TU-Berlin

Fakultät IV

IMSEM: Data Mining mit Apache Mahout (WS09/10)

Dozentin: Isabel Drost

Erkennen von Duplikaten und nahen Duplikaten in Textkorpora

Vorgelegt von:

Dirk Dieter Flamming
Pfahlerstr. 4
13403 Berlin
Flamming´at´cs.tu-berlin.de
Matrikelnr.: 305467

Inhaltsverzeichnis

1. Vorwort

Digitale Dokumente kommen sind aus dem heutigen Alltag nicht mehr wegzudenken, sei es nun im privaten Gebrauch, wie dem surfen in Webseiten und der dadurch resultierenden Suche nach bestimmten Dingen auf diesen, oder im kommerziellen Bereich, in dem Anwälte auf der Jagd nach Urheberrechts-Verletzungen sind, die in veröffentlichen PDF-Dokumenten zu finden sind.

Viele von uns verwendeten Funktionen verwenden die Erkennung von Duplikaten und nahen Duplikaten in Textkorpora ohne dass wir es erwarten, wie zum Beispiel Suchmaschinen für wissenschaftliche Arbeiten wie CiteSeer oder SPAM-Filter für E-Mail-Server. Also spielt es nicht einmal eine Rolle ob man die Duplikate vermeiden möchte oder ob man gerade auf der Suche nach nahen Duplikaten ist um zum Beispiel eine Web-Seite zu finden die die gleichen Themen behandelt wie eine favorisierte bereits bekannte Seite.

In dieser Arbeit soll nahe gelegt werden, was Duplikate und nahe Duplikate überhaupt sind und einen kleinen Überblick über allgemein bekannte Techniken zu geben die zur Duplikat-Erkennung verwendet werden. Darüber hinaus wird noch auf einen bestimmten Algorithmus eingegangen der die Suche von nahen Duplikaten in Form von Clustering behandelt.

2. Duplikate

Ein Duplikat ist eine exakte Kopie eines Objektes. Während dies in der klassischen Duplikat-Erkennung, wie sie zu *Information Retrieval*-Zwecken in der Informationsintegration, genauer der Datenintegration, bei der Fusion mindestens zweier Datenbanken verwendet wird, vor allem solche Daten betrifft, die durch einen Tippfehler oder der unterschiedlichen Anordnung von Begriffen (z.B. die Reihenfolge von Vor- und Nachnamen in einem Namensfeld einer Datenbank) zustande gekommen sind, so hat sich heutzutage das Erkennen von Duplikaten auch in anderen Bereichen als nützlich erwiesen.

Nun werden nicht nur einfache Einträge in Datenbanken auf Duplikate überprüft, sondern zum Teil komplexe Dokumente. So wurden bereits Duplikats-Erkennungs-Systeme entwickelt, die für Anwendungsbereiche wie

- Web- Dokumente
- Dateien in Dateisystemen
- E-Mails
- Domain-spezifische Korpora

verwendet werden.

Bei Duplikaten von Dokumenten zielt man allerdings nicht darauf ab Tippfehler oder sonstiges auszumerzen, sondern auf dem jeweiligen Anwendungsbereich passende Ziele.

Zum Zwecke der Erläuterung ziehen wie die oben genannten Anwendungsbereiche heran:

Die Suche von nahen Duplikaten zum Beispiel, lässt sich in Web-Dokumenten nutzen um Seiten zu finden die inhaltlich miteinander in Beziehung stehen, jedoch nicht miteinander verlinkt sind oder um in einer schnellen Suche erkennen zu können ob eine Web-Seite von einer anderen exakt (inhaltlich nicht strukturell) kopiert wird. Durch die Identifizierung von nahen Duplikaten von Web-Seiten lassen sich durch die daraus kosten-effektivere Datensätze speichern, in denen man durch die Abwesenheit von (nahen) Duplikaten wiederum Suchanfragen wesentlich schneller und besser beantworten kann.

Desweiteren können nahe Duplikate dazu verwendet werden um zum Beispiel News-Einträge von Nachrichtenseiten wie „Spiegel Online“ und diversen Blogs zu vergleichen um herausfinden zu können wo ein bestimmter Beitrag seinen Ursprung hat.

