String Matching

Komplexe Informationssysteme

Fabian Panse

panse@informatik.uni-hamburg.de

Universität Hamburg

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Einordnung

- Identifzierung von Stringwerten die das gleiche Realwelt-Objekt referenzieren
  - Namen von Personen (z.B. 'Christina' vs. 'Kristina')
  - Namen von Einrichtungen (z.B. 'HPI' vs. Hasso-Plattner Institut')
  - Adressen
    ('1210 W. Dayton St Madison WI' vs. '1210 West Dayton Madison WI 53706')
  - Telefonnummern ('040-12345' vs. '(040)12354')
- Spielt eine tragende Rolle in vielen Integrationsaufgaben
  - Schema Matching, Duplicate Detection, etc.
- Ausblick
  - Problemdefinition
  - Maße um die Ähnlichkeit zweier Stringwerte numerisch zu quantifizieren
  - Techniken zur Performanzoptimierung

Problemdefinition

- Gegeben sind zwei Mengen A and B an Stringwerten
  - Finde alle Paare (x, y) mit $x \in A$ und $y \in B$ die dasselbe Realwelt-Objekt referenzieren
  - Ergebnispaare werden als *Match* bezeichnet
  - Beispiel (aus [DHI12])

Menge A	**Menge** B	**Matches**
x_1 =Dave Smith	y_1 =David D. Smith	(x_1, y_1)
x_2 =Joe Wilson	y_2 =Daniel D. Smith	(x_3, y_2)
x_3 =Dan Smith		

- Herausforderungen: Genauigkeit und Skalierbarkeit

Herausforderungen: Genauigkeit

- Google meldet 593* verschiedene Schreibweisen von 'Britney Spears' in Suchanfragen
  - 'Britney Spears' (488941 Suchen)
  - 'Brittany Spears' (40134 Suchen)
  - 'Brittney Spears' (36315 Suchen)
  - 'Britany Spears' (24342 Suchen)
  - 'Britny Spears' (7331 Suchen)
  - ...
- wieviele verschiedene Schreibweisen für komplexere Namen?
  - 'Giannis Antetokounmpo' (griechischer Basketballspieler)
  - 'Dharmavarapu Subramanyam' (indischer Schauspieler)
  - 'Janice Keihanaikukauakahihuliheekahaunaele' (hawaiianische Frau)
  - 'Venkatanarasimharajuvaripeta railway station' (Bahnstation in Indien)

*Quelle: http://www.netpaths.net/blog/britney-spears-spelling-variations/

Herausforderungen: Genauigkeit

Arten von Fehlerquellen:

- Tippfehler (z.B. 'Spears' vs. 'Speasr')
- Unterschiedliche Konventionen
  (z.B. 'Britney Jean Spears' vs. 'Spears, Britney J.')
- fehlerhafte Transformation vom Gesprochenen ins Geschriebene
  (z.B. 'Spears' vs. 'Speers')
- fehlerhafte Transformation vom Handschriftlichen ins Maschinelle (z.B. 'Spears' vs. 'Speans')
- Spitznamen, Künstlernamen oder Synonyme
  (z.B. 'Allan Stewart Konigsberg' vs. 'Heywood Allen' vs. 'Woody Allen')
- Abkürzungen (z.B. 'AStA' vs. 'Allgemeiner Studierendenausschuss')
- Unterschiedliche Maßeinheiten (z.B. '62 sec' vs. '1:02 h')

Herausforderungen: Genauigkeit

- Lösungsansatz:
  - Verwendung von einem Ähnlichkeitsmaß $s(x, y) \in [0, 1]$ um die Ähnlichkeit zweier Stringwerte zu quantifizieren
  - Je größer $s(x, y)$, desto größer ist die Wahrscheinlichkeit das (x, y) ein Match ist
  - Klassifiziere (x, y) als ein Match, wenn $s(x, y) \geq \theta$

- Oft werden Distanz- oder Kostenmaße verwendet
  - Je geringer die Distanz (Kosten), desto größer die Ähnlichkeit
  - Umrechnung für normiertes Distanzmaß d: $s(x, y) = 1 - d(x, y)$

Herausforderungen: Skalierbarkeit

- Vergleichen jedes Stringwertes $x \in A$ mit jedem Stringwert $y \in B$ ist bei großen Mengen zu ineffizient (quadratische Komplexität)
- Verwendung einer Methode *FindCands* welche eine Menge an möglichen Matches $C \subseteq B$ für jedes $x \in A$ effizient bestimmt

  Für jeden Stringwert $x \in A$
  - verwende *FindCands* um eine Menge $C \subseteq B$ zu bestimmen
  - für jeden Stringwert $y \in C$
    - wenn $s(x, y) \geq \theta$, dann klassifiziere (x, y) als Match

Klassifizierung von Ähnlichkeitsmaßen

- Mengenbasiert / Tokenbasiert
  - Overlap Measure, Jaccard, Adamic, Kosinus mit TF/IDF
- Sequenzbasiert
  - Levenshtein, Affine Gap, Jaro
- Phonetisch
  - Soundex, Kölner Phonetik
- Semantisch
  - Thesauri, Ontologien
- Hybrid
  - Extended/Generalized Jaccard, Kosinus mit Soft-TF/IDF, Monge-Elkan

Mengenbasierte Ähnlichkeitsmaße

- betrachten Stringwerte als (Multi-)Mengen von Tokens
- verwenden Mengeneigenschaften um Ähnlichkeit zu bestimmen
- verschiede Möglichkeiten um einen Stringwert in Tokens zu zerlegen
  - Wörter des Stringwertes
    - Trennung durch Leerzeichen oder Trennsymbole
    - Entfernung von Stopp-Wörtern wie 'der', 'und', oder 'von'
    - Bsp.: 'david smith' zu {'david','smith'}
  - q-grams (Substrings der Länge q)
    - Spezielles Zeichen # an Anfang und Ende des Stringwertes damit jeder Buchstabe gleich häufig vorkommt
    - Bsp.(q=3): 'david smith' zu {'##d','#da','dav','avi','vid','id#','d##'}

Overlap Measure & Common Neighbor Score

- Input: Zwei Stringwerte x und y, Tokenfunktion tok
- Genierung der zwei Tokenmengen $X = tok(x)$ und $Y = tok(Y)$
- Overlap Measure: $O(X, Y) = |X \cap Y|$ (= Anzahl gemeinsamer Tokens)
- Nachteil: nicht normalisiert (Werte daher schwer vergleichbar)

$\Rightarrow$ Common Neighbor Score [BG07]: $CNS(X, Y) = O(X, Y)/k$

- Normalisierung des Overlap Measures mit geeignet großer Konstante k

Overlap Measure & Common Neighbor Score (Beispiel)

- x='dave', y='dav'
- Tokenbildung durch 2-grams

$$X = \{\text{'\#d','da','av','ve','e\#'}\}$$

$$Y = \{\text{'\#d','da','av','v\#'}\}$$

$\Rightarrow X \cap Y = \{\text{'\#d','da','av'}\}$

$\Rightarrow O(X, Y) = 3$

$\Rightarrow CNS(X, Y) = 0.3$ mit $k = 10$

Jaccard Coefficient

- Normalisierung durch feste Konstante unrepräsentativ wenn Größe der ingesamt betrachteten Tokenmengen stark variiert

