Dublettenerkennung in Adress-Daten mit Unix Bordmitteln

Dirk Weimar, Februar 2014

Zusammenfassung

Der folgende Beitrag beschreibt ein einfaches, performantes Verfahren zur Erkennung und Extraktion von Duplikaten in Adressdaten, sowie die Umsetzung des Verfahrens mit Kommandozeilen-Tools, die auf fast jedem Unix System vorhanden sind (*awk*, *sed*, *sort* und *iconv*). Duplikate oder Dubletten sind unterschiedliche Datensätze in Adressdaten, die die selbe Person repräsentieren.

Im Gegensatz zu verbreiteten Verfahren, die eine *unscharfe Suche* zur Dubletten-Erkennung nutzen (z.B. auf Basis von N-Grammen oder der Edit-Distanz), basiert das hier vorgestellte auf reinen String-Vergleichen, was einen erheblichen Performance-Gewinn mit sich bringt. (Im Anwendungsfall benötigte die zuvor verwendete Software über zwei Stunden, das Unix-Tool dagegen nur wenige Sekunden).

Verwendete Daten

Es wurde eine aus ca. 600.000 Adressdatensätzen bestehende Kundendatenbank eines deutschen Handels-Unternehmens auf Dubletten untersucht. Es handelt sich zu ca. 85% um Adressen aus Deutschland, weitere 10% aus Österreich und der Schweiz und ca. 5% aus anderen Europäischen Ländern und dem Rest der Welt. Die Daten wurden von den Kunden selbst im Zuge des Bestellprozesses über ein Webformular eingegeben („echte“ Daten).

Die aus dem ERP extrahierte flache Textdatei (Input-Datei), in der nach Dubletten gesucht werden soll, hat den folgenden Aufbau:

```
ID    Vorname      Nachname      Strasse      Haus-Nr.    PLZ   Ort    Land     E-Mail-Adresse
```

Vorgehensweise

Die Input-Datei wird zunächst **normalisiert**. Dabei werden unter anderem Umlaute ersetzt, Sonderzeichen entfernt und alle Buchstaben in Versalien umgewandelt, um eine bessere Vergleichbarkeit zu erreichen.

Danach werden aus den für die Erkennung von Dubletten relevanten Feldern (E-Mail-Adresse, Vorname, Nachname, Postleitzahl) zwei unterschiedliche **Schlüssel** gebildet. Die Schlüssel werden als nächstes in zwei zusätzliche Felder an das Ende jeder Zeile in die Datei geschrieben.

Dann wird die Datei in zwei Durchläufen nach dem jeweiligen Schlüssel **sortiert**.

Bis hierhin ist das Verfahren bekannt unter dem Namen *Sorted Neighborhood*. Im Unterschied dazu werden jedoch bei dem hier vorgestellten Vorgehen die Schlüssel so gewählt, dass zwei Datensätze mit dem selben Schlüssel mit nahezu 100%iger Wahrscheinlichkeit die selbe Person repräsentieren. D. h. zwei Datensätze mit identischem Schlüssel werden sofort als **Dubletten** angesehen und **extrahiert**. Extrahiert bedeutet, die IDs der Kundedatensätze werden in eine neue Datei (Output-Datei) geschrieben. Dadurch entfallen alle folgenden Schritte gängiger unscharfer Such-Algorithmen, woraus der enorme Performance-Gewinn resultiert.

Der Nachteil gegenüber herkömmlichen Verfahren besteht darin, dass aufgrund von Tippfehlern entstandene Dubletten nur bedingt erkannt werden. Im Anwendungsfall erzielte das Verfahren im Vergleich mit der zuvor eingesetzten Software, die mit unscharfen Suchalgorithmen arbeitet, dennoch qualitativ bessere Ergebnisse. Bei erheblich größeren Datenmengen müssten vermutlich andere Schlüssel gewählt werden bzw. die Anwendbarkeit des Vorgehens an sich erst noch geprüft werden.

Vorgehen im Einzelnen

1.) Normalisierung

Nachfolgend werden die einzelnen Schritte der Normalisierung näher erläutert. Die Tabellen zeigen jeweils in der oberen Zeile den Zustand *vor* und in der unteren Zeile den Zustand *nach* dem jeweiligen Normalisierungs-Schritt. Die hellrote Hinterlegung zeigt an, welche Felder im jeweiligen Schritt bearbeitet werden. Keine Hinterlegung bedeutet, dass alle Felder behandelt wurden.

