Quantmetry
Building AI with pioneers

Fuzzy Matching with FlashText

Rémi Adon
ML Engineer

Plan

1) Le Fuzzy Matching, R&D et DataScience au quotidien
2) FlashText, introduction
3) Levenshtein, encore et encore
4) Limites et améliorations possibles

Fuzzy Matching au quotidien

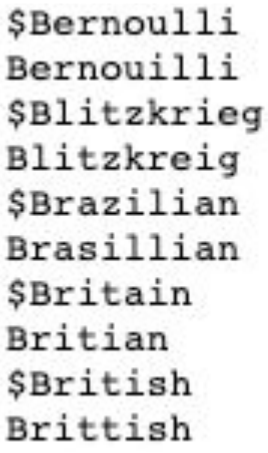
Recherche/correspondance approximative
Nettoyer des données clients

Vu sur beaucoup de missions + R&D en NLP
Données non structurées + pd.Series

Workflow classique ?

- ***Linear scan*** en utilisant une fonction de distance
  - n keywords, n*dist ∀ entrée
- **Regex**
  - texte entier …
  - nombre de mots clés …

« Brit »



```
$Bernoulli
Bernouilli
$Blitzkrieg
Blitzkreig
$Brazilian
Brasillian
$Britain
Britian
$British
Brittish
```

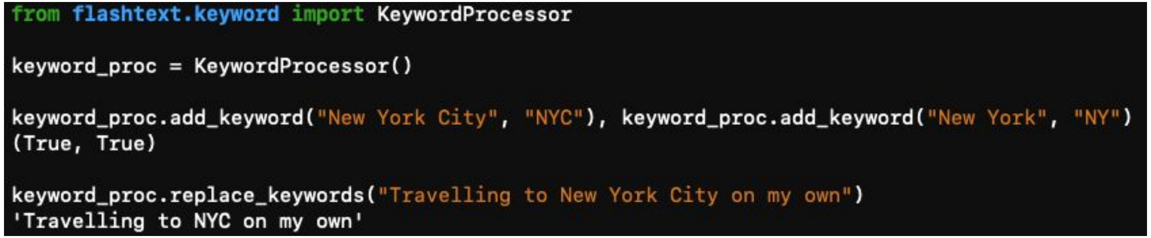
wikipedia dataset

Peut-on faire mieux, et rendre cette tâche moins fastidieuse ?
Peut-on trouver une méthode générique, qui d'adapter si besoin ?

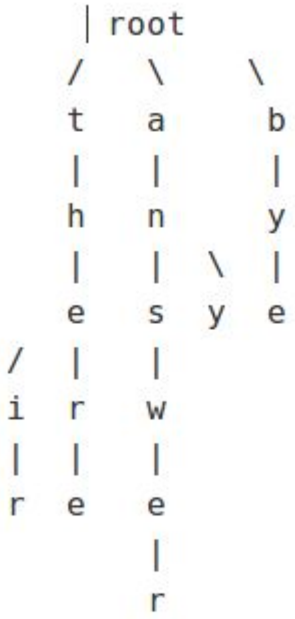
FlashText, Introduction

Aho-Corasik + Trie Dict → O(n)

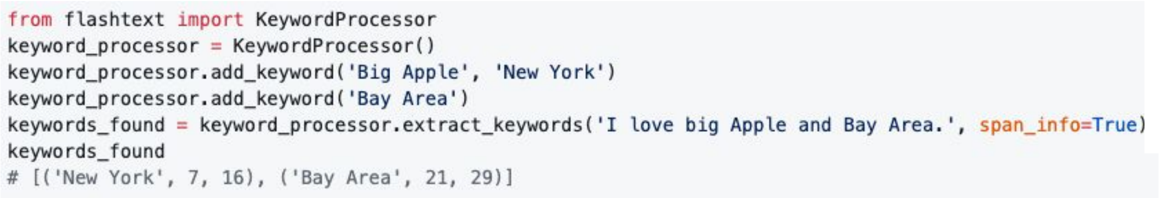
Keyword = un élément, composé de plusieurs mots (words)
Conçu pour extraire/remplacer **le keyword le plus long**



```
from flashtext.keyword import KeywordProcessor

keyword_proc = KeywordProcessor()

keyword_proc.add_keyword("New York City", "NYC"), keyword_proc.add_keyword("New York", "NY")
(True, True)

keyword_proc.replace_keywords("Travelling to New York City on my own")
'Travelling to NYC on my own'
```

