



2



Implementando un corrector ortográfico en Python, utilizando la distancia de Levenshtein



Miguel Ángel Flores
Apr 18, 2019 · 3 min read



¿Qué es la distancia de Levenshtein?

Supongamos que vamos a escribir la palabra *algoritmo*, pero hemos cometido algunos errores y en su lugar escribimos *algratmo*.



¿Cuál es el número mínimo de operaciones (eliminaciones, inserciones o sustituciones) requeridas para que *algratmo* sea igual a *algoritmo*.

Si $x = \text{algratmo}$ y $z = \text{algoritmo}$:

$DL(x,z) = 2$, ya que necesitamos sustituir “t” por “o” y “a” por “i”.

Un corrector ortográfico medirá esta distancia entre 2 palabras, mientras mayor sea la distancia de Levenshtein, mayor será su diferencia. De manera que buscará en un diccionario aquellas palabras cuya distancia sea menor, y las mostrará como candidatas para el reemplazo.

Implementando nuestro autocorrector en Python

No voy a reinventar la rueda creando el código desde cero. Ya hay toda una comunidad detrás que lo ha optimizado.

Por acá puedes ver el código para Python y otros lenguajes de programación:

How to Write a Spelling Corrector
I figured they, and others, could benefit from an explanation. The full details of an industrial-strength spell...
www.norvig.com

Como nosotros estamos interesados en Python, utilizaremos el siguiente:

```
1 import re
2 from collections import Counter
3
4 def words(text): return re.findall(r'\w+', text.lower())
5
6 WORDS = Counter(words(open('big.txt').read()))
7
8 def P(word, N=sum(WORDS.values())):
9     "Probability of `word`."
10    return WORDS[word] / N
11
12 def correction(word):
13     "Most probable spelling correction for word."
14     return max(candidates(word), key=P)
15
16 def candidates(word):
17     "Generate possible spelling corrections for word."
18     return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])
19
20 def known(words):
21     "The subset of `words` that appear in the dictionary of WORDS."
22     return set(w for w in words if w in WORDS)
23
24 def edits1(word):
25     "All edits that are one edit away from `word`."
26     letters = 'abcdefghijklmnopqrstuvwxyz'
27     splits = [(word[:i], word[i:])] for i in range(len(word) + 1)
28     deletes = [L + R[1:] for L, R in splits if R]
29     transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R) > 1]
30     replaces = [L + c + R[1:] for L, R in splits if R and c in letters]
31     inserts = [L + c + R for L, R in splits for c in letters]
32     return set(deletes + transposes + replaces + inserts)
33
34 def edits2(word):
35     "All edits that are two edits away from `word`."
36     return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

El algoritmo se alimenta de un diccionario de palabras llamado “big.txt”, tu puedes crear tu propio diccionario de palabras o descargar alguno en el idioma que desees y adaptarlo a tus necesidades. Procura nombrar correctamente el archivo y guardarlo en la misma carpeta que tu notebook.

Aquí puedes descargar el diccionario usado en la página antes mostrada, se trata del libro de “Las aventuras de Sherlock Holmes” en su versión en inglés.

http://www.norvig.com/big.txt

Como yo quise implementarlo en español, lo alimenté con un libro en español, las “21 leyes de liderazgo” de John C. Maxwell. Aquí puedes descargarlo :

21 LEYES DEL LIDERAZGO - JOHN C. MAXWELL.txt
Edit description
drive.google.com

Si vas a usar este libro como diccionario, recuerda cambiar el nombre del archivo a “big.txt”.

Implementando el código

Una ves que ya tengas el código en el notebook, basta con ejecutar la función *correction(palabra)* y el autocorrector hará lo suyo.

```
In [8]: correction('nafuraleza')
Out[8]: 'naturaliza'
```

```
In [9]: correction('ptrsona')
Out[9]: 'persona'
```

Un ejemplo más:

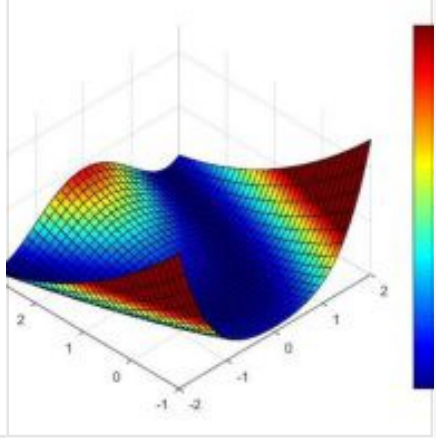
```
In [10]: correction('algratmo')
Out[10]: 'algratmo'
```

En este caso nuestro autocorrector no fue capaz de encontrar una palabra adecuada, se corregiría agregando dicha palabra (algoritmo) al diccionario.

Hasta aquí llegamos con este post.

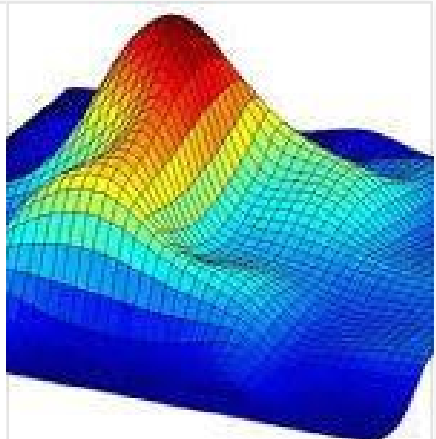
Te invito a seguir el blog que he creado para compartir artículos de programación con Python, Matlab y R con enfoque en ciencias:

Programación para ingeniería y ciencias
 Un blog dedicado a la programación con Python, Matlab y R con fines científicos
 programandomate.wordpress.com



Así mismo te invito a seguirme en facebook:

Programación para ingeniería y ciencias
 Programación para ingeniería y ciencias. 15 likes · 1 talking about this.
 Página dedicada a la difusión de información...
 www.facebook.com



Python Programming
 Naturallanguageprocessing
 Mathematics
 Algoritmos
 Python



2 claps



WRITTEN BY

Miguel Ángel Flores

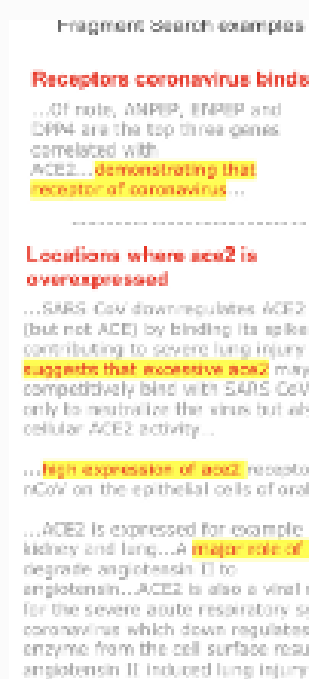
Ing. Civil dedicado a la geomática[Aficionado a la computación científica]

Follow

Write the first response

More From Medium

Also tagged Naturallanguageprocessing



Document search with fragment embeddings

Ajit Rajasekharan in Towards Data...
Apr 7 · 12 min read



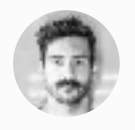
31



Also tagged Python



Optical Character Recognition (OCR) with less than 12 Lines of Code using Python



Hucker Marius in Towards Data...
Apr 9 · 3 min read



4



Also tagged Mathematics



Convergence of Reinforcement Learning Algorithms



Nathan Lambert in Towards Data...
Apr 9 · 9 min read



7



Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage — with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. Upgrade