1.1) Deutsche Umlaute ersetzen. Da später ohnehin alle Buchstaben in Großbuchstaben umgewandelt werden, werden auch klein geschriebene Umlaute in ihre großgeschriebenen Äquivalente umgewandelt.

21310	Dr. René-Karl	Schönweiß	Kantstr.	22a	D-44789	Bochum	D	r.k.s@gmail.com
21310	Dr. René-Karl	SchOEnweiß	Kantstr.	22a	D-44789	Bochum	D	r.k.s@gmail.com

1.2) Konvertieren in 7 Bit ASCII Zeichensatz. Nicht-ASCII-Zeichen werden dabei in möglichst ähnliche Zeichen bzw. Zeichenkombinationen umgewandelt.

21310	Dr. René-Karl	SchOEnweiß	Kantstraße	22a	D-44789	Bochum	D	r.k.s@gmail.com
21310	Dr. Ren'e-Karl	SchOEnweiss	Kantstrasse	22a	D-44789	Bochum	D	r.k.s@gmail.com

1.3) Umwandeln in Großbuchstaben

21310	Dr. Ren'e-Karl	SchOEnweiss	Kantstrasse	22a	D-44789	Bochum	D	r.k.s@gmail.com
21310	DR. REN'E-KARL	SCHOENWEISS	KANTSTRASSE	22A	D-44789	BOCHUM	D	R.K.S@GMAIL.COM

1.4) Entfernen von Titlen wie Dr. oder Prof. aus den Feldern Vor- und Nachname

21310	DR. REN'E-KARL	SCHOENWEISS	KANTSTRASSE	22A	D-44789	BOCHUM	D	R.K.S@GMAIL.COM
21310	REN'E-KARL	SCHOENWEISS	KANTSTRASSE	22A	D-44789	BOCHUM	D	R.K.S@GMAIL.COM

1.5) Ersetzen von STRASSE durch STR im Feld Straße

21310	REN'E-KARL	SCHOENWEISS	KANTSTRASSE	22A	D-44789	BOCHUM	D	R.K.S@GMAIL.COM
21310	REN'E-KARL	SCHOENWEISS	KANTSTR	22A	D-44789	BOCHUM	D	R.K.S@GMAIL.COM

1.6) Entfernen von Buchstaben im Feld Postleitzahl (außer bei Ländern, bei denen Buchstaben erlaubter Bestandteil der Postleitzahl sind (z.B. Niederlande und Groß Britannien).

21310	REN'E-KARL	SCHOENWEISS	KANTSTR	22A	D-44789	BOCHUM	D	R.K.S@GMAIL.COM
21310	REN'E-KARL	SCHOENWEISS	KANTSTR	22A	-44789	BOCHUM	D	R.K.S@GMAIL.COM

1.7) Löschen aller Sonderzeichen und Leerzeichen (außer im Feld E-Mail-Adresse)

21310	REN'E-KARL	SCHOENWEISS	KANTSTR	22A	-44789	BOCHUM	D	R.K.S@GMAIL.COM
21310	RENEKARL	SCHOENWEISS	KANTSTR	22A	44789	BOCHUM	D	R.K.S@GMAIL.COM

2.) Schlüssel erzeugen

Im zweiten Schritt werden aus den normalisierten Adress-Daten zwei verschiedene Schlüssel gebildet. Die Schlüssel bestehen aus Zeichenketten, die aus Aneinanderreihung relevanter Datenfelder oder von Teilen relevanter Datenfelder gebildet werden. Sie wurden so gewählt, dass zwei Datensätze, die sich den selben Schlüssel teilen, mit nahezu 100%iger Wahrscheinlichkeit dieselbe Person repräsentieren.

Die Zusammensetzung der Schlüssel, also die Auswahl der verwendeten Felder sowie der Längen der Teilstrings, die in den jeweiligen Schlüssel einfliessen, sind entscheidend für die Qualität des Ergebnisses.

Die im Folgenden beschriebenen beiden Schlüssel sind das Ergebnis von Vorüberlegungen einerseits („Was macht eine Person aus bzw. was unterscheidet sie am deutlichsten von einer anderen Person?“) und anderseits das Ergebnis eines „Trial and Error“ Prozesses mit den vorliegenden Daten. Die Beurteilung des jeweiligen Ergebnisses erfolgte im Vergleich mit dem Ergebnis, welches von der Software *Fuzzy Dupes* von Kroll Software erzielt wurde.