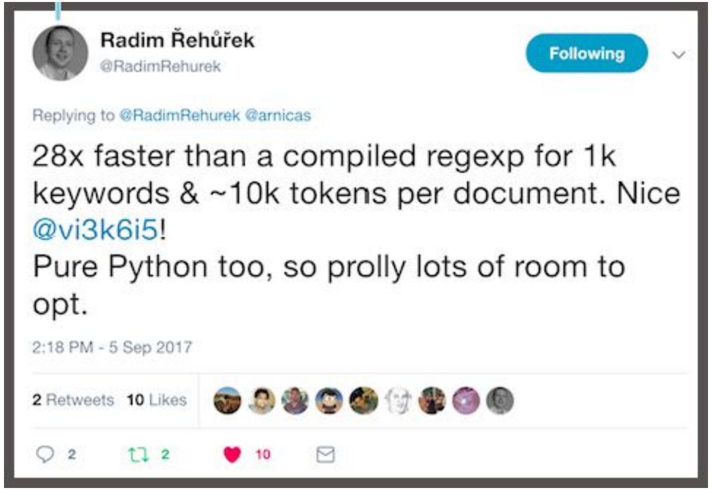


Informations additionnelles lors de l'extraction, comme les Regex



```
from flashtext import KeywordProcessor
keyword_processor = KeywordProcessor()
keyword_processor.add_keyword('Big Apple', 'New York')
keyword_processor.add_keyword('Bay Area')
keywords_found = keyword_processor.extract_keywords('I love big Apple and Bay Area.', span_info=True)
keywords_found
# [('New York', 7, 16), ('Bay Area', 21, 29)]
```

FlashText, Performances



Radim Řehůřek
@RadimRehurek

Following

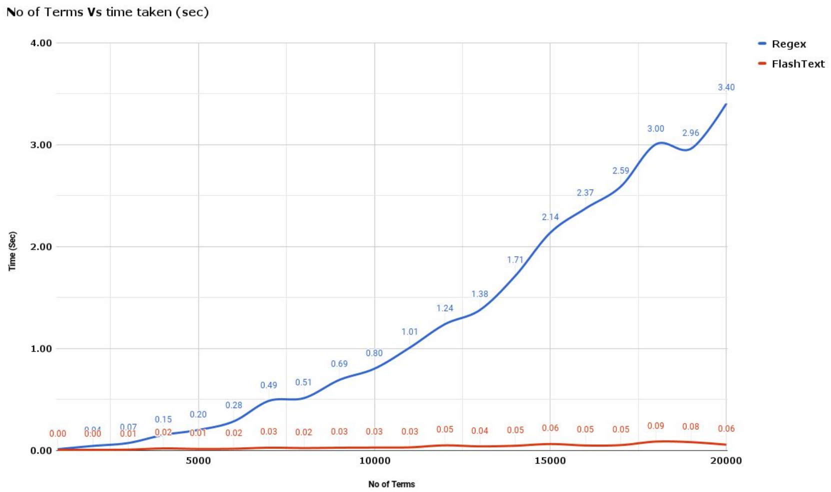
Replying to @RadimRehurek @arnicas

28x faster than a compiled regexp for 1k keywords & ~10k tokens per document. Nice @vi3k6i5!
Pure Python too, so prolly lots of room to opt.