$\Rightarrow$ Normalisierung durch Anzahl aller betrachteter Tokens

$$Jacc(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

(= Anteil gemeinsamer Tokens)

Jaccard Coefficient (Beispiel)

- x='dave', y='dav'
- Tokenbildung durch 2-grams

$$X = \{\text{'\#d','da','av','ve','e\#'}\}$$

$$Y = \{\text{'\#d','da','av','v\#'}\}$$

$\Rightarrow$ $X \cap Y = \{\text{'\#d','da','av'}\}$

$X \cup Y = \{\text{'\#d','da','av','ve','e\#','v\#'}\}$

$\Rightarrow$ $Jacc(X, Y) = 3/6 = 0.5$

Adamic/Adar Ähnlichkeit

- Einige Tokens sind bessere Unterscheidungsmerkmale als andere

$\Rightarrow$ zwei Stringwerte sind umso ähnlicher je mehr seltene Token sie gemeinsam haben

$\Rightarrow$ Berücksichtigung der *uniqueness* $u(t)$ eines Tokens t

$$Adar(X, Y) = \frac{\sum_{t \in |X \cap Y|} u(t)}{\sum_{t \in |X \cup Y|} u(t)}$$

Adamic/Adar Ähnlichkeit (Beispiel)

- x='Apple Corporation, CA', y='IBM Corporation, CA', z='Apple Corp'
- Tokenbildung durch Trennzeichen

$$X = \{\text{'Apple','Corporation','CA'}\}$$
$$Y = \{\text{'IBM','Corporation','CA'}\}$$
$$Z = \{\text{'Apple','Corp'}\}$$

- Eigentlich x wahrscheinlicher ein Match mit z als mit y, aber $Jacc(X, Y) > Jacc(X, Z)$
- aber wenn $u(\text{'Apple'}) \gg u(\text{'Corporation'})$, dann gilt $Adar(X, Z) > Adar(X, Y)$

TF/IDF

- Verbreitetes Maß für *uniqueness* (Information Retrieval)
- term frequency $tf(t, d)$:
  - Häufigkeit des Terms t in Dokument d
- document frequency $df(t)$:
  - Anteil aller Dokumente (Menge N) die Term t beinhalten (Menge N_t), d.h. $df(t) = N_t/N$
- inverse document frequency $idf(t)$: $idf(t) = N/N_t$
- In String Matching: Term $\simeq$ Token und Dokument $\simeq$ Multimenge aller Tokens eines Stringwerts
- verschiedene Varianten um tf und idf zu TF/IDF vereinen
- Gängige Varianten [DHI12]:
  - $TF/IDF(t, d) = tf(t, d) \times idf(t)$
  - $TF/IDF(t, d) = \log(tf(t, d) + 1) \times \log(idf(t))$

TF/IDF: Beispiel

$X = \{\text{'Apple','Corporation','CA'}\}$
$Y = \{\text{'IBM','Corporation','CA'}\}$
$Z = \{\text{'Apple','Corp'}\}$

$N = 100$ $N_{\text{'Apple'}} = 5$
$N_{\text{'CA'}} = 25$ $N_{\text{'Corporation'}} = 40$
$N_{\text{'IBM'}} = 1$ $N_{\text{'Corp'}} = 20$

tf	X	Y	Z
'Apple'	1	0	1
'CA'	1	1	0
'Corp'	0	0	1
'Corporation'	1	1	0
'IBM'	0	1	0

idf	
'Apple'	$100/5 = 20$
'CA'	$100/25 = 4$
'Corp'	$100/20 = 5$
'Corporation'	$100/40 = 2.5$
'IBM'	$100/1 = 100$

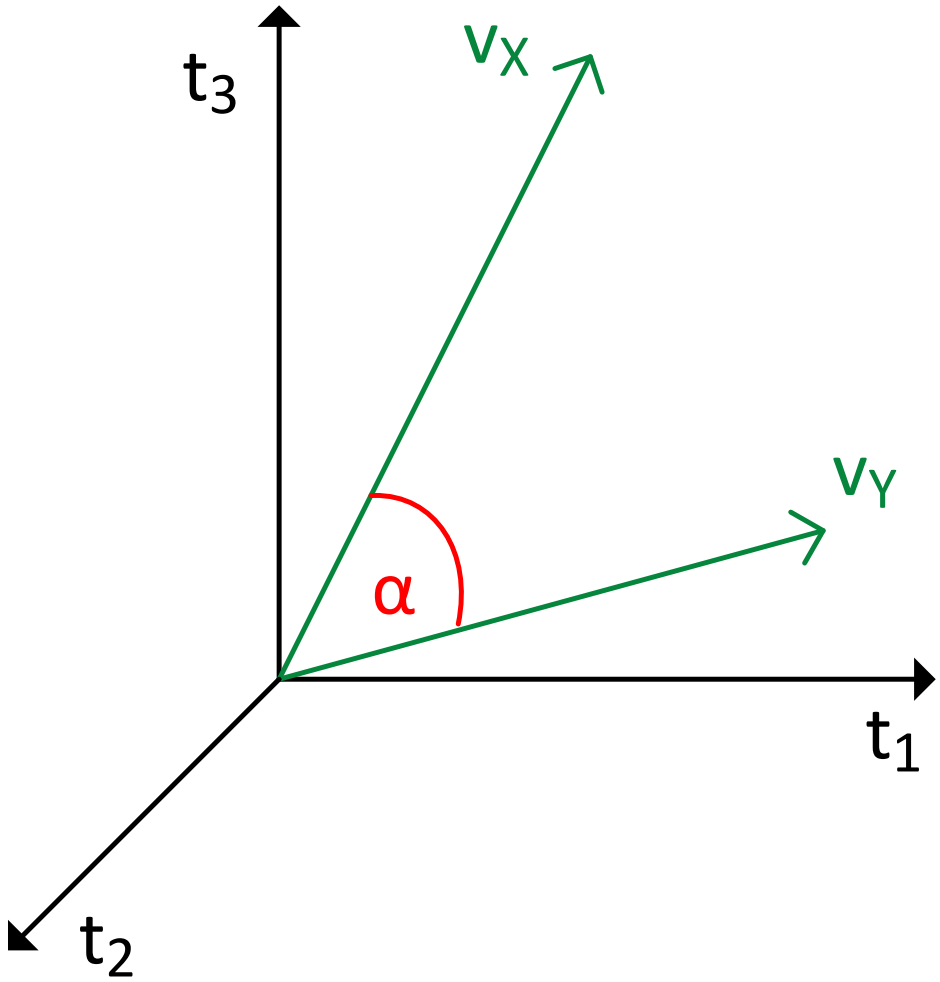
Kosinus Ähnlichkeit

- Für jede Tokenmenge (Dokument) X wird ein mehrdimensionaler Feature-Vektor v_X erstellt
- Die Ähnlichkeit zwischen zwei Tokenmengen X und Y entspricht dem Kosinus des Winkels zwischen ihren Vektoren v_X und v_Y

$$
\begin{aligned}
CosSim(X, Y) &= \frac{v_X \cdot v_Y}{||v_X|| \times ||v_Y||} \\
&= \frac{\sum_{i=1}^{k} v_X(i) \times v_Y(i)}{\sqrt{\sum_{i=1}^{k} v_X(i)^2} \times \sqrt{\sum_{i=1}^{k} v_Y(i)^2}}
\end{aligned}
$$

