Verfahrensskizze
zur
Identifizierung und Ähnlichkeitsprüfung digitaler Personenidentitäten auf Basis von n-Grammen und Bloomfiltern

Autor(en):

Helge Kampf

HK-Businessconsulting
Kelloggstraße 24
22045 Hamburg

Nummer:

Standort:

Art: Technische Notiz

Einstufung: offen

Thema/Projekt: Identity Management

Version: 0.14

Stand: 3. April 2021

Status:								
	x	in Bearbeitung		vorgelegt		fertig gestellt		freigegeben

Änderungshistorie:

Datum	Bearbeiter	Version	Änderungen	Bemerkung
22.01.2007	H. Kampf	0.01	Neuerstellung	
17.09.2007	H. Kampf	0.09	Vorlage als Diskussionsgrundlage	
23.10.2007	H. Kampf	0.10	Redaktionelle Überarbeitung	
25.06.2008	H. Kampf	0.11	Redaktionelle Überarbeitung	
08.05.2010	H. Kampf	0.12	Redaktionelle Überarbeitung	
29.01.2013	H. Kampf	0.13	Redaktionelle Überarbeitung	
06.04.2013	H. Kampf	0.14	Redaktionelle Überarbeitung	

© Helge Kampf, 2007-2021
HK-Businessconsulting, Hamburg

Inhalt

© Helge Kampf, 2007-2021
HK-Businessconsulting, Hamburg

0 Zusammenfassung

Die hier vorliegende Skizze beschreibt ein Verfahren, das eine Duplikatserkennung in Identitätenrepositories mit hoher Wahrscheinlichkeit ermöglicht.

Grundlage hierfür ist die Zerlegung relevanter, Identitäten beschreibender Attribute in n-Gramme mit anschließender Transformation zu (zusammengesetzten) Bloomfiltern. Über den Jaccard-Koeffizienten als Ähnlichkeitsmaß kann dann die Ähnlichkeit zweier digitaler Identitäten bestimmt werden.

1 Einleitung

1.1 Problemstellung

Existierende Identitätenrepositories, insbesondere solche, die zu einer bestimmten, kaum überschaubaren Größe angewachsenen sind, beinhalten aus der Historie heraus oft ungewollt Dubletten digitaler Identitäten[1], d.h. aus unterschiedlichen Gründen Datensätze, die in Wirklichkeit ein und dieselbe natürliche Person referenzieren, jedoch in ihrer Ausprägung nicht identisch sind, aber ggf. zueinander teilw. identische resp. ähnliche Werte, ihrer zugeordneten, sie hinreichend beschreibenden Attributmenge aufweisen. Es existieren somit ggf. mehrere digital über jeweils einen eineindeutigen Identifikator[2] referenzierbare Repräsentanten, d.h. Dubletten ein und derselben natürlichen Identität, die somit das Abbild der realen Welt verfälschen. Es kann aber auch passieren, dass die Eineindeutigkeit gebrochen wird, was ebenfalls zu einer Diskrepanz zur realen Welt führt.

Ist aufgrund fehlender eindeutiger oder (ungewollt) falsifizierter Referenzierungen eine Zuordnung zu verarbeitender Informationen auf entsprechende digitale Identitäten nicht gegeben, so ist nicht immer eindeutig erkennbar, ob es sich trotz teilw. identischer oder sehr ähnlicher Attributwerte um eine Erstaufnahme ins Repository oder Zuweisung neuer Attributwerte zu einer bestehenden digitalen Identität im Respository handelt.

Wird fälschlich eine Erstaufnahme konstatiert, so entsteht eine Dublette, mit der Folge, dass eine natürliche Person nun zwei digitale Entsprechungen im Repository besitzt. Hierdurch könnte diese Person ggf. entweder über die eine oder über die andere digitale Identität (Zugriffs-)Rechte ausüben, zu denen sie nicht mehr autorisiert ist, da der Rechteentzug durch Löschung eines bestimmten Datensatzes oder durch Eintrag in einem bestimmten Datensatz stattgefunden hat, die korrelierenden Rechte jedoch im anderen weiter bestehen.

Bei vermeintlicher Aktualisierung einer digitalen Identität werden ggf. identifizierende Attribute überschrieben, deren Werte real einer anderen, natürlichen Person gehören. In diesem Falle geht eigentlich eine bestehende Relation zwischen digitaler Identität und natürliche Person unter und eine neue wird geschaffen, streng genommen jedoch repräsentiert diese digitale Identität zwei natürliche Personen. In diesem Falle ist die neue Person aber nicht der Lage, (Zugriffs-)Rechte auszuüben, vielmehr werden diese von der ehemalig referenzierten weiter genutzt. Hierdurch kann es dann bei unerlaubten Handlungen zu einer Fehlidentifizierung kommen, wobei die falsche Person zur Verantwortung gezogen würde.

Da das Problem ungewollter Dubletten zu den kostenaufwendigsten Datenfehlern gehört[3], gilt es, wie auch im Falle eines Eineindeutigkeitsbruch, den inkonsistenten Abbildern von vornherein zu begegnen. So sollten die Anomalien frühzeitig, d.h. am besten bei ihrer Entstehung im Datenerhebungsprozess aufgespürt und eliminiert werden. Hierfür müssen schon vor Aufnahme einer digitalen Identität ins Repository Maßnahmen mehr oder minder automatisierte Ver-

[1] Eine digitale (Personen-)Identität sei definiert als ein eindeutiger, digitaler Identifikator, dem je nach situativem Kontext entsprechend personifizierende Attribute mit ihrer jeweiligen Menge an Ausprägungen zugeordnet sein können. Dieser Identifikator steht während seiner gesamten Lebensdauer im System in einer „1:1“-Relation zu einer natürlichen Person und ist somit eineindeutig.

[2] Da die eine in realiter natürliche existente Person beschreibenden Attribute, in ihrer verfügbaren Menge und Ausprägung als Kriterium für die Unterscheidung nicht immer ausreichen und außerdem sich ein Teil dieser Menge in seiner Ausprägung im Laufe der Zeit ändern kann, existiert ein systemimmanentes Surrogat als Identifikator, der Personen eineindeutig referenziert und sie prägnant voneinander trennt.

[3] vgl. [Naumann, Felix (2007)]

fahren ergriffen werden, die eine ausreichende Datenqualität[4] sicherstellen und es über ein Verfahren zur Duplikaterkennung[5] ermöglichen, zu erkennen, ob eine digitale Identität schon einmal erfasst wurde; geschieht dies nicht, muss zu mindest in kurzen Abständen im Nachgang eine Dublettenprüfung durchgeführt werden.

Eine andere Problematik ist gegeben, wenn Personen resp. deren digitale Identitäten aus bestimmten Gründen nicht in das Identitätenrespository Einzug finden dürfen und hierfür entsprechend gelistet sind (Blacklist). Da nicht immer alle Datenquellen, d.h. die Personen erfassenden Stellen über diesen Sachverhalt informiert sind, kann es durchaus passieren, dass Datensätze zur Verarbeitung anstehen, die solche Personen repräsentieren. Hier muss die entsprechende Blacklist fehlertolerant abgeglichen werden können, um vorgegebene Compliance-Verstöße mit entsprechenden Sanktionen a priori zu vermeiden.

1.2 Datenbereinigung

Zur Datenbereinigung (engl. data cleaning oder data scrubbing) gehören verschiedene Verfahren zum Entfernen und Korrigieren von Datenfehlern in Informationssystemen. Die Fehler können beispielsweise aus inkorrekten (ursprünglich falschen oder veralteten), redundanten, inkonsistenten oder falsch formatierten Daten bestehen.

Die Datenbereinigung ist ein Beitrag zur Verbesserung der Informationsqualität. Allerdings betrifft Informationsqualität auch viele weitere Eigenschaften von Datenquellen (Glaubwürdigkeit, Relevanz, Verfügbarkeit, Kosten etc.), die sich mittels Datenbereinigung nicht verbessern lassen. Wesentliche Schritte zur Datenbereinigung sind die Duplikaterkennung (Erkennen und Zusammenlegen von gleichen Datensätzen) und Datenfusion (Zusammenführen und Vervollständigen lückenhafter Daten).

Aufgrund unterschiedlicher Datenqualität entstehen beim Erkennen von Dubletten und Datenfusion von Identitäten resp. deren Datensätze aus unterschiedlichen Quellen Datenkonflikte auf unterschiedlichen Ebenen der Semiotik, die durch

- syntaktische (Datenformat),
- semantische (Einzelwortbedeutung) und/oder
- pragmatische (kontextbezogene Bedeutung)

Unzulänglichkeiten hervorgerufen werden können. So werden z. B. Dubletten häufig ungewollt durch Tippfehler, Hörfehler, Buchstaben- und Wortdreher, Abkürzungen, unterschiedliche Schreibweisen usw. produziert. Viele Datenfehler lassen sich von vornherein durch domänenspezifische Normalisierung und/oder Transformation sowie durch Standardisierung der Daten beheben[6].

[4] „Informationsqualität oder Datenqualität bezeichnet die Qualität, also Bedeutsamkeit, Relevanz und Korrektheit von Informationen. Sie beschreibt, wie gut eine Information (bzw. ein Datensatz) geeignet ist, die Realität zu beschreiben, das heißt, inwieweit sie ein Modell tatsächlicher Situationen bildet. Insbesondere besagt sie, wie verlässlich eine Information ist und inwieweit man sie als Grundlage für eine Planung des eigenen Handelns verwenden kann.“ (vgl. http://de.wikipedia.org/wiki/Informationsqualität)

[5] „Unter Duplikaterkennung oder Objektidentifizierung versteht man verschiedene automatische Verfahren, mit denen sich Datensätze identifizieren lassen, die dasselbe Objekt in der realen Welt repräsentieren. Dies ist beispielsweise beim Zusammenführen mehrerer Datenquellen oder bei der Datenbereinigung notwendig. Die Schwierigkeit besteht darin, dass Datensätze für gleiche Objekte unterschiedliche Werte aufweisen können und deshalb Heuristiken angewandt werden müssen.“ (vgl. http://de.wikipedia.org/wiki/Duplikaterkennung)

[6] vgl. [Naumann, Felix (2007)]

1.3 Lösungsidee

Eine einfache Prüfung einzelner identifizierender Attribute auf Identität reicht bei weitem nicht aus, um ungewollte Dubletten aufdecken zu können. Denn z. B. unterschiedliche Schreibweisen, wie auch Schreibfehler usw. führen dazu, dass eine Identität der Attribute nicht gegeben ist und somit Dubletten nicht erkannt werden können. Eine zunehmende Internationalisierung führt dazu, dass sprachabhängige Verfahren, wie z. B. das Soundex-Verfahren[7], hier mehr oder weniger versagen werden, auch der Rückgriff auf sog. Domainwissen wird seine Grenzen finden.

Ein weiteres Problem liegt in der Performanz, riesige Datenmengen zeitnah miteinander vergleichen und ggf. zusätzliche Informationen über diese Datenmengen speichern zu müssen, sodass z. B. die Verwendung eines Levenshtein-Distance-Algorithmus[8] nicht immer als adäquateste Wahl erscheint. Dem Grunde nach lässt sich die gesamte Problemstellung auf die einfache Fragestellung reduzieren:

Wie lassen sich effizient ähnliche Identitäten in einem großen Repository extrahieren?

Hierfür muss zunächst ein Maß, eine Art Fingerabdruck (Fingerprint[9]), definiert werden, das in der Lage ist, die zu komparierende Datenmenge auf eine erträgliche Größe zu reduzieren und gleichfalls die Ähnlichkeit einzelner Attributwerte und/oder zu Teilmengen kombinierter mit denen im Repository eingetragenen feststellen zu können, indem die ermittelten Ähnlichkeitswerte mit einem oder mehreren Schwellwerten verglichen werden können. Die Wahl eines solchen Maßes scheint hier nicht die Verwendung sog. kryptografischer Hash-Funktionen[10] zu sein, denn kleine Änderungen in den Attributwerten führen, dem Ziel dieser Hash-Funktionen folgend, prinzipiell zu stark differierenden Fingerabdrücken.

Sinnvoll nach Auffassung des Autors scheint hier, IT-Ressourcen schonend, eine Kandidatenvorauswahl von zueinander ähnlichen Identitäten zu treffen, um nachfolgend, auf einer so entstandenen ggf. sehr kleinen Mengen von Identitäten, inhaltlich intensiver prüfen zu können. Dies entweder mit Hilfe anderer automatisierter exakter arbeitender Berechnungsverfahren und/oder final durch Einsatz kognitiver Fähigkeiten menschlicher Ressourcen (Experten).

Schon 1970 hat Bloom in einem Aufsatz[11] eine gleich geartete, allgemein gehaltene Fragestellung behandelt, in der für den Test auf Nichtmitgliedschaft in einer Gesamtmenge von Fällen ein einfaches, Speicherplatz[12] und Rechenzeit schonendes Verfahren zur Anwendung kommt, das jedoch mit einer „erlaubten“ Fehlerwahrscheinlichkeit behaftet ist, Mitgliedschaften fälschlicherweise auszuweisen (false positive resp. false drops). Konnten aus der gesamten Fallmenge eine Menge ähnlicher Mitglieder gefiltert werden (Bloomfilter), so kann für einen intensiveren Test in dieser verbliebenen wesentlich kleineren Fallmenge mit differenzierteren und komplexeren Methoden die tatsächliche Mitgliedschaft überprüft werden.

---

[7] „Soundex ist ein phonetischer Algorithmus zur Indizierung von Wörtern und Phrasen nach ihrem Klang in der englischen Sprache. Gleichklingende Wörter sollen dabei zu einer identischen Zeichenfolge kodiert werden.“ (vgl. http://de.wikipedia.org/wiki/Soundex)

[8] Stringvergleichender Algorithmus, der die Differenz zweier Zeichenketten über eine Kostenfunktion ermittelt (vgl. http://en.wikipedia.org/wiki/Levenshtein_distance oder http://en.wikipedia.org/wiki/Damerau-Levenshtein_distance

[9] vgl. http://en.wikipedia.org/wiki/Fingerprint_(computing)

[10] vgl. http://en.wikipedia.org/wiki/Cryptographic_hash_function

[11] vgl. [Bloom, Burton H. (1970)]

[12] Reduzieren des Speicherplatzes, führt zum Anwachsen der Fehlerwahrscheinlichkeit, da die Wahrscheinlichkeit von Bit-Kollisionen wächst.

Die o. g. Methode einer ungefähren Suche nach Identitäten mit Hilfe von speziellen Signaturen, den sog. Bloomfiltern[13], liefert zwar effizient eine um mögliche „false positives“ erweiterte oder auch eine leere Ergebnismenge, anhand derer dann mittels festgelegter, die Identität hinreichend beschreibender Attribute die exakte Übereinstimmung der Attributausprägungen ermittelt und so homonym- und phantombereinigt[14] festgestellt werden kann, ob eine Identität sich im Repository befindet oder nicht, jedoch muss, da aufgrund der aufgeführten Probleme hinsichtlich Datenkonflikten und -qualität die Forderung besteht, inexakt, also nach Ähnlichkeiten in den Merkmalsausprägungen von Identitäten zu suchen, die Verarbeitung der Bloomfilter um ein bestimmtes Verfahren der Ähnlichkeitsbestimmung ergänzt werden[15].