2.1) Der erst Schlüssel besteht aus der vollständigen E-Mail-Adresse und dem erstem Buchstaben des Vornamens. Der erste Buchstabe des Vornamens wurde mit aufgenommen, da sich in einigen Fällen zwei Personen eine E-Mail-Adresse teilen (z.B. Ehepaare), es aber nur sehr selten vorkommt, dass die Vornamen dieser beiden Personen mit dem selben Buchstaben beginnen.

Dieser Schlüssel ist relativ „sicher“, d.h. die Wahrscheinlichkeit, dass es sich bei zwei Datensätzen, bei denen dieser Schlüssel identisch ist, um Dubletten handelt, ist sehr hoch. Im Fall einer geänderten oder nicht bekannten E-Mail-Adresse ist dieser Schlüssel allerdings nutzlos, weshalb ein weiterer Schlüssel benötigt wird.

Der erste Schlüssel für die Zeile

21310	RENEKARL	SCHOENWEISS	KANTSTR	22A	44789	BOCHUM	D	R.K.S@GMAIL.COM

lautet **R.K.S@GMAIL.COMR**

2.2) Der zweite Schlüssel besteht aus den jeweils ersten vier Buchstaben des Vor- und Nachnamens sowie der vollständigen Postleitzahl.

Dieser zweite Schlüssel ist weniger sicher, funktioniert aber auch bei geänderter oder fehlender E-Mail-Adresse, sofern Vor- und Nachname sowie die PLZ weitgehend identisch sind. Das Kürzen der Namen auf die ersten vier Zeichen bewirkt, dass Dubletten mit gewissen Unterschieden in der Schreibweise trotzdem erkannt werden (z.B. bei Tippfehlern oder Buchstabendrehern *nach* dem 4. Zeichen oder Doppelnamen). Die Gefahr, dass zwei unterschiedliche Personen sich diesen Schlüssel teilen, ist jedoch höher als beim ersten Schlüssel. So hätten z.B. Maria Schreiber und Marianne Schröder aus Bochum beide den Schlüssel MARISCHR44789.

Der zweite Schlüssel für die Zeile

21310	RENEKARL	SCHOENWEISS	KANTSTR	22A	44789	BOCHUM	D	R.K.S@GMAIL.COM

lautet **RENESCHO44789**

3.) Sortieren

Nachem die Schlüssel erzeugt und in zwei neue Felder an des Ende jeder Zeile in die Datei geschrieben wurden, wird die Datei in zwei Durchläufen nach dem jeweiligen Schlüssel sortiert. So muss beim Prüfen auf mehrfach vorkommende Schlüssel im nächsten Schritt nur die jeweils aktuelle Zeile mit der vorigen Zeile verglichen werden.

4.) Dubletten extrahieren

Im letzten Schritt werden die sortierten Dateien in zwei Duchläufen (ein Durchlauf pro Schlüssel) Zeile für Zeile durchlaufen und auf mehrfach vorkommende Schlüssel geprüft. Datensätze mit identischen Schlüsseln werden als Duplikate betrachtet. Die Kunden-IDs solcher Datensätze werden schließlich als Dubletten-Gruppe in die jeweilige Output-Datei geschrieben.

Die beiden resultierenden Output-Dateien haben folgenden Aufbau:

```
SCHLÜSSEL           ID1, ID2, ... Idn
```

Eine Zeile enthält also jeweils den Schlüssel der Dubletten-Gruppe sowie, durch Tabstopp getrennt, die IDs der als Dubletten erkannten Datensätze, untereinander jeweils durch Komma getrennt.

Sobald ein Schlüssel zum zweiten mal gefunden wird, wird in der Output-Datei eine neue Zeile mit dem Schlüssel und der ID der *vorherigen* Zeile begonnen, da die vorherige Zeile das erste Mitglied in einer Dubletten-Gruppe repräsentiert. Bei erneutem Findes des selben Schlüssels in der folgenden Zeile wird der Output-Zeile wiederum die ID der vorherigen Zeile hinzugefügt. Dies wird so lange wiederholt, bis ein neuer Schlüssel gefunden wird. Dann wird wiederum die ID der vorherigen Zeile in die begonnene Zeile der Output-Datei geschrieben und die Zeile beendet.