Für beliebige Dateien in einem Dateisystem lassen sich die Vorteile für eine Duplikat-Erkennung leicht sehen, da man statt der Duplikate (sofern es sich nicht um eine gewollte Redundanz von Daten handelt) dann falls überhaupt nur „Verknüpfungen“ benötigt und somit viel Platz sparen kann und man ggf. sogar ähnliche Dateien (nahe Duplikate) finden kann, die unter Umständen nicht benötigt werden, da der Unterschied der Dateien lediglich eine weitere Leerzeile am Dateiende sein könnte.

In der Software von E-Mail-Servern werden ebenfalls nahe Duplikate verwendet, hier allerdings zur SPAM-Erkennung. Da bei SPAM-Mails davon auszugehen ist, dass sie an möglichst viele Adressen des Servers gesendet werden und diese sich Erfahrungsgemäß in ihren Textkörpern nicht oder zumindest nur wenig im Textkörper unterscheiden, lässt sich hier ein Mechanismus einsetzen, der Mails die in großer Zahl an die Adressen des Servers geschickt werden aussortiert werden noch bevor sie im Post-Eingang eines Mail-Kontos landen.

Bei der Duplikat-Erkennung in Domänen-spezifischen Korpora geht es vor allem um die Erkennung von nahen Duplikaten von mehreren Revisionen von Dokumenten und das Erkennen von Dokumenten die aus mehreren verschiedenen Dokumenten zusammengesetzt sind und sogar um die Erkennung von Urkunden die einer anderen ähneln.

2.1. Nahe Duplikate

Duplikate zeichnen sich durch den einfachen Vergleich aus, der benötigt wird um diese zu erkennen. Nahe Duplikate hingegen unterscheiden sich unter Umständen nur in einigen wenigen Worten von einem anderen Dokument.

Ein gutes Beispiel für ein solches Dokument wäre eines, das eine exakte Kopie eines anderen wäre, wenn es nicht einen Fachbegriff durch ein Synonym ersetzt hätte und daher einen kleinen Unterschied aufweist. Natürlich würde einem Menschen so etwas leicht auffallen, wenn er beide Dokumente vor sich liegen hat, aber einem Computer-System nicht, da sich durch diese Änderung unter Umständen einiges an der Datei-Struktur ändern kann.

Ein weiteres Beispiel wäre bei einem Vergleich von News-Blogs möglich, bei denen der eine Blog Zeitstempel und Ort des Geschehens direkt in den Titel des Beitrages einfügt und der andere nicht. Auf diese Weise könnte es sein, dass ein Erkennungs-System diese für Unterschiedlich erachtet, da wenn man den gesamten Titel verwendet zwar bemerkt, dass der eine Titel ein Teil des anderen ist, das System allerdings nur feststellen kann, dass sich die Bit-Codes der beiden Titel stark unterscheiden.

3. Techniken und Werkzeuge

Mit der Zeit sind, wie in jedem wissenschaftlichen Gebiet, mit der Zeit immer mehr Techniken und Werkzeuge aufgetreten, die einem dabei helfen sein Problem zu lösen. Im Fall der Duplikat-Erkennung kann man sehen, dass einige dieser Ideen besonders oft leichten Erweiterungen und/oder Verbesserungen erliegen. Im Folgenden sollen einige dieser Techniken und Werkzeuge vorgestellt werden um einen kleinen Überblick in diesem Bereich zu verschaffen.

3.1. Levenshtein-Distanz

Die auch als Edit-Distanz bekannte Levenshtein-Distanz bezeichnet die minimale Anzahl an Einfüge-, Lösch- und Änderungs-Operationen die benötigt werden um eine Zeichenkette in eine bestimmte andere zu überführen.

Diese Distanz wird vor allem in der Duplikat-Erkennung in Datenbanken verwendet um zum Beispiel Duplikate zu vermeiden die durch Tippfehler aufgetreten sind, da dort eher kürzere Zeichenketten vorkommen und der Aufwand somit noch nicht all zu hoch ist.

3.2. Fingerprints

Bei der Erkennung von identischen Duplikaten (vor allem in Suchmaschinen) hat es sich für praktisch erwiesen, dass man Dokumenten einen Fingerprint zuteilt, der für dieses Dokument immer identisch sein muss. Für diesen Zweck hat man Hash-Algorithmen verwendet, die zum einen platzsparend sind und zum anderen Duplikate schneller zu ermitteln sind. Der Nachteil dieser Methode ist allerdings, das kleine Änderungen schon ausreichen können um zwei sehr ähnlichen Dokumenten (z.B. eins mit und eines ohne der Signatur des Webmasters) Fingerprints zuzuteilen, die sich sehr stark voneinander unterscheiden.

Dieses Problem konnte *Charikar* beheben, indem er einen Algorithmus entwickelte, der auf Dokumenten Vektoren basiert und diese soweit komprimiert, dass zwei Fingerprints einander ähnlich sind, wenn deren dazugehörige Dokumenten Vektoren einander ähneln. Dieser Algorithmus ist nun allgemein bekannt als *simhash*.

Zusammen mit dem Hamming-Abstand, der im Grunde eine Edit-Distanz auf gleich lange Zeichenketten ist, kann man durch *simhash* nun auch nahe Duplikate von Dokumenten erkennen, da die von *simhash* generierten Fingerprints sich in diesen Fällen in einem geringen Hamming-Abstand zueinander befinden sollten.

3.3. Stemming

Vor allem in relativ kurzen Text-Segmenten können kleine Abweichungen der Wörter bei gleich bleibenden Wortstamm vorkommen, die allerdings zu großen Abweichungen bei z.B. Hash-Funktionen führen.

Aus diesem Grund ist es bei solch kurzen Text-Segmenten einzelne Wörter auf ihren Wortstamm zurückzuführen und diese weiter mit den Wörtern (und deren Wortstämmen) zu Vergleichen.

Einer der bekanntesten dieser Sorte ist der Porter-Stemmer, der zwar nicht auf die linguistischen Wortstämme abbildet, jedoch trotzdem verwandte Worte auf eine durch sinnvolle Verkürzung der Zeichenketten basierte Methode zurückführt. Dabei geht der Porter-Stemmer nicht auf die wirkliche Silbenbildung der Worte ein, sondern bezieht sich ausschließlich auf die Vokal-

Konsonanten-Sequenzen und trennt entsprechend immer mehr dieser Sequenzen ab um mehr Begriffe erfassen zu können.

Daraus ergibt sich aber auch das Problem eines Stemmers, da dieser auch viele Falsche Ergebnisse bekommen kann. Diese falschen Ergebnisse sind allerdings verkraftbar, wenn man die außerordentlich hohe Zahl an richtigen Treffern betrachtet.

3.4. Shingles (Dachziegel)

Um mit einem Fingerprint nicht nur einen einfachen Hash-Wert für ein Dokument zu erzeugen hat man die Verwendung von *shingles* eingeführt. Um *shingles* zu erzeugen wird ein Dokument in Tokens zersetzt, wobei hierbei Dinge wie HTML-Tags, Satzzeichen etc. herausgefiltert werden.

Danach werden Teilfolgen dieser Tokens als *shingles* verwendet, sodass diese sich wie Dachziegel überlappen. So wird das *w- shingling* S(D, *w*) als Menger aller einzigartiger *shingles* der Länge *w* in einem Dokument D definiert

Ein 4- *shingling* für die Tokensequenz

(a, rose, is, a, rose, is, a, rose)

würde die Menge

{ (a, rose, is, a), (rose, is, a, rose), (is, a, rose, is) }

ergeben.

Beispiel aus [8]

Durch solche *shingles* lassen nahe Duplikate gut erkennen, wie sich im folgenden Kapitel dieses Dokuments zeigen wird.

4. Syntactic clustering of the Web

In diesem Kapitel wird der Einsatz von *shingles* zur Duplikat-Erkennung zum Zwecke des Clustering von digitalen Dokumenten von *Broder et al.* ([3] und [8]) behandelt. Mögliche Einsätze dieser Methode wären

- Berechnung für ähnliche Dokumente zu einem Dokument von einer bestimmten URL

- "Lost and Found"- Service, der Webseiten, die ihre URL geändert haben, automatisch bemerkt, sodass man bei Angabe der alten URL die neue Adresse von diesem Service erhält

- Zusammenlegen von Suchergebnissen, sodass der Benutzer einer Suchmaschine ähnliche Dokumente kompakter anzeigt, sodass dieser sehen kann, das diese ähnliches beinhalten, sich jedoch für ein bevorzugtes entscheiden kann.

