2020_edit

January 1, 2023

1 Algo - distance d'édiction

La distance d'édition ou distance de Levenshtein permet de calculer une distance entre deux mots et par extension entre deux séquences.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

[2]:
```
%matplotlib inline
```

1.1 Enoncé

1.1.1 Q1 : Distance simple entre deux mots de même longueur

Une distance entre deux mots… c'est plus simple si les deux mots ont la même longueur, on calcule le nombre de différences.

[3]:
```

```

1.1.2 Q2 : Distance simple entre deux mots de longueur différente

On construit cette distance comme la différence des longueurs + la distance entre le mot le plus court et toutes les sous-séquences de même longueur issues de la chaîne la plus longue.

[4]:
```

```

1.1.3 Q3 : Distance alambiquée...

Cette fois-ci, on coupe chacun des deux mots en deux, au hasard. On applique la distance précédente sur chacun des deux tronçons. On fait la somme. Il ne reste plus qu'à minimiser cette somme sur l'ensemble des coupures possibles.

[5]:
```

```

1.1.4 Q4 : implémenter l'algorithme de la page wikipedia

Levenshtein

[6]:
```

```

[7]:
```

```

1.2 Réponses

1.2.1 Q1 : Distance simple entre deux mots de même longueur

Une distance entre deux mots… c'est plus simple si les deux mots ont la même longueur, on calcule le nombre de différences.

[8]:
```
def distance_meme_longueur(m1, m2):
    if len(m1) != len(m2):
        raise ValueError("m1 et m2 sont de longueurs différentes")
    d = 0
    for c1, c2 in zip(m1, m2):
        if c1 != c2:
            d += 1
    return d

distance_meme_longueur('abcef', 'abcde')
```

[8]:
```
2
```

On vérifie que la fonctionne jette bien une exception lorsque les chaînes de caractères sont de longueurs différentes.

[9]:
```
try:
    distance_meme_longueur('a', 'bb')
except Exception as e:
    print(e)
```

```
m1 et m2 sont de longueurs différentes
```

1.2.2 Q2 : Distance simple entre deux mots de longueur différente

On construit cette distance comme la différence des longueurs + la distance entre le mot le plus court et toutes les sous-séquences de même longueur issues de la chaîne la plus longue.

[10]:
```
def distance(m1, m2):
    if len(m1) < len(m2):
        return distance(m2, m1)
    if len(m1) == len(m2):
        return distance_meme_longueur(m1, m2)
    d = len(m1) - len(m2)
    mind = [distance_meme_longueur(m1[i:i+len(m2)], m2)
            for i in range(0, d)]
    return d + min(mind)

distance('aa', 'aa'), distance('aa', 'aaa'),  distance('aa', 'bbb')
```

[10]:
```
(0, 1, 3)
```

1.2.3 Q3 : Distance alambiquée...

Cette fois-ci, on coupe chacun des deux mots en deux, au hasard. On applique la distance précédente sur chacun des deux tronçons. On fait la somme. Il ne reste plus qu'à minimiser cette somme sur l'ensemble des coupures possibles.

[11]:
```
def distance_alambiquee(m1, m2):
    mini = None
```

```
    for i in range(len(m1)):
        for j in range(len(m2)):
            d = distance(m1[:i], m2[:j]) + distance(m1[i:], m2[j:])
            if mini is None or d < mini:
                mini = d
            # Option verlan
            d = distance(m1[:i], m2[j:]) + distance(m1[i:], m2[:j]) + 0.5
            if d < mini:
                mini = d
    return mini

(distance('abc', 'ac'),
 distance_alambiquee('abc', 'ac'),
 distance_alambiquee('abc', 'ca'),
 distance_alambiquee('b', 'b'))
```

[11]: (2, 1, 1.5, 0)

[12]:

1.2.4 Q4 : implémenter l'algorithme de la page wikipedia

Levenshtein
La première implémentation reprend l'algorithme décrit sur la page wikipédia.

[13]:
```
def levenstein(m1, m2):
    d = {}
    d[0,0] = 0
    for i in range(len(m1) + 1):
        d[i, 0] = i
    for j in range(len(m2) + 1):
        d[0, j] = j
    for i in range(1, len(m1) + 1):
        for j in range(1, len(m2) + 1):
            d[i, j] = min(d[i-1,j] +1, d[i,j-1] +1,
                d[i-1, j-1] + (1 if m1[i-1] != m2[j-1] else 0))
    return d[len(m1), len(m2)]

levenstein('abc', 'ac')
```

[13]: 1

La seconde version est plus alambiquée, elle modifie légèrement la version alambiquée. C'est une version récursive.

[14]:
```
def distance_alambiquee_levenstein(m1, m2):
    mini = None
    for i in range(len(m1)):
        for j in range(len(m2)):
            if i > 0 and i < len(m1) - 1 and j > 0 and j < len(m2) - 1:
                d1 = distance_alambiquee_levenstein(m1[:i], m2[:j])
                d2 = distance_alambiquee_levenstein(m1[i:], m2[j:])
            else:
                d1 = distance(m1[:i], m2[:j])
                d2 = distance(m1[i:], m2[j:])
```

```
            d = d1 + d2
            if mini is None or d < mini:
                mini = d
    return mini

(distance_alambiquee('abcde', 'ace'),
 levenstein('abcde', 'ace'),
 distance_alambiquee_levenstein('abcde', 'ace'))
```

[14]: (3, 2, 2)

[15]: