney [32] stellen ein Lernverfahren auf Basis der TF/IDF Ähnlichkeit von Cohen vor, welches einen SVM-Klassifikator nutzt. Dazu wird ein Vektor erzeugt, indem die Kosinus-Ähnlichkeit zwischen den Attributen eines Datensatzpaares berechnet wird. Dabei werden die Komponenten der bekannten Summe $\frac{x_i \cdot y_i}{\|x\|\|y\|}$, welche zum i$^{\text{ten}}$ Element des Vokabulars gehören, einem d-dimensionalen Vektor zugeordnet $\mathbf{p}(x, y) = \langle \frac{x_i \cdot y_i)}{\|x\|\|y\|} \rangle$. In Abbildung 2.5 sind zwei Datensätze gezeigt. Jedes Attributspaar ist dabei ein Teil eines Vektors. Die Vektoren werden dann von einem SVM-Model in Matches und Non-

Name	*Address*	*City*	*Cuisine*
Fenix	8358 Sunset Blvd. West	Hollywood	American
Fenix at the Argyle	8358 Sunset Blvd.	W. Hollywood	French (new)

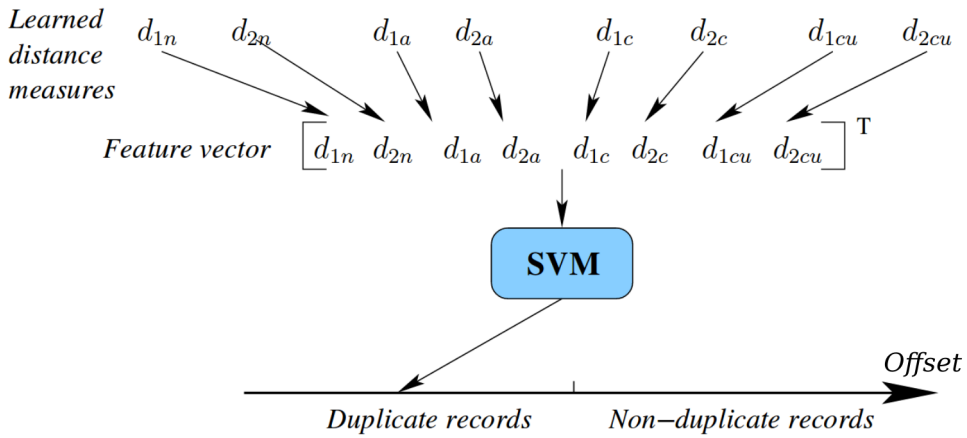

Abbildung 2.5 Datensatzklassifikation nach [32]. Der Featurevektor für die Klassifikation wird aus den Attributsähnlichkeiten von Name, Address, City und Cuisine erzeugt. Eine SVM klassifiziert diesen Vektor anschließend in Match (duplicate records) oder Non-Match (Non-duplicate records).

Matches klassifiziert. Trainiert wird das SVM-Model anhand von Match und Non-Match Vektoren.

Christen [33] erweitert dieses Verfahren, indem mehrere SVM Modelle trainiert werden. Die initiale SVM wird mit der Mengen der offensichtlichen Matches und Non-Matches der Gesamttrainingsmenge trainiert. Bei offensichtlichen Matches sind die Werte der Vektoren sehr nahe an 1 und bei offensichtlichen Non-Matches sehr nahe bei 0. Alle nicht offensichtlichen Vektoren der Trainingsmenge werden anschließend mit der initialen SVM klassifiziert. Je nach Klassifizierungsergebnis werden diejenigen, welche am weitesten von der seperierenden Hyperebene entfernt sind, zur Menge der offensichtlichen Matches bzw. Non-Matches hinzugefügt. Die zweite SVM wird dann mit den erweiterten Trainingsmengen trainiert. Diese Schritte werden solange wiederholt, bis ein Stopkriterium erfüllt ist.

2.4.3 Aktives Lernen

Ein großer Nachteil der überwachten Lernverfahren ist, dass die Trainingsmenge viele Beispiele benötigt und das diese repräsentativ für die zu Gesamtmenge von Entitäten sein muss. Wie Trainingsdaten akquiriert werden wird in Abschnitt 2.5 diskutiert. Als Alternative dazu gibt es die aktiven Lernverfahren, welche initial nur eine sehr kleine Trainingsmenge (*seed*) benötigen. Auf Basis des initialen Models werden in Interaktion

mit einem erfahrenen Benutzer Datensatzpaare selektiert, die helfen das Klassifikationsmodell zu verbessern. Eine initiales Modell kann relativ einfach über offensichtliche Matches und Non-Machtes erzeugt werden. Anschließend ist aktives Lernen ein iterativer Prozess. Zunächst wird die Trainingsmenge mit dem Modell klassifiziert. Anschließend ist aktives Lernen ein iterativer Prozess. Zunächst wird die Trainingsmenge mit dem Model klassifiziert. Aus der Menge klassifizierte Daten werden die Interessantesten ausgewählt, die manuelle von einem Benutzer klassifiziert werden. Anschließend werden diese zu den initialen Daten hinzugefügt und es wird ein neues Model trainiert. Diese Schleife wird solange wiederholt bis ein Stopkriterum (Anzahl von Iterationen oder minimale Genauigkeit) erreicht wurde.

