Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Speed up Entity Resolution with Bit Arrays

Big Data Praktikum

Einleitung:
Das Ziel des Praktikums besteht darin, durch experimentelle Auswertungen zwei Ansätze zur Entity-Resolution miteinander zu vergleichen. Der erste Vertreter ist dabei die Jaccard-Ähnlichkeit von N-Grammen. Dieses Vorgehen liefert gute Ergebnisse, lässt jedoch aus Sicht der Ausführungszeit insbesondere für große Datenmengen Wünsche offen.

Aus diesem Grund wird die zweite Strategie auf der Basis von Bloom-Filtern realisiert. Bloom-Filter sind dabei Datenstrukturen, die approximativ und sehr effizient entscheiden können, ob ein gegebenes Element in einer Menge enthalten ist. Durch diese Objekte sollen damit die für die Jaccard-Ähnlichkeit notwendigen Berechnungen effizient approximiert werden.

Die Hoffnung besteht dabei darin, dass dieser Ansatz möglichst wenig Genauigkeit einbüßt, und dabei signifikante Verbesserungen in Bezug auf die Ausführungszeit liefert.

Leipzig, Mai 2017

vorgelegt von

Maik Fröbe	2515360
Moritz Engelmann	2527532

Betreuer:

Ziad Sehili
Fakultät für Mathematik und Informatik
Abteilung Datenbanken

1 Entity-Resolution

Zwei eigentlich identische Objekte können sich durch verschiedene Schreibweisen, Abkürzungen, oder der Verwendung unterschiedlicher Persistierungen unterscheiden. So können z.B. die beiden Instanzen „Peter Müller“und „Peter Mueller“die selbe Person darstellen, obwohl sie unterschiedlich geschrieben sind.

Ziel der Entity-Resolution ist es durch automatische Verfahren herauszufinden, ob zwei Instanzen aus verschiedenen Datensätzen dasselbe Objekt in der realen Welt repräsentieren. Das ist z.B. nötig, wenn man verschiedene Datenquellen zusammenführen oder eine Datenbereinigung vornehmen möchte. Dazu ist es nötig, geeignete Methoden zu verwenden bzw. zu entwickeln, welche identische Instanzen möglichst schnell und genau identifizieren können. Im folgenden sollen zwei Verfahren vorgestellt und in der weiteren Arbeit verglichen werden.

1.1 Entity-Resolution mit N-Grammen

Für die Berechnung von Ähnlichkeiten mittels N-Grammen müssen aus den zu vergleichenden Instanzen zuerst N-Gramme erzeugt werden. Beispielsweise werden aus den beiden Instanzen „Peter Müller“und „Peter Mueller“die folgenden Trigramme erzeugt:

- A = trigrams(„Peter Müller“) = { „Pet“, „ete“, „ter“, „Mül“, „üll“, „lle“, „ler“}
- B = trigrams(„Peter Mueller“) = { „Pet“, „ete“, „ter“, „Mue“, „uel“, „ell“, „lle“, „ler“}

Mittels der Jaccard-Ähnlichkeit kann nun die Ähnlichkeit von A und B berechnet werden:

$Jaccard(A, B) = \frac{A \cap B}{A \cup B} = \frac{5}{10} = \frac{1}{2}$

Wenn man nun einen Schwellwert festlegt und die berechnete Ähnlichkeit ist größer oder gleich diesem Schwellwert, so geht man davon aus, das es sich um die selben Instanzen handelt.

1.2 Entity-Resolution mit N-Grammen unter Verwendung von Bloom-Filtern

Eine zentrale Rolle bezüglich der Performance der Entity-Resolution stellt der Aufwand für die Bestimmung der Ähnlichkeit zweier Objekte dar. Insbesondere ein wie oben vorgestellter Ansatz, der auf der Berechnung des Jaccard-Index zwischen Mengen basiert, ist aufgrund teurer Schnitt- und Vereinigungs-Berechnungen aufwändig. Ein alternativer Ansatz stellt eine approximative Lösung, zum Beispiel unter Verwendung von Bloom-Filtern, dar.

1.2.1 Bloom-Filter

Ein Bloom-Filter ist eine Datenstruktur, die effizient - aber dafür approximativ - entscheidet, ob ein beliebiges Element in einer, durch den Bloom-Filter repräsentierten, Menge enthalten ist. Für einen Bloom-Filter A_{BF} einer Menge A gilt dabei stets: $x \in A \Rightarrow x \in A_{BF}$.

Realisiert wird ein solches Objekt mit einem Bit-Array der Länge n, sowie einer Anzahl k unterschiedlicher Hashfunktionen. Diese Hashfunktionen haben jeweils einen Wertebereich von 0 bis $n-1$. Initial ist der Bit-Array mit Nullen gefüllt.