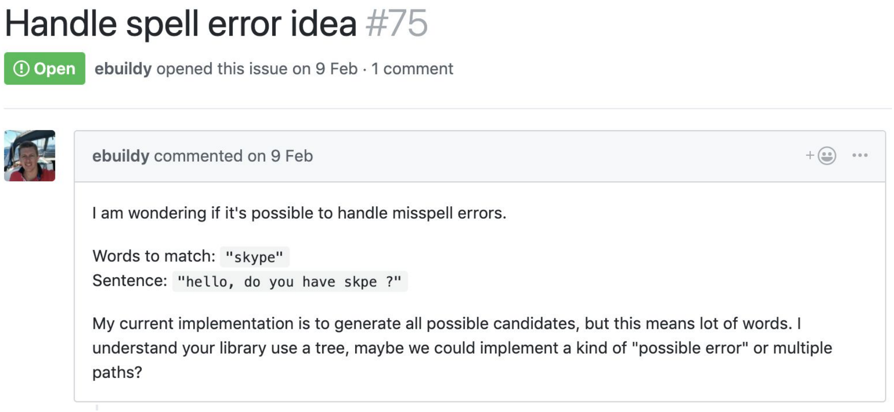
2:18 PM - 5 Sep 2017

2 Retweets 10 Likes

2 2 10



Issues github



Handle spell error idea #75

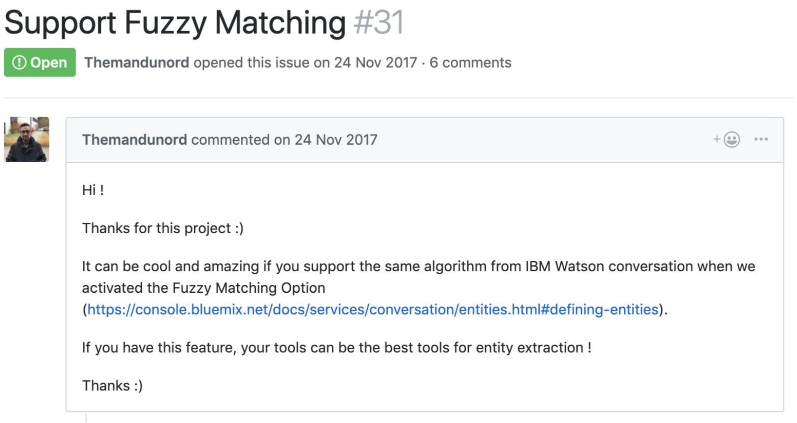
ⓘ Open ebuildy opened this issue on 9 Feb · 1 comment

ebuildy commented on 9 Feb

I am wondering if it's possible to handle misspell errors.

Words to match: `"skype"`
Sentence: `"hello, do you have skpe ?"`

My current implementation is to generate all possible candidates, but this means lot of words. I understand your library use a tree, maybe we could implement a kind of "possible error" or multiple paths?



Support Fuzzy Matching #31

ⓘ Open Themandunord opened this issue on 24 Nov 2017 · 6 comments

Themandunord commented on 24 Nov 2017

Hi !

Thanks for this project :)

It can be cool and amazing if you support the same algorithm from IBM Watson conversation when we activated the Fuzzy Matching Option (https://console.bluemix.net/docs/services/conversation/entities.html#defining-entities).

If you have this feature, your tools can be the best tools for entity extraction !

Thanks :)

Comment conserver au maximum les **performances** ?
Se reposer au maximum sur l’existant

Levenshtein, encore et encore

3 opérations

- Insertion : cou □ coup
- Suppression : coup □ cou
- Substitution : coup □ cout

Max cost = 1

« Mar »

« Marin » ?