Arasu et al. [34] kombinieren eine aktives Lernvorgehen mit einem Blockingmechanismus, welcher entweder mit einem Decision Tree oder einer SVM funktioniert. Dabei gibt der Benutzer als Stopkriterium die Mindestpräzision (siehe Abschnitt 2.5) an, welcher das finale Modell entsprechen muss. Der Lernprozess versucht dann ein Model zu finden, welches einen hohen Recall liefert und gleichzeitig die Anzahl der manuell zu klassifizierenden Paare gering hält.

2.5 Messen von Qualität- und Komplexität[1]

Aus den bis hier vorgestellten Verfahren zu Entity Resolution stellt sich die Frage: Wie kann die Qualität und Komplexität dieser Verfahren gemessen werden? Dies soll dazu dienen, ein Verfahren zu bewertet und gleichzeitig eine Vergleichbarkeit zwischen anderen Verfahren bieten. Damit die Qualität und die Komplexität der ER-Verfahren überprüft werden kann, ist es unerlässlich über die Ground Truth Daten (auch Gold Standard Daten) zu verfügen. Die Ground Truth beschreibt eine Menge gekennzeichneter Daten, welcher einer oder mehreren Klassen angehören. Für Entity Resolution sind die Daten Datensatzpaare und die Klassen Matches und Non-Matches. Damit die Ground-Truth repräsenativ für die zu überprüfenden Daten ist, sollte diese möglichst deren Charakteristik wiederspiegeln. Daraus entsteht die nächste Frage: Woher kommen die Ground Truth Daten?

- Wird versucht einen entwickelten Algorithmus/Verfahren zu bewerten, dann empfiehlt es sich einen der frei verfügbaren Datensätze zu nehmen, zu welchen bereits Ground Truth Daten existieren und beispielsweise von Wissenschaftlern oder Domainexperten manuell klassifiziert wurden. Das Problem ist, dass viele dieser Datensätze nur wenige Einträge (meist < 10.000) haben und daher kaum Bezug zu Realdaten haben. Diese Datensätze werden in Abschnitt 2.6 diskutiert.

[1] Dieser Abschnitt bezieht sich auf Analysen und Erklärungen zu Qualität und Komplexität von Entity Resolution Systemen aus Christen [30].

- Soll ein Verfahren auf Daten einer Domäne angewendet werden, zu welcher keine Ground Truth existiert, müssen diese manuell erzeugt werden. Dabei werden Datensatzpaare zufällig erzeugt und müssen anschließend von einem Prüfer in Matches und Non-Matches klassifiziert. Ein großer Nachteil dieser Methode ist, selbst wenn ein Blockingverfahren angewendet wurde, dass die Zahl der zu klassifizierenden Paare riesig ist. Hinzu kommt, dass die Anzahl der Matches nur einen Bruchteil der Paare betrifft, weshalb die Ground Truth ein deutliches Ungleichgewicht aufweisen wird. Ein weiteres Problem ist, dass in diesem Prozess Fehler gemacht werden. Dabei entstehen die Fehler nicht bei den offensichtlichen Match und Non-Matches, sondern meist in Paaren, die auch für den Menschen nur schwer zu bewerten sind. Des Weiteren kann es zu unterschiedlichen Klassifizierungen je nach Prüfer kommen und auch derselbe Prüfer kann je nach Gemütslage und Konzentrationslevel unterschiedliche Aussagen über dasselbe Paar treffen.

Vogel et al. [35] haben deshalb einen sogenannten *Annealing Standard* entwickelt, welcher das Erstellen einer Ground Truth über einen iterativen Prozess vereinfachen sollen. Dabei wird zunächst mit einem Klassifikatoren eine Baseline erzeugt, die den Annealing Standard darstellt. Anschließend werden mit einem weiteren Klassifikatoren, welcher der vorherige mit anderen Parametern sind kann, Paare erzeugt und mit der Baseline verglichen. Die Übereinstimmung der beiden bildet den neuen Annealing Standard. Die übrigen Paare werden zu manuellen Inspektion Prüfern vorgelegt und die dadurch erzeugten Matches und Non-Matches werden mit dem Annealing Standard verschmolzen. Diese Iteration wird solange wiederholt, bis das Delta der Klassifikatoren einen bestimmten Maximalwert an Paaren unterschreitet.

Ein weiteres Verfahren haben Kejriwal & Mirankern [17] entwickelt. Ihre Idee ist es eine Menge schwach klassifizierter Daten zu generieren. Dabei werden sowohl positive, als auch negative Datensatzpaare klassifiziert. Über zwei Schranken kann der Benutzer festlegen wie ähnlich (obere Schranke ut) bzw. wie verschieden (untere Schranke lt) die Paare sein sollen. Anschließend wird ein Blocking der Daten per Standard Blocking und Sorted Neighborhood durchgeführt. Zunächst werden dazu alle Attribute jedes Datensätzes in Token zerlegt. Anhand der Token wird das Standard Blocking durchgeführt, wobei jeder Datensatz in mehreren Blöcken vertreten sein kann. Um innerhalb der Blöcke den Paarvergleichsaufwand zu reduzieren, wird zusätzlich ein Fenster der Größe c über den Block geschoben, was der Sorted Neighborhood entspricht. Nachdem das Fenster über jeden Block geschoben wurde, steht die Menge möglicher Kandidatenpaare fest. Diese Paare werden nun mit der TF/IDF-Ähnlichkeit (sim) [12] verglichen. Diese ermöglicht, nachdem die TF/IDF Statistik über die kompletten Daten erfasst wurde, eine Komplexität von $O(1)$, da lediglich die Werte des Paares nachgeschlagen werden müssen. Ist $sim \geq ut$ wird das Paar positiv klassifiziert. Ebenso, ist $sim < lt$ wird ein Paar

negativ klassifiziert. Damit die Menge der klassifizierten Daten nicht beliebig groß wird, kann der Anwender festlegen, wie viele positive max_p bzw. negative Paare max_n maximal erzeugt werden sollen. Von allen positiven Paaren werden abschließend die besten d ausgewählt, aber maximal max_p. Analog werden ebenfalls die besten nd negative Paare gewählt. Bei den negativen Paaren soll dadurch verhindern, dass lediglich Paare mit $sim \approx 0.0$ ausgewählt werden, da diese für gewöhnlich zu niedrigen Klassifikationsraten führen. Die Gesamtkomplexität des Algorithmus ist $O(n + nm + nm)$, welcher sich in die Erzeugung der TF/IDF Statistik ($O(n)$) , die Erzeugung der Blöcke über m Attribute ($O(nm)$) und die Erzeugung der Kandidatenpaare $O(nm)$ gliedert.

- Werden schnell große Datensätze mit entsprechender Ground Truth benötigt„ bieten sich synthetisch generierte Datensätze an. Damit diese repräsentativ sind, sollten Sie die gleichen Attribute haben wie die echten Datensätze. Dazu wird eine Datenbank möglicher Attributswerte benötigt, welche der Generator verwenden soll. Zusätzlich gibt es Parameter, um die Größe des Datensatzes und Anzahl der Duplikate, die Häufigkeitsverteilung der einzelnen Attribute und die Modifikationen der Duplikate gegenüber dem Original, in typographische, OCR oder phonetisch Fehler, zu bestimmen. Beispiele solcher Datensätze finden sich in Abschnitt 2.6.

- Anstatt synthetische Datensätze zu generieren und anschließend Fehler einzufügen, ist stattdessen auch möglich in einen bestehenden Datensatz Fehler einzubauen und diese als entsprechende Ground Truth zu verwenden. Dadurch werden allerdings die tatsächlichen Matches unterschlagen, was zu Konflikten bei der Entity Resolution führen kann.

2.5.1 Qualitätsmaße

Ist zu einem Datensatz die Ground-Truth verfügbar, so können die klassifizierten Datensätze einer der Kategorien in Tabelle 2.1 zugeordnet werden.

- True Positives (TP), sind alle Paare, die als Matches klassifiziert wurden und nach Ground Truth tatsächlich Matches sind.
- False Positives (FP), sind alle Paare, die als Matches klassifiziert wurden aber keine sind.
- False Negatives (FN), sind alle Paare, die als Non-Matches klassifiziert wurden aber tatsächlich Matches sind.
- True Negatives (TN), sind alle Paare, die als Non-Matches klassifiziert wurden und auch tatsächlich zwei verschiedene Entitäten identifizieren.

Das Ergebnis eines idealen Klassifikatoren ist, dass soviele Matches wie möglich True Positives sind und die Anzahl der False Positives, sowie False Negatives klein ist. Auf Basis

		Predicted classes	
		Match	Non-Match
Actual Matches	Match	True Positives (TP)	False Negatives (FN)
	Non-Match	False Positives (FP)	True Negatives (TN)

Tabelle 2.1 Matrix mit den vier Klassifikationszuständen. TP wenn tatsächliches und klassifiziertes Match, FN wenn tatsächlich Non-Match, aber klassifiziert als Match, FP wenn tatsächlich Match, aber klassifiziert als Non-Match und TN wenn tatsächliches und klassifiziertes Non-Match.

der vier Klassifikationsklassen können Qualitätsmaße bestimmt werden. Die folgende Liste zeigt die beliebtesten Methoden und erklärt ihre Stärken und Schwächen.

- *Accuracy.*

$$acc = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.2}$$

  Die Genauigkeit ist ein weit verbreitetes Qualitätsmaß für Binär- und Multi-Klassen Probleme im Maschine-Learning Bereich. Die Accuracy ist nützlich in Situationen, in welchen die Klassen möglichst gleich verteilt sind. Für Entity Resolution ist dieses Maß daher nur bedingt geeignet, da zwischen Matches und Non-Matches fast immer ein Ungleichgewicht zugunsten von Non-Matches besteht. Daher sind die meisten klassifizierten Ergebnisse True Negatives, welche die Gleichung dominieren. Dadurch wird fälschlicherweise, trotz weniger True Positives und vieler False Positives, sowie False Negatives, eine hohe Accuracy gemessen.