- Tokenmengen ähnlich $\Rightarrow$ Winkel $\rightarrow 0 \Rightarrow$ Kosinus $\rightarrow 1$

Kosinus Ähnlichkeit



Kosinus Ähnlichkeit

- Eine Dimension per vorkommendem Token (Term)
- Jeder Vektor v_X hat einen Featurewert $v_X(t)$ für jedes Token t
- Der Featurewert $v_X(t)$ wird gewöhnlich mit $TF/IDF(t, X)$ berechnet
- Je mehr sich die Vektoren in den einzelnen Featurewerten ähneln, desto geringer wird der Winkel zwischen ihnen

Kosinus Ähnlichkeit (Beispiel)

$X = \{\text{'Apple','Corporation','CA'}\}$
$Y = \{\text{'IBM','Corporation','CA'}\}$
$Z = \{\text{'Apple','Corp'}\}$

$N = 100$ $N_{\text{'Apple'}} = 5$
$N_{\text{'CA'}} = 25$ $N_{\text{'Corporation'}} = 40$
$N_{\text{'IBM'}} = 1$ $N_{\text{'Corp'}} = 20$

- Gegeben $TF/IDF(t, X) = tf(t, X) \times idf(t)$

$\Rightarrow v_X = \langle 20, 4, 0, 2.5, 0 \rangle$
$v_Y = \langle 0, 4, 0, 2.5, 100 \rangle$
$v_Z = \langle 20, 0, 5, 0, 0 \rangle$

$\Rightarrow CosSim(X, Y) = (22.25)/(\sqrt{422.25} \times \sqrt{10022.25}) = 0.011$
$\Rightarrow CosSim(X, Z) = (400)/(\sqrt{422.25} \times \sqrt{425}) = 0.944$

Sequenzbasierte Ähnlichkeitsmaße

- betrachten Stringwerte als Sequenzen von Schriftzeichen
- Edit-Distanz berechnet Ähnlichkeit basierend auf den minimalen Kosten die notwendig sind um die erste Sequenz in die zweite zu transformieren
- Mehrere Varianten mit unterschiedlichenn Mengen an erlaubten Operationen und deren Kosten
- Grundlegende Variante: Levenshtein Distanz

Levenshtein Distanz/Ähnlichkeit

- Erlaubte Operationen: Einfügen, Löschen und Ersetzen
- Jede Operation hat die Kosten 1
- Normalisierung durch die Länge der größeren Zeichenkette

$$LevSim(x, y) = 1 - \frac{LevDst(x, y)}{max(|x|, |y|)}$$

Levenshtein Distanz/Ähnlichkeit (Beispiel)

- x='hase', y='rasen'
- Schritt 1: Ersetze 'h' durch 'r'
- Schritt 2: Füge ein 'n' ans Ende der Sequenz

$\Rightarrow$ Gesamtkosten sind 2

- Da gilt: $|x| = 4$ und $|y| = 5$

$\Rightarrow$ Levenshtein Ähnlichkeit ist $1 - 0.4 = 0.6$

Levenshtein Distanz/Ähnlichkeit

- Berechnung durch dynamische Programmierung

  - Edit-Distance-Matrix (Berechnung)

		H	A	S	E
	0	1	2	3	4
R	1	1	2		
A	2				
S	3				
E	4				
N	5				

$$D(i, 0) = i$$

$$D(0, j) = j$$

$$D(i, j) = min \{$$

$$D(i-1, j) + 1,$$

$$D(i, j-1) + 1,$$

$$D(i-1, j-1) + d(i, j)$$

$$\}$$

wobei $d(i, j) = 0$ bei Gleichheit, $d(i, j) = 1$ sonst

Levenshtein Distanz/Ähnlichkeit

- Ermittlung der zugehörigen Transformation

Edit-Distance-Matrix
(Ergebnis)

		H	A	S	E
	0	1	2	3	4
R	1	1	2	3	4
A	2	2	1	2	3
S	3	3	2	1	2
E	4	4	3	2	1
N	5	5	4	3	**2**

- Transkript durch Traceback rückwärts zum kleinst-möglichen Wert nach
  - links = DELETE
  - oben = INSERT
  - diagonal = MATCH oder REPLACE

Levenshtein Distanz/Ähnlichkeit

- Beispiel: 'dva' vs. 'dave'

		y_0	y_1	y_2	y_3	y_4
			d	**a**	**v**	**e**
x_0		0	1	2	3	4
x_1	**d**	1	0	1		
x_2	**v**	2				
x_3	**a**	3				

		y_0	y_1	y_2	y_3	y_4
			d	**a**	**v**	**e**
x_0		0	1	2	3	4
x_1	**d**	1	0	1	2	3
x_2	**v**	2	1	1	1	2
x_3	**a**	3	2	1	2	2

$x = \mathrm{d}\ -\ \mathrm{v}\ \mathrm{a}$
$\quad\quad\ |\ \ |\ \ |\ \ |$
$y = \mathrm{d}\ \mathrm{a}\ \mathrm{v}\ \mathrm{e}$

Substitute a with e
Insert a (after d)

Quelle: Doan, Halevy and Ives. Principles of data Integration, 2012 [DHI12]

Affine Gap Distanz

- Erweitert die Levenshtein Distanz um das Einfügen und Löschen von kompletten Substrings
- Modifizieren von ganzen Blöcken ist günstiger als alle Zeichen einzeln zu modifizieren
  (Ziel: Robustheit gegen fehlende Mittelnamen)
- Kosten w_g für das Öffnen einer Lücke
- Kosten w_s für jedes weitere Zeichen in einer Lücke
- Kosten einer Lücke der Länge l: $w(l) = w_g + (l - 1) \times w_s$
- Kann mit Hilfe dynamischer Programmierung berechnet werden

Affine Gap Distanz (Beispiel)

- $x =$'Hans J Wurst' und $y =$'Hans Jürgen Wurst'
- Kosten für Öffnen einer Lücke: $w_g = 1$
- Kosten für Weiterführen einer Lücke: $w_s = 0.1$
- Die Lücke umfasst den substring 'ürgen'
- Gesamtkosten der Lücke: $w(l) = 1 + 4 \times 0.1 = 1.4$
- Die Levenshtein Distanz zwischen x und y beträgt hingegen 5

Jaro Ähnlichkeit

- Konstruiert zum Vergleich kurzer Stringwerte wie Namen
- Berechnung in 3 Schritten:
  - Finde allen gemeinsamen Buchstaben x_i und y_j $(x_i = y_j)$ so dass $|i - j| \leq min(|x|, |y|)/2$
  - Vergleiche den $i$ten gemeinsamen Buchstaben von x mit dem $i$ten gemeinsamen Buchstaben von y. Sind sie verschieden, liegt eine Transposition vor.
  - Sei c die Anzahl der gemeinsamen Buchstaben und sei t die Anzahl der Transpositionen, die Jaro Ähnlichkeit zwischen x und y ist definiert als:

$$Jaro(x, y) = \frac{1}{3}\left(\frac{c}{|x|} + \frac{c}{|y|} + \frac{c - t/2}{c}\right)$$