Mit Verwendung des sog. Jaccard-Index[16] als Ähnlichkeitsmaß, der die Schnittmenge zweier Mengen ins Verhältnis zu ihrer Vereinigungsmenge setzt, ergibt sich in Verbindung mit Bloomfiltern die in diesem Dokument u. a. nachzuweisende Forderung, dass je wahrscheinlicher die Gleichheit zweier Bloomfilter, desto ähnlicher die beiden Wertemengen W_i, W_j relevanter Identitätenattribute sind, über die die jeweiligen Filter als binäres Abbild derer berechnet wurden.

$$prob\left[sig_{Bloom}(W_i) = sig_{Bloom}(W_j)\right] = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} \equiv sim_{Jaccard}(W_i, W_j) = \frac{|W_i \cap W_j|}{|W_i \cup W_j|}$$

Die jeweiligen Mengen *M* repräsentieren eine bestimmte Anzahl gesetzter Bits in den involvierten Bloomfiltern und sind wie folgt zu interpretieren:

- M_{11}: Gesamtanzahl in beiden Filtern an identischen Positionen gesetzter Bits
- M_{10}: Gesamtanzahl gesetzter Bits an Positionen im ersten Filter, an denen im zweiten keine gesetzt sind
- M_{01}: Gesamtanzahl gesetzter Bits an Positionen im zweiten Filter, an denen im ersten keine gesetzt sind
- M_{00}: Gesamtanzahl in beiden Filtern an identischen Positionen ungesetzter Bits (hier irrelevant)

Dieser Lösungsansatz scheint, neben vielen anderen, ein gangbarer Weg zu sein, die verwendeten Identitätenrepositories zu konsolidieren resp. konsolidiert zu halten.

[13] Ein Bloom-Filter ist eine über eine bestimmte Anzahl unabhängiger überlagerter Hash-Funktionen gebildete Signatur, die mit einer Menge gleichartiger Signaturen verglichen werden kann.

[14] Homonyme und Phantome sind Repräsentanten sog. false drops. Homonyme: unterschiedliche Identitäten im Repository weisen eine identische Signatur auf (Surjektivität durch z. B. Hashing). Phantome: Signaturen spiegeln Identitäten wider, die im Repository nicht vorhanden sind (Ursache z. B.: Überlagerung/Superimposed Coding i.V.m. Doppelhashing).

[15] Eine auf Bloom-Filter und Hamming-Abstand basierende Idee für eine mögliche Verwendung im kriminaltechnischen Umfeld der Identitätsfestellung mit verbundenem schnellem Merkmalsabgleich zur Fallaufklärung ist in [Kirsch, A.; Mitzenmacher, M. (2006a)] beschrieben.

[16] Der Jaccard-Index, auch bekannt als Jaccard-Ähnlichkeitskoeffizient, ist eine mengentheoretische, auf Stichproben basierende Statistik, die für Ähnlichkeits- und (biologische) Diversivitätsvergleiche genutzt werden kann (vgl. http://en.wikipedia.org/wiki/Jaccard_index)

2 Identifizierungsbasis

2.1 Merkmalseigenschaften

Nicht immer reichen einzelne Identifizierungsmerkmale, die eine Person resp. Identität beschreiben, in Quantität und Qualität aus, um in einem Identitätenrepository diese mit einer hohen Wahrscheinlichkeit (wieder)erkennen oder mit einer bestimmten Wahrscheinlichkeit, dass eine vermeintlich erkannte Identität, nicht die ist, für die sie gehalten wurde, ausschließen zu können. Dies gilt insbesondere, wenn einer Identität noch kein eineindeutiger Identifikator als Surrogat zugewiesen wurde. D.h. dass bei Fehlen eines solchen Identifikators ein anderes Merkmal oder eine geschickte, anhand von Domänenwissen konstruierte, ggf. auf Aussagenlogik basierte Kombination aus mehreren, eine Identität beschreibenden Merkmalen an die Stelle eben dieses Identifikators treten muss.

Um beim Durchsuchen des Identitätenrepository die endgültige Trefferquote, d.h. die Ergebnismenge ähnlicher und/oder identischer Datensätze gering zu halten, ist es notwendig, sich auf mehrere Merkmale einer Person abstützen zu können. Hierdurch lässt sich die Wahrscheinlichkeit, dass viele Datensätze der Abfragebedingung genügen, enorm reduzieren. So lässt sich schon anhand des sog. Geburtstagsparadoxon[17] festmachen, dass allein der Geburtstag für eine effektive Identifizierung nicht ausreicht, da schon bei einer Repositorygröße von 50 erfassten Personen die Wahrscheinlichkeit über 97% beträgt, dass zwei dieser Personen am selben Tag Geburtstag haben (Anm.: der Jahrgang wurde hier nicht berücksichtig).

Die Eigenschaften der identifizierenden Merkmale müssen bestimmten Kriterien gehorchen, damit sie eine hinreichende Aussagekraft allein oder in Kombination mir anderen über die zu identifizierende Person besitzen. So eignen sich Merkmale, die zu einer hohen Änderungsrate neigen, eher weniger, als solche deren Ausprägung (Attributwert) dem Individuum ein Leben lang anhaften bleiben[18]. Gleichzeitig muss eine ausreichende Menge[19] an prägnanten Merkmalen vorhanden sein, um durch Kombination eine Quasi-Eindeutigkeit erreichen und damit die für eine extensivere Prüfung disponierte Ergebnismenge einer Abfrage sehr klein halten zu können.

Des Weiteren sind Merkmale, die aufgrund einer kleinen Wertemenge einen geringen Differenzierungsgrad aufweisen, wie z. B. die z. Z. zweiwertige Gender[20]-Ausprägung, wenig hilfreich, da sie die Gesamtmenge an Identitäten in wenige, quantitativ große, aber qualitativ wenig prägnant beschreibende Untermengen teilen. Ein solches Merkmal könnte höchstenfalls, nach Ausschöpfung aller anderen Modalitäten, als ergänzendes Kriterium für die Identifizierung resp. Ähnlichkeitsmessung herangezogen werden.

2.2 Identifizierende Merkmalskandidaten

Für die Ermittlung, welche Merkmale, die eine Person hinreichend beschreiben, um sie für den hier vorliegenden Zweck der Identifizierung in einem Repository zu verwalten, herangezogen

[17] s. http://de.wikipedia.org/wiki/Geburtstagsparadoxon

[18] Das Alter einer Person z. B. ändert sich jedes Jahr, dass Geburtsdatum aber nicht.

[19] Es ist nicht immer sinnvoll, Eigenschaften eines Individuums, die man an ihm beobachten, benennen und beschreiben kann, wie z. B. Augenfarbe, Größe etc. vollständig in einem Identitätenrepository abzuspeichern. Auch wird es aus datenschutzrechtlichen oder anderen normativen Gründen nicht immer möglich sein, geeignete Merkmale, wie z. B. Nationalität, ins Repository aufnehmen zu können.

[20] Der Begriff Gender bezeichnet das soziale oder psychologische Geschlecht einer Person im Unterschied zu ihrem biologischen Geschlecht (vgl. http://de.wikipedia.org/wiki/Gender); mit hiesiger Verwendung dieses Begriffes versucht der Autor dem Gender-Mainstream gerecht zu werden.

werden können, diene hier, in Analogie zur Feststellung einer Person, der Personalausweis, in dem u. a. folgende Grundinformationen aufgeführt sind:

- Vor- und Nachname
- Geburtstag und –ort
- Adresse

Bei näherer Betrachtung ist zu konstatieren, dass die Attribute „Vorname“, „Geburtstag“ und „Geburtsort“ bessere Identifizierungseigenschaften aufweisen, da sie der natürlichen Person „lebenslänglich“ anhaften[21]. Das Attribut „Nachname“, dessen Inhalt z. B. durch Heirat sich ändern kann, käme eher in Kombination mit den eben genannten Attributen als Zusatzinformation in Frage; ähnliches gilt für das Attribut Adresse.

Genügen z. B. die Attribute „Vorname“, „Geburtstag“ und „Geburtsort“ aufgrund fehlender Datenqualität und bestimmten, rigorosen Prüfbedingungen allein nicht, um in die Ergebnismenge ähnlicher Datensätze aufgenommen zu werden, obwohl sie eigentlich Kandidaten hierfür gewesen wären, so lässt sich mit Hilfe des Attributes „Nachname“[22] dafür Sorge tragen, diese Kandidaten trotzdem der Ergebnismenge zuzuschlagen. Gleichfalls lässt sich aber anhand von Zusatzinformationen die Trefferquote reduzieren, um die Ergebnismenge für die weitere Behandlung entsprechend einschränken zu können.

3 n-Gramme

Die Zerlegung (Tokenization[23]) von Datensätzen resp. deren Attribute in sog. n-Gramme (Token) ermöglicht es, diese Attribute, interpretiert als Menge von Zeichenketten über ein bestimmtes Alphabet, durch Vektoren definieren und mathematische Verfahren auf diese anwenden zu können.

Die intuitive Idee dieser Art von Tokenization liegt zugrunde, dass je mehr sich zwei Zeichenketten ähneln, desto größer die Anzahl ihrer gemeinsamen n-Gramme ist. Ausgedrückt werden, kann dieses Verhalten z. B. über ein Ähnlichkeitsmaß, das die Schnittmenge zweier n-Grammengen, die durch die Zerlegung zweier Zeichenketten entstanden sind, ins Verhältnis zur ihrer Vereinigungsmenge gesetzt wird.

3.1 Schreibweisennormalisierung und Alphabet

Um dem Problem unterschiedlicher Schreibeweisen von Namen (Personen-, Orts- und Straßennamen sowie Organisations-, Funktionsbezeichnungen usw.) in Attributen, gerade in Bezug auf die Verwendung von Akzenten[24], vorzubeugen und des Weiteren das Alphabet in seiner Mächtigkeit (Kardinalität) überschaubar zu gestalten, werden alle diakritischen Zeichen eliminiert und die verbleibenden Grundkörper der Buchstaben entweder durchgehend in Majuskel- oder in der hier präferierten Minuskelform verwendet. Bei Umlauten und Zusammentreffen von Umlauten und Tremata, da „syntaktisch“ nicht unterscheidbar, wird bei ihrer Existenz in

[21] Die Änderungsmöglichkeit von Vornamen gem. Namenänderungsrecht möge hier aufgrund der Anwendungshäufigkeit keine Rolle spielen.

[22] Voraussetzung ist natürlich, dass der Nachname sich in der Zwischenzeit nicht geändert hat.

[23] vgl. [Järvelin, Anni et al. (2006)]

[24] z. B. Betonungszeichen wie Akut (Áá), Gravis (Àà), Trema (Ää), Cedille (Çç), nordischer Akzent (Åå), Tilde (Ññ, Õõ), Zirkumflex (Ââ), Trema (ë) usw.

Namen resp. Zeichenketten hinter dem entsprechenden, um das Diakritikum bereinigten Buchstaben jeweils ein „e“ eingefügt; ein „ß“ z. B. wird durch die Zeichenfolge „ss“ ersetzt[25].

Attribute mit Zahlenangaben, insbesondere Datum, Uhrzeit, Telefonnummer usw., sollten jeweils in ein vorab definiertes einheitliches Format gebracht werden; d.h. sie bestehen nur noch aus Zahlen, wobei die Semantik zwar teilw. verloren gehen, jedoch aufgrund der Kenntnis der spezifischen wohlkonstruierten Formatsyntax ggf. über (Teil-)Längen und Anfangspositionen reproduziert werden könnte[26]. Die Formatunifizierung unterstützt so i. V. m. der Kenntnis über den Attributtyp und dessen Wertemenge die Plausibilitätsprüfung[27] auf eine einfache Art und Weise.

Die Behandlung verkürzender Schreibweisen, wie z. B. im Deutschen „…str.“ für „…straße“ oder im Spanischen „c/“ oder „C/“ für „Calle“, setzt bestimmtes Domainwissen voraus, um sie z. B. durch Vervollständigung in eine normalisierte Form bringen zu wollen.

Alle Zeichen, die i. e. S. keine Buchstaben und/oder Ziffern sind[28], werden nicht berücksichtigt und für die spätere Weiterverarbeitung aus der Zeichenkette gelöscht oder ggf. durch ein Füllzeichen ersetzt.

Die hier betrachteten Zeichenketten setzen sich nun aus dem nachstehend konstruierten Zeichensatz, dem Alphabet *A*, zusammen, das um ein Hilfszeichen – hier sei es der Unterstrich '_' – für die Zerlegung der Zeichenketten in n-Gramme ergänzt wird. Der Zeichenvorrat dieses Alphabets umfasst 37 Symbole, d.h. die Mächtigkeit resp. Kardinalität |*A*| dieses Alphabets beträgt dann 37. Das hier zugrunde gelegte Alphabet präsentiert sich dann wie folgt:

A ={ '_','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
'0','1','2','3','4','5','6,'7','8','9'}

Der Zeichensatz resp. das Alphabet *A* sei als geordnete Menge von Zeichen gegeben, sodass

$$pos : A \rightarrow / 0, |A| - 1 / \in N$$

Somit liefert die Funktion

$$Ord(a) \, mit \; a \in A$$

die jeweilige Ordinalzahl des Zeichens a in diesem Alphabet. [29]

[25] Weitere mögliche Transliterationen z. B. aus dem Zeichensatz ISO 8859-1 sind vorstellbar: Æ→A, æ→A, ª→a, Ð→Dh, ð→dh, º→o, Þ→Th, þ→th etc. aber auch ggf. Mª→Maria

[26] 27. Mai 1964 als ein mögliches Geburtsdatum ergäbe mit dem vorgegebenen Format „JJJJMMTT“ die Ziffernfolge 19640527 oder eine Telefonnummer 040/422838-4625 würde mit +49404228384625 standardisiert.

[27] Es ließe sich relativ einfach erkennen, dass an der Position der Tagesangabe innerhalb eine Datums z. B. ein Wert von „32“ nicht sein kann und es sich hier womöglich um einen sog. Zahlendreher handele.

[28] z. B. Leerzeichen, Bindestrich, Schrägstrich, Apostroph, Interpunktionen usw.

[29] Der Zeichensatz ISO-8859-1 liefert z. B. Ord('_') → $5F_{16}$, Ord('a') → 61_{16}, Ord('z') →$7A_{16}$, Ord('0') →30_{16}, Ord('9') →39_{16}

3.2 Zeichenkettenzerlegung

Durch die hier gewählte Art der Zerlegung einer Zeichenkette entstehen Textfragmente, sog. n-Gramme[30], wobei *n* (hier: *n* > 1 (!)) der Länge der sich um *n-1* Zeichen überlappenden Teilzeichenketten entspricht. Um eine Benachteiligung aufgrund der Unterrepräsentanz des ersten und letzten Symbols einer Zeichenkette in der Gesamtheit der aus ihr generierten n-Gramm-menge ausschließen und Zeichenketten aus einem Zeichen nach gleichem Schema zerlegen zu können, wird von vornherein das Alphabet *A* um ein Hilfszeichen ergänzt.