Anschließen werden die Elemente der verwalteten Menge eingefügt. Dazu werden mittels der k verschiedenen Hashfunktionen k Hashwerte ermittelt. Jeder dieser Hashwerte wird nun auf die entsprechende Position im Bit-Array übertragen. Dazu werden diese Positionen auf 1 gesetzt. Wenn eine Position schon durch einen anderen Hashwert auf 1 gesetzt worden ist, kommt es zu einer Kollision und diese Position wird auf 1 belassen.

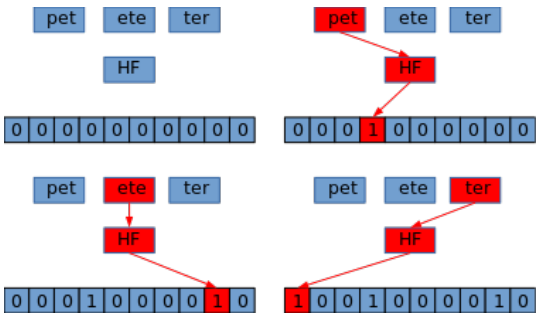

Abbildung 1: Einfügen vom Trigrammen in einen Bloom-Filter

Um zu überprüfen, ob ein bestimmtes Element im Bloom-Filter enthalten ist, berechnet man seine k Hashwerte. Ist an einer der k Stellen im Bit-Array eine 0, so ist dieses Element nicht in der Menge enthalten. Steht jedoch an allen Positionen eine 1, so ist der Wert mit einer hohen Wahrscheinlichkeit in der vom Bloom-Filter repräsentierten Menge enthalten. Aufgrund möglicher Kollisionen kann man im Allgemeinen weder exakt bestimmen, wie viele dis-

junkte Elemente ein Bloom-Filter beherbergt, noch sicher sagen, dass ein Element wirklich enthalten ist.

Abbildung 1 visualisiert das Einfügen der Trigramme des Wortes „Peter“. Es wird veranschaulicht, wie das zuerst leere Bit-Array nach und nach gefüllt wird. Zur Vereinfachung wird nur eine Hashfunktion verwendet.

1.2.2 Einsatzszenario in der Entity-Resolution

Ein Bloom-Filter ist unter den oben beschriebenen Eigenschaften in der Lage, eine Menge zu repräsentieren. Des weiteren lässt sich ein logisches UND beziehungsweise ein logisches ODER zwischen den Bit-Arrays, die den Kern einer jeden solchen Datenstrukturen bilden, extrem performant bilden. Dieser Gedanke führt zu der zentralen Fragestellung dieses Praktikums:
Lässt sich die Berechnung der Jaccard-Ähnlichkeit zwischen zwei Mengen durch den Einsatz von Bloom-Filtern optimieren? Wenn ja, zu welchem Preis?

Anschaulich betrachtet bedeutet dies, dass für zwei Mengen A und B - mit entsprechenden Bloom-Filtern A_{BF} bzw. B_{BF} - der Jaccard-Index gegeben ist durch:

$$Jaccard(A_{BF}, B_{BF}) = \frac{|A_{BF} \wedge B_{BF}|}{|A_{BF} \vee B_{BF}|} = \frac{\text{geschätzte Anzahl Elemente in } A_{BF} \wedge B_{BF}}{\text{geschätzte Anzahl Elemente in } A_{BF} \vee B_{BF}}$$

Abbildung 2 zeigt diese Funktionsweise für die Berechnung der Ähnlichkeit mittels Bloom-Filtern. Dabei werden die beiden Instanzen „Peter Müller“ und „Peter Mueller“ vergleichen. Wenn man beide Bit-Arrays vergleicht, sieht man, das die Anzahl der UND's gleich 5 und die Anzahl der OR's gleich 10 ist. Dementsprechend würde die Jaccard-Ähnlichkeit $\frac{5}{10}$ entsprechen.

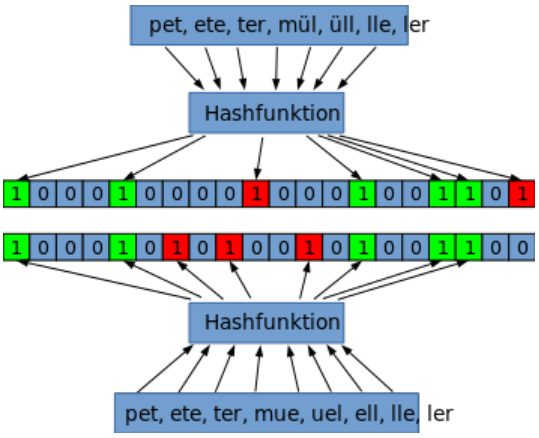

Abbildung 2: Vergleich mit Bloom-Filtern

1.2.3 Gefahren beim Einsatz eines Bloom-Filters

Für eine approximative Lösung des Entity-Resolution-Problems über den vorgestellten Bloom-Filter-Ansatz wäre es vorteilhaft, wenn eine allgemeine Beziehung des Jaccard-Indizes für Mengen zu denen mit Bloom-Filtern existiert. Insbesondere ein Bezug wie $Jaccard(A, B) \leq Jaccard(A_{BF}, B_{BF})$ wäre wünschenswert.

