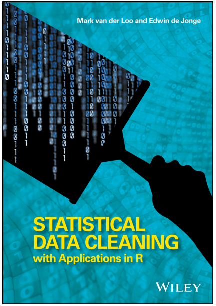
Approximate Matching



Mark van der Loo and Edwin de Jonge

Statistics Netherlands Research & Development
Twitter: `@markvdloo` `@edwindjonge`

Tokyo | Institute of Statistical Mathemathics | 2020

Try the code together with your neighbour

`02input/raw_to_input.R`

大学共同利用機関法人 情報・システム研究機構
統計数理研究所
The Institute of Statistical Mathematics

String distance

Default (Optimal String Alignment distance)

Count number of character deletions, insersions, substitutions and transpositions (of adjacent characters)

```
library(stringdist)
stringdist("Ross Ihaka", "Robert Gentleman")
```

```
## [1] 12
```

Exact Matching with `match`

```
lookup <- c("Alice","Bob","Carol","Danny")
raw    <- c("Bob","Carl","Rob","bob","Dan","Alice")
i <- match(raw, lookup)
data.frame(raw=raw, matched=lookup[i])
```

```
##     raw matched
## 1   Bob     Bob
## 2  Carl    <NA>
## 3   Rob    <NA>
## 4   bob    <NA>
## 5   Dan    <NA>
## 6 Alice   Alice
```

Approximate Matching with `stringdist::amatch`

```
library(stringdist)
j <- amatch(raw, lookup, maxDist=2)
data.frame(raw=raw, matched=lookup[i], amatched=lookup[j])
```

```
##     raw matched amatched
## 1   Bob     Bob      Bob
## 2  Carl    <NA>    Carol
## 3   Rob    <NA>      Bob
## 4   bob    <NA>      Bob
## 5   Dan    <NA>    Danny
## 6 Alice   Alice    Alice
```

$\rightarrow$ Match with closest match, and distance ≤ 2.

Optimal string alignment?

```
stringdist("Robert Gentleman", "Gentleman, Robert")
```

```
## [1] 15
```

```
stringdist("Robert Gentleman", "Ross Ihaka")
```

```
## [1] 12
```

→ OSA wil give a false match (if we allow `maxDist` of 12)

Alternative: cosine distance

```r
stringdist("Robert Gentleman","Gentleman, Robert"
           , method="cosine", q=2)
```

```
## [1] 0.1608536
```

```r
stringdist("Robert Gentleman","Ross Ihaka"
           , method="cosine", q=2)
```

```
## [1] 0.9139337
```

Notes

- Based on counting co-occurrence of character *q*-grams (here: pairs).
- Always between 0 and 1

More on `amatch`

```
amatch(x, table, method, maxDist,...)
```

`x`	character data to be matched
`table`	the lookup table with clean values
`method`	string distance type
`maxDist`	Maximum distance allowed (depends on `"method"`!)
`...`	Extra options depending on `"method"`

Example

```
amatch(raw, lookup, method="cosine", maxDist=0.5, q=3)
```

Assignment

Merge data from the `companies` dataset with data from `backbone.csv`.

- Using approximate matching on the `"name"` and `"company"` column.
- Think about and try different distance functions and `maxDist`
- Keep your best solution
- Remove rows that cannot be matched
- Write to `02input/myinput.csv`

More on String distances

More on string distances

Main idea

Define a sence of *distance* between two text strings

Distance

A function $d(s, t)$ that takes two arguments and

- returns a nonnegative number,
- returns zero if and only if $s = t$
- is symmetric: $d(s, t) = d(t, s)$
- is the length of a shortest path between s and t: $d(s, t) \leq d(s, u) + d(u, t)$

Note

Some string distances violate one or more of the above assumptions.

Distance types

- Edit based
- q-gram based
- Heuristic

Edit-based string distances

Idea

1. Choose basic steps to alter a string
2. Find the smallest nr of steps that changes s into t
3. The distance equals the nr of steps needed.

Basic steps

- deletion: hihi $\rightarrow$ hii
- insertion: hihi $\rightarrow$ hihih
- substitution: hihi $\rightarrow$ hiha
- transposition: hihi $\rightarrow$ ihhi

Edit-based string distances

	Allowed operation			
Distance	substitution	deletion	insertion	transposition
Hamming	✔	✖	✖	✖
LCS	✖	✔	✔	✖
Levenshtein	✔	✔	✔	✖
OSA	✔	✔	✔	✔*
Damerau-Levenshtein	✔	✔	✔	✔

*Substrings may be edited only once.

Example

Levenshtein distance

$$\mathit{leia} \xrightarrow[+1]{\text{sub}} \mathit{lela} \xrightarrow[+1]{\text{ins}} \mathit{leela}$$

Longest common subsequence distance

$$\mathit{leia} \xrightarrow[+1]{\text{del}} \mathit{lea} \xrightarrow[+1]{\text{ins}} \mathit{leea} \xrightarrow[+1]{\text{ins}} \mathit{leela}$$

q-gram based distances (I)

Algorithm

- Tabulate substrings of length q (= q-gram profile)
- Compute a distance between the profiles

Example

2-gram profile of banana

ba	an	na
1	1	2

q-gram based distances (II)

q-gram distance

Manhattan distance between q-gram profiles

$$\sum_{\text{qgram}} |n_{\text{qgram}}(s) - n_{\text{qgram}}(t)|$$

Cosine-distance

1 minus the cosine of the angle between the profiles

$$1 - \frac{\mathbf{n}(s) \cdot \mathbf{n}(t)}{\|\mathbf{n}(s)\|\|\mathbf{n}(t)\|}$$

Note

- Does not satisfy the 'identity' demand.
- Often one chooses $q = 2$ of $q = 3$

Jaro-Winkler distance

Jaro distance

$$d_j(s,t) = 1 - \frac{1}{3}\left(\frac{m}{|s|} + \frac{m}{|t|} + \frac{m+T}{m}\right)$$

- m number of matching characters (within a window)
- T number of matches that need swapping
- $|s|, |t|$ number of characters in s, t.

Jaro-Winkler distance

$$d_{jw}(s,t) = [1 - p\ell(s,t)]d_j(s,t)$$

- $\ell(s,t)$ length of longest equal prefix (up to 4 characters)
- p a number between 0 and 0.25 (usually 0.1)

Soundex

Algorithm

- Strings are appointed a code: same code means 'sounds the same'
- Equal codes: distance zero, otherwise 1

Example

- Farnsworth $\to$ H652
- Fnarswort $\to$ H562

Note

- Based on English pronunciation
- Many extensions exist (see the `phonics` R package)

Which one to use?

Considerations

- Fixed versus variable structure/length
- Why would strings differ? (typos, speech, deliberate changes)
- Performance