Bildlich gesehen wird ein Fenster mit n Facetten sukzessive beginnend mit dem ersten Zeichen unter dem rechten Fensterrand (n-te Facette) Zeichen für Zeichen über die aktuelle Zeichenkette geschoben und die jeweils erscheinenden *n* Symbole ergebenen die korrelierenden n-Gramme; in Facetten, in denen kein Zeichen der Zeichenkette erscheint, erscheint das Hilfszeichen ‚_'. Aus einer Zeichenkette mit *m* Zeichen werden so (*m+n-1*) n-Gramme generiert, wobei die ersten und letzten n-Gramme am Anfang resp. Ende jeweils eine bestimmte Anzahl des Hilfszeichens aufweisen.

Ein n-Gramm beschreibt also eine bestimmte Wahrscheinlichkeit, mit der ein Zeichen in einer Zeichenkette auf n-1 vorhergehenden folgt, gleichzeitig entsteht durch die Überlappung eine Abhängigkeit der aus einer Zeichenkette entstandenen n-Gramme zueinander. Die Vorteile dieser Art von Zerlegung ergeben sich aus[31]:

- Robustheit: gewisse Toleranz gegenüber Schreibfehlern (z. B. Buchstabendrehern) und anderen morphologischen Variationen, da sich der Fehler nur lokal, d.h. auf eine kleine Anzahl von Textfragmenten auswirkt, während der Rest davon unberührt bleibt[32]
- Abgeschlossenheit: Vokabular besteht aus einer geordneten Menge von unterscheidbaren n-Grammen mit einer wohldefinierten, einfach bestimmbaren Größe (Kardinalität) resp. Wortschatz mit $W_n = |A|^n$ [33]; d.h. gegenüber einer natürlichen Sprache ist der Wortschatz (Vokabular) überschaubar und fest definiert und hat einen geringeren Umfang[34] als natürliche Wortschätze.
- Domänenunabhängigkeit: Sprach- und Themenneutralität
- Wirtschaftlichkeit: nur ein Verarbeitungsdurchgang
- Einfachheit: keine linguistischen Kenntnisse erforderlich (z.. B. Stammformreduktion, Silbenzerlegung); alle Wörter habe identische Länge

Diese Zerlegung in sinnneutrale Einheiten und der Vergleich dieser Einheiten miteinander sind besonders gut geeignet, Ähnlichkeiten von Eigennamen zu überprüfen. Ein Nachteil ist vielleicht, dass in Bezug auf das menschliche Wahrnehmungsvermögen sich eine Ähnlichkeit nicht immer intuitiv sofort erschließt.

---

[30] n=1: Unigramm (nicht sinnvoll); n=2: Bi- oder Digramm; n=3: Trigramm; n=4: Tetra- oder Quad(ri)gramm usw.

[31] vgl. [Tauritz, Daniel R. (2002)]

[32] vgl. [Cavnar, William B.; Trenkle, John M. (1994)]

[33] Für Bigramme ergäbe sich hier ein Vokabular W2 von 1.369 (37^2), für Trigramme W3 von 50.653 (37^3) und für Tetragramme W4 von 1.585.080 (37^4) möglicher Alphabetkombinationen.

[34] Wikipedia gibt für den englischen Wortschatz eine Zahl von 500.000 bis 600.000 Wörtern an, der deutsche liege knapp darunter, der französische bei etwa 300.000 Wörtern (s. https://de.wikipedia.org/wiki/Wortschatz)

Für die hier betrachtete Anwendungspraxis scheint die Wahl von n-Grammen mit *n = 2* (Digramme) und *n = 3* (Trigramme) angemessen. Für kleine Zeichenketten kommen eher Digramme infrage, da sie noch ausreichende syntaktische Informationen liefern und im Gegensatz zu Trigrammen die semantische Aussagekraft erhalten bleibt. Für große Zeichenketten jedoch führen Digramme ggf. zu einer überproportionalen Ähnlichkeit, da ihre Auftrittswahrscheinlichkeit steigt; eine Verwendung von Trigrammen scheint aufgrund der größeren Überdeckungen somit sinnvoller. n-Gramme *n > 3* führen zu einer exponentiell wachsenden Anzahl von n-Grammen bei gleichzeitig sinkender Auftrittswahrscheinlichkeit und sind hier wohl eher theoretischer Natur, aber in mächtigeren Textkorpora durchaus sinnvoll.

3.3 Einfache Indexberechnung

Für jede n-Gramm-Familie existiert jeweils ein abgeschlossenes Vokabular V_n als kartesisches Produkt über ein gegebenes Alphabet *A*, sodass eine (geordnete) Menge mit endlich vielen Elementen entsteht

$$V_n = \prod_1^n A_i = \left\{ w_j = \left(a_1, \ldots, a_n\right) \right\} \; mit \; a_i \in A_i; \; i = 1,..,n \wedge 1 \le j \le \left|V_n\right| = \left|A\right|^n$$

Um n-Gramme für die weitere rechnergestützte Verarbeitung effizienter gestalten zu können, reicht für die hiesigen Zwecke eine äquivalente bijektive Zahlendarstellung mit

$$idx : V \rightarrow I = [1, |V|] \in N$$

aus, sodass für alle möglichen paarweise verschiedenen n-Gramme jeweils genau eine Zahl ungleich Null, aus einem gegebenen Intervall, gebildet über die Mächtigkeit des Vokabulars, existiert

$$idx(w_i) = i \quad mit \; w_i \in V \wedge 0 \le i \le \left|V\right| - 1$$

Das Ergebnis *i* entspricht dann dem Index eines n-Gramms resp. der Position eines n-Gramms im Vokabular.

Ein n-Gramm setzt sich aus *n* Zeichen des Alphabetes *A* zusammen, sodass der Index eines jeden n-Gramms wie folgt berechnet werden kann:

$$idx(w) = \sum_{j=0}^{n-1} \left(\left(Ord(a_j) + 1 \right) * \left|A\right|^{n-j-1} \right) mit \; w \in V \wedge a \in A$$

3.4 Rekursive Indexberechnung

Die Verwendung sich überlappender n-Gramme bietet die Gelegenheit, die Berechnung der jeweiligen Indizes beschleunigen zu können[35], da jedes n-Gramm zu seinen Vorgängern *n-1* gleiche Symbole resp. Zeichen aufweist, d.h. für ein neues n-Gramm wird ein neues Symbol hinzugefügt und ein altes fallen gelassen. Diese Redundanz lässt sich für die Indexberechnung der aus einer Zeichenkette generierten n-Gramme für die Optimierung hinsichtlich des Rechenaufwands nutzen.

Sei ein n-Gramm ein in der Zerlegungsreihenfolge *i-ter* Teil einer Zeichenkette aus *n* Symbolen des Alphabetes *A*, so dass

$$w_i = \left(a_i, a_{i+1}, \ldots, a_{i+n-1}\right)$$

[35] vgl. [Cohen, Jonathan D. (1997)]

so ist mit Blick auf den Vorgänger w_{i-1} das Zeichen a_{i+n-1} hinzugekommen und a_{i-1} eliminiert worden. Gesucht ist nun eine neue Indexfunktion, so dass

$$idx(w_i) = f(idx(w_{i-1}), a_{i-1}, a_{i+n-1})$$

Unter Verwendung der bekannten Indexfunktion ergibt sich der Index für ein n-Gramm in Abhängigkeit zur der Zerlegungsreihenfolge wie folgt

$$idx(w_i) = \sum_{j=0}^{n-1} \left((Ord(a_{i+j}) + 1) * |A|^{n-j-1} \right)$$

Rekursiv formuliert ergibt sich für die neue Indexfunktion für alle n-Gramme einer Zeichenkette dann durch

$$idx(w_1) = \sum_{i=1}^{n} \left((Ord(a_i) + 1) * |A|^{n-i} \right)$$

und nachfolgend

$$idx(w_i) = idx(w_{i-1}) * |A| - Ord(a_{i-1}) * |A|^n + (Ord(a_{i+n-1}) + 1) \; mit \; 1 < i \leq N$$

3.5 Beispiel: Trigrammzerlegung und Indexberechnung

Die Zerlegung der normalisierten Zeichenkette *erika*, bestehend aus 5 Zeichen, in Worte zu drei Zeichen zerlegt, ergibt dies 7 Trigramme, resultierend, indem ein fiktives Fenster mit 3 Facetten in 7 Schritten sukzessive über die Zeichenkette geschoben wird; in Facetten, in denen kein Zeichen der Zeichenkette vorhanden ist, erscheint das sog. Hilfszeichen '_'.

		e	**r**	**i**	**k**	**a**			**5 Zeichen**
_	_	e							**1. Trigramm**
	_	e	r						**2. Trigramm**
		e	r	i					**3. Trigramm**
			r	i	k				**4. Trigramm**
				i	k	a			**5. Trigramm**
					k	a	_		**6. Trigramm**
						a	_	_	**7. Trigramm**

Ein Wort resp. Trigramm setzt sich aus 3 Zeichen (a_0, a_1, a_1) des gegeben Alphabetes *A* mit der Mächtigkeit 37 (s. Kap. 3.1) zusammen, sodass der Index eines jeden Trigramms wie folgt berechnet werden kann (s. Kap. 3.3):

$$idx(w = (a_0, a_1, a_2)) = (Ord(a_0) + 1) * 37^2 + (Ord(a_1) + 1) * 37 + (Ord(a_2) + 1)$$

$Ord(a_0)$	a_0	a_1	a_2	$idx_{Tri}(w=(a_0,a_1,a_2))$	
0	_	_	e	1.591	**1. Trigramm**
0	_	e	r	1.610	**2. Trigramm**
5	e	r	i	8.280	**3. Trigramm**
18	r	i	k	26.467	**4. Trigramm**
9	i	k	a	14.136	**5. Trigramm**
11	k	a	_	16.503	**6. Trigramm**
1	a	_	_	2.776	**7. Trigramm**

4 Bloomfilter

Bloomfilter, auch in ihrer kombinierten Form, sind spezielle Signaturausprägungen[36], die in vielen Bereichen der Informations- und Kommunikationstechnik (Netzwerke, Datenbanken usw.)[37] für effiziente Vergleiche herangezogen werden können.

Ein (Standard-)Bloomfilter ist eine einfache Platz sparende Datenstruktur (Bit-Array), mithilfe derer die Mitgliedschaft in einer bestimmten Menge, d.h. das Vorhandensein einer Identität im Identitätenrepository getestet werden kann. Die effiziente Verwendung von Speicherplatz jedoch geht zu Lasten, nicht exakt die Mitgliedschaft feststellen zu können; d.h. dass mit einer bestimmten, jedoch gestaltbar kleinen Wahrscheinlichkeit sog. False-Positives (eine Form der sog. False-Drops) auftauchen, also fälschlicherweise eine Mitgliedschaft nachgewiesen wird. Auf jeden Fall treten bei der hier verwendeten Bloomfilterart sog. False-Negatives, eine wirkliche Mitgliedschaft wird nicht erkannt, nicht auf[38].

4.1 Von der Wortmenge zur Signatur

Sei ein mengentheoretisches Ähnlichkeitsmaß über Wortmengen durch eine Funktion

$$sim : V \times V \rightarrow [0,1] \in R$$

gegeben, so lässt sich die durchschnittliche Ähnlichkeit zweier Wortmengen, konstruiert aus Zeichenketten eines oder mehreren Attributen durch jeweilige n-Gramm-Zerlegung, mit Hilfe des Verhältnisses – auch bekannt als Jaccard-Koeffizient oder -index[39] – aus der Anzahl Wörter, die in beiden Wortmengen vorkommen, zu der Gesamtzahl aller jeweils einmal vorkommenden Wörter aus beiden Wortmengen, durch eine heuristische Ähnlichkeitsfunktion quantitativ wie folgt beschreiben:

[36] Diese Art Signatur sei hier grundsätzlich als ein weiteres Identifizierungsmerkmal einer Identität verstanden, die über eine bestimmte Attributmenge gebildet, eben diese Menge für bestimmte Zwecke hinreichend beschreibt. Ein Anzahl möglicher Bloomfilterarten (Standard, Counting, Compressed, Attenuated) sind u. a. in [Broder, A.; Mitzenmacher, M. (2005)] kurz skizziert.

[37] vgl. [Blustein, J.; El-Maazawi, A (2002)] sowie [Roozenburg, Jelle (2005)]

[38] Es gibt auch Varianten (Generalized u. Distance-Sensitive Bloom Filters), die sog. False-Negatives enthalten, vgl. z. B. [Laufer, R. P.; Velloso, P. B.; Duarte, O. C. M. B. (2005)], [Kirsch, A.; Mitzenmacher, M. (2006a)]

[39] vgl.[Jaccard, Paul (1912)], [Cross, Valerie; Cabello, Cesar (1995)] und [Matthé, Tom et al. (2006)]

$$sim(W_Q, W_R) = \frac{|W_Q \cap W_R|}{|W_Q \cup W_R|} \quad mit\ W_Q, W_R \subseteq V$$

Der Quotient als Ähnlichkeitsmaß ergebe nun, dass je ähnlicher zwei Wortmengen eines Vokabulars *V*, je mehr Wörter resp. n-Gramme sind ihnen gemein und desto näher liegt das Ergebnis bei Eins.[40]

Über einen gewählten Schwellwert γ für die Relevanz lässt sich nun bestimmen, ob zwei Wortmengen ähnlich resp. fast identisch sind. Es gelte

$$sim(W_Q, W_R) > \gamma \Rightarrow W_Q \cong W_R \quad mit\ \delta \in [0,1]$$

Es lassen sich unter Verwendung eines Vektorraummodells für ein Vokabular und einer gegebenen minimalen, perfekten Hash-Funktion[41] die eben genannten Wortmengen äquivalent durch eine Transformations-

$$trans : hash(w) \rightarrow \{0,1\}^m$$
$$mit \quad w \in W \in V \wedge m = |V| \wedge w_i \neq w_j \Rightarrow hash(w_i) \neq hash(w_j)$$

und Überlagerungsfunktion (Superimposed Coding[42])

$$imp : \{trans(hash(w))\}^n \rightarrow \{0,1\}^m \quad mit \quad n = |W|$$

in zwei m-dimensionale Bit-Vektoren mit der Vorgabe überführen, dass zum einen das Resultat der Hash-Funktion eine Dimension der Signatur repräsentiert, dessen Bit die Transformationsfunktion auf Eins setzt und zum anderen durch Überlagerung die einzelnen Bits der relevanten Vektoren mittels boolescher Disjunktion miteinander verknüpft werden, sodass

$$s := (b_1, b_2, \ldots, b_{m-1}, b_m) \ \ mit\ 1 \leq m \leq |V| \wedge b_i \in \{0,1\}$$

und

$$S := (b_1, b_2, \ldots, b_{m-1}, b_m) \ mit\ 1 \leq m \leq |V| \wedge b_i \in \{0,1\}$$

Diese Art der Überlagerung kann zu einer Erhöhung der sog. False-Positive-Rate führen, da die Entstehung sog. Phantome begünstigt wird.

s repräsentiert nun die über ein Wort $w \subseteq W$ resp. n-Gramm und *S* dann die über eine Wortmenge[43] $W \subseteq V$ aus *n* n-Grammen (Wörtern) gebildete Signatur mit der Länge *m*.