Beispielhaft wird im Folgenden die Verarbeitung von vier Zeilen gezeigt, von denen drei die selbe Person repräsentieren. Im Beispiel befinden wir uns im zweiten Durchlauf (Verarbeitung der nach **SCHLÜSSEL 2** sortierten Datei).

1. Zeile

ID	VORNAME	NACHNAME	PLZ	EMAIL	SCHLÜSSEL 1	SCHLÜSSEL 2
21310	RENEKARL	SCHOENWEISS	44789	R.K.S@GMAIL.COM	R.K.S@GMAIL.COMR	RENESCHO44789

Der Schlüssel RENESCHO44789 tauch zum ersten mal auf. Der Zähler für diesen Schlüssel ist „1“.

2. Zeile

144310	RENE	SCHOENWEISS	44789	R.K.S@WEB.DE	R.K.S@WEB.DER	RENESCHO44789

Der selbe Schlüssel taucht erneut auf (Dublette gefunden!) und der Zähler wird auf „2“ gesetzt. Da in der vorherigen Zeile *keine* Dublette gefunden wurde, wird in der Output-Datei eine neue Zeile begonnen. In diese Zeile wird der Schlüssel, sowie, durch Tabstopp `\t` getrennt, die ID der voherigen Zeile geschrieben, gefolgt von einem Komma:

```
RENESCHO44789  21310,
```

3. Zeile

226108	RENE	SCHOENWIESS	44789	R.K.S@WEB.DE	R.K.S@WEB.DER	RENESCHO44789

Der selbe Schlüssel taucht erneut auf. Da bereits in der vorhergehenden Zeile eine Dublette gefunden wurde, wird *keine* neue Zeile begonnen, sondern der angefangenen Zeile die ID der vorherigen Zeile hinzugefügt, gefolgt von einem Komma:

```
RENESCHO44789  21310,144310,
```

4. Zeile

```
226100   PETER        WALTHER          10243   P.WALTHER@GMX.DE  P.WALTHER@GMX.DEP  PETEWALT10243
```

Ein neuer Schlüssel! Das bedeutet, dass die vorangegangene Zeile das letzte Mitglied der gebildeten Dubletten-Gruppe darstellt. Also wird der Zeile in der Output-Datei abermals die ID der vorherigen Zeile hinzugefügt und die Zeile dann durch ein Linebreak `\n` beendet.

```
RENESCHO44789  21310,144310,226108
```

Umsetzung

Die Umsetzung erfolgte mit den Unix Kommandozeilen-Tools awk bzw. gawk, sed, sort und iconv. Die einzelnen Kommandos/Programmaufrufe wurden in einem Shell-Skript (*dedupe.sh*) zusammengefasst.

Die Datei- und Verzeichnisstruktur des Projektes sieht wie folgt aus:

```
dedupe.sh
/in
  Customers.txt
/out
  DupesByEmail.txt
  DupesByName.txt
/app
  /bin
    addKeys.awk
    extractDupes.awk
    normalize.gawk
  /trans
    1_Customers_ReplacedMutatedVovels
    2_Customers_ASCII
    3_Customers_Normalized
    4_Customers_Keys
    5a_Customers_SortedByKeyEmail
    5b_Customers_SortedByKeyName
```

Die Datei **`dedupe.sh`** beinhaltet das übergreifende Shell-Skript, das die einzelnen Kommandos ausführt. Im Verzeichnis **`/in`** liegt initial die zu verabeitende Adress-Datei. Im Verzeichnis **`/out`** landen die beiden finalen Output-Dateien mit den Schlüssel-Dubletten-Gruppen. Im Verzeichnis **`/app/bin`** liegen die awk- bzw. gawk-Skripte, die für die Normalisierung, das Hinzufügen der Schlüssel und das eigentliche Extrahieren der Dubletten zuständig sind. Im Verzeichnis **`/app/trans`** werden zur Laufzeit die aus den einzelnen Schritten der Normalisierung, dem Hinzufügen der Schlüssel und der Sortierung resultierenden Dateien abgelegt.