Jaro Ähnlichkeit (Beispiel)

- $x =$'jon' und $y =$'ojhn'
- Die maximale Distanz zwischen zwei gemeinsamen Buchstaben beträgt $\lfloor min(3,4)/2 \rfloor = \lfloor 1.5 \rfloor = 1$
- Die Sequenz gemeinsamer Buchstaben in x ist 'jon'
- Die Sequenz gemeinsamer Buchstaben in y ist 'ojn'
- Anzahl an gemeinsamen Buchstaben ist: $c = 3$
- Anzahl an Transpositionen ist: $t = 2$

$$\Rightarrow \quad Jaro(x,y) = \frac{1}{3}\left(\frac{3}{3} + \frac{3}{4} + \frac{3-1}{3}\right) = 0.81$$

Phonetische Ähnlichkeit

- Ähnlichkeit anhand der Wortklänge anstatt der Syntax (sinnvoll wenn Daten telefonisch übermittelt werden)
- Verfahren sind zumeist sehr sprachabhängig
- Verwendung oft auf Personennamen beschränkt
  z.B. 'Meier' vs. 'Mayer', 'Schmidt' vs. 'Schmitt'
- Stringwerte werden in Codes transformiert
- Phonetische Ähnlichkeit ergibt sich aus der Ähnlichkeit der Codes. Im einfachsten Fall:
  - identische Codes $\Rightarrow$ Ähnlichkeit $= 1$
  - unterschiedliche Codes $\Rightarrow$ Ähnlichkeit $= 0$
- Häufig verwendetes Verfahren ist Soundex [Rus18, Rus22]

Soundex

1. Mit Ausnahme des ersten Buchstaben werden alle Vorkommnisse von 'h' und 'w' entfernt
2. Mit Ausnahme des ersten Buchstaben werden alle Nichtvokale durch folgende Ziffern ersetzt

b, f, p, v	$\longrightarrow$	1
c, g, j, k, q, s, x, z	$\longrightarrow$	2
d, t	$\longrightarrow$	3

l	$\longrightarrow$	4
m, n	$\longrightarrow$	5
r	$\longrightarrow$	6

3. Alle aufeinanderfolgenen Vorkommnisse der gleichen Ziffer werden durch ein einzelnes ersetzt (z.B. '1001' wird zu '101')
4. Mit Ausnahme des ersten Buchstaben werden alle Nicht-Ziffern entfernt
5. Der Code wird auf die Länge vier beschränkt (notfalls Auffüllen mit der Ziffer '0')

Soundex: Beispiele

	'Tukker'	**'Tacker'**	**'William'**	**'Bill'**
Schritt 1	'Tukker'	'Tacker'	'William'	'Bill'
Schritt 2	'Tu22e6'	'Ta22e6'	'Wi44ia5'	'Bi44'
Schritt 3	'Tu2e6'	'Ta2e6'	'Wi4ia5'	'Bi4'
Schritt 4	'T26'	'T26'	'W45'	'B4'
Schritt 5	**'T260'**	**'T260'**	**'W450'**	**'B400'**

- Mit Ausnahme des ersten Buchstaben werden alle Vorkommnisse von 'h' und 'w' entfernt
- Mit Ausnahme des ersten Buchstaben werden alle Nichtvokale durch Ziffern ersetzt (k→2, r→6, l→4, m→5)
- Alle aufeinanderfolgenen Vorkommnisse der gleichen Ziffer werden durch ein einzelnes Vorkommnis ersetzt
- Mit Ausnahme des ersten Buchstaben werden alle Nicht-Ziffern entfernt
- Der Code wird auf die Länge vier beschränkt (Auffüllen mit '0')

Soundex (alternative Definition)

1. Mit Ausnahme des ersten Buchstaben werden alle Vorkommnisse der Buchstaben 'a', 'e', 'i', 'o', 'u', 'y', 'h' und 'w' entfernt
2. Mit Ausnahme des ersten Buchstaben werden alle verbliebende Buchstaben durch Ziffern ersetzt

b, f, p, v	$\longrightarrow$	1
c, g, j, k, q, s, x, z	$\longrightarrow$	2
d, t	$\longrightarrow$	3

l	$\longrightarrow$	4
m, n	$\longrightarrow$	5
r	$\longrightarrow$	6

3. Alle aufeinanderfolgenen Vorkommnisse der gleichen Ziffer werden durch ein einzelnes ersetzt (z.B. '1001' wird zu '101'), wenn die den Ziffern zugehörigen Buchstaben im ursprünglichen String nicht durch einen Vokal (inkl. 'y') getrennt waren
4. Der Code wird auf die Länge vier beschränkt (notfalls Auffüllen mit der Ziffer '0')

Soundex: Beispiele (alternative Definition)

	'Tukker'	**'Tacker'**	**'William'**	**'Bill'**
Schritt 1	'Tkkr'	'Tckr'	'Wllm'	'Bll'
Schritt 2	'T226'	'T226'	'W445'	'B44'
Schritt 3	'T26'	'T26'	'W45'	'B4'
Schritt 4	**'T260'**	**'T260'**	**'W450'**	**'B400'**

- Mit Ausnahme des ersten Buchstaben werden alle Vorkommnisse der Buchstaben 'a', 'e', 'i', 'o', 'u', 'y', 'h', und 'w' entfernt
- Mit Ausnahme des ersten Buchstaben werden alle verbliebende Buchstaben durch Ziffern ersetzt (k$\rightarrow$2, r$\rightarrow$6, l$\rightarrow$4, m$\rightarrow$5)
- Alle aufeinanderfolgenen Vorkommnisse der gleichen Ziffer werden durch ein einzelnes Vorkommnis ersetzt
- Der Code wird auf die Länge vier beschränkt (Auffüllen mit '0')

Soundex

- Gleiche Codes resultieren nur wenn der erste Buchstabe gleich ist (daher Probleme mit 'Bill' vs. 'William' oder 'Jawornicki' vs. 'Yavornitzky')
- konzipiert für kaukasische Namen, funktioniert aber auch zufriedenstellend für Datensätze mit Namen verschiedener Herkünfte
- funktioniert allerdings weniger gut für asiatische Namen
- weitere phonetische Maße vorhanden (z.B. Kölner Phonetik [Pos69], Metaphone [Phi90] oder NYSIIS Algorithmus [Taf70])

Semantische Ähnlichkeit

- Erkennen von semantischen Beziehungen zwischen Begriffen
- Verwendung von Ontologien
- Spektrum reicht von formalen Regeln bis hin zu informalen Beschreibungen
- Weitverbreitete informale Ontologien sind Thesauri
- Thesauri eignen sich um Synonymen und Antonymen, sowie um Hyponymen und Hyperonymen zu erkennen