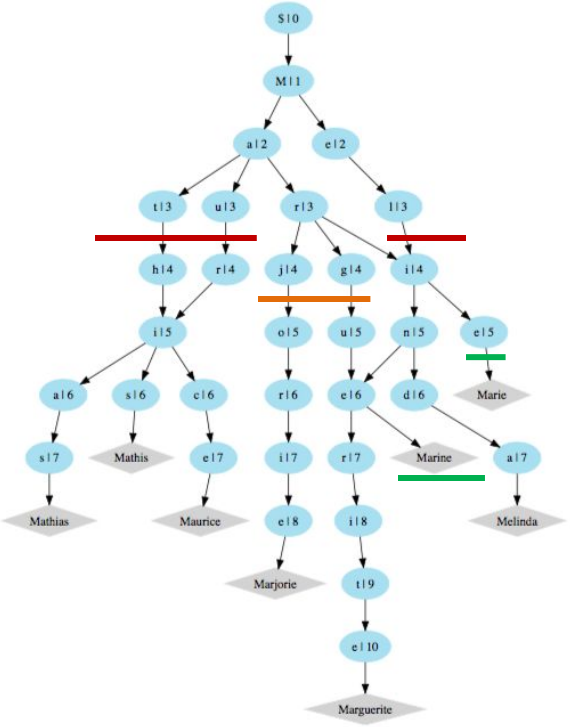
« Marina »

32 nœuds dans l’arbre

~20 non observés pendant la recherche

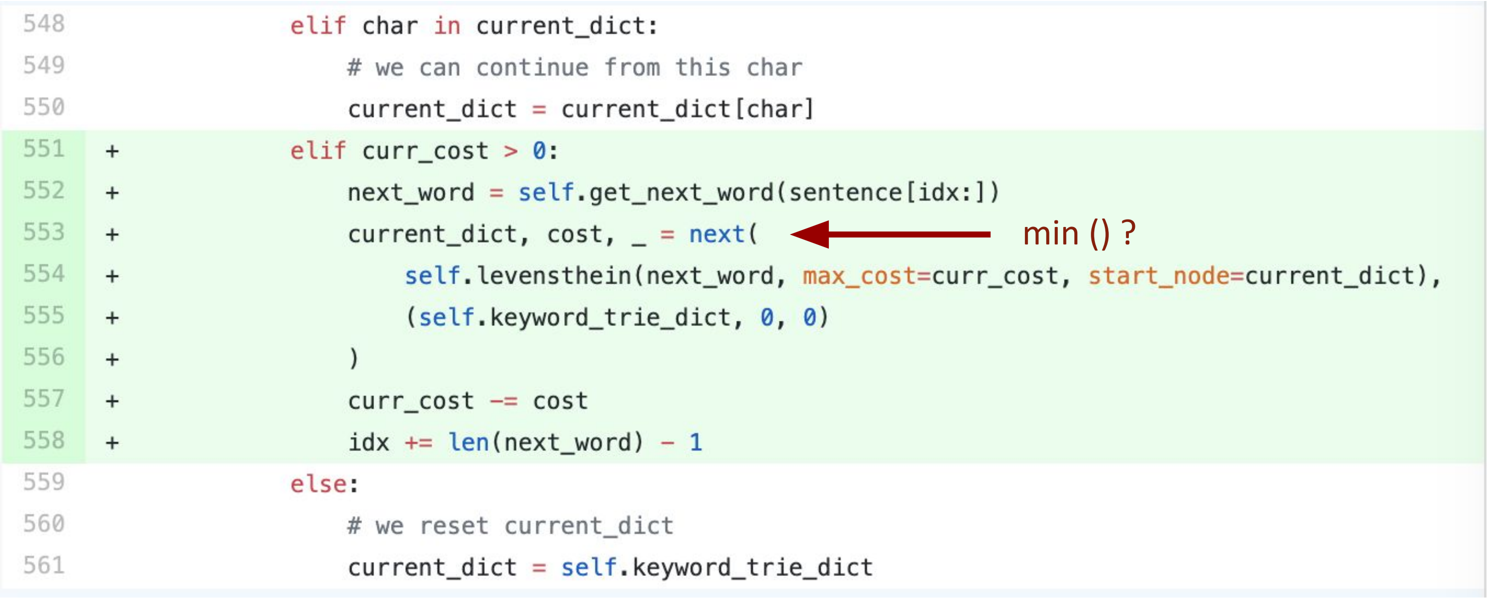
Pourquoi Levenshtein ?