Sei die kollisionsfreie[44] Hash-Funktion gegeben durch

$$hash(w) = idx(w)\ mit\ w \in V \wedge idx(w) \in [0, |V|] \in N$$

[40] Lieferte die Funktion den Wert 1, bedeutete dies nicht unbedingt, dass zwei Zeichenketten identisch sind, da hier die Häufigkeit eines n-Gramms sowie dessen Position(en) in einer Zeichenkette unberücksichtigt bleiben.

[41] vgl. [Czech, Zbigniew J. (1998)]; Die Mächtigkeit des Vokabulars ist gleich der Mächtigkeit der über das Vokabular mit Hilfe der Hash-Funktion erzeugten Ergebnismenge und sind Wörter paarweise verschieden, so sind auch deren Hash-Funktionsergebnisse verschieden.

[42] vgl. [Faloutsos, Christos (1988)]

[43] Besteht eine Wortmenge nur aus einem Element, so ist *w* = *W* und somit *s* = *S*.

[44] Im Gegensatz zu „kollisionsresistent“ – es treten Kollisionen auf, sind aber nicht einfach nachzuweisen resp. zu berechnen – bedeutet hier „kollisionsfrei“ i.e.S. des Wortes, dass keine Kollisionen auftreten.

und die Transformationsfunktion mit

$$s = trans(hash(w)) = 2^{idx(w)} \; mit \; w_i \in W$$

so ergibt sich die Signatur mittels bitoperativen Disjunktor *U* wie folgt

$$s_i \left(b_1^{s_1} \vee b_1^{s_2} \ldots \vee b_1^{s_n} \ldots, b_m^{s_1} b_m^{s_2} \ldots \vee b_m^{s_n} \right) mit \; 1 \le m \le |V| \wedge b_i \in \{0,1\}$$

Die auf Eins gesetzten Bits in der Bit-Folge entsprechen nun den paarweise verschiedenen n-Grammen in der jeweiligen Wortmenge, so dass aufgrund des hier vorliegenden minimalen, perfekten Hashing[45]

$$s \equiv w \; \forall w \in W \Rightarrow S \equiv W \; mit \; W \subseteq V$$

und es gilt:

$$s_i \wedge S = s_i \Rightarrow w_i \in W \; mit \; i \in [1, |W| = n]$$

und

$$s_j \wedge S \neq s_j \Rightarrow w_j \quad W \wedge w_j \in \{V \setminus V \cap W\}$$

4.2 Übers Zählen zur Ähnlichkeit

Mit einer Zählfunktion[46], die die auf Eins gesetzten Bits summiert, ergibt sich die Mächtigkeit einer einzelnen resp. zusammengesetzten Signatur[47]. Das Äquivalent für eine Wortmenge sei gegeben durch

$$|S| = \sum_{i=0}^{m} b_i \equiv |W| \quad mit \; W \subseteq V$$

die Schnittmenge zweier Wortmengen über Skalarprodukt (innere Produkt) zweier Signaturen

$$|S_Q \wedge S_R| = \sum_{i=1}^{m} b_i^{S_Q} \cdot b_i^{S_R} \equiv |W_Q \cap W_R| \quad mit \; W_Q, W_R \subseteq V \wedge m = |V|$$

und die Vereinigungsmenge zweier Wortmengen durch

$$|S_Q \vee S_R| = |S_Q| + |S_R| - |S_Q \wedge S_R| \equiv |W_Q \cup W_R| \; mit \; W_Q, W_R \subseteq V$$

S_Q entspricht z. B. einer Anfrage-Signatur (Index Q = Query) eines Mitgliedsaspiranten, der in das Identitätenrepository aufgenommen werden soll, und S_R einer Repository-Signatur eines vorhandenen Mitgliedes im Identitätenrepository.

[45] kollisionsfrei, da bijektive Abbildung vorausgesetzt

[46] In der programmiertechnischen Umsetzung der Zählfunktion („population count“) können aus Performanzgründen ggf. sog. Lookup-Tabellen (vgl. [El-Qawasmeh, Eyas; Al-Qarqaz, Wafa'a (2006)]) eingesetzt werden. Eine Signatur oder eine durch Konjunktion resp. Disjunktion zusammengesetzte Signatur wird in eine Anzahl Bitmuster bestimmter Länge (z. B. 8 Bit-Pattern) zerlegt. Die jeweils durch ein Bitmuster repräsentierte Zahl entspricht einem Tabellenindex, über den die entsprechende Anzahl gesetzter Bits aus der Tabelle ausgelesen werden kann (z. B.: $10011001_2 = 153_{10}$ = i; LookUp [i] $\rightarrow$ 4). Die resultierenden Zahlen aller Bitmuster werden addiert und ergeben die Mächtigkeit (Hamming-Gewicht) der entsprechenden Signatur.

[47] Eine Unterscheidung zwischen einer Signatur über ein Wort und einer über eine Wortmenge ist hier irrelevant.

Für die Wahrscheinlichkeit, dass zwei so konstruierte Signaturen identisch resp. ähnlich zueinander sind, bedeutet dies:

$$sim(W_Q, W_R) = sim(S_Q, S_R) = \frac{|S_Q \wedge S_R|}{|S_Q \vee S_R|}$$

Unter Verwendung eines vorher bestimmten Schwellwertes γ für die Relevanz der gefundenen Mitglieder im Identitätenrepository ergibt sich dann

$$sim(S_Q, S_R) > \gamma \Rightarrow W_Q \cong W_R \quad mit\ \gamma \in [0,1]$$

Die Signaturen belegen aufgrund der oben verwendeten Hash-Funktion schon für n-Gramm-Familien mit kleinem *n* einen mit *n* exponentiell wachsenden prohibitiv großen Speicherplatz für die vollständige fiktive Abbildung des Adressraumes[48] für jede einzelne Wortmenge. Da jedoch augenscheinlich, dass in der realen Welt, d.h. in der Wertemenge der betrachteten Attribute resp. deren Kombinationen nicht alle möglichen n-Gramm-Ausprägungen sowie Wortmengen vorkommen, diese Mengen sogar wesentlich kleiner sind als die Kardinalität der Kreuzprodukte über das gegebene Vokabular, so ließe sich zwar durch Speicherung der Signaturen nur real vorkommender Wortmengen in eine Liste der Speicherverbrauch drastisch reduzieren[49], was jedoch für die hier vorliegenden Anforderungen nicht ausreichend erscheint. Auch verbliebe in Analogie zur o. g. Problematik der veritable Rechenaufwand, da jede der Signatur(en) eines Kandidaten mit allen in der Liste eingetragenen Bit für Bit abgeglichen werden müsste. Für eine Vorselektion scheint dieser Sachverhalt auf jeden Fall nicht tragbar.

4.3 Konstruktion und Verfahren

Ein eine bestimmte Wortmenge *W* eines Vokabulars *V* aus *n* Wörtern *w* repräsentierender Bloomfilter *B* wird durch eine initial auf Null gesetzte Bitfolge der Länge *m*, die wesentlich kleiner als die Mächtigkeit von *V* gewählt sein soll, beschrieben, in der für jedes dem Filter hinzugefügte Wort aus der Wortmenge bestimmte Bits, mittels einer bestimmten Anzahl *k* aus einer Menge universeller, unabhängiger Hash-Funktionen berechnet, auf Eins gesetzt werden.

$$trans(hash_j(w_i))\, mit\ W \subseteq V \wedge w_i \in W \wedge n = |W|$$

und

$$s = trans(hash_j(w_i)) = 2^{hash_j(w_i) \bmod m} \quad mit\ \ 1 \le j \le k \wedge 1 \le i \le n \wedge m << |V|$$

Sei eine Wortmenge *W* durch einen Bloomfilter *B* und ein beliebiges Wort *w* durch seine mittels der *k* Hash-Funktionen gebildete Bit-Folge *s* repräsentiert und ist $s \wedge B = s$, so lässt sich konstatieren, dass w ein Element von *W* ist. Ist jedoch $s \wedge B \neq s$ ist *w* auf jeden Fall nicht in *W* enthalten.

Diese Art der Konstruktion kann jedoch zu Bitkollisionen sowie zu nicht gewollten Bitmustern in Bloomfiltern führen und begründen die Entstehung sog. False-Positives. Entstehen bei der Ausführung der Hash-Funktionen Bitfolgen, sodass Wörter aus dem Vokabular auf ein und dieselbe (Teil-)Bitfolge abgebildet werden, handelt es sich um Bitkollisionen, die das Phänomender sog. Homonyme hervorrufen. Werden durch Überlagerung (Superimposed Coding)

[48] Mit m gleich der Mächtigkeit des Vokabulars einer n-Gramm-Familie, ergibt sich z. B. für Trigramme eine Bit-Folge mit 50.653 Stellen mit einem Adressraum von 2^{50653} möglichen Einträgen.

[49] Mit einer Größe von jeweils ca.6,3 kByte (50653 / 8) für die Darstellung nur eines Attributes in Form einer aus Trigrammen gebildeten Signatur entsteht für ein Repository mit 100.000 Identitäten ein Speichervolumen von ca. ≤ 633 MByte.

von Wortsignaturen *s* z. B. ein Bitmuster erzeugt, das einem Wort aus dem Restvokabular (*V* \ *V*∩*W*) entspricht, nicht aber dem Filter miithilfe der Hash-Funktionen hinzugefügt wurde, entspricht dies dem Phänomen der sog. Phantome.

Aus den eben beschriebenen Ereignissen heraus resultiert, dass

$$s \wedge B = s \; obwohl \; w \notin \mathrm{W}$$

Als Konsequenz ist zu konstatieren, dass nicht mehr eindeutig auf die Wortmenge geschlossen werden kann, über die der Filter gebildet wurde[50].

4.4 Bloomfilterkonstruktion und Mitgliedschaftsprüfung

Das hier vorgestellte Beispiel ist unrealistisch und dient nur der Veranschaulichung!

Gegeben sei ein Bloomfilter der Länge *m* = *7* und eine Anzahl *k* = *2* Hash-Funktionen mit

$$hash_1(w) = idx(w) \bmod 7$$
$$hash_2(w) = idx(w) \bmod 6 + 1$$

[50] Fügte man ein Wort *w* nach obigem Verfahren einem Bloom-Filter *B* hinzu und führte diese Addition (bitweise „odern“) zu keiner Veränderung dieses Filters, so ließe sich somit nicht erkennen, ob wirklich *w* ∈ *W*, ein Test auf Mitgliedschaft jedoch würde immer positiv ausfallen.

Trigramm	*idx*	$hash_1$	$hash_2$	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]
__e	1591	2	2	0	0	1	0	0	0	0
_er	1610	0	3	1	0	0	1	0	1	0
eri	8280	0	2	1	0	1	0	0	0	0
rik	26467	0	2	1	0	1	0	0	0	0
ika	14136	3	1	0	1	0	1	0	0	0
ka_	16503	4	4	0	0	0	0	1	0	0
a__	2776	4	5	0	0	0	0	1	1	0
B_R				**1**	**1**	**1**	**1**	**1**	**1**	**0**
abc	2853	4	4	**0**	**1**	**0**	**0**	**1**	**0**	**0**
mno	19737	4	4	**1**	**1**	**0**	**0**	**0**	**0**	**0**

Weder das Wort resp. Trigramm „abc“ noch "mno" werden als Elemente von B_R gewertet.

4.5 False-Positive-Rate

Die einem Bloomfilter aufgrund der Bitkollisionen immanenten Wahrscheinlichkeit, dass ein beliebiges Wort w_i fälschlich als Element einer Wortmenge W angesehen wird, heißt False-Positive-Rate. Sie repräsentiert die Wahrscheinlichkeit, bei wieviel positven Antworten durchschnittlich eine falsch ist, und lässt sich nachfolgend für den hier genutzten Verwendungsfall von Bloomfiltern exakt genug beschreiben[51]. Die False-Positve-Rate wird durch folgende Phenomäne bstimmt:

- Homonyme
- Phantome

Dass nach Hinzufügen aller n Wörter w einer Wortmenge $W \in V$ zum Bloomfilter durch Anwenden der k Hash-Funktionen ein bestimmtes Bit nicht auf Eins gesetzt worden ist, liegt bei[52]

$$prob(b_i = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}} = p$$

Die Wahrscheinlichkeit, dass ein Bit im Bloomfilter auf Eins gesetzt ist, ergibt sich aus

$$prob(b_j = 1) = 1 - \left(1 - \frac{1}{m}\right)^{kn} \approx 1 - e^{-\frac{kn}{m}}$$

[51] vgl. [Broder, A.; Mitzenmacher, M. (2005)] und http://en.wikipedia.org/wiki/Bloom_filter

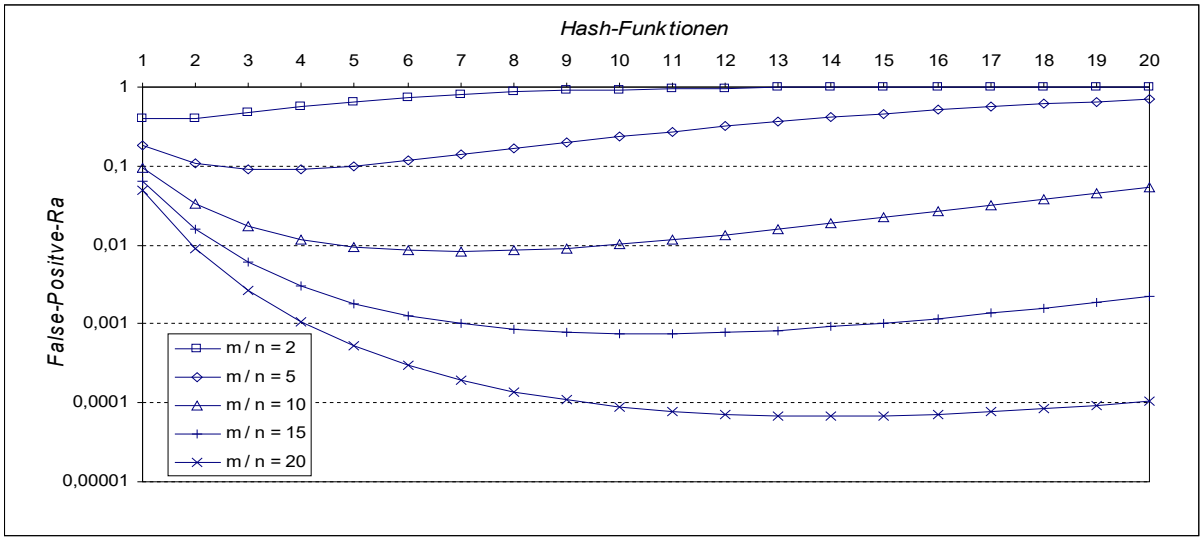
[52] Grundlage: $\lim_{m\to\infty}\left(1 - \frac{1}{m}\right)^m = e^{-1}$

Die False-Positive-Rate lässt sich nun anhand der Wahrscheinlichkeit, dass ein Wort an genau denselben *k* Positionen eines Bloomfilters eine Eins aufweist[53], was dazu führt, dass dieses Wort fälschlich als Element der durch den Bloomfilter repräsentierten Wortmenge angesehen wird, wie folgt berechnen[54]

$$fpr(B) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k = (1-p)^k = \varepsilon$$

Die optimale Gestaltung eines Bloomfilters hängt also, unabhängig, so scheint es, von der Mächtigkeit des verwendeten Vokabulars, allein von der Wahl der Parameter *k* und *m* zu *n* ab und liefert in letzter Konsequenz einen Kompromiss[55] zwischen Rechenzeit (Anzahl *k* der Hash-Funktionen) und Speicherplatz (Länge *m* des Filters) sowie der Restfehlerwahrscheinlichkeit (False-Positiv-Rate ε).

