Australian National University

COMP3430 / COMP8430
Data wrangling

Lecture 16: Record pair comparison (2)
(Lecturer: Peter Christen)

Lecture outline

- Comparing strings for record linkage
- Approximate string comparison functions
  - Q-gram based (Jaccard and Dice coefficient)
  - Edit and bag distances
  - Jaro-Winkler
- Statistical linkage key SLK-581

Comparing strings for record linkage

- Strings (text) are the most commonly used attributes (fields) when comparing records
  - Names: Title; first, middle, and last name; name suffix and prefix, etc.
  - Addresses: Street name and type, postcode/zipcode (e.g. in the UK: “CB3 0EH”), suburb/town name, state/territory and country names
  - Telephone numbers, emails, credit card numbers, drivers license numbers, etc.
- The aim is to calculate a normalised similarity between two strings with $0 \leq sim_{approx} \leq 1$
- Many different techniques available
  - Some general to any types of strings, others specific to certain types (such as personal names or long genome sequences)

Q-gram based string comparison (1)

- Convert a string into its set of q-grams
  - Often with $q = 2$ (bigrams) or $q = 3$ (trigrams)
  - For example, with bigrams: “peter” → [‘pe’, ‘et’, ‘te’, ‘er’]
- Calculate the similarity between two strings based on counting the number of q-grams that occur in both strings
  - Jaccard similarity: $sim_{Jacc}(s_1,s_2) = |intersection(Q_1,Q_2)| / |union(Q_1,Q_2)|$
  - Dice coefficient: $sim_{Dice}(s_1,s_2) = 2* |intersection(Q_1,Q_2)| / (|Q_1|+|Q_2|)$

  where:
  - Q_x is the set of q-grams extracted from string s_x
  - $intersection(Q_1,Q_2)$ is the set of q-grams that occur in both strings
  - |..| denotes the number of elements in a set

Q-gram based string comparison (2)

- For example, with s_1 = “peter” and s_2 = “pete” and $q = 2$:
  - Q_1 = [‘pe’, ‘et’, ‘te’, ‘er’], Q_2 = [‘pe’, ‘et’, ‘te’], $|Q_1| = 4$, $|Q_2| = 3$
  - intersection(Q_1,Q_2) = [‘pe’, ‘et’, ‘te’] and *union*(Q_1,Q_2) = [‘pe’, ‘et’, ‘te’, ‘er’]
  - $sim_{Jacc}(s_1, s_2)$ = | [‘pe’, ‘et’, ‘te’] | / | [‘pe’, ‘et’, ‘te’, ‘er’] | = 3 / 4 = 0.75
  - $sim_{Dice}(s_1, s_2) = 2*3 / (3+4) = 6 / 7 = 0.857$

$Q1 \wedge Q2$

- **Questions**: *Which one is correct? Which one is better? What are the Jaccard and Dice similarities between* s_1 = 'peter' and s_2 = 'pedro' for q = *1, 2, and 3 ?*

Edit distance (1)

- Idea: Count how many basic *edit operations* are needed to convert one string into another (known as *Levenshtein* edit distance)
  - Insertion of a character: “pete” → “peter”
  - Deletion of a character: “miller” → “miler”
  - Substitution of a character: “smith” → “smyth”
  - Transpositions of two adjacent characters: “sydney” → “sydeny” (known as *Damerau-Levenshtein* edit distance)

$Q1 \wedge Q2$

- **Questions**: *What is the Levenshtein edit distance between “peter” and “petra”, and between “gayle” and “gail”?*

Edit distance (2)

- Convert an edit distance into a similarity $0 \leq sim_{edit\_dist} \leq 1$ by calculating $sim_{edit\_dist}(s_1, s_2) = 1 - edit\_dist(s_1, s_2) / max(len(s_1), len(s_2))$
- For example, with s_1 = “peter” and s_2 = “petra”:

  $sim_{edit\_dist}(s_1, s_2) = 1 - 2 / max(5, 5) = 1 - 2 / 5 = 3 / 5 = 0.6$

$Q1 \wedge Q2$

- Edit distance can be calculated using a dynamic programming algorithm based on the *edit matrix*
  - Which has a quadratic complexity in the lengths of the two strings (i.e. requires $len(s_1) * len(s_2)$ *computational steps*)