Thesauri: Beispiel von Synonymen

- Vornamen die mit den Vornamen 'Alexander' verwandt sind*

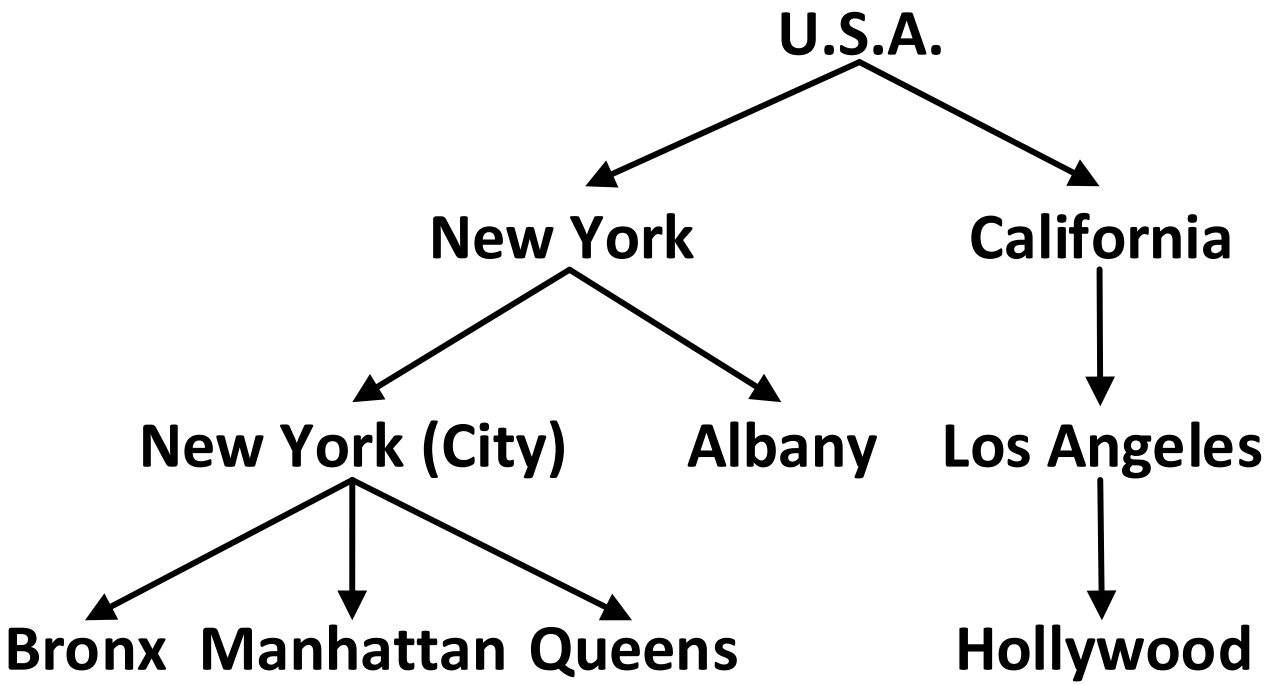
name	**LevSim**
Alejandro	0.67
Aleksandr	0.67
Alessandro	0.6
Alexandros	0.8
Alexis	0.44
Leszek	0.33
Sascha	0.11
Xander	0.67

- Andere Beispiele: 'Bill' vs. 'William', 'James' vs. 'Jim', 'Australien' vs. 'Down Under'

*Quelle: http://www.onomastik.com/

Thesauri: Beispiel von Hyponymen und Hyperonymen

- Hierarchie von Statdteilen, Städten und Staaten in den U.S.A.



Hybride Ähnlichkeitsmaße

- Nachteil Mengenbasiert: Typos werden nicht kompensiert
- Nachteil Sequenzbasiert: Wortvertauschungen werden nicht kompensiert
- Idee: Beide Konzepte in einem Maß zu vereinen
- Zerlegung eines Stringwertes x in eine Multimenge von Token $X = tok(x)$
- Verwendung eines Sequenzbasiertes Maßes *TokenSim* um die Ähnlichkeit zwischen Token zu bestimmen

Extended Jaccard Coefficient [NH10]

- Ähnlichkeit zwischen zwei Stringwerten x und y als Anteil ähnlicher Token in $X = tok(x)$ und $Y = tok(y)$
- Menge an ähnlichen Tokenpaaren:

$$shared(X, Y) = \{(t_1, t_2) \mid t_1 \in X, t_2 \in Y, TokenSim(t_1, t_2) \geq \theta\}$$

- Menge an Token von X ohne ähnliches Token in Y:

$$unique(X) = \{t_1 \mid t_1 \in X, \nexists t_2 \in Y, TokenSim(t_1, t_2) \geq \theta\}$$

- Extended Jaccard Coefficient:

$$ExtJacc(X, Y) = \frac{|shared(X, Y)|}{|shared(X, Y)| + |unique(X)| + |unique(Y)|}$$

Extended Jaccard Coefficient (Beispiel)

- $x =$'Henri Waternoose'
  $y =$'Henry Peter Waternose'
- $X = \{$'Henri','Waternoose'$\}$
  $Y = \{$'Henry','Peter','Waternose'$\}$
- $shared(X, Y) = \{$('Henri','Henry'),('Waternoose','Waternose')$\}$
- $unique(X) = \emptyset$
  $unique(Y) = \{$'Peter'$\}$
- $ExtJacc(X, Y) = \frac{2}{2+0+1} = \frac{2}{3}$

Generalized Jaccard Coefficient [DHI12]

- Ähnlichkeit zwischen zwei Stringwerten x und y als normaliziertes Maximum Weighted Matching ihrer Token
- Betrachtung der Ähnlichkeiten der Token von $X = tok(x)$ und $Y = tok(y)$ als bipartiter Graph
- Entferne alle Kanten deren Gewicht (Ähnlichkeit zweier Token) kleiner als θ ist
- Berechne das Maximum Weighted Matching M des reduzierten bipartiten Graphen
- Generalized Jaccard Coefficient:

$$GenJacc(X, Y) = \frac{\sum_{(t_1,t_2)\in M} TokenSim(t_1, t_2)}{|X| + |Y| - |M|}$$

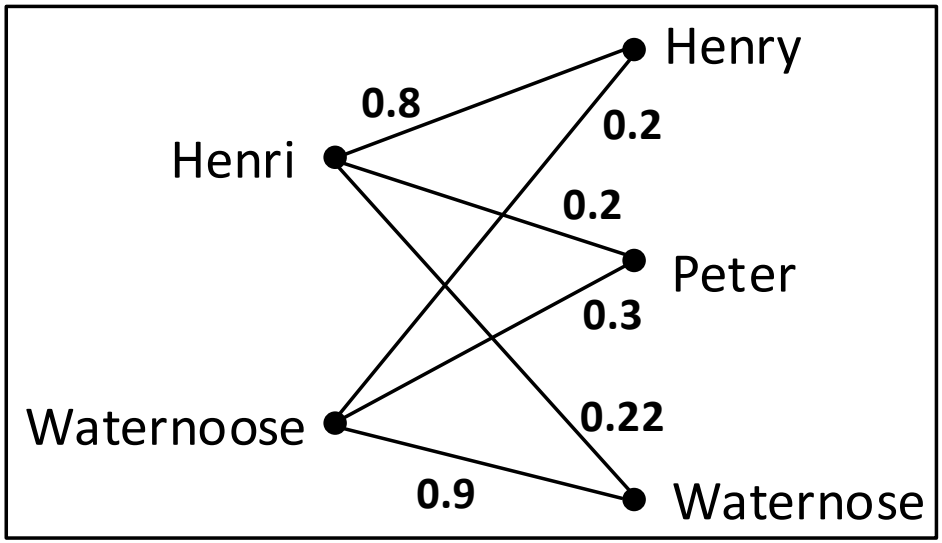
Maximum Weighted Matching

Gegeben: Ein (gewichteter) Graph $G = (V, E, w)$

- Ein Matching M von G ist eine Menge an Kanten die sich keine Knoten teilen (die Kanten sind paarweise nicht adjazent)
- Ein Maximal Matching M von G ist ein Matching das nicht erweitert werden kann (d.h. das Hinzufügen einer weiteren Kante von G bedeutet, dass M kein Matching mehr ist)
- Ein Maximum Matching M von G ist ein Matching das (unter allen mgl. Matchings) die maximale Anzahl an Kanten besitzt
- Ein Maximum Weighted Matching M von G ist ein Matching bei dem die Gesamtsumme aller Kantengewichte maximal ist

