

String distance

Hyunjoong Kim

soy.lovit@gmail.com

github.com/lovit

String distance

- 벡터는 다양한 metric 을 이용하여 거리나 유사도를 정의할 수 있습니다.
 - Euclidean, Cosine, ...
- 단어 간 거리는 형태적 거리와 의미적 거리로 정의할 수 있습니다.
 - 의미적으로 비슷한 단어
 - (점심, 저녁) vs (점심, 자동차)
 - 형태적으로 비슷한 단어
 - (서비스, 써비스) vs. (서비스, 소보루)

String distance

- 단어 간 형태적 거리를 정의하기 위해서 다양한 metric 이 제안되었습니다.
 - Levenshtein (Edit) distance
 - Jaro-Winkler
 - Cosine
 - Jaccard
 - Hamming
 - TF-IDF
 - ...

String distance

- 오타자와 정자의 관계는 형태적 거리가 아주 가까운 단어입니다.
 - 두 단어의 형태가 비슷하면서 한 단어의 빈도수가 매우 크다면 희귀한 단어를 오타자로, 빈번한 단어를 정자로 생각할 수 있습니다.
 - 희귀한 단어를 빈번한 단어로 치환하여 오타자를 수정합니다.
 - 써비스 → 서비스

Levenshtein (Edit) distance

- 가장 대표적인 string distance metric 입니다. 단어 A 에서 B 로 수정하기 위한 횟수를 거리로 정의합니다.
- 단어의 수정 방법은 세 가지로 정의됩니다.
 - **Deletion**: '서**어**비스' → '서비스' (거리 = 1)
 - **Insertion**: '데이터마닝' → '데이터마**이**닝' (거리 = 1)
 - **Substitution**: '데이**타**마이닝' → '데이**터**마이닝' (거리 = 1)

Levenshtein (Edit) distance

- String metric 중 가장 대표적인 척도입니다. 단어 A 에서 B 로 수정하기 위한 횟수를 거리로 정의합니다.
 - 데이타마닝 → 데이터마이닝
 - 1단계: 데이~~타~~마닝 → 데이~~터~~마닝 (누적 거리 = 1)
 - 2단계: 데이터마닝 → 데이터마~~이~~닝 (누적 거리 = 2)

Levenshtein (Edit) distance

- 거리 행렬을 이용하여 edit distance 를 계산합니다.
 - $d(\text{데이타마닝} \rightarrow \text{데이터마이닝}) = d(\text{데이터마이닝} \rightarrow \text{데이타마닝})$
 - '데' \rightarrow '데이타마닝'거리

	데	이	타	마	닝
데	0	1	2	3	4
이					
터					
마					
이					
닝					

← Insertion [이, 타, 마, 닙]

Levenshtein (Edit) distance

- 거리 행렬을 이용하여 edit distance 를 계산합니다.
 - '데이' → '데이타마닝' 거리

The image shows a Levenshtein distance matrix for the strings '데이' (Day) and '데이타마닝' (Daytamaning). The matrix is a 6x6 grid. The first row and column represent the empty string. The first four rows and columns represent the characters of '데이' and '데이타마닝'. The cell (2,2) containing '0' is highlighted with a green box, with a green arrow pointing to it from the text 'Deletion [이]'. The cells (2,3), (2,4), and (2,5) containing '1', '2', and '3' respectively are highlighted with a red box, with a red arrow pointing to them from the text 'Insertion [타, 마, 닙]'. The cells (2,2), (2,3), (2,4), and (2,5) are highlighted with a yellow background.

	데	이	타	마	닝
데	0	1	2	3	4
이	1	0	1	2	3
터					
마					
이					
닝					

Levenshtein (Edit) distance

- 거리 행렬을 이용하여 edit distance 를 계산합니다.
 - '데이터' → '데이터마닝' 거리

	데	이	타	마	닝
데	0	1	2	3	4
이	1	0	1	2	3
터	2	1	1	2	3
마					
이					
닝					

Substitution [터 → 타]

Insertion [마, 닙]

Levenshtein (Edit) distance

- 거리 행렬을 이용하여 edit distance 를 계산합니다.
- 첫 행과 열을 각각 $[0, 1, 2, \dots, m]$, $[0, 1, 2, \dots, n]$ 로 초기화 합니다.

