Métricas de distancia Levenshtein, N-Gramas y Bigram

Introducción

Al trabajar con sistemas de Procesamiento de Lenguaje Natural (NLP), es frecuente toparnos con la necesidad de comparar palabras o frases entre sí, o de buscar patrones de caracteres dentro de un texto. En muchos casos es de interés encontrar no solamente las coincidencias exactas entre dos cadenas de texto, sino también tener una medida de aproximación o similitud entre éstas cuando la coincidencia no es perfecta.

Para dichos casos resulta de utilidad apoyarse en las medidas de similitud o distancia en cadenas de caracteres, así como para secuencias de palabras o tokens.

Levenshtein

La distancia de Levenshtein es una métrica de edición basada en caracteres, por lo que generalmente se le conoce simplemente como “distancia de edición”, aunque este término se refiere en realidad a una familia más grande de métricas.

Esta distancia puede definirse como el número mínimo de operaciones de edición requeridas para convertir una cadena en la otra, considerando como operaciones válidas el borrado, la inserción o la sustitución. Algunos de los campos de aplicación que utilizan la distancia de Levenshtein son: la corrección ortográfica automática, los sistemas de reconocimiento del habla, el análisis de ADN y en la detección de plagio.

El algoritmo convencional para calcular esta métrica usa programación dinámica, aunque existen diversas adaptaciones y aproximaciones para reducir el costo computacional de su cálculo. En la implementación con programación dinámica, se utiliza una matriz en la que se van almacenando los resultados intermedios, producto de calcular la distancia de Levenshtein, entre los prefijos de las cadenas originales a comparar.

Por ejemplo, la diferencia entre “books” y “back” es de tres.

books -> baoks (sustitución de “o”por “a”)
baoks -> backs (sustituyendo “o” por “c”)
backs -> back (supresión de la “s”)

Teniendo como resultado el pseudocódigo, debido a las ecuaciones para calcular la distancia de Levenshtein, las cuales se muestran a continuación:

Levenshtein Distance Algorithm Pseudo-Code

```
LevenshteinDistanceAlgo (Source, Target)
Int Distance = new Int [Source.Length + 1, Target.Length + 1)
If (Source.Lenght = = 0)
|       return Target.Length
End If
If (Target.Lenght = = 0)
|       return Source.Length
End If
For Int I=0 to Source.Length
|       Distance [I, 0] = I
End For
For Int J=0 to Target.Length
|       Distance [0, J] = J
End For
For Int I=1 to Source.Length
|       For Int J=1 to Target.Lengh
|       |       Int Cost = (Target [J - 1] = = Source [I - 1] ? 0 : 1
|       |       Distance [I, J] = Min (Min (Distance [I - 1, J ] + 1, Distance [I, J - 1] + 1),
|       |       Distance [I - 1, J - 1] + cost)
|       End For
End For
return Distance [Source.Length, Destination.Length]
```

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

N-Gramas

Un n-gramas es un conjunto de n elementos consecutivos en un documento de texto, que puede incluir palabras, números, símbolos y puntuación. Los modelos de n-gramas son útiles en muchas aplicaciones de análisis textual en que la secuencia de palabras es relevante, tales como análisis de sentimiento, clasificación de texto y generación de texto. El modelado de n-gramas es una de las técnicas utilizadas para convertir texto de un formato no estructurado a un formato estructurado.

Sea una secuencia S de elementos ordenados $s_1s_2s_3...s_k...$ se denomina n-grama a cualquier subsecuencia $A=s_{i+1}s_{i+2}...s_{i+n}$, donde i es un valor entre 0 y $|S|$-n para garantizar que la longitud de A sea siempre n o lo que es lo mismo $|A|=n;\ n>1$.

Uno de los usos más interesantes es el de buscar la similitud entre un documento y un grupo de documentos. Este hecho cobra una gran importancia en la estimación del idioma predominante en el texto.