- *Precision.*

$$prec = \frac{TP}{TP + FP} \tag{2.3}$$

  Precision wird oft als Qualitätsmaß vor Suchergebnisse genommen, da es den Anteil der True Positives in den Matches berechnet. Für Entity Resolution misst die Precision wie viele Matches (TP + FP) ein Klassifikatoren korrekt bestimmt hat.

- *Recall.*

$$rec = \frac{TP}{TP + FN} \tag{2.4}$$

  Recall misst den Anteil der tatsächlichen Matches (TP + FN), welche korrekt (TP) als Matches klassifiziert wurden. Zwischen Recall und Precision gibt es einen Kompromiss. Beispielsweise kann der Recall verbessert werden, indem die Precision abgesenkt bzw. die Precision verbessert indem der Recall gesenkt wird.

- *F-measure*. Auch bekannt als *f-score* oder *f_1-score*.

$$fmeas = 2 \cdot \left(\frac{prec \cdot rec}{prec + rec} \right) \tag{2.5}$$

  Das F-measure berechnet das harmonische Mittel zwischen Precision und Recall. Eine guter F-measure Wert ist daher ein Kompromiss zwischen den beiden.

Die oben genannten Qualitätsmaße berechnen alle einen exakten Wert für die Qualität eines Klassifikators. Aus den bekannten Verfahren für Blocking, Vergleich und Klassifizierung geht hervor, dass diese eine Reihe von Parametern haben, um das Ergebnis zu kalibrieren. Deshalb ist es sinnvoll eine Reihe von Werten zu erzeugen, um diese miteinander zu vergleichen. Ein solcher Vergleich funktioniert am einfachsten per Visualisierung. Die folgenden drei Visualisierungen werden dazu oft verwendet:

- *Precision-recall Graph*. Diese Visualisierung zeigt den Kompromiss zwischen Precision und Recall. Für jede Parametereinstellung eines Klassifikators wird ein Punkt im Graph erzeugt. Dabei ist die X-Achse stets der Recall und die Y-Achse die Precision. Durch den Kompromiss startet die Kurve meist in der oberen linken Ecke mit hoher Precision und niedrigen Recall und endet in der linken unteren Ecke mit entgegengesetzten Werten. Dabei ist das Ziel die Kurve möglichst Nahe an die linke obere Ecke zu bekommen, in welcher Precision und Recall maximal sind.
- *F-measure Graph*. Anstatt zwei Qualitätmaße gegeneinander zu zeichen, kann man diese auch zusammen im Bezug auf einen bestimmten Parameter darstellen. Der F-measure Graph, beispielsweise plottet Precision, Recall und F-measure gegen einen Parameter, wie etwa die akkumulierte Gesamtwahrscheinlichkeit bei den schwellenbasierten Klassifikatoren genutzt. Darüber kann dann abgelesen werden, bei welchem Wert (z.B. Schwelle) die beste Precision, der beste Recall und das beste F-measure erreicht wird.
- *ROC Kurve*. Wie der Precision-recall Graph vergleicht die Receiver-Operating-Characteristic (ROC) Kurve zwei Qualitätmaße. In diesem Fall die auf der X-Achse die False-Positve-Rate und auf der Y-Achse der Recall. Obwohl die ROC Kurve robust gegen ungleichgewichtige Klassen ist, so ist diese mit Vorsicht für Entity Resolution zu genießen, da die False Positive Rate die True Negatives miteinbezieht, hat die Kurve das Problem etwas zu optimistische zu sein. Verschiedene ROC Kurven verschiedener Klassifikatoren mit unterschiedlichen Parametern zu vergleichen, kann dennoch nützlich sein, um deren Qualität zu bewerten.

2.5.2 Komplexitätsmaße

Neben der Qualität bestimmt auch die Effektivität wie gut Entity Resolution Systeme funktionieren. Die offensichtlichste Art Effektivität zu messen ist, die Laufzeit des Ver-

fahrens zu messen und miteinander zu vergleichen. Allerdings ist dieser Ansatz abhängig von der genutzten Hardware und bietet keinen plattformübergreifenden Vergleich. Für die folgenden Maße müssen zunächst einige Mengen definiert werden. Zunächst wird in die Menge aller tatsächlichen Matches n_M und die Menge aller tatsächlichen Non-Matches n_N gegliedert. Dementsprechend ist $n_M + n_N = m \cdot n$ für Entity-Linking und $n_M + n_N = m(m-1)/2$ für Deduplizierung. Die Menge der durch Blocking gruppierten Datensatzpaare wird ebenso in Matches und Non-Matches geteilt und mit s_M bzw. s_N bezeichnet, wobei $s_M + s_N \leq n_M + n_N$.