Edit distance (3)

- Matrix shows the number of edits between sub-strings
  (for example, between ‘ga’ and 'gayle’ → 3 inserts)

  “gail” → substitute ‘i’ with ‘y’,
  then insert ‘e’ → “gayle”
  (final edit distance is 2)

- **Question**: *Calculate edit distance between s_1 = “peter” and s_2 = “petra”*

		g	a	y	l	e
	0	1	2	3	4	5
g	1	**0**	1	2	3	4
a	2	1	**0**	1	2	3
i	3	2	1	**1**	2	3
l	4	3	2	2	**1**	**2**

Bag distance

- Main drawback of edit distance is its quadratic complexity in the lengths of the two strings, i.e. $len(s_1) * len(s_2)$ computational steps
- A fast approximation of edit distance is *bag distance*
  - A bag is a multi set of the characters in a string: “peter” → [‘e’, ‘e’, ‘p’, ‘r’, ‘t’]
- Bag distance is defined as:
  $bag\_dist(s_1,s_2) = max(|x - y|, |y - x|)$, where $x = bag(s_1)$ and $y = bag(s_2)$
- It has been shown that always: $bag\_dist(s_1, s_2) \leq edit\_dist(s_1, s_2)$, and therefore: $sim_{bag\_dist}(s_1, s_2) \geq sim_{edit\_dist}(s_1, s_2)$
  - If $sim_{bag\_dist}(s_1, s_2)$ is below a threshold then edit distance does not need to be calculated

Jaro-Winkler string comparison (1)

- Developed by the US Census Bureau specifically to compare personal name strings, taking various heuristics into account that are based on extensive practical experiences of name matching
- A combination of q-gram and edit distance string comparison
- Basic idea of the Jaro comparison function:
  - Count *c*, the number of agreeing (common) characters within half the length of the longer string
  - Count *t*, the number of transposed characters ('pe' versus 'ep') in the set of common strings
  - Calculate $sim_{Jaro}(s_1, s_2) = (c / len(s_1) + c / len(s_2) + (c - t) / c) / 3$

Jaro-Winkler string comparison (2)

- Further modifications, named Jaro-Winkler, aim to improve name matching further
  - Increase similarity if the first few characters (p, with $p \leq 4$) are the same:
    $sim_{Jaro\_Winkler}(s_1, s_2) = \ = sim_{Jaro}(s_1, s_2) + (1 - sim_{Jaro}(s_1, s_2)) * p/10$
    For example, for s_1 = “peter” and s_2 = “petra”: $p = 3$ (“pet”)
  - Further increase the similarity if both strings are at least 5 characters long and contain two common characters besides the prefix
  - Adjust similarity if certain similar character pairs (like in Soundex) occur in two strings (for example ‘w’ ↔ ‘v’ or ‘s’ ↔ ‘z’)

Statistical linkage key SLK-581 (1)

- Developed by the Australian Institute for Health and Welfare http://meteor.aihw.gov.au/content/index.phtml/itemId/349895
- Aims to identify records that likely correspond to the same person
- Combines blocking and comparison functionalities
- Basic idea:
  - Take the 2nd, 3rd, and 5th letters of a record's family name (surname)
  - Take the 2nd and 3rd letters of the record's given name (first name)
  - Take the day, month and year of the person, concatenated in that order (*ddmmyyyy*) to form the date of birth
  - Take the gender of the person (1=male, 2=female, 9=unknown)
  - If names too short use 2, if full name component missing use 999

Statistical linkage key SLK-581 (2)

- Examples: (spaces added for illustration only)
  - “marie miller”, 13/04/1991, “f” → “ile ar 13041991 2”
  - “john smith”, 31/03/2001, “m” → “mih oh 31032001 1”
  - “ashley lee”, 11/12/1963, “u” → “ee2 sh 11121963 9”

- **Question**: *Calculate SLK-581 for yourself* $Q1 \wedge Q2$