Nachfolgend der Programmcode der vier Scripte, aus denen das Tool besteht:

Listing 1: /dedupe.sh

```
1.  #!/bin/bash
2.
3.  # Find dupes in input file and write IDs into output files
4.  # Expected input file fields:
5.  # iId        = $1;
6.  # sFname     = $2;
7.  # sLname     = $3;
8.  # sStreet    = $4;
9.  # sZIP       = $5;
10. # sCity      = $6;
11. # sCountry   = $7;
12. # sEmail     = $8;
13.
14. # Start timer
15. T0=$(date +%s)
16. echo ""
17.
18. ##############################################################################
19. # Normalize
20. ##############################################################################
21. echo 'Normalize input file ...'
22.
23. # Replace mutated vovels
24. echo "  Replace mutated vovels"
25. sed 's/[Üü]/UE/g;s/[Öö]/OE/g;s/[Ää]/AE/g;' ./in/Customers.txt >
    ./app/trans/1_Customers_ReplacedMutatedVovels
26.
27. # Convert to ASCII
28. echo "  Convert to ASCII"
29. iconv -c -f UTF-8 -t US-ASCII//TRANSLIT ./app/trans/1_Customers_ReplacedMutatedVovels
    > ./app/trans/2_Customers_ASCII
30.
31. # Normalize
32. echo "  Normalize"
33. gawk -f ./app/bin/normalize.gawk ./app/trans/2_Customers_ASCII >
    ./app/trans/3_Customers_Normalized
34.
35. # Timer
36. T1=$(date +%s)
37. TD=$(($T1 - $T0))
38. echo "  > Done in $TD seconds"
39. echo ""
40.
41. ##############################################################################
42. # Add keys and sort files by key
43. ##############################################################################
44. echo 'Add blocking keys, sorting ...'
```

```
45.
46. # Add keys
47. echo "  Add blocking keys"
48. awk -f ./app/bin/addKeys.awk ./app/trans/3_Customers_Normalized >
    ./app/trans/4_Customers_Keys
49.
50. # Sort by email key
51. echo "  Sort by email key"
52. sort -t$'\t' -k9 ./app/trans/4_Customers_Keys >
    ./app/trans/5a_Customers_SortedByKeyEmail
53.
54. # Sort by name key
55. echo "  Sort by name key"
56. sort -t$'\t' -k10 ./app/trans/4_Customers_Keys >
    ./app/trans/5b_Customers_SortedByKeyName
57.
58. # Timer
59. T2=$(date +%s)
60. TD=$(($T2 - $T1))
61. echo "  > Done in $TD seconds"
62. echo ""
63.
64. ##############################################################################
65. # Extract dupes
66. ##############################################################################
67. echo 'Extract dupes ...'
68.
69. # Extract dupes by email key
70. echo "  Extract dupes by email key"
71. awk -f ./app/bin/extractDupes.awk -v FK=9 ./app/trans/5a_Customers_SortedByKeyEmail >
    ./out/DupesByEmail.txt
72.
73. # Extract dupes by name key
74. echo "  Extract dupes by name key"
75. awk -f ./app/bin/extractDupes.awk -v FK=10 ./app/trans/5b_Customers_SortedByKeyName >
    ./out/DupesByName.txt
76.
77. # Timer
78. T3=$(date +%s)
79. TD=$(($T3 - $T2))
80. echo "  > Done in $TD seconds"
81. echo ""
82.
83. # End timer
84. TD=$(($T3 - $T0))
85. echo "-----------------------------------------------------------------------"
86. echo "Finished! Total runtime: $TD seconds"
87. echo "-----------------------------------------------------------------------
```

Listing 2: normalize.gawk

```
1.  #!/bin/gawk -f
2.
3.  # Used for duplicate detection in address data
4.  # Method: sorted neighborhood
5.  # Normalizes address fields
6.  #
7.  # Expected fields:
8.  # iId           = $1;
9.  # sFname        = $2;
10. # sLname        = $3;
11. # sStreetAndNr  = $4;
12. # sZIP          = $5;
13. # sCity         = $6;
14. # sCountry      = $7;
15. # sEmail        = $8;
16.
17. BEGIN { FS = "\t"; OFS = "\t" }
18.
19. {
20.
21.     # Walk through fields
22.     for(i = 1; i <= NF; i++) {
23.
24.         # ToUpper
25.         $i = toupper($i);
26.
27.         # Remove titles from names
28.         if(i == 2 || i == 3 ) {
29.             gsub(/PROF\.|DR\.|MAG\./, "", $i);
30.         }
31.
32.         # Normalize street
33.         if(i == 4) {
34.             sub(/STRASSE/, "STR", $i);
35.         }
36.
37.         # Remove chars from zipcodes
38.         # Chars in countries with alphanumeric zipcodes must be kept
39.         # http://de.wikipedia.org/wiki/Liste_der_Postleitsysteme
40.         if(i == 5) {
41.             if(!match($7, /AR|GB|NL|CA/)) {
42.                 gsub(/[a-zA-Z]/, "", $i);
43.             }
44.         }
45.
46.         # Remove special chars and blanks inside fields
47.         # Special chars in email must be kept
```

```
48.         if(i != 8) {
49.             gsub(/[^a-zA-Z0-9]/, "", $i);
50.         } else {
51.             gsub(/[<> ]/, "", $i); # Remove brackets and blanks from email addresses
52.         }
53.
54.     }
55.
56.     print $0;
57.
58. }
59.
```