- Auffinden von Plagiaten: Man kann Dokumente ausfindig machen, die zumindest Teilweise auf dem geistigen Eigentum beruhen, dass nicht zur Weiterverbreitung freigegeben ist oder sogar als eigenes geistiges Eigentum ausgegeben wird.

4.1. Definition der Ähnlichkeit von Dokumenten

In *Broder et al.´*s Arbeit werden die Begriffe *resemblance*, welcher die Ähnlichkeit zweier Dokumente zueinander beschreibt, und *containment*, der eine Aussage darüber trifft wie stark ein Dokument in einem anderen enthalten ist, verwendet.

Sowohl die *resemblance*, als auch das *containment* nehmen dabei einen Wert zwischen 0 und 1 an, wobei 1 für eine maximale Ähnlichkeit steht bzw. zeigt, dass ein Dokument A vollständig in einem Dokument B enthalten ist.

Um die *resemblance* zu berechnen wird hier die Methode des *shingling* (siehe Kapitel 3.4) verwendet um die Problematik auf ein Schnittmengen-Problem zu reduzieren. Wie bereits beschrieben kann man so für jedes Dokument eine Menge S(D, *w*) bilden, die alle einzigartigen shingles des Dokuments D enthält. Da wir im folgenden immer ein für alle Dokumente festgelegtes beliebiges *w* verwenden und um Speicherkapazität für die Berechnung und vor allem auch für die weitere Datenhaltung zu sparen werden diese shingles mit Fingerprints versehen, sodass die unter Umständen langen shingles nur wenige Bits Speicherplatz benötigen.

"In der Praxis sind 64-bit Rabin-Fingerprints ausreichend" (Broder 2000), da bei diesen die Wahrscheinlichkeit, dass zwei ungleiche Fingerprints den gleichen Wert zugewiesen bekommen.

Def. 1: Die *resemblance r*(A, B) von zwei Dokumenten A und B, ist definiert als

$$\mathrm{r}(A,B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|}$$

Def. 2: Das containment von zwei Dokumenten A und B ist definiert als

$$c(A,B) = \frac{|S_A \cap S_B|}{|S_A|}$$

Da wir nun die shingles mit Fingerprints versehen haben, benötigen wir eine neue Menge S_D in der alle Fingerprints (, welche Zahlen sind,) der *shingles* aus dem Dokument D enthalten sind.

An den Definitionen 1 und 2 aus [8] lässt sich leicht erkennen wie nützlich die shingles sein können, da wir aus ihnen direkt ihre Ähnlichkeit berechnen können indem wir Vereinigungs- und Schnittmengen der eben genannten Mengen anfertigen.

4.2. Schätzen von *resemblance* und *containment*

Da S_D allerdings eine relativ große Menge ist (genau genommen: $|S_D| = n - w + 1$ wobei n die Anzahl und *w* wie gehabt die Länge der shingles sind), ist die laufende Speicherung dieser großen Datenmenge nicht praktikabel.

Um diesen Missstand zu umgehen nutzt Broder die Möglichkeit die *resemblance* und das *containment* zu schätzen, indem er nur eine bestimmte Menge zufällig gewählter *shingles* verwenden möchte um diese zu berechnen.

Dieser *sketch*, wie diese Teilmenge an Fingerprints aus S_D genannt werden, kann für ein großes Dokument kleiner als 50 Bytes sein und ist damit ungleich platzsparender als ein ganzes Dokument im Speicher laden zu müssen.

Im Folgenden soll Erläutert werden wie man diese *sketches* erhält:

Nehmen wir an, das für alle Dokumente die wir betrachten $S_D \subseteq \{0, \dots, n-1\} \stackrel{\text{def}}{=} [n]$ gilt, wobei n eine beliebige Zahl ist ($n = 2^{64}$ hat sich in der Praxis als nützlich erwiesen).

Nun wird π zufällig aus der Menge der Permutationen von $[n]$ gewählt, sodass wir nun eine feste Zahl t wählen können um eine Menge von Permutationen zu generieren, wobei t günstig zu wählen ist um weder zu viel Speicher aufzuwenden, noch übermäßig an Genauigkeit zu verlieren.