Supóngase que se tienen dos conjuntos de trigramas I_1 e I_2, asociados a los idiomas inglés y español, provenientes de separar en n-gramas de grandes volúmenes de textos en uno y otro idioma, para que $I1$ e $I2$ sean representativos. Luego se tienen el conjunto de n-gramas T de un documento que podría estar escrito en inglés o español. Puede estimarse el idioma predominante mediante la expresión de evaluación de semejanza:

$$f(T_1, T_2) = 2\frac{|T_1 \cap T_2|}{|T_1| + |T_2|}$$

$$Donde\ T_1\ y\ T_2\ son\ conjuntos\ de\ n-gramas$$

Se calculan $f(T, I_1) = t_1$ y $f(T, I_2) = t_2$ y si $t_1 > t_2$ el documento asociado a T está escrito en inglés, de lo contrario está en español. En caso de $t_1 = t_2$ no puede decidirse.

Por lo tanto, de manera general, la estimación de n-gramas por conteo es:

$$f(w_1 | w_{i-n+1}, \dots, w_{i-1}) = \frac{C(w_{i-n+1}, \dots, w_{i-1}, w_i)}{C(w_{i-n+1}, \dots, w_{i-1})}$$

Bigram

El algoritmo de bigrama compara datos en función de la aparición de caracteres consecutivos en ambas cadenas de datos en un par coincidente, buscando pares de caracteres consecutivos que sean comunes a ambas cadenas. Cuanto mayor sea el número de pares idénticos comunes entre las cadenas, mayor será la puntuación de coincidencia. Este algoritmo es útil en la comparación de cadenas de texto largas, como líneas de direcciones de formato libre.

Los Bigramas ayudan a proporcionar la probabilidad condicional de una palabra dada la palabra precedente, cuando la relación de la probabilidad condicional se aplica:

$$P(w_n | w_{n-1}) = \frac{P(w_{n-1}, w_n)}{P(w_{n-1})}$$

Es decir, la probabilidad $P()$ de una palabra w_n dada la palabra w_{n-1} es igual a la probabilidad de su bigrama, o la co-ocurrencia de las dos palabras $P(w_{n-1}, w_n)$, dividido por la probabilidad de que la palabra precedente.

Por ejemplo, si se tienen las siguientes cadenas: larder y lerder. Estás cadenas se agrupan en los siguientes grupos de Bigram respectivamente:

- La, ar, rd, de, er
- Le, er, rd, de, er

Tenga en cuenta que la segunda aparición de la cadena "er" dentro de la cadena "lerder" no coincide, ya que no hay una segunda aparición correspondiente de "er" en la cadena "larder".

Para calcular la puntuación de coincidencia de Bigram, la transformación divide el número de pares coincidentes (6) por el número total de pares en ambas cadenas (10). En este ejemplo, las cadenas son 60% similares y la puntuación de coincidencia es 0,60.

Bibliografía.

- Bhagwan, J. (2015, 1 septiembre). *Design and Analysis of a Levenshtein Distance Based Code Clones Detection Algorithm*. csjournals. http://csjournals.com/IJCSC/PDF7-1/45.%20Jai.pdf
- *Bigram*. (s. f.). Informatica. Recuperado 28 de junio de 2021, de https://docs.informatica.com/data-integration/data-services/10-1/developer-transformation-guide/comparison-transformation/field-matching-strategies/bigram.html
- EcuRed. (s. f.). *N-grama - EcuRed*. Recuperado 28 de junio de 2021, de https://www.ecured.cu/N-grama
- Sharp, B. (2010, 21 marzo). *Informatica Data Quality Workbench Matching Algorithms*. The Data Quality Chronicle. https://dqchronicle.wordpress.com/2009/12/10/informatica-data-quality-workbench-matching-algorithms/#:%7E:text=The%20bigram%20algorithm%20matches%20data,the%20higher%20the%20match%20score.
- Sobrino, D. C. (2019, 19 diciembre). *Métricas de similitud para cadenas de texto. Parte II: Métricas basadas en operaciones de edición.* SoldAI. https://soldai.com/metricas-operaciones-edicion/