Listing 3: addKeys.awk

```
1.  #!/bin/awk -f
2.
3.  # Used for duplicate detection in address data
4.  # Method: sorted neighborhood
5.  # Adds blocking keys
6.  #
7.  # Expected fields:
8.  # iId           = $1;
9.  # sFname        = $2;
10. # sLname        = $3;
11. # sStreetAndNr  = $4;
12. # sZIP          = $5;
13. # sCity         = $6;
14. # sCountry      = $7;
15. # sEmail        = $8;
16.
17. BEGIN { FS = "\t"; OFS = "\t" }
18.
19. {
20.
21.     # Add blocking key - EmailFnam
22.     $9 = $8 substr($2,1,1);
23.
24.     # Add blocking key - FnamLnamZIP
25.     $10 = substr($2,1,4) substr($3,1,4) $5;
26.
27.     print $0;
28.
29. }
```

Listing 4: extractDupes.awk

```
1.  #!/bin/awk -f
2.
3.  # Extracts IDs of duplicate customers from file_in
4.  # file_in must be sorted by blocking key
5.  # Data sets with Identical blocking keys are considered duplicates
6.  # Field containing the customer ID must be the first field in file_in
7.  # Field containing the blocking key must be passed by parameter "FK"
8.  #
9.  # Usage examples:
10. # extractDupes.awk -v FK=12 Customers_SortedByKeyEmail > DupesByEmail
11. # extractDupes.awk -v FK=13 6ustomers_SortedByKeyName > DupesByName
12.
13. BEGIN { FS = "\t"; }
14.
15. {
16.
17.     if(!FK) {
18.         exit "Please provide Field No. containing blocking key.";
19.     }
20.
21.     sId  = $1;     # Customer ID
22.     sKey = $FK;    # Blocking key
23.
24.     # Count occurs
25.     a[sKey]++;
26.
27.     # Dupe found?
28.     if(a[sKey] > 1 && index(sKey, "NULL") == 0) {
29.
30.         if(!bDupeInPreviousLine) {
31.
32.             # Start output line
33.             printf("%s\t", sKey);
34.
35.         }
36.
37.         printf("%s,", sPrevId);
38.
39.         bDupeInPreviousLine = 1;
40.
41.     } else {
42.
43.         if(bDupeInPreviousLine) {
44.
45.             # Finish output line
46.             printf("%s\n", sPrevId);
47.
```

```
48.         }
49.
50.         bDupeInPreviousLine = 0;
51.
52.     }
53.
54.     sPrevId = sId; # Remember this line's Id for matching in next line
55.
56. }
```

Performance

Da bei diesem Vorgehen durch die vorangegangene Sortierung und die Annahme, dass zwei Zeilen mit identischem Schlüssel ein und dieselbe Person repräsentieren immer nur ein Feld der aktuellen Zeile mit einem Feld der vorherigen Zeile verglichen werden muss, geht das ganze sehr schnell. Für 600.000 Datensätze benötigt das Skript wenige Sekunden. Im Vergleich dazu benötigte die als Benchmark dienende Software über zwei Stunden für die Verarbeitung derselben Datei.

Literatur

https://de.wikipedia.org/wiki/Dublette_(Datenbank)

https://de.wikipedia.org/wiki/Duplikaterkennung

https://de.wikipedia.org/wiki/Sorted_Neighborhood

http://www.profilingcompany.de:8080/html/PDF/Management_Summary_Dublettensuche_in_Adressdaten.pdf

https://mediatum.ub.tum.de/doc/644174/644174.pdf

http://epb.bibl.fh-koeln.de/frontdoor/index/index/docId/270

http://www.kroll-software.de/de/products/fuzzydupes/