Nun können wir eine Funktion $min(S_D)$ so definieren, dass diese Funktion das kleinste Element aus S_D wiedergibt, wobei sich dabei auf die numerische Reihenfolge in S_D bezogen wird.

Durch diese Funktion können wir nun unseren sketch für das Dokument A folgendermaßen definieren:

$$\overline{S}_D = \{\min\{\pi_1(S_A)\}, \min\{\pi_2(S_A)\}, \dots, \min\{\pi_t(S_A)\}\}$$

Es hat sich zwar erwiesen, dass man in einigen Situation auch eine einzelne zufällige Permutation wählen und dann einfach die t niedrigsten Elemente aus S_D wählen kann (wie es in [8] angewendet wurde), aber man sollte den allgemeinen Fall bevorzugt erwähnen.

Broder [3] hat gezeigt, dass für die *resemblance* zwischen zwei solcher sketches der *resemblance* der gesamten *shingles* der Dokumente hinreichend ähnelt um für die Zwecke der Erkennung von nahen Duplikaten auszureichen.

Als Alternative zur ausschließlichen Nutzung einer bestimmten Menge von *shingles* für einen *sketch*, kann man auch wie *Broder et al.* [8] bemerken eine Modulo-Funktion definieren, die jedes m-te Element aus der ‚nach einem zufälligen π-permutierten, Menge S_D in die Menge $\overline{S}_D$ einfügt. Diese sketches hätten zwar den Nachteil, dass sie im gleichen Maße größer werden wie das Dokument D auf das sie sich beziehen, würden jedoch den Vorteil haben, dass man aus ihnen nicht nur die resemblance, sondern auch das *containment* hinreichend gut bestimmen könnte.

4.3. Clustering-Algorithmus

Den Ablauf des Algorithmus für das Clustering könnte man kurzfassen, indem man

1. Sich alle Dokumente heranzieht die man betrachten möchte. (zum Beispiel jedes Web-Dokument)
2. Für jedes Dokument einen sketch berechnen
3. Die sketches für jedes mögliche Dokumenten paar vergleichen und nachsehen ob sie einen bestimmten resemblance-Wert übersteigen
4. Einen Cluster für diejenigen Dokumente anfertigen, die sich untereinander ähneln

Auch wenn dies sehr einfach klingt, so würde eine einfache Implementation quasi unausführbar sein, da für den Testfall in [8] 30 millionen HTML- und Text-Dokumente verwendet wurden und man für den paarweisen Vergleich 10^{15} Vergleiche vorzunehmen hätte.

Also wird der der Algorithmus in vier Phasen aufgeteilt, wobei die erste Phase für die Anfertigung der *sketches* für alle Dokumente beinhaltet.

In der zweiten Phase erweitern wir die *sketches* für jedes Dokument so, dass sie Listen von <*shingle value, document ID*>-Paaren entsprechen, wobei die *shingle value* für den Fingerprint steht den der dazugehörige *shingle* erhalten hat und die *document ID* eine für jedes Dokument

eindeutige Zahl ist. Die daraus resultierende Gesamtliste wird dann mithilfe eines „divide, sort, merge“-Ansatzes nach *shingle value ´*s sortiert, sodass für jede Teilliste nur der Arbeitsspeicher zur Berechnung verwendet werden muss, was die Geschwindigkeit des Algorithmus drastisch steigert.

In der dritten Phase wird die sortierte Liste aus der zweiten Phase verwendet um die Anzahl der gemeinsamen shingles zweier Dokumente zu ermitteln und dadurch unsere sketches ein weiteres Mal zu erweitern. Dieser Erweiterung erfolgt indem wir die vorhanden Liste auf das Tripel <*ID, ID, Anzahl gemeinsamer shingles*> erweitern Dafür werden zuerst alle Tripel <ID, ID, 1> erzeugt um anschließend wieder mit einem eines „divide, sort, merge“-Ansatz zu arbeiten und alle ID-ID-Paare zu zählen und entsprechend den gewollten fertigen Triple erhalten.