Leider existieren im Allgemeinen keine derartigen Beziehungen. Als Ursachen dafür können Kollisionen an verschiedenen Stellen im Workflow identifiziert werden.
So bewirken diese beim Aufbau eines Bloom-Filters, dass eine später repräsentierte Vereinigung kleiner, und der Schnitt größer wird. Dem entgegen bewirken Kollisionen bei der Bildung der Vereinigung von Bloom-Filtern (logisches oder) eine unberechtigte Vergrößerung der Mächtigkeit der Vereinigung, während bei dem Schnitt (logisches und) keine derartigen Effekte auftreten.

2 Vorgehen

Im Rahmen der Praktikumsaufgabe werden verschiedene Vorgehensweisen zur Lösung des Entity-Resolution-Problems untersucht. Um die Vergleichbarkeit der Ansätze zu wahren und eine größere Bandbreite von Auswertungen durchführen zu können, wollen wir stets Standard-Java-Klassen verwenden. Dabei sind insbesondere verschiedene Implementierungen der List- und Set-Schnittstellen aus dem Collections-Framework sowie MessageDigest und BitSet relevant.

Zur Lösung der Aufgabe implementieren wir ein Programm (z.Bsp.: PersonEntityResolution.jar), welches für eine Ausführung die ähnlichen Personen in zwei Personendatensätzen findet. Dabei soll das Programm durch seine Parametrisierung flexibel aus den oben vorgestellten Lösungsansätzen, inklusive relevanter Details, wählen können.

Zur vereinfachten Auswertung wird dieses Programm dann durch ein Skript (z.Bsp.: run_evaluations.sh) sequentiell für alle interessanten Konstellationen ausgeführt. Die entstehenden Metadaten sowie Ergebnisse werden aufgehoben. Abbildung 3 stellt dieses Vorgehen schematisch dar.

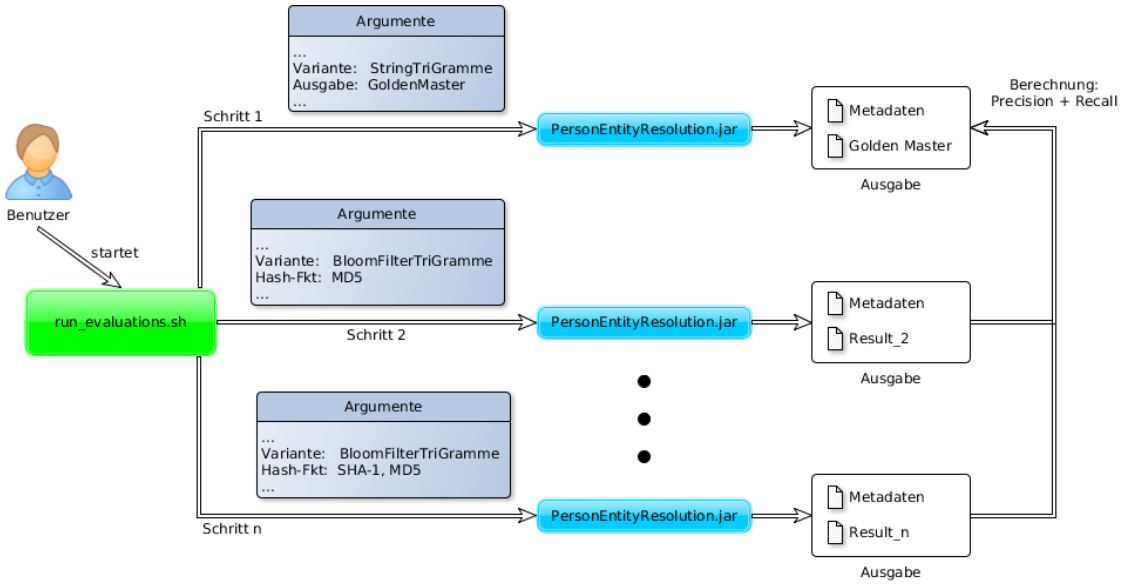

Abbildung 3: Workflow zur Evaluierung verschiedener Entity-Resolution-Ansätze

So wird im ersten Schritt die Entity-Resolution in der ursprünglichen Variante mit String-N-Grammen durchgeführt. Die dabei gefundenen Paare von ähnlichen Personen werden für die Ausführungen anderer Konstellationen als Golden-Master abgelegt.