- **Intelligible**
- Debuggable
- Facilement **adaptable**



Levenshtein, minimum locaux

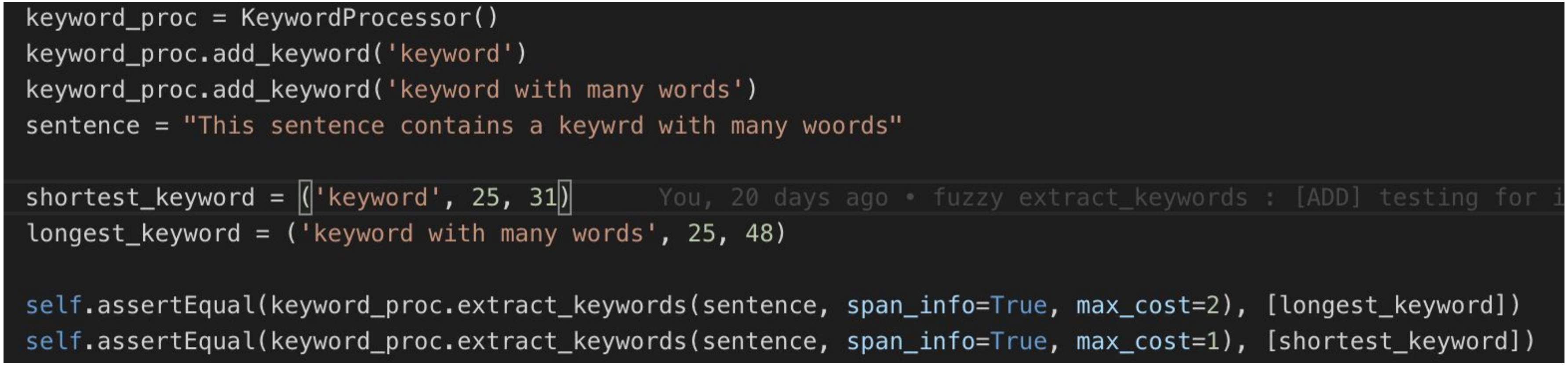
Levensthein(« string », « strong ») == levenshtein(« ing», « ong»)



```
            elif char in current_dict:
                # we can continue from this char
                current_dict = current_dict[char]
+           elif curr_cost > 0:
+               next_word = self.get_next_word(sentence[idx:])
+               current_dict, cost, _ = next(
+                   self.levensthein(next_word, max_cost=curr_cost, start_node=current_dict),
+                   (self.keyword_trie_dict, 0, 0)
+               )
+               curr_cost -= cost
+               idx += len(next_word) - 1
            else:
                # we reset current_dict
                current_dict = self.keyword_trie_dict
```

min () ?