Generalized Jaccard Coefficient (Beispiel)

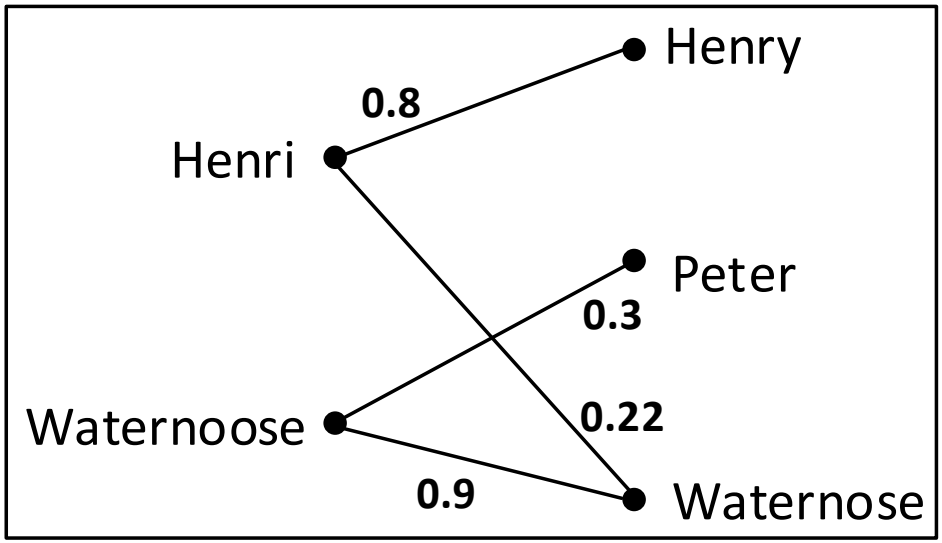
- $X = \{\text{'Henri'},\text{'Waternoose'}\}$
  $Y = \{\text{'Henry'},\text{'Peter'},\text{'Waternose'}\}$
- threshold $\theta = 0.21$



- $GenJacc(X, Y) = \frac{0.8+0.9}{2+3-2} = \frac{1.7}{3} = 0.567$

Generalized Jaccard Coefficient (Beispiel)

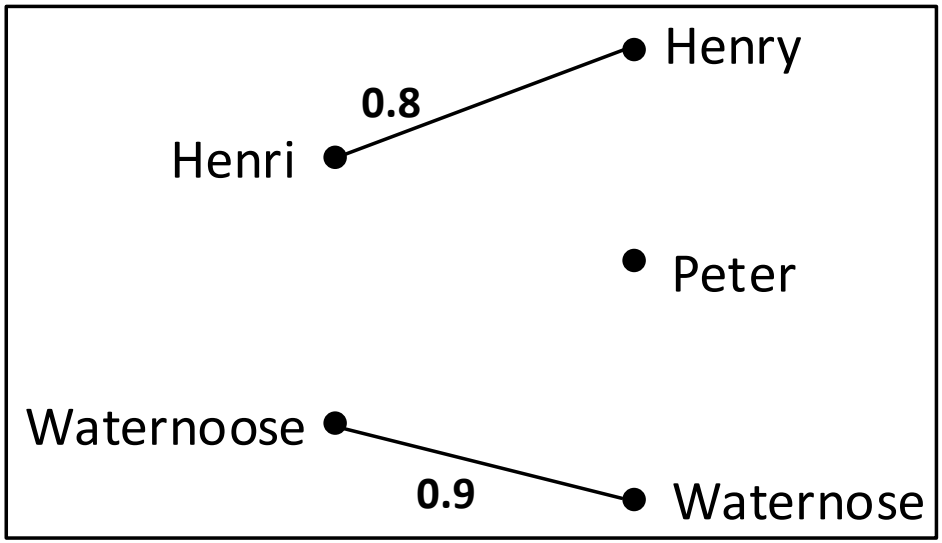
- $X = \{\text{'Henri'},\text{'Waternoose'}\}$
  $Y = \{\text{'Henry'},\text{'Peter'},\text{'Waternose'}\}$
- threshold $\theta = 0.21$



- $GenJacc(X, Y) = \frac{0.8+0.9}{2+3-2} = \frac{1.7}{3} = 0.567$

Generalized Jaccard Coefficient (Beispiel)

- $X = \{\text{'Henri'},\text{'Waternoose'}\}$
  $Y = \{\text{'Henry'},\text{'Peter'},\text{'Waternose'}\}$
- threshold $\theta = 0.21$



- $GenJacc(X, Y) = \frac{0.8+0.9}{2+3-2} = \frac{1.7}{3} = 0.567$

Monge-Elkan Similarity [ME97]

- Ähnlichkeit von Stringwert x zu Stringwert y als durchschnittliche maximale Ähnlichkeit der Token in $X = tok(x)$ zu Token in $Y = tok(y)$
- Monge-Elkan Similarity:

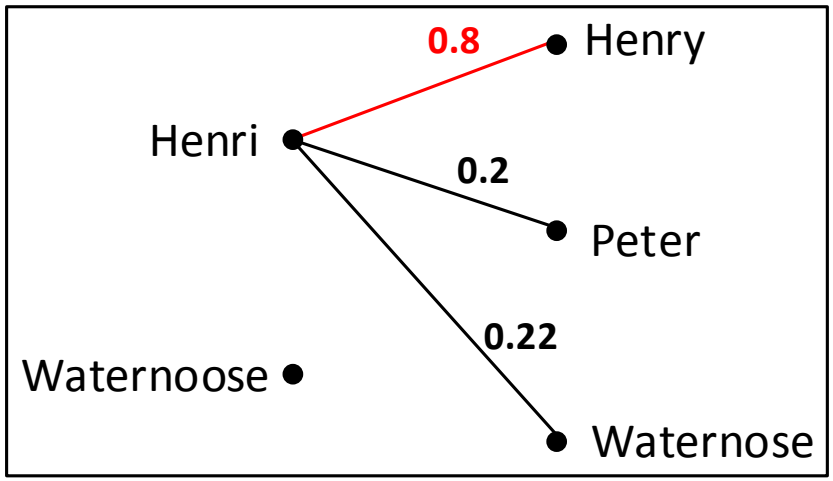
$$MongeElkan(X, Y) = \frac{1}{|X|} \sum_{i=1}^{|X|} max_{j=1}^{|Y|} TokenSim(t_i, t_j)$$