Alle weiteren Ausführungen (2 bis n) stellen nun verschiedene Varianten der Entity-Resolution mit Bloom-Filtern dar. Die Menge der anwendbaren Parameter ist vielfältig (Abbildung 4 gibt eine Skizze), was eine ausführliche Auswertung ermöglicht. Für die Ergebnisse wird dabei stets mittels Precision und Recall gemessen, wie nah sie an die Wahrheit des Golden-Masters heranreichen.

Zusätzlich müssen weitere Kennzahlen erhoben werden. Dazu gehören im Wesentlichen gemonitorte Details wie Ausführungszeiten und der benötigte Speicherplatz. Auf Basis der so gesammelten Informationen können ausgiebige Vergleiche der verwendeten Ansätze durchgeführt werden.

Gleichzeitig ist der in Abbildung 3 dargestellte Ansatz flexibel genug, um ohne zusätzlichen Aufwand Tests mehrfach auszuführen und somit Seiteneffekte auszuschließen.

N-Gramme

Parameter	Wertebereich
N	TBA
Set	{ HashSet, LinkedHashSet }

Bloom-Filter

Parameter	Wertebereich
N	TBA
Attributschachtelung	{ Ja, Nein }
Länge Bitliste	
Hash-Funktionen	{MD5, SHA, ...}

Abbildung 4: Skizze möglicher Parameter

3 Ziele der Implementierung

Die Hauptaufgabe des Praktikums besteht in der Realisierung des in Abschnitt 2 beschriebenen Programms, sowie der Erstellung der relevanten Auswertungen. Die bereitzustellenden Aspekte sind dabei in Pflicht- und Optionale Bereiche aufgegliedert, die im folgenden beschrieben werden.

3.1 Pflicht: Vergleich verschiedener Strategien

Das im vorigen Abschnitt beschriebene Vorgehen beruht auf der Idee, dass verschiedene Implementierungen einer Ähnlichkeitsfunktion zur Entity-Resolution austauschbar in dem gleichen Workflow ausgeführt werden können. Um dies zu gewährleisten, wird insbesondere auch der Import der Daten durch die Ähnlichkeitsfunktion so beeinflusst, dass beim Import die Daten bereits in der von der verwendeten Strategie bevorzugten Datenstruktur bereitgestellt werden.

Tabelle 5 beschreibt alle Implementierungen von Ähnlichkeitsfunktionen, die wir im Rahmen der Aufgabe vergleichen wollen. Viele Ansätze sind redundant vertreten, da wir der Auffassung sind, dass kleine Implementierungsdetails schwer abzuschätzende Auswirkungen haben können. Des weiteren existieren verschiedene Implementierungen, die entweder keine oder unvollständige Berechnung ausführen. Die Motivation dafür besteht in der Möglichkeit, den Workflow hinsichtlich eines isolierten Aspekts zu bewerten.

3.2 Optional: Parallelisierung

- Ausführen des Workflows in Flink

Komponentenbasierte Personen-Ähnlichkeit Ähnlichkeit entspricht arithmetischem Mittel der Jaccard-Ähnlichkeiten der N-Gramm-Mengen der Geburtstags-, Name-, Address-Komponenten.
konstante Abbildung auf 0**:** Fokus: Bewertung Import + Transformation
Jaccard-KISS-Ähnlichkeit: Fokus: Lesbarkeit
Jaccard-Ähnlichkeit: Fokus: Performance
Dritt-Bibliothek-Bloom-Ansatz: Vergleich: Eigenimplementierung vs Verbreitete
Bloom-Ansatz (Eigenimplementierung): Mit BitSet
Optional: Bloom-Ansatz (Eigenimplementierung): Mit Long-Array
Optional: Bloom-Ansatz (Eigenimplementierung): Mit Boolean-Array

Personen-Ähnlichkeit Alle Komponenten einer Person werden in einer gemeinsamen Menge aufbewahrt, wobei die Jaccard-Ähnlichkeiten dieser Menge die Ähnlichkeit der Personen entspricht. Ähnlichkeitsfunktionen haben die Bedeutung von oben.
konstante Abbildung auf 0
Jaccard-KISS-Ähnlichkeit
Jaccard-Ähnlichkeit
Dritt-Bibliothek-Bloom-Ansatz
Bloom-Ansatz (Eigenimplementierung)
Optional: Bloom-Ansatz (Eigenimplementierung)
Optional: Bloom-Ansatz (Eigenimplementierung)

Abbildung 5: Übersicht der zu implementierenden Ähnlichkeitsfunktionen