- *Reduction Ratio*. Dieses Maß gibt an wie viele Datensatzpaare von einem Blockingverfahren generiert worden sind und setzt diese ins Verhältnis mit der Anzahl aller möglichen Datensatzpaaren, welche ohne Blocking generiert worden wären. Das Reduction Ratio ist definert als

$$rr = 1 - \left(\frac{s_M + s_N}{n_M + n_N} \right). \tag{2.6}$$

- *Pairs completeness*. Dieses Maß berechnet den Anteil der möglichen Matches. Es wird berechnet mit

$$pc = \frac{s_M}{n_M}. \tag{2.7}$$

  Pairs completeness ist mit dem Recall aus Formel 2.4 verwandt. Je geringer die Pairs completeness ist, desto geringer ist auch die Matchingqualität, da dieses Maß eine Obergrenze für einen möglichen Recall bestimmt. Denn tatsächliche Matches, die von einem Blocking Mechanismus nicht selektiert werden, können auch nicht klassifiziert werden. Zwischen Reduction Ratio und Pairs Completness gibt es einen offensichtlichen Kompromiss, je mehr Datensatzpaare erzeugt werden, desto mehr tatsächliche Matches können gefunden werden.
- *Pairs quality*. Dieses Maß berücksichtigt die Qualität eines Blockingverfahren, indem es selektierten tatsächlichen Matches in Relation mit mit allen selektierten Paaren stellt. Es wird berechnet mit

$$pq = \frac{s_M}{s_M + s_N}. \tag{2.8}$$

  Die Pairs quality ist verwandt mit der Precision aus Formel 2.3. Eine hohe Pairs qualtiy bedeutet, dass ein Blockingverfahren hauptsächlich Paare erzeugt, welche tatsächlich Matches sind. Auch hier gibt es ähnlich zu Precision und Recall einen Kompromiss zwischen Pairs completness und Pairs quality.

2.6 Datensätze

2.6.1 CORA

Der CORA Datensatz beinhaltet 1879 bibliographische Einträge über wissenschaftliche Veröffentlichungen aus dem Maschine Learning Bereich. Die Einträge bestehen aus Autoren, Titel, Publikationsjahr und Konferenz bzw. Journal. Insgesamt beinhaltet dieser Datensatz 64.577 Duplikate. Dieser Datensatz ist besonders schwierig zu Deduplizieren, da teilweise nur Initialen der Autoren vorhanden sind bzw. Attribute zusammengefügt oder getauscht wurden.

2.6.2 Abt-Buy & Amazon-GoogleProducts

Diese beiden Datensätze beinhalten Produkte aus dem Onlinehandel verschiedener Plattformen mit Name, Beschreibung, Hersteller und Preis. Der Abt-Buy Datensatz beinhaltet 2171 Einträge mit 1096 Duplikaten. Im Amazon-GoogleProducts Datensatz sind es 4587 Einträge mit 1299 Duplikaten.

2.6.3 DBLP-ACM & DBLP-Scholar

Diese beiden Datensätze beinhalten bibliografische Einträge mit Titel, Autor(en), Konferenz, und Jahr. Der DBLP-ACM Datensatz beinhaltet 4908 Einträge und 2223 Duplikate. Im DBLP-Scholar Datensatz sind sind 66877 Einträge mit 5346 Duplikaten. Dabei ist zu beachten, dass der DBLP-ACM Datensatz einfach zu klassifizieren ist, da ein Großteil der Daten durch eine Instanz gepflegt wird.

2.6.4 Restaurant

Der Restaurant Datensatz ist ein kleiner mit lediglich 864 Einträgen, welche aus Restaurantname, Adresse, Telefonnummer und der Küchenart bestehen. Es gibt insgesamt 112 Restaurantduplikate, welche doppelt vorkommen.

2.6.5 NCVR

Der NC Voter Registration (NCVR) Datensatz beinhaltet ca. 6 mio Datensätze aus dem Wählerverzeichnis des Bundestates North Carolina in den USA. Eine genaue Analyse des Datensatzes wurde von Christen [36] durchgeführt. Der Datensatz beinhaltet ca. 145.000 Duplikate zwischen zwei Einträgen, sowie 3.500 zwischen drei und mehr Einträgen. Die Zuordnung der Duplikate wurde dabei über die Wählerregistriernummer getätigt. Weitere Attribute sind Namenspräfix, Vorname, Zweiter, Vorname, Nachname, Namenssuf-

fix, Alter, Geschlecht, Rassenziffer, Ethnizitätsziffer, Strasse + Hausnummer, Stadt, Bundesland, Postleitzahl, Telefonnummer, Geburtsort und Registrierdatum.

2.6.6 Febrl

Die Febrl-Datensätze wurden synthetisch durch den Febrlgenerator erzeugt. Die Attributsdaten dafür liefert ein australisches Telefonbuch. Die generierten Einträge haben folgende Attribute: Kultur, Geschlecht, Alter, Geburtsdatum, Titel, Vorname, Nachname, Bundesland, Vorort, Postleitzahl, Hausnummer, Straße und Telefonnummer.

