

Estructura de Datos y Algoritmos

ITBA 2024-Q2

String Matching – Q-Grams

Q-Grams (o N-Grams)







Es un algoritmo que consiste en generar los pedazos que componen un string. La distancia entre 2 strings estará dada por la cantidad componentes que tengan en común.

Si Q es 1 se generan componentes de longitud 1, Si Q es 2 se generan bi-gramas, si Q es 3 se generan tri-gramas.

Por ejemplo, para el string "JOHN" si se quiere generar hasta tri-gramas ($Q \leq 3$), puede completarse al comienzo y al final con $Q-1$ símbolos especiales (que no pertenezcan al alfabeto) y deslizando la ventana imaginaria de tamaño Q , se va generando los Q -gramas. Sea '###JOHN##':

Q-grams (John) = { 'J', 'O', 'H', 'N', '#J', 'JO', 'OH', 'HN', 'N#',
'###J', '#JO', 'JOH', 'OHN', 'HN#', 'N##' }

String Matching – Q-Grams

Q-grams (John) = {, , 'H', 'N', , , 'OH', 'HN', 'N#',
, , 'JOH', 'OHN', 'HN#', 'N###'}

Qué tan distinto es 'JOHN' de 'JOE' ? ¿Qué calculamos?

Q-grams (Joe) = {, , 'E', , , 'OE', 'E#', , , 'JOE', 'OE#', 'E###' }

Los Q-gramas que tienen en común son:      

Distancia(John, Joe) = 6

Como siempre, se precisa alguna fórmula para pasarlo a un número [0, 1]. Existen varias formas de hacerlo.

Variantes más sofisticadas existen. Por ejemplo: no sólo calculan los Q-Gramas sino la posición que ocupan en el string. Por lo tanto, no es lo mismo que haya coincidencia exactamente en la misma posición a que coincidan pero en otro lugar.

String Matching – Q-Grams

Algunas implementaciones de los Q-Gramas

Usa directamente Tri-gramas (=3) no posicionales.

La fórmula para normalizarlo a un número en $[0, 1]$ es la siguiente:

Q-Gram ($str1, str2$) =

$$\frac{\#TG(str1) + \#TG(str2) - \#TGNoShared(str1, str2)}{\#TG(str1) + \#TG(str2)}$$

Donde:

$\#TG(str1)$ es la cantidad de trigramas que se generaron de $str1$.

$\#TG(str2)$ es la cantidad de trigramas que se generaron de $str2$.

$\#TGNotShared(str1, str2)$ son la cantidad que no matchearon.

String Matching – Q-Grams

Ejemplo, $Q\text{-Gram}(\text{'JOHN'}, \text{'JOE'})$ para $Q=3$, sabiendo que

$Q\text{-grams}(\text{John}) = \{\text{'###J'}, \text{'#JO'}, \text{'JOH'}, \text{'OHN'}, \text{'HN#'}, \text{'N###'}\}$

$Q\text{-grams}(\text{Joe}) = \{\text{'###J'}, \text{'#JO'}, \text{'JOE'}, \text{'OE#'}, \text{'E###'}\}$

Y los Q-gramas que tienen en común son: $\text{'###J'}, \text{'#JO'}$

$$Q\text{-Gram}(\text{John}, \text{Joe}) = (6 + 5 - 7) / (6 + 5) = 0.3636$$

Notar que si tuvieran TODOS los Q-grams en común (matching exacto) tendríamos $(N + N - 0) / (N + N) = 1$

Q-Gram de 2 parámetros da “similitud” entre 2 strings: 1 es máxima similitud, 0 es nula

Para Q exactamente 2:

¿Cuál es la similitud entre **salesal** y **vale** ?

Justificar el cálculo.

Rta

Q-grams(salesal)= { #s, sa, **al**, **le**, es, sa, al, l#}

Q-grams(vale)= { #v, va, **al**, **le**, e#}.

En común 2 (ojo con los repetidos!).

$Q\text{-Gram}(\text{salesal}, \text{vale}) = (8 + 5 - 9) / (8 + 5) = 0.3076$

Para Q exactamente 2:

¿Cuál es la similitud entre **salesal** y **alale** ?

Justificar el cálculo.

Rta Q-grams(salesal)= { #s, sa, **al**, **le**, es, sa, **al**, l#)

Q-grams(alale)= { #a, **al**, la, **al**, **le**, e#}.

En común 3 (ojo con los repetidos!).

$$Q\text{-Gram}(\text{salesal}, \text{alale}) = (8 + 6 - 8) / (8 + 6) = 0.4285$$

Escribir una aplicación Java que calcule los Q-grams para Q exactamente “N” de un string dado (N genérico).

Deberá contar con un método printTokens() tal que por cada grama encontrado devuelva la cantidad de apariciones.

También debe contar con el método similarity(“”, “”) que devuelva la “similitud” entre 2 strings

Caso de Uso:

```
QGram g= new Qgram(2); // 1, 2, 3 .etc
```

```
g.printTokens("alal");
```

```
    // #a 1
```

```
    // al 2
```

```
    // la 1
```

```
    // l# 1
```

```
...
```

```
double value= g.similarity("salesal", "alale");
```

```
System.out.println(value); //
```

```
...
```

TP 2A- Ejer 10



Implementar Q-Grams
paramétrico y testeos de
unidad.

Cuando implementaron QGrams paramétrico:

```
QGram g= new Qgram(2); // 1, 2, 3 .etc
g.printTokens("alal");           // no importa el orden del output
// #a 1
// al 2
// la 1
// l# 1
...
double value= g.similarity("salesal", "alale");
System.out.println(value); //
...
```

Qué ventajas/desventajas tiene: implementación con ArrayList vs HashMap?

Comparando algoritmos...

Análisis de un caso extremo:

Si quiero la similitud entre 2 frases (por lo menos 2 palabras), en qué tipo de frases resulta muchííísimo mejor Qgrams que Levenshtein/Metaphone/Soundex.

Exhibir un caso.

Resumiendo, los algoritmos representantes que hemos analizado sobre procesamiento de Strings