Levenshtein, exemples simples

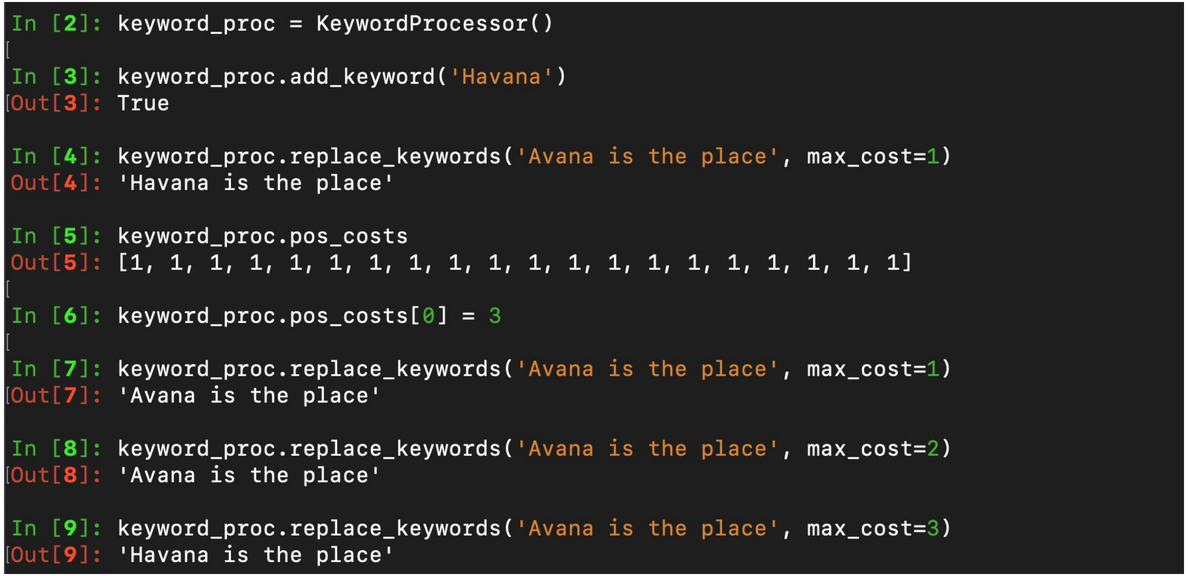


```
keyword_proc = KeywordProcessor()
keyword_proc.add_keyword('keyword')
keyword_proc.add_keyword('keyword with many words')
sentence = "This sentence contains a keywrd with many woords"

shortest_keyword = ('keyword', 25, 31)      You, 20 days ago • fuzzy extract_keywords : [ADD] testing for i
longest_keyword = ('keyword with many words', 25, 48)

self.assertEqual(keyword_proc.extract_keywords(sentence, span_info=True, max_cost=2), [longest_keyword])
self.assertEqual(keyword_proc.extract_keywords(sentence, span_info=True, max_cost=1), [shortest_keyword])
```

Limites et améliorations possibles

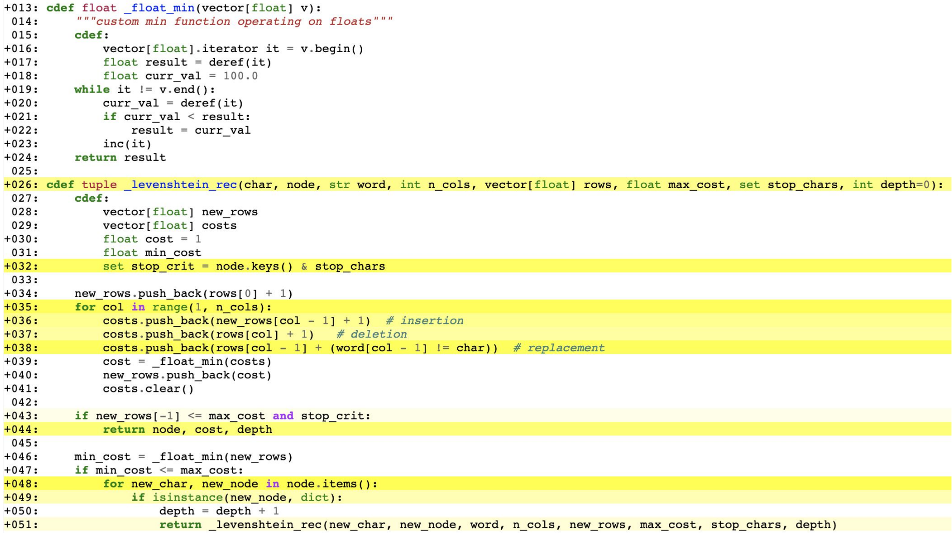
PR toujours en attente …

- Poids custom additions/suppressions/remplacements (défaut à 1)
- Cython
- Poids positionnels
- Benchs dédiés