Zum Entwickeln:

- Febrl-4k-1k: 5.000 Einträge mit 1.000 Duplikaten zwischen zwei Datensätzen
- Febrl-9k-1k: 10.000 Einträge mit 1.000 Duplikaten zwischen zwei Datensätzen
- Febrl-90k-10k: 100.000 Einträge mit 10.000 Duplikaten zwischen zwei Datensätzen

Zum Evaluieren:

- 5.000.000 Einträge 100.000 Duplikate zwischen zwei und mehr und Attributsverteilung (Uniform, Poisson, Zipfian).

3 Analyse

3.1 DySimII

3.1.1 Problem: DNF-Blocking

Vermeiden von Blöcken mit max 1 var 1 und Kandidatenmenge max 1 var 1, da diese keinen Mehrwert bringen.

3.1.2 Problem: Kandidatenmenge

Der DySimII Index hat das Problem, dass er die Kandidatenmenge nicht gut kontrollieren kann. Daher müssen die Attribute, welche durch den DySimII indiziert werden sollen, mit bedacht gewählt werden. Attribute wie das Geschlecht, Nationalität oder Bundesland führen dazu, dass über den Record Index eine riesige Anzahl an Kandidatenpaaren selektiert werden. So werden beispielsweise, beim Überprüfen eines Datensatzes mit Deutscher Nationalität alle Deutschen als Kandidaten gewählt. Dadurch wird die Reduction Ratio des DySimII Algorithmus dramatisch verschlechtert.

Beispiel: Restaurant, Ferbl-9k, Ferbl-90k

Zwar kann über die Eingrenzung auf die Besten n bzw. durch das Setzen einer Schranke für die Mindestähnlichkeit ein Großteil der Kandidaten ausgeschlossen werden, dennoch muss bleibt die Reduction Ratio gleich, da jeden einzelnen die Ähnlichkeit aus dem Index geholt und abgeglichen werden muss

Lösung 1: Die offensichtlichste Lösung ist, Attribute, welche rießige Kandidatenmengen erzeugen nicht zu indizieren. Allerdings kann es durchaus vorkommen, dass die Konjunktion im DNF-Schema mit anderen Attributen Block Key Values erzeugt, die eine hohe Präzision ermöglichen. In diesem Fall landet das Attribut zwangsweise im Record Index.

Vgl.[8][15] State mit Sicherheit nicht gewählt!

Lösung 2: Die zweite Lösung setzt voraus, dass die Blockschlüssel so gewählt wurden, dass Kandidatenmengen nicht zu groß werden können. Das Löschen des Record Index ist keine Option, da dadurch die Zuordnung zu den eigentlichen Datzensätzen wegfällt. Um dennoch den Einträge im Record Index zu minimieren wird dieser in den Blocking Index verschoben. D.h. jedes Attribut eines Blockes verweist nun auf die Datensätze, welche nicht lediglich das gleiche Attribute haben, sondern den gleichen Blockschlüssel. Statt pauschal alle Deutschen bei einer Anfrage als Kandidaten zu selektieren, werden dadruch beispielsweise nur die Deutschen, die ein der selben Stadt wohnen und den gleichen Nachnamen haben ausgewählt.

Beispiel/Bild/Pseudo-Code

Dieses Verfahren erhöht zwar die Komplexität etwas von $O(?)$ auf $O(?)$ erhöht aber gleichzeitig die Reduction Ratio dramatisch.

Vgl. MDySimII, MDySimIII

3.2 Problem: Weak Labels

Dabei empfiehlt es sich die Daten vorher zu sortieren, um deterministische Ergebnisse zu erzielen.

3.3 Ähnlichkeitsmetriken

Aus der Vielfalt der möglichen Ähnlichkeitsmaße gibt es keines das allen anderen klar überlegen ist. Es ist daher sehr domainabhängig, welcher Algortihmus gute Ergebnisse liefert. Beim Vergleich von Datensätzen sind diese Domänen meist durch die unterschiedlichen Attribute getrennt. Daher ist es notwendig herauszufinden, für welches Attribute welche Ähnlichkeitsmetric besonders gut funktioniert. Daraus folgt das Problem der Vergleichbarkeit der Ähnlichkeitsmetriken. Für zwei Strings a und b liefert der Jaccard-Koeffizient beispielweise Werte zwischen 0 und 1, die Levenshtein-Distanz hingegen Werte zwischen 0 und $maxlen(a, b)$. Deshalb ist es notwendig die verschiedenen Ähnlichkeitsmaße zu normalisieren. Dafür wird das Intervall von 0 bis 1 gewählt, wobei 1 totale Übereinstimmung und 0 keine Übereinstimmung bedeutet.

3.3.1 Edit-distance

Die Edit-distance ist eine der beliebtesten und meistgenutzten Metriken, um die Ähnlichkeit zweier Strings zu bestimmen. Dabei bestimmt die Edit-distance die benötigten

Schritte um einen String in einen anderen zu überführen. Dafür werden die Tranformationen einfügen, löschen und ersetzen im klassischen Verfahren von Levenshtein und zusätzlich transponieren in der Erweiterung von Damerau eingesetzt. Das Ergebnis sind die Anzahl der benötigten Transformationen. Diese Anzahl alleine ist noch kein genaues Maß zur Bestimmung der Ähnlichkeit, da zwei Transformationen in einem kürzeren String kritischer sind als in einem langen. Um die Ähnlichkeit zu normalisieren gibt es zwei Möglichkeiten. Entweder auf Basis des Transformationspfades oder der Stringlänge. Es ist zu beachten das beide Methoden nicht der Dreiecksungleichung stand halten.