	데	이	타	마	닝
데	0	1	2	3	4
이	1				
터	2				
마	3				
이	4				
닝	5				

```
for j from 1 to (n=6) :  
  for i from 1 to (m=5) :  
    if s[i] = t[j]: substitutionCost := 0  
    else: substitutionCost := 1  
  
    d[i, j] := minimum(d[i-1, j] + 1, // deletion  
                      d[i, j-1] + 1, // insertion  
                      d[i-1, j-1] + substitutionCost) // subs.  
  
return d[m, n]
```

Levenshtein (Edit) distance

- 거리 행렬을 이용하여 edit distance 를 계산합니다.

	데	이	타	마	닝
데	0	1	2	3	4
이	1	0	1	2	3
터	2	1	1	2	3
마	3	2	2	1	2
이	4	3	3	2	2
닝	5	4	3	2	2

길이가 1 짧은 단어로 바꾸는 것이 최소비용

```
for j from 1 to (m=5) :  
  for l from 1 to (n=6) :  
    if s[i] = t[j]: substitutionCost := 0  
    else: substitutionCost := 1  
  
    d[i, j] := minimum(  
      d[i-1, j] + 1, // deletion  
      d[i, j-1] + 1, // insertion  
      d[i-1, j-1] + substitutionCost // subs.  
    )  
  
return d[m, n]
```

$$d[3,2] = \min(d[2,2] + 1, \\ d[3,1] + 1, \\ d[2,1] + \text{subs})$$

Levenshtein (Edit) distance

- 거리 행렬을 이용하여 edit distance 를 계산합니다.

	데	이	타	마	닝
데	0	1	2	3	4
이	1	0	1	2	3
터	2	1	1	2	3
마	3	2	2	1	2
이	4	3	3	2	2
닝	5	4	3	2	2

'터' 대신 '타' 를 쓰는 것이 최소 비용

```
for j from 1 to (m=5) :  
  for i from 1 to (n=6) :  
    if s[i] = t[j]: substitutionCost := 0  
    else: substitutionCost := 1  
  
    d[i, j] := minimum(d[i-1, j] + 1, // deletion  
                      d[i, j-1] + 1, // insertion  
                      d[i-1, j-1] + substitutionCost) // subs.  
  
return d[m, n]
```

$$d[3,3] = \min(d[2,3] + 1, \\ d[3,2] + 1, \\ d[2,2] + \text{subs})$$

Levenshtein (Edit) distance

- 거리 행렬을 이용하여 edit distance 를 계산합니다.

	데	이	타	마	닝
데	0	1	2	3	4
이	1	0	1	2	3
터	2	1	1	2	3
마	3	2	2	1	2
이	4	3	3	2	2
닝	5	4	3	2	2

'데이' 다음에 '타' 를 입력하는 것이 최소비용

```
for j from 1 to (m=5) :  
  for i from 1 to (n=6) :  
    if s[i] = t[j]: substitutionCost := 0  
    else: substitutionCost := 1  
  
    d[i, j] := minimum(d[i-1, j] + 1, // deletion  
                      d[i, j-1] + 1, // insertion  
                      d[i-1, j-1] + substitutionCost) // subs.  
  
return d[m, n]
```

$$d[2,3] = \min(d[1,3] + 1, \\ d[2,2] + 1, \\ d[1,2] + \text{subs})$$

Jaccard distance

- Jaccard distance 는 집합 간의 유사도를 정의하는 방법입니다.
- 각 단어를 글자 (units) 의 집합으로 생각합니다.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Jaccard distance

- 각 단어를 글자 (units) 의 집합으로 생각합니다.

$$J(\text{'데이터마이닝'}, \text{'데이타마닝'}) = \frac{|\{\text{데, 이, 마, 닝}\}|}{|\{\text{데, 이, 터, 타, 마, 닝}\}|} = \frac{4}{6}$$

- Similarity 를 distance 로 변환합니다. $J_D(A, B) = 1 - J(A, B)$
- **한 글자(unit)**의 등장 횟수는 고려하지 않습니다.

Cosine distance

- Cosine 은 단어 별로 글자 (units) 의 등장 횟수를 고려합니다.