Die False-Positive-Rate verhält sich bei jeweils entsprechend angepasstem Verhältnis von *m* zu *n* reziprok proportional zur Anzahl der angewendeten Hash-Funktionen. Viele Hash-Funktionen erhöhen die Wahrscheinlichkeit mit der ein gefundenes Wort relevant ist (Precision), da der Füllungsgrad im Filter mit Einsen zunimmt und der Informationsgehalt steigt, wodurch ε sowohl sinkt als auch wahrscheinlicher wird[56]. Die nachfolgende Grafik verdeutlicht das eben beschriebene Verhalten der False-Positive-Rate in Abhängigkeit steigernder Anzahl angewendeter Hash-Funktionen.



[53] Die o. a. Berechnung spiegelt, nach Auffassung des Autors, das Auftreten sog. Homonyme wider inwieweit in diese Berechnung das mögliche Vorkommen sog. Phantome Berücksichtigung findet, scheint ungeklärt.

[54] In [Bose, Prosenjit et al. (2004)] wird ein Beweis geführt, in dem festgestellt wird, dass die implizite Annahme Blooms, dass das Setzen von Bits im Filter auf Eins unabhängige Ereignisse seien, nicht korrekt sei und zu einer niedrigen und damit falschen False-Positve-Rate führt, als sie tatsächlich sein müsste – mit groß genug gewählten *m* bei entsprechend kleinem *k* sei jedoch, so die Autoren, dieser Unterschied vernachlässigbar.

[55] vgl. [Bloom, Burton H. (1970)]

[56] Wären alle Bits eines Filters auf Eins gesetzt, führte dies unweigerlich dazu, dass alle möglichen Wörter $w \in W$ immer als Element von W_R ausgewiesen würden, obwohl dies augenscheinlich nicht immer der Fall sein dürfte.

Durch justieren der Parameter *k,m,n* unter der Bedingung, dass der Bloomfilter optimal bei einem Füllungsgrad *p* von 50% operiert, haben u. a. Broder und Mitzenbacher[57] gezeigt, dass die optimale Anzahl von Hash-Funktionen[58] *k*, abhängig vom konstanten Verhältnis *m/n*, bei

$$k = -\frac{\ln(2)}{n \cdot \ln\left(1 - \frac{1}{m}\right)} \approx \frac{m}{n}\ln(2) \approx 0{,}6931 \cdot \frac{m}{n}$$

liegt und die optimale False-Positive-Rate mit *p = 0,*5 konstant bei ungefähr[59]

$$\varepsilon_{opt} = 2^{-k} = (\frac{1}{2})^{k} = 0{,}6185^{\frac{m}{n}}$$

Allgemein gesehen bedeutet das, dass je größer der Bloomfilter und je größer die Anzahl Hash-Funktionen, desto geringer die False-Positive-Rate; gleichzeitig aber wächst die False-Positive-Rate mit größer werden der aufgenommenen Wortmenge. Mit der vorgegebenen False-Positive-Rate *fpr* und der bekannten Anzahl im Bloomfilter aufzunehmenden Wörtern *n* sowie Einsetzen von *k,* die Anzahl benötigter Hash-Funkionen, kann durch Umformen der Gleichungen die Größe des Bloomfilters *m* bestimmt werden:

$$fpr = 2^{-\frac{m}{n}\ln(2)} \rightarrow \ln(fpr) = -\ln(2)\ln(2) * \frac{m}{n}$$

$$m = -\frac{\ln(fpr)n}{\ln(2)^2}$$

Wie leicht zu sehen ist, wird wächst die Größe eines Bloomfilters proportional zur Anzahl der abzubildenden Wörter *n* und einem Koeffizienten, der hauptsächlich von der gewählten False-Positive-Rate *fpr* bestimmt wird

$$f(fpr) = -\frac{\ln(fpr)}{\ln(2)^2}$$

Um von einem effizienten und sinnvollen[60] Bloomfilter hinsichtlich Größe, d.h. > *0* und < *n*[61] sprechen zu können, sollte folgende Bedingung gelten (s. nachfolgende Graphik)

$$0 < f(fpr) < 1$$

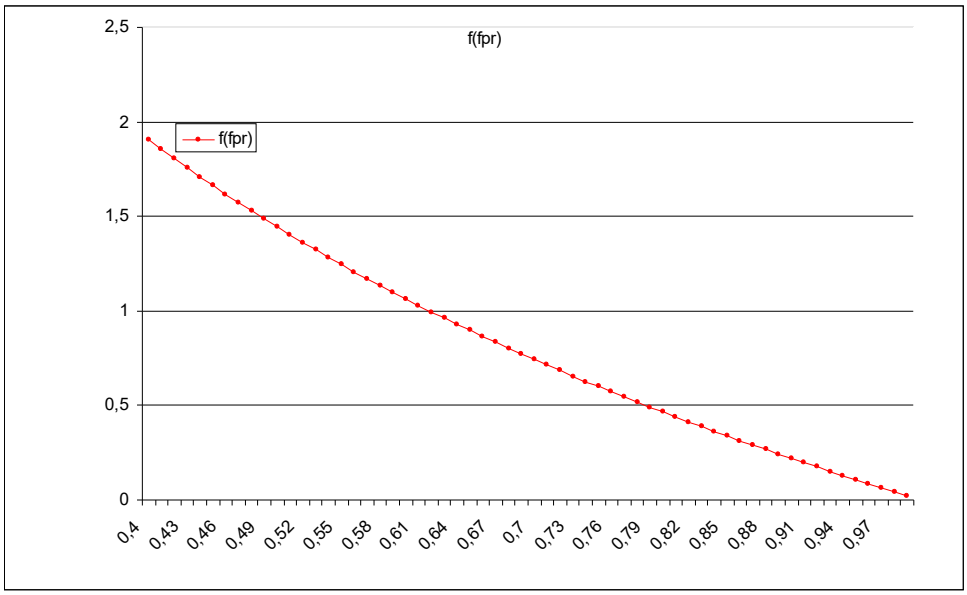
[57] vgl. [Broder, A.; Mitzenmacher, M. (2005)] und [Blustein, J.; El-Maazawi, A (2002)]

[58] Da $k \in N$, ergibt sich die sog. optimale Anzahl der Hash-Funktionen durch Abrunden von $\lfloor k+0{,}5 \rfloor$

[59] Bei einem Verhältnis von ungefähr *m/n=10* liegt das Optimum für *k* bei *7* mit einer Fehlerwahrscheinlichkeit von unter 1% (ca. 0,00819)

[60] Ein Bloomfilter mit Größe Null oder negativ ist nicht sinnvoll

[61] Läge die Größe des Bloomfilters über der Anzahl der abzubildenden Wörter, so reichte doch schon *m = n* vollkommen aus, die einzelnen Wörter eindeutig, d.h. ohne False-Positive-Rate und mit nur einer entsprechenden Hash-Funktion abzubilden.



Nachfolgende Tabelle stellt ein paar Werte in Relation[62], zu denen *k* als optimal eingestuft wird:

m/n	*k*	*⌊k+0,5⌋*	*fpr*
2	1,39	1	0,393
3	2,08	2	0,237
4	2,77	3	0,147
5	3,46	3	0,092
6	4,16	4	0,0561
7	4,85	5	0,0347
8	5,55	6	0,0216
9	6,24	6	0,0133
10	6,93	7	0,00819
11	7,62	8	0,00509
12	8,32	8	0,00314
13	9,01	9	0,00194
14	9,7	10	0,0012
15	10,4	10	0,000744

4.6 Hashing

Als Grundlage des Hashing-Verfahrens für Bloomfilter dienen *k* sog. universelle, unabhängige Hash-Funktionen, sodass[63]

[62] Weitere tabellarische Auflistungen sind in [Fan, Li et al. (1998)] zu finden

[63] vgl. [Kutzelnigg, Reinhard (2005)]

$$\forall w_i, w_j \in V \wedge hash_i \in \{hash_1,...,hash_k\} \rightarrow prob\left(hash_i(w_i) = hash_i(w_j)\right) \leq \frac{1}{m}$$

und aufgrund der Unabhängigkeit zusätzlich

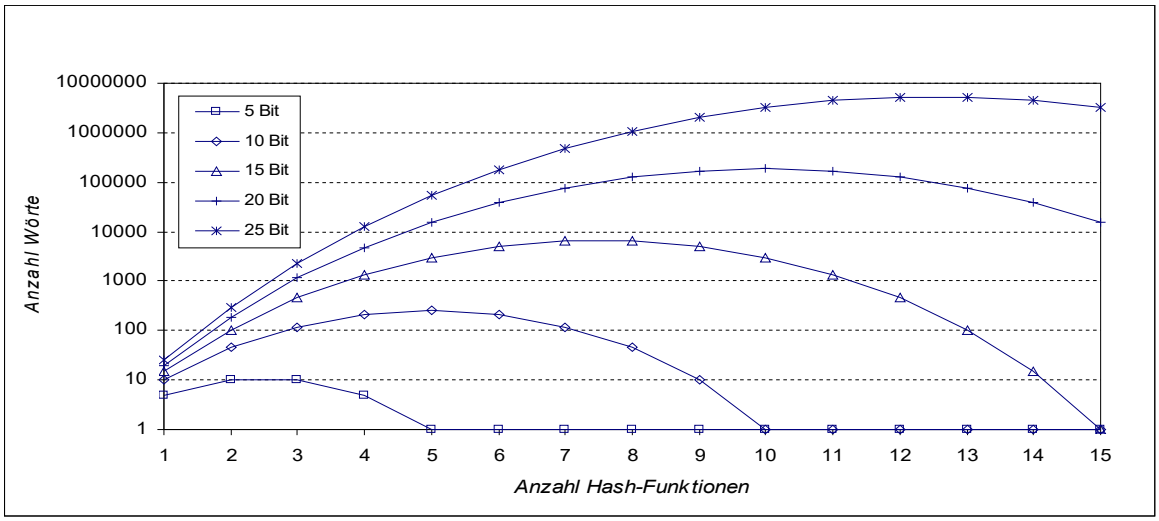
$$hash_1(w) \neq hash_2(w) \neq ... \neq hash_k(w)$$

gelte.

Die Anzahl in einen Filter maximal differenziert darstellbarer Wörter *n* entspricht der Partialsumme der Anzahl adressierbarer Bits und kann mithilfe von Filterlänge *m* und Anzahl angewendeter Hash-Funktionen *k* wie folgt (rekursiv) berechnet werden[64]:

$$n = f_k(m) = \frac{(m-k+1)}{k} \cdot f_{k-1}(m) \;\; mit \; f_0(m) = 1$$

Durch Verwendung mehrerer Hash-Funktionen steigt, wie die nachfolgende Grafik eindrucksvoll zeigt, die Anzahl unterschiedlicher, darstellbarer Wörter aufgrund ihrer unterscheidbaren Bitkombinationen[65] in Abhängigkeit der jeweiligen konstant gehaltenen Filterlänge (5-25 Bit) bis zu einem bestimmten Punkt; gleichzeitig jedoch geht einher, dass aufgrund des drastisch ansteigenden Füllungsgrades ebenfalls die Wahrscheinlichkeit möglicher Bitkollisionen und damit die False-Positive-Rate in gleichem Maße anwächst.

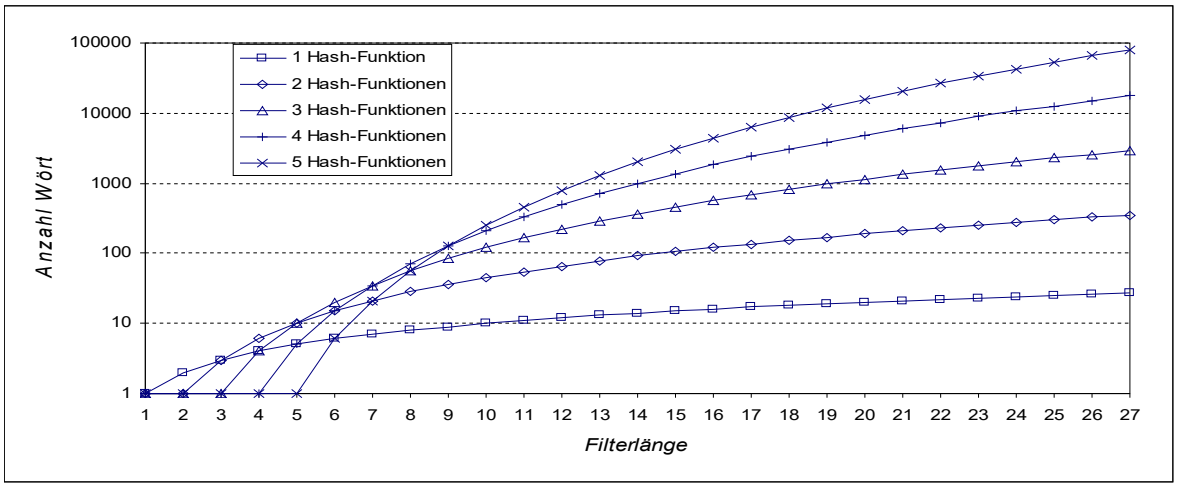


Nach Überschreiten des Maximums lassen sich in zunehmendem Maße die Wörter aufgrund wachsender Anzahl identischer Bitkombinationen nicht mehr differenzieren und die mögliche Anzahl unterscheidbarer aufzunehmender Wörter nimmt wieder ab, bis die jeweiligen Bitkombinatikonen der einzelnen Wörter paarweise identisch sind und der Füllungsgrad sowie die False-Positive-Rate 100% entsprechen.