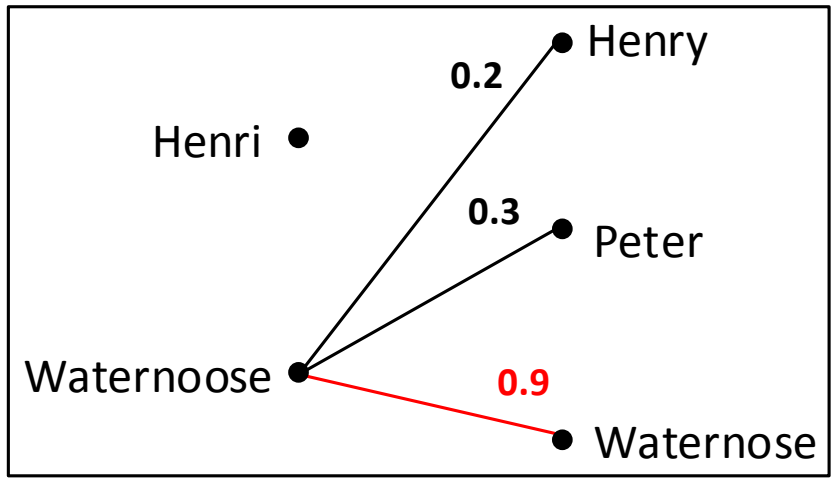
- gerichtetes Ähnlichkeitsmaß (nicht symmetrisch)
  $\Rightarrow$ für manche X und Y gilt:

$$MongeElkan(X, Y) \neq MongeElkan(Y, X)$$

Monge-Elkan Similarity (Beispiel)

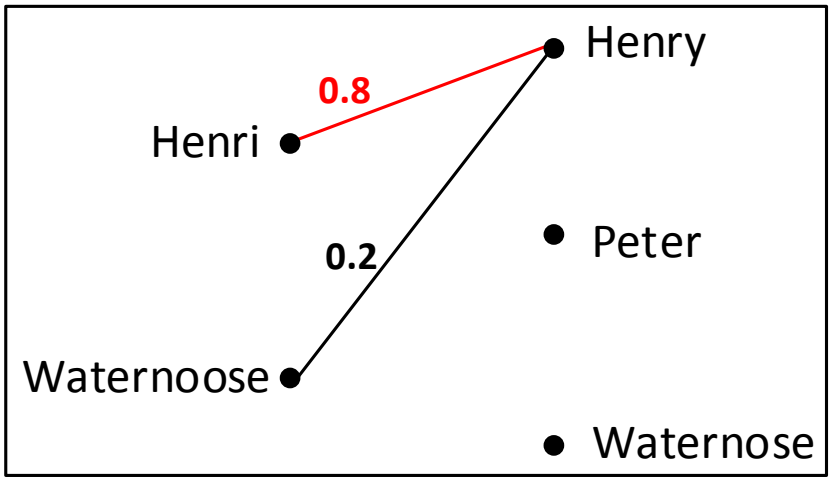
- $X = \{\text{'Henri'},\text{'Waternoose'}\}$
  $Y = \{\text{'Henry'},\text{'Peter'},\text{'Waternose'}\}$

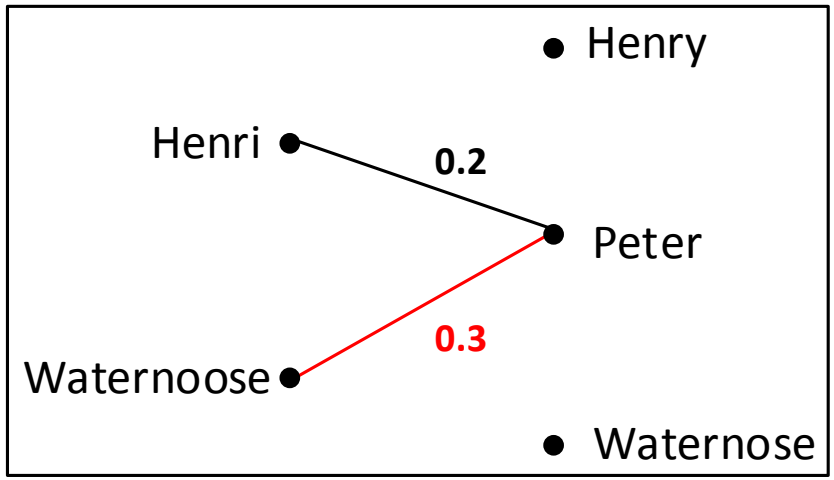


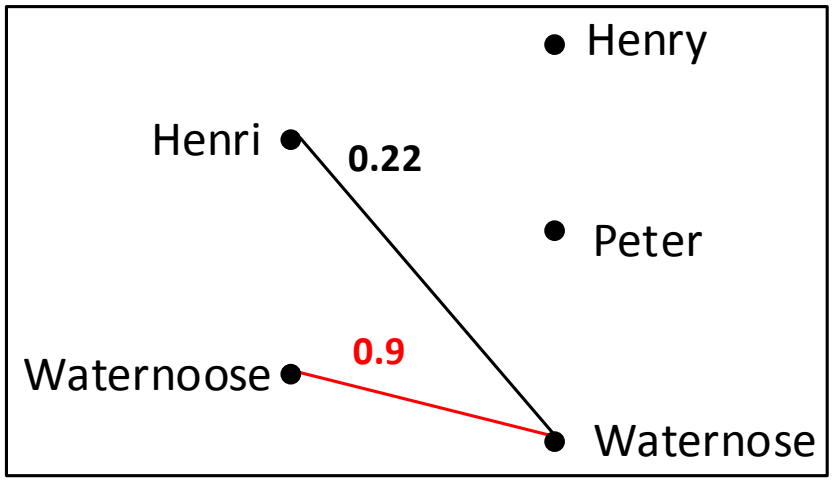


- $\mathit{MongeElkan}(X, Y) = \frac{1}{2} \times (0.8 + 0.9) = 0.85$

Monge-Elkan Similarity (Beispiel)







- $MongeElkan(Y, X) = \frac{1}{3} \times (0.8 + 0.3 + 0.9) = 0.67$

Skalierbarkeit

- Vergleichen jedes Stringwertes $x \in A$ mit jedem Stringwert $y \in B$ ist bei großen Mengen zu ineffizient (quadratische Komplexität)
- Verwendung einer Methode *FindCands* welche eine Menge an möglichen Matches $C \subseteq B$ für jedes $x \in A$ effizient bestimmt

  Für jeden Stringwert $x \in A$
  - verwende *FindCands* um eine Menge $C \subseteq B$ zu bestimmen
  - für jeden Stringwert $y \in C$
    - wenn $s(x, y) \geq \theta$, dann klassifiziere (x, y) als Match

- Hilfreich bei Schema Matching
- Bei Duplikatenerkennung wenig hilfreich, da θ stark kontextabhängig

Invertierter Index [DHI12]

- Aufbauen eines invertierten Indexes über die Token der einzelnen Stringwerte von B

Set A

1:{lake, mendota}
2:{lake, monona, area}
3:{lake, mendota, monona, dane}

Set B

4:{lake, monona, university}
5:{monona, research, area}
6:{lake, mendota, monona, area}

Token in B	ID Listen
area	5,6
lake	4,6
mendota	6
monona	4,5,6
research	5
university	4