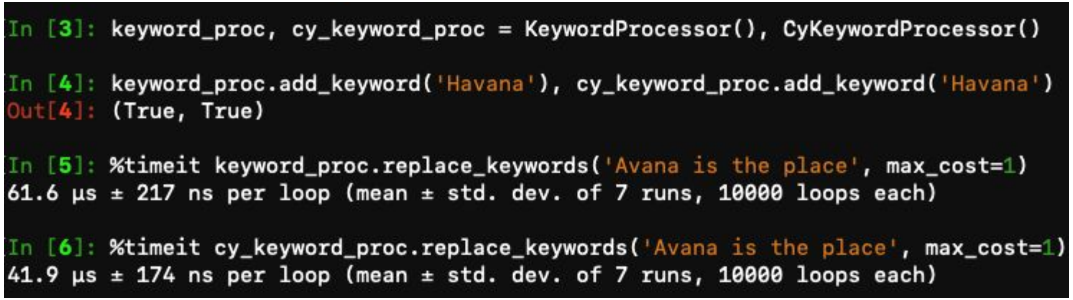


```
In [2]: keyword_proc = KeywordProcessor()

In [3]: keyword_proc.add_keyword('Havana')
Out[3]: True

In [4]: keyword_proc.replace_keywords('Avana is the place', max_cost=1)
Out[4]: 'Havana is the place'

In [5]: keyword_proc.pos_costs
Out[5]: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

In [6]: keyword_proc.pos_costs[0] = 3

In [7]: keyword_proc.replace_keywords('Avana is the place', max_cost=1)
Out[7]: 'Avana is the place'

In [8]: keyword_proc.replace_keywords('Avana is the place', max_cost=2)
Out[8]: 'Avana is the place'

In [9]: keyword_proc.replace_keywords('Avana is the place', max_cost=3)
Out[9]: 'Havana is the place'
```

Cython



```
cdef float _float_min(vector[float] v):
    """custom min function operating on floats"""
    cdef:
        vector[float].iterator it = v.begin()
        float result = deref(it)
        float curr_val = 100.0
    while it != v.end():
        curr_val = deref(it)
        if curr_val < result:
            result = curr_val
        inc(it)
    return result

cdef tuple _levenshtein_rec(char, node, str word, int n_cols, vector[float] rows, float max_cost, set stop_chars, int depth=0):
    cdef:
        vector[float] new_rows
        vector[float] costs
        float cost = 1
        float min_cost
        set stop_crit = node.keys() & stop_chars

    new_rows.push_back(rows[0] + 1)
    for col in range(1, n_cols):
        costs.push_back(new_rows[col - 1] + 1)  # insertion
        costs.push_back(rows[col] + 1)    # deletion
        costs.push_back(rows[col - 1] + (word[col - 1] != char))  # replacement
        cost = _float_min(costs)
        new_rows.push_back(cost)
        costs.clear()

    if new_rows[-1] <= max_cost and stop_crit:
        return node, cost, depth

    min_cost = _float_min(new_rows)
    if min_cost <= max_cost:
        for new_char, new_node in node.items():
            if isinstance(new_node, dict):
                depth = depth + 1
                return _levenshtein_rec(new_char, new_node, word, n_cols, new_rows, max_cost, stop_chars, depth)
```



```
In [3]: keyword_proc, cy_keyword_proc = KeywordProcessor(), CyKeywordProcessor()

In [4]: keyword_proc.add_keyword('Havana'), cy_keyword_proc.add_keyword('Havana')
Out[4]: (True, True)

In [5]: %timeit keyword_proc.replace_keywords('Avana is the place', max_cost=1)
61.6 µs ± 217 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)

In [6]: %timeit cy_keyword_proc.replace_keywords('Avana is the place', max_cost=1)
41.9 µs ± 174 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

MERCI

Quantmetry
Building AI with pioneers

Quantmetry
52 rue d’Anjou
75008 Paris

SIRET : 53117239300039