[64] vgl. [Nafe, Clemens (2005)].

[65] Die o. a. rekursive Formel spiegelt die Anzahl der Möglichkeiten wider, in einem Filter mit der Länge *m* verschiedene Wörter aus Kombinationen von *k* Bits darstellen zu können – sei *m = 49* und *k = 6*, so ergeben sich 13.983.816 Kombinationsmöglichkeiten (6 aus 49):

$$\binom{m}{k} = \frac{m!}{k! \cdot (m-k)!} \rightarrow \binom{49}{6} = \frac{49!}{6! \cdot (49-6)!}$$



Nimmt neben der Anzahl Hash-Funktionen ebenfalls die Länge des Filters zu, so wächst, wie in der oben stehende Grafik dargestellt, die Anzahl unterschiedlicher Wörter, die mittels eines Filters repräsentiert werden kann, exponentiell.

Als Rechenzeit reduzierendes Hashing-Verfahren für Bloomfilter schlagen Kirsch und Mitzenbacher[66] vor, den Filter in disjunkte Teile zu zerlegen, mit der Forderung, dass die Anzahl der Teile der Anzahl anzuwendender Hashfunktionen und die Länge der jeweiligen Teile einer Primzahl entspricht. Des Weiteren werden mit Hilfe zweier stochastisch unabhängiger Hash-Funktionen[67] als Hilfsfunktionen in Analogie zum sog. Doppel-Hashing[68] die benötigte Anzahl Hash-Funktionen simuliert, die auf die jeweiligen Filterfragmente angewendet werden. Dieser modus operandi hat hinsichtlich Ressourcenschonung und Reduktion in der Komplexität des Hash-Verfahrens erhebliche Vorteile, ohne dabei die unerwünschte False-Positive-Rate[69] signifikant zu beeinflussen.

Habe ein Bloomfilter die Länge $m = k\,p$, wobei k die „optimale“ Anzahl unabhängiger Hash-Funktionen, die jeweils auf einen definierten Teil des in k Bereiche zerlegten Bloomfilters (Disjoint Coding[70]) angewendet werden, und repräsentiere p eine Primzahl[71], die ungefähr der Länge eines Teiles, d.h. eines Fragmentes des Bloomfilters entspreche, so ergäbe sich nach der

[66] vgl. [Kirsch, A.; Mitzenmacher, M. (2006b)]

[67] Unabhängigkeit bedingt in Bezug auf Kollisionswahrscheinlichkeit, dass

$$prob\left[h_1(x) = h_1(y) \wedge h_2(x) = h_2(y)\right] = \frac{1}{p^2}$$

[68] vgl. [Kutzelnigg, Reinhard (2005)] und http://de.wikipedia.org/wiki/Doppel-Hashing

[69] Durch Aufteilung des Filters in k disjunkte Teile ergibt sich für die Wahrscheinlichkeit, dass ein Bit auf Eins gesetzt ist, durch $(1 - k/m)^n \leq (1 - 1/m)^{kn} = p$ (Anm.: p entspricht hier der Wahrscheinlichkeit, dass ein Bit im Bloomfilter auf Eins gesetzt ist); dies würde – eine gute Wahl von k, m sowie n vorausgesetzt – die False-Positive-Rate $\varepsilon = (1 - p)$ nach [Broder, A.; Mitzenmacher, M. (2005)] vernachlässigbar höher als im nicht aufgeteilten Filter erscheinen lassen.

[70] vgl. [Pfaltz, John L.; Berman, William J.; Cagley, Edgar M. (1980)]

[71] Vorstellbar wäre, eine Zahl mit $p \leq m / k$ zu wählen, hierbei wäre ggf. m entsprechend anzupassen mit der Folge, dass sich die False-Positive-Rate minimal erhöhte.

Divisionsrest-Methode i. V. m. dem sog. linearem Sondieren folgende Form, aus der die jeweiligen partiellen Hash-Funktionen generiert werden können[72]:

$$hash_j(w, j) = (h_1(w) + j * h_2(w)) \bmod p$$
$$mit\ 0 \le j \le k-1 \wedge h_1(w) \in [0, p-1] \wedge h_2(w) \in [1, p-1]$$

Die zusammengesetzte Hash-Funktion für einen Bloomfilter wird also um den Laufparameter *j* erweitert. Hinsichtlich der geforderten Unabhängigkeit sollte u. a. $h_2(w) \neq 0$ sein, da sonst die Position, an der eine Eins zu setzen wäre, also der Index in jedem Fragment identisch wäre und die Kollisionswahrscheinlichkeit negativ beeinflösse; Ähnliches wäre gegeben, wenn *p* ein Vielfaches von $h_2(w)$ resp. $h_2(w)$ ein Teiler von *p* wäre, denn die Anzahl unterschiedlicher Indices betrüge $min\ (k, p / h_2(w))$ und läge damit im schlimmsten Falle bei 2; um dies zu vermeiden sollte $h_2(w)$ relativ prim, d.h. teilerfremd zu *p* sein[73].

Mit diesen gegebenen Forderungen lassen sich nun zwei simple Hilfsfunktionen sehr einfach konstruieren:

$$h_1(w) = idx(w) \bmod p$$
$$h_2(w) = idx(w) \bmod (p-2) + 1$$

Beispiel:

Gewählt sei eine Zerlegung eines Attributes in Trigramme, wobei dieses Attribut als Zeichenkette eine Aufnahmekapazität von max. 30 Zeichen habe, so ergäbe ausgedrückt durch die Zeichenkettenlänge (30) und Länge des n-Gramms (3) mit $n_{max} = 32$ (30+3-1) die maximale Anzahl möglicher Trigramme (Wörter), die aus dieser Zeichenkette generiert werden können. Mit einer gewünschten False-Positve-Rate $fpr \le 1\%$ ergäbe sich mit Blick auf o. a. Tabelle mit $k = 7$ der Wert für die Anzahl Hash-Funktionen und mit $m = 320$ (10 x 32) die (Bit-)Länge des Bloomfilters – da $m = k\ p$, wäre $p = 45{,}7$ (320 / 7).

Mit $p = 43$ wäre die Forderung, dass *p* eine Primzahl sei, erfüllt, sodass *m* auf *301* korrigiert werden würde. Mit dieser Korrektur wäre mit fpr[74] = 0,01096 die gewünschte False-Positve-Rate von ca.1% noch hinreichend erfüllt.

Der Algorithmus für die Berechnung und Bildung der jeweiligen Bloomfilter über eine 4x48-Bitmatrix für diese Art von Attributen könnte wie folgt aussehen:

[72] vgl. auch [Dillinger, P. C.; Manolios ,P. (2004)]

[73] Da gefordert, dass *p* eine Primzahl, sind alle Zahlen $z \in [1,p-1]$ zu *p* teilerfremd, da der jeweils größte gemeinsamer Teiler $ggT(p,z) = 1$. Eine andere Möglichkeit zur Erfüllung der Teilerfremdheit wäre, *p* als Zweierpotenz zu wählen, wobei $h_2(w)$ stets ungerade sein müsste.

[74] $\left(1 - e^{-\frac{7*32}{301}}\right)^7 = 0{,}01096$

```
FOR EACH trigram IN string
   h1 := idx(trigram) MOD 43
   h2 := idx(trigram) MOD 41 + 1
   bloom[0,h1] := 1
   FOR i := 1 TO (7-1)
      h1 := (h1 + h2) MOD 43
      bloom[i,h1] := 1
   END FOR
END FOR
```

4.7 Ähnlichkeitsmaß

Als Voraussetzung zur Ähnlichkeitsbestimmung zweier Bloomfilter gilt, dass jeweils nur solche herangezogen werden, deren Konstruktion auf denselben Parametern *m* für die Länge des Bloomfiltesr, *k* für die Anzahl der Hash-Funktionen und *n* für die maximale Anzahl von Wörtern, die ein Bloomfilter aufnehmen können soll, beruhen.

Aufgrund der Mächtigkeit einer einzelnen Wortsignatur von $|S| = k$ und der Überlagerung mehrerer Wörter, deren Signatur zu Signaturen anderer Wörter an einer oder mehreren identischen Positionen eine Eins aufweisen (False-Positive-Rate), lässt sich die Anwendung der Zählfunktionen für einfache Signaturen auf Basis kollisionsfreier Hash-Funktionen, um die Ähnlichkeit zweier Bloomfilter und damit die zweier Wortmengen bestimmen zu können, nicht einfach auf diese Filter übertragen. D.h. die Mächtigkeit eines Bloomfilters kann aufgrund möglicher aufgetretener Bitkollisionen nur bedingt mit der Mächtigkeit der korrelierenden Wortmenge gleichgesetzt werden, da ggf. nach Setzen einer Eins weitere Einsen an einer bestimmten Position gesetzt werden und somit diese keine zählbaren Einsen darstellen. Es gilt

$$\frac{\left\lceil |B| = \sum_{i=0}^{m} b_i \right\rceil}{k} \leq |W|$$

Nachfolgend soll analytisch skizziert werden, dass die Anwendung des Jaccard-Koeffizienten als Ähnlichkeitsmaß zweier Wortmengen auch hier seine Berechtigung hat, unter bestimmten Randbedingungen die Ähnlichkeit zweier durch Bloomfilter repräsentierte Wortmengen bestimmen zu können.

Als Grundlage der weiteren Betrachtung gelte in Analogie zu den einfachen kollisionsfreien Signaturen

$$sim(B_Q, B_R) = \frac{|B_Q \wedge B_R|}{|B_Q \vee B_R|} = \frac{\sum_{i=1}^{m} b_i^{B_Q} \cdot b_i^{B_R}}{\sum_{i=0}^{m} b_i^{B_Q} + \sum_{i=0}^{m} b_i^{B_R} - \sum_{i=1}^{m} b_i^{B_Q} \cdot b_i^{B_R}} \approx \frac{|W_Q \cap W_R|}{|W_Q \cup W_R|} = sim(W_Q, W_R)$$

Nun spiegelt aber gerade die Bitkollisionsrate δ prozentual das Verhältnis von nicht zählbaren zu zählbaren Einsen wider, wodurch die Mächtigkeit eines Bloomfilters um eine Menge von

Einsen ergänzt werden müsste, die aus der mit dem Prozentsatz δ multiplizierten Mächtigkeit eben dieses Filters resultiert, so dass[75]

$$\frac{|B|\cdot(1+\delta)}{k}=|W| \qquad mit\ \delta\to 0$$

Diesen Sachverhalt in obige Formel eingesetzt, ergibt, unter Beachtung, dass konstruktionsbedingt $\delta_Q = \delta_R$,

$$\frac{\frac{|B_Q\wedge B_R|}{k}\cdot(1+\delta^2)}{\frac{|B_Q|}{k}\cdot(1+\delta)+\frac{|B_R|}{k}\cdot(1+\delta)-\frac{|B_Q\wedge B_R|}{k}\cdot(1+\delta^2)}=\frac{|B_Q\wedge B_R|\cdot(1+\delta^2)}{(|B_Q|+|B_R|)\cdot(1+\delta)-|B_Q\wedge B_R|\cdot(1+\delta^2)}$$

und mit $B = B_Q = B_R$

$$\frac{|B|\cdot(1+\delta^2)}{2\cdot|B|\cdot(1+\delta)-|B|\cdot(1+\delta^2)}$$

und somit ein Korrekturkoeffizient durch

$$\frac{1+\delta^2}{1+2\delta-\delta^2}$$

so dass

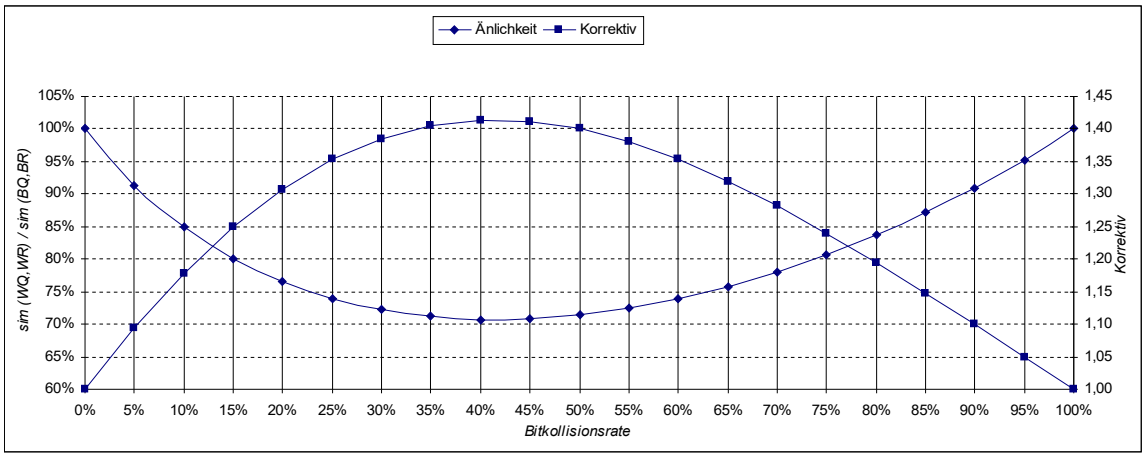
$$sim(B_Q,B_R)\cdot\frac{1+\delta^2}{1+2\delta-\delta^2}=sim(W_Q,W_R) \qquad mit\ \delta\to 0$$

Unter Verwendung eines vorher bestimmten Schwellwertes γ resultiert unter Einbezug einer Restfehlerwahrscheinlichkeit δ als Korrektiv dann

$$sim(B_Q,B_R)>\gamma\cdot\frac{1+2\delta-\delta^2}{1+\delta^2}\Rightarrow B_Q\cong B_R \quad mit\ \gamma\in[0,1]$$

Für eine Kollisionswahrscheinlichkeit von z. B. 10% bedeutete dies, dass, stochastisch gesehen, *sim(W_Q,W_R)* 15% unter dem Wert von *sim(B_Q,B_R)* liegt (s. nachfolgende Grafik) liegt, wodurch fehlerhafterweise eine höhere als reale Ähnlichkeit zweier Wortmengen durch die sie repräsentierenden Bloomfilter ermittelt würde und somit γ mit einem Wert von *1,18* multiplikativ korrigiert, d.h. von z. B. *75%* auf *88%* angehoben werden müsste.

[75] $|B|\cdot\delta$ beschreibt die fehlende fiktive Anzahl der mehrfach auf Eins gesetzten Bits.