Eine Erweiterung der Edit-distance ist die Tranformationen zu gewichten. Dazu wird jedem Transformationstyp ein positiver reeller Kostenfaktor zugewiesen, mit welchem die Anzahl seiner Transformationen gewichtet wird. Durch die Gewichtung ist es allerdings nicht mehr möglich wie bei uniformer Gewichtung zu Normalisieren. Eine Möglichkeit, welche aus die Dreiecksungleichung erfüllt stellen Yujian & Bo in [37] vor.

Durchprobieren der Gewichte, welche Ergebnisse liefert das? Ist es überhaupt Aussagekräfig?

3.4 Berechnung der Metriken für Real-time ER

Können nicht pauschal auf den Index berechnet werden, sondern sind Kennziffern, die kontinuierlich, mit dem verarbeiten von Anfragen, berechnet und akkumuliert werden.

- Pair Completeness
- Reduction Ratio

Beispiel eines Code-listings

Auflistung 3.1 Listing caption

```
main :: IO ()
main = putStrLn "Hello World!"
```

Literaturverzeichnis

[1] Fellegi, Ivan P.; Sunter, Alan B.: A Theory for Record Linkage. In: *Journal of the American Statistical Association* Bd. 64 (1969), Nr. 328, S. 1183–1210

[2] Köpcke, Hanna; Rahm, Erhard: Frameworks for Entity Matching: A Comparison. In: *Data & Knowledge Engineering* Bd. 69 (2010), Nr. 2, S. 197–210

[3] Kolb, Lars: *Effiziente MapReduce-Parallelisierung von Entity Resolution-Workflows*, University of Leipzig, Dissertation, 2014

[4] Kolb, Lars; Rahm, Erhard: Parallel Entity Resolution with Dedoop. In: *Datenbank-Spektrum* Bd. 13 (2013), Nr. 1, S. 23–32

[5] Malhotra, Pankaj; Agarwal, Puneet; Shroff, Gautam: Graph-Parallel Entity Resolution Using LSH & IMM. In: *EDBT/ICDT Workshops*, 2014, S. 41–49

[6] Whang, S. E.; Marmaros, D.; Garcia-Molina, H.: Pay-As-You-Go Entity Resolution. In: *IEEE Transactions on Knowledge and Data Engineering* Bd. 25 (2013), Nr. 5, S. 1111–1124

[7] Ramadan, Banda; Christen, Peter; Liang, Huizhi; Gayler, Ross W.: Dynamic Sorted Neighborhood Indexing for Real-Time Entity Resolution. In: *J. Data and Information Quality* Bd. 6 (2015), Nr. 4, S. 15:1–15:29

[8] Christen, Peter; Gayler, Ross: Towards Scalable Real-Time Entity Resolution Using a S Imilarity-Aware Inverted Index Approach. In: *Proceedings of the 7th Australasian Data Mining Conference - Volume 87*, *AusDM '08*. Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2008 — ISBN 978-1-920682-68-2, S. 51–60

[9] Köpcke, Hanna; Thor, Andreas; Rahm, Erhard: Evaluation of Entity Resolution Approaches on Real-World Match Problems. In: *Proceedings of the VLDB Endowment* Bd. 3 (2010), Nr. 1-2, S. 484–493

[10] Draisbach, Uwe; Naumann, Felix: A Comparison and Generalization of Blocking and Windowing Algorithms for Duplicate Detection. In: *Proceedings of the International Workshop on Quality in Databases (QDB)*, 2009, S. 51–56

[11] Christen, P.: A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication. In: *IEEE Transactions on Knowledge and Data Engineering* Bd. 24 (2012), Nr. 9, S. 1537–1555

[12] Aizawa, A.; Oyama, K.: A Fast Linkage Detection Scheme for Multi-Source Information Integration. In: *International Workshop on Challenges in Web Information Retrieval and Integration*, 2005, S. 30–39

[13] Hernández, Mauricio A.; Stolfo, Salvatore J.: The Merge/Purge Problem for Large Databases. In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, *SIGMOD '95*. New York, NY, USA : ACM, 1995 — ISBN 978-0-89791-731-5, S. 127–138

[14] Cohen, William W.; Richman, Jacob: Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *KDD '02*. New York, NY, USA : ACM, 2002 — ISBN 978-1-58113-567-1, S. 475–480

[15] Ramadan, Banda; Christen, Peter; Liang, Huizhi; Gayler, Ross W.; Hawking, David: Dynamic Similarity-Aware Inverted Indexing for Real-Time Entity Resolution. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining* : Springer, 2013, S. 47–58