- Ermöglicht ein schnelles Auffinden aller Stringwerte $y \in B$ die ein Token mit einem Stringwert $x \in A$ teilen
- Die Reduktion an Vergleichen ist aber oft noch nicht groß genug

Size Filtering [DHI12]

- Auswahl an Kandidaten anhand ihrer Länge
- B-Baum mit allen Stringwerten in B als Datensätze und deren Länge als Schlüsselwert.
- Besonders nützlich für Jaccard Coefficient

$$Jacc(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

- Es gilt: $Jacc(X, Y) \leq |Y|/|X| \leq 1/Jacc(X, Y)$

  (Anm.: $Jacc(X, Y)$ maximal wenn $Y \subseteq X$ oder $X \subseteq Y$)

- Vorgabe: x und y sind ein Match wenn $Jacc(X, Y) \geq \theta$

$\Rightarrow$ y kann nur ein Match von x sein wenn $|X| \times \theta \leq |Y| \leq |X|/\theta$

Size Filtering (Beispiel) [DHI12]

Set A

1:{lake, mendota}
2:{lake, monona, area}
3:{lake, mendota, monona, dane}

Set B

4:{lake, monona, university}
5:{monona, research, area}
6:{lake, mendota, monona, area}

Token in B	ID Listen
area	5,6
lake	4,6
mendota	6
monona	4,5,6
research	5
university	4

- Annahme: $\theta = 0.8$ und X={lake, mendota}
- Damit y ein Match von x sein kann, muss gelten:

$$1.6 = 2 \times 0.8 \leq |Y| \leq 2/0.8 = 2.5$$

$\Rightarrow$ Da für alle $y \in B$ gilt: $|Y| \geq 3 \Rightarrow$ Kein Kandidat für x

Prefix Filtering (Overlap Measure) [DHI12]

- Idee: Wenn zwei Mengen viele gemeinsame Elemente haben, müssen sich alle großen Untermengen überschneiden
- Wenn $|X \cap Y| \geq k \Rightarrow$ jede Untermenge von $X' \subseteq X$ mit $|X'| \geq |X| - (k-1)$ überschneidet sich mit Y

- Beispiel: $X = \{\text{lake, mendota, monona, dane}\}$ und $Y = \{\text{lake, mendota, monona, area}\}$

  Überschneidung: $|X \cap Y| = 3$

  $\Rightarrow$ alle Mengen in $\{X' \mid X' \subseteq X, |X'| \geq 2\}$ schneiden Y

  z.B. $X' = \{\text{lake, dane}\}$ schneidet Y

Prefix Filtering (Overlap Measure) [DHI12]

- Vorgabe: $x \in A$ und $y \in B$ sind ein Match, wenn $|X \cap Y| \geq k$
- Aufbau eines invertierten Indexes der Token in B
- Auswahl eines beliebigen $X' \subseteq X$ mit $|X'| \geq |X| - (k-1)$ (z.B. der Präfix von X der Länge $|X| - (k-1)$)
- Bestimmung der Kandidaten von x mit Hilfe des Indexes
- Reduziert Anzahl an Kandidaten pro $x \in A$

Prefix Filtering (Beispiel 1) [DHI12]

Set A

1:{lake, mendota}
2:{lake, monona, area}
3:{lake, mendota, monona, dane}

Set B

4:{lake, monona, university}
5:{monona, research, area}
6:{lake, mendota, monona, area}
7:{dane, area, mendota}

Token in B	ID Listen
area	5,6,7
lake	4,6
mendota	6,7
monona	4,5,6
research	5
university	4
dane	7

- Annahme: $k = 2$ und X={lake, mendota}
- X'={lake}
- Kandidaten für x sind: $\{4, 6\}$
- Ohne Prefix Filtering: Kandidaten für x sind: $\{4, 6, 7\}$

Prefix Filtering (Beispiel 2) [DHI12]

Set A

1:{lake, mendota}
2:{lake, monona, area}
3:{lake, mendota, monona, dane}

Set B

4:{lake, monona, university}
5:{monona, research, area}
6:{lake, mendota, monona, area}
7:{dane, area, mendota}

Token in B	ID Listen
area	5,6,7
lake	4,6
mendota	6,7
monona	4,5,6
research	5
university	4
dane	7

- Annahme: $k = 2$ und X={lake, monona, area}
- X'={lake, monona}
- Kandidaten für x sind: $\{4, 6\}$ (lake) und $\{4, 5, 6\}$ (monona)
- Ohne Prefix Filtering: Kandidaten für x sind: $\{4, 5, 6, 7\}$

Prefix Filtering (Jaccard Koeffizient) [DHI12]

- Aus $Jacc(X, Y) \geq \theta$ folgt [DHI12]:

$$|X \cap Y| \geq k = \lceil \frac{\theta}{1+\theta} \times (|X| + |Y|) \rceil$$

- Beispiel: $X = \{\text{lake, mendota, monona, dane}\}$ und $Y = \{\text{lake, mendota, monona, area}\}$

  Annahme: $\theta = 0.5$

  $\Rightarrow k = \lceil 8/3 \rceil = \lceil 2.67 \rceil = 3$

Andere Ähnlichkeitsmaße [DHI12]

- Bisher nur Overlap Measure und Jaccard Koeffizient
- Jede dieser Techniken kann für ein anderes Ähnlichkeitsmaß $s(x, y)$ verwendet werden, wenn wir $s(x, y)$ mit Hilfe von Constraints durch eines der beiden Maße ausdrücken können

- Levenshtein: Aus $LevDst(x, y) \leq \epsilon$ folgt:

$$|X \cap Y| \geq k = \lceil max(|X|, |Y|) + q - 1) - q \times \epsilon \rceil$$

  wenn q-grams zur Tokenbildung verwendet wurden, d.h. X entspricht der Menge an q-grams von x

- Kosinus: Aus $CosSim(X, Y) \geq \theta$ folgt:

$$|X \cap Y| \geq k = \lceil \theta \times \sqrt{|X| \times |Y|} \rceil$$

Literatur I

[BG07] Indrajit Bhattacharya and Lise Getoor.
Collective entity resolution in relational data.
TKDD, 1(1), 2007.

[DHI12] Anhai Doan, Alon Halevy, and Zachary Ives.
Principles of Data Integration.
Morgan Kaufmann, 2012.

[ME97] Alvaro E. Monge and Charles Elkan.
An efficient domain-independent algorithm for detecting approximately duplicate database records.
In *DMKD*, pages 0-, 1997

[NH10] Felix Naumann and Melanie Herschel.
An Introduction to Duplicate Detection.
Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

Literatur II

[Phi90] Lawrence Philips.
Hanging on the metaphone.
Computer Language Magazine, 7(12):39-44, December 1990.

[Pos69] Hans Joachim Postel.
Die kölner Phonetik. Ein Verfahren zur Identifizierung von Personennamen auf der Grundlage der Gestaltanalyse.
IBM-Nachrichten, pages 925-931, 1969.

[Russ18] R.C. Russell.
A method of phonetic indexing, 1918.

[Russ22] R.C. Russell.
A method of phonetic indexing, 1922.

[Taf70] Robert L. Taft.
Name Search Techniques.
Bureau of Systems Development, New York State Identification and Intelligence System, 1970.