Dieses Erscheinungsbild hat den Vorteil, dass bei Verzicht auf eine mögliche Korrektur, zwar die Menge der im Repository gefundenen ähnlichen Mitgliedern ggf. größer sein könnte, aber keine Mitglieder oberhalb der vorgegebenen Relevanz verloren gingen – als Konsequenz reicht es also in der Praxis aus, für die Restfehlerwahrscheinlichkeit einen relativ kleinen Wert bei der Konstruktion der Bloomfilter zu wählen.[76]

4.8 Bitkollisionsraten (kritisch betrachted)

Verfahrens- und konstruktionsbedingt ist ein Bloomfilter der Länge *m* in *k* (Anzahl unabhängiger Hash-Funktionen) Fragmente unterteilt, in denen Bitkollisionen stattfinden können. Das sowohl in einem wie auch in den anderen Fragmenten auftretende Ereignis einer Kollision bei Hinzufügen eines Wortes, ist aufgrund der unhabhängigen Hash-Funktionen per se unabhängig. Sei außerdem die Auftrittswahrscheinlichkeit von Bitkollisionen in allen Fragmenten gleich, so kann die Bitkollisionsrate eines Bloomfilters folgendermaßen beschrieben werden:

$$\delta = prob\left(Collision\right)^k$$

Die Wahrscheinlichkeit[77] wann zum ersten Mal eine Kollision im Fragment auftritt und damit eine weitere Betrachtung notwendig wird, lässt sich, wie landläufig für die Bestimmung der Kollisionsresistenz einer Hash-Funktionen herangezogen, analog zum sog. Geburtstagspara-

[76] In [Jain, Navendu; Dahlin, Mike; Tewari, Renu (2005)] wurde für die Ähnlichkeitsbestimmung zweier Bloomfilter über das Verhältnis der Bitkollisionswahrscheinlichkeiten – Notation wurde vom Autor an die in diesem Dokument verwendeten Schreibweisen adaptiert – von Anzahl korrespondierender gesetzter Bits in B_Q und B_R zur Anzahl in B_R gesetzter Bits

$$\frac{prob_{fpr}\left(\left|B_Q \wedge B_R\right|\right)}{prob_{fpr}\left(\left|B_R\right|\right)} = \frac{1-e^{-\frac{k}{m}\left(c+(n-c)\cdot 0{,}6185^{\frac{m}{n}}\right)}}{1-e^{-\frac{nk}{m}}} \quad mit \quad c = \left|B_Q \wedge B_R\right|$$

berechnet, dass, wenn *m* = *n*, ein Wert von 0,6973 resultiere, also 69% der korrespondierenden Bits aufgrund der False-Positive-Rate nicht miteinander korrelieren. Mit *m = 2n, 4n, 8n, 10n, 11n* ergeben sich aus dem o. a. Verhältnis dann 0,4658, respektive 0,1929, 0,0295, 0,113 und 0,0070.

[77] vgl. [Henning, Christian (2001)]

doxon[78] über die Beantwortung der Frage beschreiben, wie viele Wörter ein Fragment der Länge *p* aufnehmen kann, ohne dass es zu Kollisionen kommt.

$$prob_{Classic}(p,n) = 1 - \overline{prob_{Classic}}(p,n) \quad mit \quad p = \frac{m}{k}$$

Für die Wahrscheinlichkeitsberechnung, dass keine Kollision eingetreten ist, diene als Grundlage der sog. klassische Fall der Wahrscheinlichkeitsrechnung[79]

$$\begin{aligned}\overline{prob_{Classic}}(p,n) &= \frac{Anzahl\ der\ günstigen\ Ereignisse}{Anzahl\ aller\ möglichen\ Ereignisse} \\ &= \frac{p}{p}\cdot\frac{p-1}{p}\cdot\frac{p-2}{p}\cdot\ldots\cdot\frac{p-(n-1)}{p} \\ &= \prod_{i=0}^{n-1}\left(\frac{p-i}{p}\right) = \frac{p!}{p^n\cdot(p-n)!}\end{aligned}$$

Für die Wahrscheinlichkeit, dass mindestens eine Kollision im Fragment eingetreten ist, und unter Verwendung der sog. Stirlingschen Näherungsformel[80] ergibt sich dann

$$\begin{aligned}prob_{Classic}(p,n) &\approx 1 - \frac{\sqrt{2\pi\cdot p}\cdot\left(\frac{p}{e}\right)^p}{p^n\cdot\sqrt{2\pi\cdot(p-n)}\cdot\left(\frac{p-n}{e}\right)^{p-n}} \\ &= 1 - \frac{p^{p+0,5}\cdot e^{-p}}{p^n\cdot(p-n)^{p-n+0,5}\cdot e^{-(p-n)}} \\ &= 1 - e^{-n}\cdot\left(\frac{p}{p-n}\right)^{p-n+0,5} \\ &= 1 - e^{-n}\cdot\left(1-\frac{n}{p}\right)^{-p+n-0,5} \\ &\approx 1 - e^{-n}\cdot e^{n-\frac{n^2}{p}+\frac{n}{2p}} \quad für\ n << p \\ &= 1 - e^{-\frac{2n^2+n}{2p}}\end{aligned}$$

und für den gesamten Bloomfilter

$$\delta_{Classic} = prob_{Classic}(p,n)^k \approx \left(1 - e^{-\frac{2n^2+n}{2p}}\right)^k$$

Die Wahrscheinlichkeit, dass keine Kollision stattgefunden hat, wird im klassischen Fall durch das Verhältnis von Fragmentlänge *p* zu Anzahl Wörter *n* im Filter bestimmt und weist in der

[78] Mit welcher Wahrscheinlichkeit haben 2 von n Personen – gleichverteilte und stochastisch unabhängige Geburtstage vorausgesetzt – am selben Tag Geburtstag? – die Personen entsprächen den Wörtern und die Anzahl möglicher Geburtstage (365) der Fragmentlänge (s. http://de.wikipedia.org/wiki/Geburtstagsparadoxon i.V.m. http://de.wikipedia.org/wiki/Kollisionsfreiheit)

[79] vgl. [Bronstein, I. N.; Semendjajew, K. A. (1981)]

[80] $n! \approx \left(\frac{n}{e}\right)^n \cdot \sqrt{2\pi\cdot n}$; (s. [Bronstein, I. N.; Semendjajew, K. A. (1981)])

Hauptsache ein mit *n* quadratisches Wachstum für *p* auf. Zur Veranschaulichung sei eine Kollisionswahrscheinlichkeit von z. B. 10% und hierdurch die Anzahl der Hash-Funktionen[81] mit $k \approx 3$, so dass

$$10^{-1} = \left(1 - e^{-\frac{2n^2+n}{2p}}\right)^3 \rightarrow p = -\frac{2n^2+n}{2 \cdot \ln\left(1-\sqrt[3]{10^{-1}}\right)} \approx 3771 \quad mit\ n = 30$$

Sei nun die Anzahl Wörter im Filter gleich 30, so wäre eine Fragmentlänge von $p = 3771$ erforderlich, was wenig realistisch scheint[82] und die erfolgreiche Verwendung von Bloomfiltern in praxi konterkarieren würde.

Auch steht demgegenüber das Berechnungsmodell der False-Positive-Rate, d.h. der Berechnung der Auftrittswahrscheinlichkeit von Homonymen, die durch stattgefundene Bitkollisionen im Bloomfilter entstehen, mit

$$\delta_{fpr} = prob_{fpr}(p,n)^k = \left(1 - \left(1 - \frac{1}{p}\right)^n\right)^k \approx \left(1 - e^{-\frac{n}{p}}\right)^k$$

Dieses Modell weist ein mit *n* lineares Wachstum für *p* auf – für das o. g. Beispiel ergäbe sich mit $n = 30$ eine Fragmentlänge von $p = 48$.

$$p = -\frac{n}{\ln\left(1-\sqrt[3]{10^{-1}}\right)} \approx 48$$

Nach Auffassung des Autors trägt das Berechnungsmodell der False-Positive-Rate den Umstand nicht Rechnung, dass der Bloomfilter mit Nullen initialisiert wurde; auch bedarf es zumindest der Betrachtung des jeweils vorherigen und aktuellen Zustandes der entsprechenden Position (Übergangswahrscheinlichkeiten) unter Einbezug von günstiger zu möglichen Fällen, so dass hier anhand einer präziseren Modellierung ein realistischeres Abbild aufgezeigt werden soll.

Für die Abschätzung, dass ein oder mehrere Bitkollisionen in einem Fragment stattgefunden haben könnten, sei zunächst der Fall modelliert, dass nach Aufnahme von insgesamt *n* Wörtern in einer Position – Gleichwahrscheinlichkeit vorausgesetzt – keine Kollision stattgefunden hat.

Da ein Bloomfilter mit Nullen initialisiert ist, ergibt sich die Wahrscheinlichkeit, dass beim Hinzufügen des ersten Wortes durch Überlagerung an einer beliebigen Position im Fragment stochastisch keine Eins[83] steht, auch wenn intuitiv mit 100% angenommen, aus

$$\overline{prob_1}(1 \wedge \neg b_i = 0) = \left(1 - \frac{1}{2p}\right) \quad mit\ p = \frac{m}{k}$$

und dass danach nach Hinzufügen eines *i-ten* Wortes keine Eins gesetzt wurde, aus

$$\overline{prob_i}(\neg b_{i-1} \wedge \neg b_i = 0) = \left(1 - \frac{3}{4p}\right) \quad mit\ 1 \le i \le n-1$$

---

[81] gem. Berechnung über die False-Positive-Rate: $10^{-1} = 2^{-k} \rightarrow k = \frac{\log_x(0{,}1)}{\log_x(0{,}5)} = 3{,}32$

[82] vgl. [Dillinger, P. C.; Manolios ,P. (2004)] und [Wolper, Pierre; Leroy Denis (1993)]

[83] $\overline{b_{i-1} \vee b_i = 1} \equiv \neg b_{i-1} \wedge \neg b_i = 0$

Nach *n* Wörtern ist die Wahrscheinlichkeit gegeben durch

$$\overline{prob_{Coll}}\left(\overline{prob_1}\cdot\overline{prob_2}\cdot\ldots\cdot\overline{prob_n}\right)=\left(1-\frac{1}{2p}\right)\left(1-\frac{3}{4p}\right)^{n-1}$$

und die Bitkollisionsrate für ein Fragment

$$prob_{Coll}(p,n)=1-\left(1-\frac{1}{2p}\right)\left(1-\frac{3}{4p}\right)^{n-1}$$

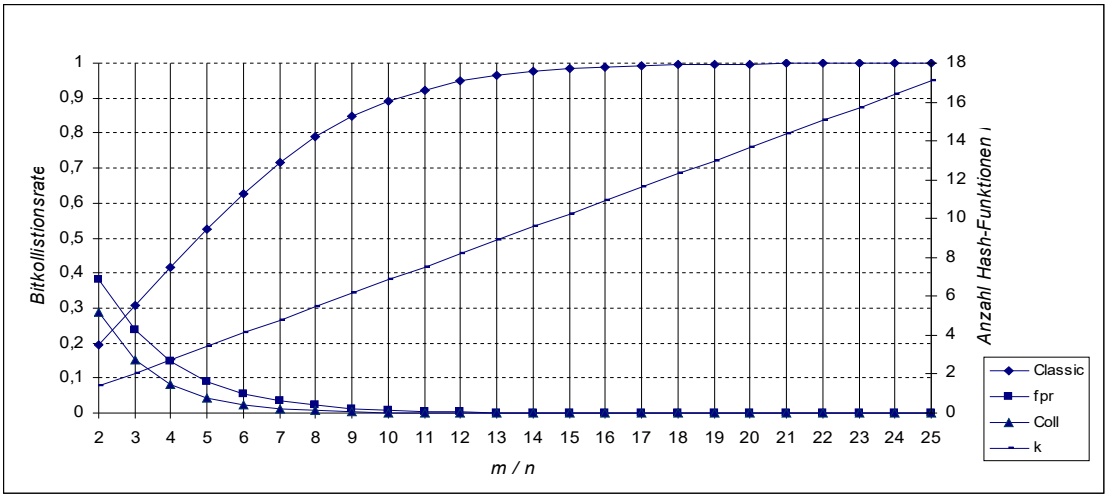
und für einen Bloomfilter dann

$$\delta_{Coll}=prob_{Coll}(p,n)^k=\left(1-\left(1-\frac{1}{2p}\right)\left(1-\frac{3}{4p}\right)^{n-1}\right)^k\approx\left(1-e^{-\frac{3n}{4p}}\right)^k$$

Für das o. g. Beispiel ergäbe sich mit *n* = *30* eine Fragmentlänge von

$$p=-\frac{0{,}75\cdot n}{\ln\left(1-\sqrt[3]{10^{-1}}\right)}\approx 36$$

Legt man nun für ein vorgegebenes Verhältnis von *m* zu *n* den aus dem Berechnungsmodell der False-Positive-Rate ermittelten Parameter *k* für die optimale Anzahl der anzuwendenden Hash-Funktionen zugrunde, so ergibt sich mit der so geschaffenen identischen Grundlage für die einzelnen Berechnungsarten der Bitkollisionsraten δ die nachfolgende Grafik.



4.9 Ähnlichkeitsprofile

Eine digitale Identität wird im Identitätenrepository über eine bestimmte Attributmenge (Attributrahmen) repräsentiert, die diese Identität hinreichend beschreiben und mehr oder weniger zum Identifizierungsprozess und damit zur Ähnlichkeitsbestimmung beitragen. Diejenigen mit einer bestimmten Relevanz versehenen Attribute, d.h. die resultierende Menge entsprechender in Bloomfilter umgewandelter Zeichenketten, die für die Ähnlichkeitsbestimmung herangezogen werden, bilden hier das Identitätsprofil einer digitalen Identität.

Für eine digitale Identität im Repository sei das Profil dargestellt als geordnete Menge von Bloomfiltern

$$P_R = \left\{B_R^1, B_R^2, ..., B_R^n\right\}$$

und für eine Anfrage als

$$P_Q = \left\{B_Q^1, B_Q^2, ..., B_Q^n\right\}$$

Je nach Relevanz der Attribute besitzen die jeweiligen Bloomfilter für die Ähnlichkeitsbestimmung innerhalb der Profile untereinander unterschiedliche Wichtigkeiten, die über partielle Gewichtungen

$$G = \left\{g_1, g_2, ..., g_n\right\} \;\; mit \sum_1^n g_i = 1 \wedge g_i \in [0,1]$$

dargestellt werden, so dass gilt

$$sim : P_Q \times P_R \times G \rightarrow [0,1] \in R$$

und somit

$$sim\left(P_Q, P_R\right) = \sum_{i=1}^{n} \left(\frac{\left|B_Q^i \wedge B_R^i\right|}{\left|B_Q^i \vee B_R^i\right|} \cdot g_i \right) \in [0,1]$$

Haben verschiedene Attributtypen in ihrer Aussagekraft mehr oder weniger identische Wichtigkeit zur Gesamtmenge, so lassen sich die resultierenden Bloomfilter durch Superimposed Coding überlagern und somit verschmelzen. Dies führt zu einer Reduktion der zu vergleichenden Filtern mit dem Nachteil, dass die False-Positive-Rate mit jedem Filter rein rechnerisch anwächst[84]. Voraussetzung für die Zusammenführung ist jedoch, dass die entsprechenden Filter in ihrer Konstruktion gleichlang sind und dieselbe Anzahl maximal aufnehmbarer Wörter sowie dieselbe False-Positve-Rate aufweisen.