	데	이	터	타	마	닝
데이터마이닝	1	2	1	0	1	1
데이터마닝	1	1	0	1	1	1

- $\cos_D(A, B) = 1 - \cos(A, B)$
- $\cos_D(\text{데이터마이닝}, \text{데이터마닝}) = 0.79$

한국어를 위한 String distance

- String distance 의 unit 을 '초/중/종성' 이나 '어절' 로 정의할 수 있습니다.
 - $d(\text{서비스}, \text{써비스}) \neq d(\text{서비스}, \text{자비스})$
 - Unit = 초/중/종성
 - $d(\text{서비스}, \text{써비스}) = 1 \rightarrow 0.33$
 - Unit = 어절
 - $d(\text{'어디야 지금'}, \text{'지금 어디야'}) = 6 \rightarrow 2$

한국어를 위한 String distance

- Character n-gram 을 units 으로 이용할 수 있습니다.

$$J(\text{'데이터마이닝'}, \text{'데이터마닝'}) = \frac{|\{\text{데이}\}|}{|\{\text{데이, 이터, 터마, 마이, 이닝, 이타, 타마, 마닝}\}|} = \frac{1}{8}$$

한국어를 위한 String distance

- n 음절을 unit 으로 이용하면 더 정확한 문맥을 반영할 수 있습니다.

$$J(\text{'도시보다'}, \text{'보도다시'}) = \mathbf{1} - \frac{|\{\text{도, 시, 보, 다}\}|}{|\{\text{도, 시, 보, 다}\}|} = \mathbf{0}$$

$$J(\text{'도시보다'}, \text{'보도다시'}) = \mathbf{1} - \frac{|\{\ \ \ \ \}|}{|\{\text{도시, 시보, 보다, 보도, 도다, 다시}\}|} = \mathbf{1}$$

한국어를 위한 String distance

- Edit distance 에서 ($a \rightarrow b$) 마다 다른 비용을 부여할 수 있습니다.
 - $d(\text{서비스}, \text{써비스}) < d(\text{서비스}, \text{거비스})$
 - $\text{cost}(\text{서}, \text{써}) < \text{cost}(\text{서}, \text{거})$

```
for j from 1 to (n=6):
    for i from 1 to (m=5):
        if s[i] == t[j]: substitutionCost := 0
        else: substitutionCost := cost.get(s[i], t[j])

        d[i, j] := minimum(d[i-1, j] + 1, // deletion
                           d[i, j-1] + 1, // insertion
                           d[i-1, j-1] + substitutionCost) // subs.

return d[m, n]
```

Inverted index 를 이용한 빠른 Edit distance

- Edit distance 는 각 글자를 unit 으로 이용합니다.
- Edit distance 가 작은 두 단어는 길이가 비슷하고 겹치는 단어가 많아야 합니다.
 - 단어는 bag of characters 로 표현할 수 있습니다.
 - Sparse vector 이기 때문에 inverted index 를 이용할 수 있습니다.

Inverted index 를 이용한 빠른 Edit distance

- Edit distance 의 길이가 d 이하인 두 단어는 다음의 조건을 만족합니다.
 - $n_1 = \text{len}(w_1), n_2 = \text{len}(w_2)$ 일 때, $|n_1 - n_2| \leq d$
 - 두 단어의 길이의 차이는 d 보다 작다.
 - $\text{len}(\text{set}(w_1)) - \text{len}(\text{set}(w_1) \cap \text{set}(w_2)) \leq d$
 - 두 단어에 공통으로 있지 않는 글자의 숫자의 개수가 d 보다 작다.

Inverted index 를 이용한 빠른 Edit distance

- Character 기준으로 inverted index 를 만듭니다.
 - $\text{Index}[c] = [\text{word1}, \text{word2}, \dots]$
- 주어진 query word 의 characters 를 이용하여 다음의 조건을 만족하는 후보를 탐색합니다.
 - 두 단어의 길이의 차이는 d 보다 작다.
 - 두 단어에 공통으로 있지 않는 글자의 숫자의 개수가 d 보다 작다.

Inverted index 를 이용한 빠른 Edit distance

- 132,864 개의 단어 사전에서 Edit distance 를 이용하여 거리가 1 이하인 단어를 탐색하는 예시입니다.
 - 132,864 개의 단어와의 거리계산 및 정렬을 위하여 약 2.49 초가 소요됩니다.
 - Inverted index 를 이용하면 7 개의 단어와 실제 거리 계산을 수행합니다.
 - 0.0056 초가 소요됩니다.

Inverted index 를 이용한 빠른 Edit distance

```
from fast_hangle_levenshtein import LevenshteinIndex

nouns = ['양식어가', '식품유통사', 'ETN전담팀', '도로주행', '로우프라이스펀드', ...]
print(len(nouns)) # 132,864
indexer = LevenshteinIndex(nouns, verbose=True)
print(indexer.levenshtein_search('분식회계'))
```

```
[('분식회계', 0), ('분식회', 1), ('분식회계설', 1), ('분석회계', 1)]
```