In der vierten und letzten Phase werden alle generierten Tripel untersucht und die resemblance berechnet. Wenn die berechnete resemblance für ein Paar ausreichen hoch war, so kann man eine Verbindung dieser Dokumente durch einen Union-Find Algorithmus eingeben, um am Ende die endgültigen Cluster zu erhalten.

Um Anfragen mit größerem Nutzen ausführen zu können ließe sich außerdem noch eine Zuordnung von einer URL zu einem Dokument und ähnliches erstellen um bestimmte Arten von Anfragen leichter beantworten zu können.

4.4. Performance

Durch diverse Testläufe dieses Algorithmus hat sich ergeben, dass manche *shingles* besonders häufig aufgetreten sind, mit häufig wurden von *Broder et al*. [8] shingles bezeichnet, die in mehr als 1000 Dokumenten aufgetreten sind. Nach einer Analyse dieser *common shingles* hat sich gezeigt, dass die meisten dieser shingles von Programmen, wie HTML-Editoren, automatisch generiert wurden. Da diese *common shingles* in den meisten Fällen wenn überhaupt die *resemblance* von Dokumenten eher verfälschten, wurden diese im Praxisfall herausgefiltert.

Desweiteren ist es vorgekommen, das identische Dokumente vorkamen, die zumindest für Ansätze für Suchmaschinen irrelevant sind und bei ihnen nur Rechenzeit und Speicherplatz kosten. Aus diesem Grund kann man Dokumente für das Clustering entfernen, die den gleichen Fingerprint haben, wie ein bereits vorhandenes Dokument.

Eine weitere Möglichkeit Ressourcen zu sparen ist es sogenannte *super shingles* einzusetzen. *Super shingles* sind im Grunde *shingles* die aus *shingles* bestehen, wodurch in Phase zwei und drei des

Algorithmus bedeutend weniger Paare bzw. Tripel zustande kommen, da nur sehr ähnliche Dokumente mehrere Folgen von *shingles* gemeinsam haben, was wiederum sehr viele Berechnungen und Speicherplatz spart.
Da zwei Dokumente die einen gemeinsamen *super shingle* haben, gleichzeitig mehrere *shingles* gemeinsam haben, kann man bei diesen Dokumenten davon ausgehen das sie einander Ähnlich sind. Wenn die Anzahl der *shingles* in einem *super shingle* gut gewählt ist, ist wahrscheinlich, dass zwei ähnliche Dokumente mindestens einen *super shingle* gemein haben, was wiederum darauf schließen lässt, dass zwei Dokumente die einen *super shingle* gemein haben sich einander ähneln. Wenn man diese Vermutung, welche sich laut *Broder et al.* [8] bewahrheitet, trifft, kann man also bei dem Algorithmus schon bei bereits einem gemeinsamen *super shingle* einen Eintrag im Cluster vornehmen. Der Nachteil von *super shingles* liegt allerdings in kurzen Dokumenten, da sie sehr unflexibel gegenüber den normalen shingles sind.

5. Literaturverzeichnis

[1] M.Henzinger: Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms. In SIGIR´06.

[2] M, Bendersky, W. Bruce Croft: Finding Text Reuse on the Web. In WSDM´09

[3] A. Z. Broder: Identifying and Filtering Near-Duplicate Documents. In CPM 2000

[4] H. Chang, J. Wang: Organizing News Archives by Near-Duplicate Copy Detection in Digital Libraries. In ICADL 2007

[5] C. Gong, Y. Huang, X. Cheng und S. Bai: Detecting Near-Duplicates in Large-Scale Short Text Databases. In PAKDD 2008

[6] S. Brin, J. Davis und H. Garcia-Molina: Copy Detection Mechanism for Digital Documents. In SIGMOD´95

[7] D. Metzler, S. Dumais und C. Meek: Similarity Measures for Short Segments of Text. In ECIR 2007

[8] A. Z. Broder, S. C. Glassman, M. S. Manasse und G. Zweig: Syntactic clustering of the Web.1997 veröffentlicht von Elsevier Science B.V.

[9] G. S. Manku, A. Jain, Anish Das Sarma: Detecting Near-Duplicates for Web Crawling. In WWW 2007

[10] O. Abdel-Hamid, B. Behzadi, S. Christoph und M. Henzinger: Detecting the Origin of Text Segments Efficiently. In WWW 2009