Unter Verwendung eines definierten Schwellwertes ξ lässt sich die semantische Identität resp. Non-Identität zweier digitaler Identitäten, repräsentiert durch ihre relevanten Attributmengen *{W_Q}* und *{W_R}*, anhand ihrer Profile bestimmen

$$sim\left(P_Q, P_R\right) > \xi \Rightarrow \left\{W_Q\right\} \cong \left\{W_R\right\}$$
$$oder$$
$$sim\left(P_Q, P_R\right) \leq \xi \Rightarrow \left\{W_Q\right\} \neq \left\{W_R\right\} \;\; mit \; \xi \in [0,1]$$

[84] $fpr\left(\varepsilon_1 \circ \varepsilon_2 \circ ... \circ \varepsilon_k\right) = 0{,}6185^{\frac{m}{k \cdot n}}$

5 Literatur

Almeidaa, P. S.; Baqueroa, C.; Preguiça, N.; Hutchison, D. (2007): Scalable Bloom Filters; Information Processing Letters, Volume 101, Issue 6, 31 March 2007, pp. 255-261; verfügbar unter: http://gsd.di.uminho.pt/members/cbm/ps/dbloom.pdf

Bloom, Burton H. (1970): Space/Time Trade-off in Hash Coding with Allowable Errors; Communication of the ACM, Vol. 13, No. 7, July 1970, pp. 422-426; verfügbar unter: http://delivery.acm.org/10.1145/370000/362692/p422-bloom.pdf

Blustein, J.; El-Maazawi, A (2002): Bloom filters. a tutorial, analysis, and survey; Technical Report CS-2002-10, Dalhousie University; verfügbar unter: http://www.cs.dal.ca/research/techreports/2002/CS-2002-10.pdf

Boitsov, L. M. (2002): Using Signature Hashing for Approximate String Matching; Computional Mathematics and Modelling, Vol. 13, No. 3, 2002, pp. 314-326

Bose, Prosenjit et al. (2004): On the False-Positive Rate of Bloom Filters; School of Computer Science, Carleton University; verfügbar unter: http://citeseer.ist.psu.edu/649161.html

Bronstein, I. N.; Semendjajew, K. A. (1981): Taschenbuch der Mathmatik; Harri Deutsch Verlag, Thun und Frankfurt/M., 21. Auflage, 1981

Broder, A.; Mitzenmacher, M. (2005): Network Applications of Bloom Filters: A Survey; Internet Mathematics, Vol. 1, No. 4, 2005, pp. 485-509; verfügbar unter: http://www.eecs.harvard.edu/~michaelm/postscripts/im2005b.pdf

Cavnar, William B.; Trenkle, John M. (1994): N-gram-based text categorization; in: Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, NV: UNLV Publications/Reprographics, pp. 161–175

Charikar, Moses S. (2002): Similarity Estimation Techniques from Rounding Algorithms; in: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, Montreal, Quebec, Canada, May 2002, pp. 380 - 388

Christen, Peter (2006): A Comparison of Personal Name Matching: Techniques and Practical Issues; Technical Report TR-CS-06-02, Department of Computer Science, The Australian National University Canberra ACT 0200, Australia; verfügbar unter: cs.anu.edu.au/techreports/2006/TR-CS-06-02.pdf

Cohen, Jonathan D. (1997): Recursive Hashing Functions for n-Grams; ACM Transactions on Information Systems, Vol. 15, No. 3, July 1997, pp. 291–320

Cohen, W. W.; Ravikumar, P.; Fienberg, S. E. (2003): A comparison of string distance metrics for name-matching tasks; in Proceedings of the IJCAI-2003, verfügbar unter: http://citeseer.ist.psu.edu/cohen03comparison.html

Cross, Valerie; Cabello, Cesar (1995): A Mathematical Relationship Between Set-Theoretic and Metric Compatibility Measures; isuma, 3rd International Symposium on Uncertainty Modelling and Analysis, 1995, pp. 169-174, verfügbar unter: http://csdl.computer.org/dl/proceedings/isuma/1995/7126/00/71260169.pdf

Czech, Zbigniew J. (1998): Quasi-perfect Hashing; The Computer Journal, Vol. 41, No. 6, pp. 416--421, 1998; verfügbar unter: http://citeseer.ist.psu.edu/220919.html

Dillinger, P. C.; Manolios, P. (2004): Bloom Filters in Probabilistic Verification; in Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design, 2004, pp. 367-381

Du, Mengmeng (2005): Approximate Name Matching – Finding similar personal names in large international name lists; Master Thesis, KTH Royal Institute of Technology, Dep. Numerical Analysis and

Computer Science, Stockholm, Sweden 2005; verfügbar unter: http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2005/rapporter05/du_mengmeng_05137.pdf

Elmagarmid, Ahmed K. et al. (2007): Duplicate Record Detection: A Survey ; IEEE Transactions on Knowledge and Data Enginieering, Vol. 19, No. 1, January 2007, pp. 1-16

Ekmekçioglu, F. Çuna; Lynch, Michael F.; Willett, Peter (1996): Stemming and N-gram matching for term conflation in Turkish texts; Information Research, verfügbar unter: http://informationr.net/ir/2-2/paper13.html

El-Qawasmeh, Eyas; Al-Qarqaz, Wafa'a (2006): Reducing Lookup Table Size Used For Bit-Counting Algorithm; in: Proceedings of the 4th International Multiconference on Computer Science and Information Technology CSIT 2006, Vol. 3, pp. 524-532, Amman, Jordan, April 2006; verfügbar unter: http://csit2006.asu.edu.jo/proceedings/vol3%20pdf/pg524.pdf

Faloutsos, Christos (1988): Signature files: An integrated access method for text and attributes, suitable for optical disk storage; BIT Numerical Mathematics, Volume 28, Number 4 / Dezember 1988, pp. 736-754

Fan, Li et al. (1998): Summary Cache A Scalable Wide-Area Web Cache Sharing Protocol; Technical Report 1361, Department of Computer Science, University of Wisconsin-Madison, Feb 1998; verfügbar unter: http://pages.cs.wisc.edu/~cao/papers/summarycache.html

Ferber, Reginald (2003): Information Retrieval – Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web; dpunkt.verlag, März 2003, ISBN-10 3-89864-213-5

Grefenstette, G.; Tapanainen, P (1994): What is a word, What is a sentence? Problems of Tokenization; In: 3rd Conference on Computational Lexicography an Text Research COMPLEX '94, Budapest, July 1994, pp. 79-87; verfügbar unter: http://citeseer.ist.psu.edu/grefenstette94what.html

Haustein, Michael P. (2002): Ähnlichkeitssuche in objekt-relationalen Datenbanksystemen; Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik, AG Datenbanken und Informationssysteme, Mai 2002

Henning, Christian (2001): Was ist Wahrscheinlichkeit?; Department of Statistical Science, UCL, London; Papers and Publications by Dr. Christian Henning; Foundations of Statistics and Constructivist Philosophy; verfügbar unter: http://www.homepages.ucl.ac.uk/~ucakche/

Hyyrö, Heiki (2003): A bit-vector algorithm for computing Levenshtein and Damerau edit distances; in: Nordic Journal of Computing 2003, Vol. 10, Iss. 1, pp. 29 – 39; verfügbar unter: http://citeseer-.ist.psu.edu/537930.htm

Jaccard, Paul (1912): The Distribution of the Flora in the Alpine Zone; New Phytologist, Vol. 11, No. 2, Feb. 1912, pp. 37-50

Jain, Navendu; Dahlin, Mike; Tewari, Renu (2005): Using Bloom Filters to Refine Web Search Results; Eighth International Workshop on Web and Databases (WebDB 2005), June 16-17, 2005, Baltimore, Maryland; verfügbar unter: http://www.cs.utexas.edu/users/dahlin/papers/webdb-167.pdf

Järvelin, Anni et al. (2006): Dictionaryindependent translation in CLIR between closely related languages. In: Proceedings of the 6th DutchBelgian Information Retrieval workshop, March, 2006 Delft, The Netherlands. verfügbar unter: http://hmi.ewi.utwente.nl/dir2006/proceedings.php

Kirsch, A.; Mitzenmacher, M. (2006a): Distance-Sensitive Bloom Filters; verfügbar unter: http://www.eecs.harvard.edu/~michaelm/postscripts/alenex2006.pdf

Kirsch, A.; Mitzenmacher, M. (2006b): Less Hashing, Same Performance: Building a Better Bloom Filter; in Y. Azar and T. Erlebach (Eds.): ESA 2006, LNCS 4168, Springer-Verlag Berlin Heidelberg 2006, pp. 456–467

Knappe, Rasmus (2005): Measures of Semantic Similarity and Relatedness for Use in Ontology-based Information Retrieval; Dissertation, Department of Communication, Journalism and Computer Science, Roskilde University, Denmark 2005

Kutzelnigg, Reinhard (2005): Analyse von Hash-Algorithmen; Diplomarbeit, Institut für Diskrete Mathematik und Geometrie, Technischen Universität Wien, 2005; verfügbar unter: http://dmg.tuwien.ac.at/drmota

Lanquillon, Carsten (2001): Enhancing Text Classification to Improve Information Filtering; Dissertation, Fakultät für Informatik, Otto-von-Guericke-Universität, Magdeburg, 2001

Laufer, R. P.; Velloso, P. B.; Duarte, O. C. M. B. (2005): Generalized Bloom Filters, Technical Report GTA-05-43, COPPE/UFRJ, September 2005; verfügbar unter: http://www.cs.ucla.edu/~rlaufer/publications/gbf.pdf

Matthé, Tom et al. (2006): Similarity Between Multi-valued Thesaurus Attributes: Theory and Application in Multimedia Systems; in: Flexible Query Answering Systems, 7th International Conference (FQAS 2006), Milan, Italy, June 2006, Lecture Notes in Computer Science, Heidelberg: Springer Berlin, pp. 331 - 342. ISSN: 0302-9743, ISBN: 3-540-34638-4; verfügbar unter: http://www.ipem.u-gent.be/staff/marc/Papers2006/2006-FQAS-MattheDeCaluweEtAl-ThesaurusAttributes.pdf

Mayfield, J.; McNamee, P. (1998): Indexing Using Both N-Grams and Words; In: NIST Special Publication 500-242; The Seventh Text REtrieval Conference (TREC 7), 1998, 419-424; verfügbar unter: http://citeseer.ist.psu.edu/mayfield98indexing.html

Metawally, A. et al. (2005): Duplicate Detection in Click Streams; Proceedings of the 14th WWW International World Wide Web Conference, May 2005, pp. 12–21

Mustafa, Suleiman H.; Al-Radaideh, Qasem A. (2004): Using N-Grams for Arabic Text Searching; Journal of the American Society for Information Science and Technology, 2004, Volume 55, Issue 11, pp. 1002 – 1007

Nafe, Clemens (2005): Indexierung lokaler Daten in Peer-to-Peer-Netzen; Diplomarbeit, Lehrstuhl Datenbank- und Informationssysteme, Universität Rostock, 2005; verfügbar unter: http://www.xml-und-datenbanken.de/sada/da-nafe.pdf

Naumann, Felix (2007): Datenqualität; Informatik Spektrum, Bd. 30, Heft 1, Springer Verlag Februar 2007, S. 27-31

Pfaltz, John L.; Berman, William J.; Cagley, Edgar M. (1980): Partial-Match Retrieval Using Indexed Descriptor Files; Communications of ACM, Vol. 23, No. 9, 1980, pp. 522-528

Rapp, Reinhard; Wettler, Manfred (1994): Kontextsensitive Rechtschreibfehlerkorrektur auf der Basis von Wortnachbarschaften; in: D.W. Halwachs, I. Stuetz (eds.), Sprache, Sprechen, Handeln, Akten des 28 Linguistischen Kolloquium, Graz 1993. Vol. 2, Tübingen, Niemeyer, pp. 341-348, 1994

Roozenburg, Jelle (2005): A Literature Survey on Bloom Filters; Research Assignment in Computer Science; Parallel and Distributed Systems group, Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, 2005; verfügbar unter: http://svn.tribler.org/bt2-design/bloom_filters/research_assignment_jroozenburg_20051108.pdf

Schulz, Jochen (2006): Unscharfe Suche in großen Adressbeständen; Diplomarbeit, Fachbereich Wirtschaftsinformatik, Fachhochschule Nordakademie, Elmshorn, Deutschland, 2006

Schulze, Inga (2006): Robuste Verfahren zur Erkennung von Schreibfehlern in elektronisch verfügbaren Dokumente; Diplomarbeit, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg, 2006; verfügbar unter: http://wdok.cs.uni-magdeburg.de/publikationen/dokumente/diplom_schulze.pdf

Talbot, David; Osborne, Miles (2007): Randomised Language Modelling for Statistical Machine Translation; in: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 512–519, Prague, Czech Republic, June 2007

Tauritz, Daniel R. (2002): Adaptive Information Filtering: concepts and algorithms. Dissertation, Leiden University, The Netherlands, 2002, ISBN 90-9015926-6

Wolper, Pierre; Leroy Denis (1993): Reliable Hashing without Collision Detection; in: Computer Aided Verification, Proc. Int. Workshop, Elounda, Crete, Lecture Notes in Computer Science, Vol. 697, Springer Verlag, June 1993, pp. 59-70

Ziembicki, Joanna Irena (2006): Distributed Search in Semantic Web Service Discovery; Master Thesis, University of Waterloo, Ontario, Canada 2006

Zhu, Yifeng; Jiang, Hong (2006): False Rate Analysis of Bloom Filter Replicas in Distributed Systems; Proceedings of the 2006 International Conference on Parallel Processing (ICPP'06), August 2006, pp. 255 - 262

6 Anhang

6.1 Funktion “bloomfilter” (Pseudocode)

```
FUNCTION bloomfilter (string)

   FUNCTION hash (nGramIndex)
      SubFilterLength := BloomFilterLength / NumberOfHashes
      BloomMatrix := ARRAY OF BIT [NumberOfHashes, SubFilterLength]
      h1 := nGramIndex MOD SubFilterLength
      h2 := nGramIndex MOD (SubFilterLength-1) + 1
      BloomMatrix [0,h1] := 1
      FOR i := 1 TO NumberOfHashes-1
         h1 := (h1 + h2) MOD SubFilterLength
         BloomMatrix [i,h1] := 1
      END FOR
     hash := BloomMatrix
   END FUNCTION

   NumberOfElements := Number of Elements of Alphabet A
   Signature := 0
   Index := 0

   FOR i := 1 TO nGramLength
      Index := Index * NumberOfElements + pos(String[i])
   END FOR

   Signature := Signature OR hash(Index)

   FOR i := 1 TO StringLength
      j := i + nGramLength
      Index := Index – ord(String[i] * NumberOfElements ^(nGramLength-1)
      Index := Index * NumberOfElements + ord(String[j])
      Signature := Signature OR hash(Index)
   END FOR
   bloomfilter := Signature

END FUNCTION
```