[16] Li, Shouheng; Liang, H.; Ramadan, Banda; others: *Two Stage Similarityaware Indexing for Large Scale Realtime Entity Resolution* : AusDM, 2013 – Maßstab

[17] Kejriwal, M.; Miranker, D. P.: An Unsupervised Algorithm for Learning Blocking Schemes. In: *2013 IEEE 13th International Conference on Data Mining*, 2013, S. 340–349

[18] Gu, Quanquan; Li, Zhenhui; Han, Jiawei: Generalized Fisher Score for Feature Selection. In: *arXiv preprint arXiv:1202.3725* (2012)

[19] Elmagarmid, A. K.; Ipeirotis, P. G.; Verykios, V. S.: Duplicate Record Detection: A Survey. In: *IEEE Transactions on Knowledge and Data Engineering* Bd. 19 (2007), Nr. 1, S. 1–16

[20] Levenshtein, Vladimir I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. In: *Soviet Physics Doklady*. Bd. 10, 1966, S. 707–710

[21] Damerau, Fred J.: A Technique for Computer Detection and Correction of Spelling Errors. In: *Communications of the ACM* Bd. 7 (1964), Nr. 3, S. 171–176

[22] Needleman, Saul B.; Wunsch, Christian D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. In: *Journal of Molecular Biology* Bd. 48 (1970), Nr. 3, S. 443–453

[23] Waterman, Michael S.; Smith, Temple F.; Beyer, William A.: Some Biological Sequence Metrics. In: *Advances in Mathematics* Bd. 20 (1976), Nr. 3, S. 367–387

[24] Smith, T. F.; Waterman, M. S.: Identification of Common Molecular Subsequences. In: *Journal of Molecular Biology* Bd. 147 (1981), Nr. 1, S. 195–197

[25] Monge, Alvaro E.; Elkan, Charles; others: The Field Matching Problem: Algorithms and Applications. In: *KDD*, 1996, S. 267–270

[26] Cohen, William W.: WHIRL: A Word-Based Information Representation Language. In: *Artificial Intelligence* Bd. 118 (2000), Nr. 12, S. 163–196

[27] Gravano, Luis; Ipeirotis, Panagiotis G.; Koudas, Nick; Srivastava, Divesh: Text Joins in an RDBMS for Web Data Integration. In: *Proceedings of the 12th International Conference on World Wide Web* : ACM, 2003, S. 90–101

[28] Sonnenburg, Sören; Rätsch, Gunnar; Rieck, Konrad: Large Scale Learning with String Kernels. In: *Large Scale Kernel Machines* (2007), S. 73–103

[29] Lodhi, Huma; Saunders, Craig; Shawe-Taylor, John; Cristianini, Nello; Watkins, Chris: Text Classification Using String Kernels. In: *Journal of Machine Learning Research* Bd. 2 (2002), Nr. Feb, S. 419–444

[30] Christen, Peter: *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection* : Springer Science & Business Media, 2012 – Maßstab

[31] Boser, Bernhard E.; Guyon, Isabelle M.; Vapnik, Vladimir N.: A Training Algorithm for Optimal Margin Classifiers. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*. New York, NY, USA : ACM, 1992 — ISBN 978-0-89791-497-0, S. 144–152

[32] Bilenko, Mikhail; Mooney, Raymond J.: Adaptive Duplicate Detection Using Learnable String Similarity Measures. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '03*. New York, NY, USA : ACM, 2003 — ISBN 978-1-58113-737-8, S. 39–48

[33] Christen, Peter: Automatic Record Linkage Using Seeded Nearest Neighbour and Support Vector Machine Classification. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* : ACM, 2008, S. 151–159

[34] Arasu, Arvind; Götz, Michaela; Kaushik, Raghav: On Active Learning of Record Matching Packages. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* : ACM, 2010, S. 783–794

[35] Vogel, Tobias; Heise, Arvid; Draisbach, Uwe; Lange, Dustin; Naumann, Felix: Reach for Gold: An Annealing Standard to Evaluate Duplicate Detection Results. In: *Journal of Data and Information Quality* Bd. 5 (2014), Nr. 1-2, S. 1–25

[36] Christen, Peter: Preparation of a Real Temporal Voter Data Set for Record Linkage and Duplicate Detection Research (2013)

[37] Yujian, L.; Bo, L.: A Normalized Levenshtein Distance Metric. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* Bd. 29 (2007), Nr. 6, S. 1091–1095

Abbildungsverzeichnis

Auflistungsverzeichnis

Erklärung

Erklärung gem. ABPO, Ziff. 6.4.3

Ich versichere, dass ich die Master-Thesis selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Wiesbaden, 17.02.2017

Kevin Sapper

Hiermit erkläre ich mein Einverständnis mit den im Folgenden aufgeführten Verbreitungsformen dieser Master-Thesis:

Verbreitungsform	**ja**	**nein**
Einstellung der Arbeit in die Bibliothek der Hochschule RheinMain	√	
Veröffentlichung des Titels der Arbeit im Internet	√	
Veröffentlichung der Arbeit im Internet	√	

Wiesbaden, 17.02.2017

Kevin Sapper