word_transformation

September 6, 2022

[ ]:
```python
def hamming_distance(word1: str, word2: str) -> int:
    """_summary_
        calculate hamming distance from 2 strings

    Args:
        word1 (str): 1st word
        word2 (str): 2nd word

    Returns:
        int: -1 if word1 and word2 have not the same length
            otherwise the hamming distance between them
    """
    distance = -1
    if (len(word1) != len(word2)):
        return distance
    else:
        distance = i = 0
        while i < len(word1):
            distance += int(word1[i] != word2[i])
            i += 1

        return distance



def hamming_distance_from_lists(list1: list, list2: list) -> list:
    """_summary_
        calculate hamming distance of 2 list of words

    Args:
        list1 (list): 1st list of words
        list2 (list): 2nd list of words

    Returns:
        list: list of hamming distance of the 2 lists
    """
    return (list(map(hamming_distance, list1, list2)))
```

```
print(hamming_distance("toto", "tita"))

list1 = ['LIGNE', 'BOOLE', 'POLICE', 'PASSION', 'CRANE']
list2 = ['LIANE', 'MOORE', 'PILOTE', 'RATIONS', 'ECRAN']

print(hamming_distance_from_lists(list1=list1, list2=list2))
```

```
2
[1, 2, 3, 6, 5]
```

[ ]:
```
list3 = ['SIGNE', 'SIGNE', 'SIGNE', 'SIGNE', 'SIGNE']
list4 = ['SUITE', 'LIGNE', 'SINGE', 'DIGNE', 'MIXTE']


print(hamming_distance_from_lists(list1=list3, list2=list4))
```

```
[3, 1, 2, 1, 3]
```

[ ]:
```
import pandas as pd
import numpy as np

def is_levenshtein_grey_box(str1: str, str2:str) -> list:
    """_summary_
    levenshtein grey box : compare 2 characters return (-1, 'g') if equal and (-1, 'w') otherwize

    Args:
        str1 (str): first character
        str1 (str): second character

    Returns:
        list: [-1, 'g'] if str1 == str2, otherwize [-1, 'w']
    """
    grey_white = ['w', 'g']
    return [-1, grey_white[int(str1==str2)]]

def create_levenshtein_matrix(word1: str, word2 :str) -> pd.DataFrame:
    """_summary_
    create the Levenshtein matrix from word1 and word2

    Args:
        word1 (str): 1st word
        word2 (str): 2nd word

    Returns:
```

```
        pd.DataFrame: the matrix(a DataFrame with in row the word1 and in␣
↪column word2)
    """

    list_word1 = list(word1)
    list_word2 = list(word2)

    list1 = ['Col Index'] + list_word1
    list2 = ['Row Index'] + list_word2
    df = pd.DataFrame(columns=list2, index=list1)

    #Init grey blocks
    for char in list_word2:
        df[char] = list(map(is_levenshtein_grey_box, [char]*len(list1), list1))

    #Init first row and col
    col1 = list(map(list,zip(range(0, len(word1)+1), (['w']*(len(word1)+1)))))
    row1 = list(map(list,zip(range(0, len(word2)+1), (['w']*(len(word2)+1)))))

    df['Row Index'] = col1
    df.loc['Col Index']
    df.loc['Col Index'] = row1

    #calculate levenshtein matrix while applying minimum rule
    #Minimum rule:
    #fill the 4th cell of  (2×2) table with :
    # • if the 4th cell is 'w' then we add + 1 to all 3 other numbers and we␣
↪took the minimum of the 3 numbers
    # • if the 4th cell is 'g' then we add +1 only to the cell above and on the␣
↪left and we took the minimum of the 3 numbers
    nparray = df.to_numpy()
    ro, col = nparray.shape

    i = j = 0
    while i < ro-1:
        while j < col-1:
            c_ij = nparray[i][j][0]
            c_ij1 = nparray[i][j+1][0]
            c_i1j = nparray[i+1][j][0]
            gw_i1j1 = nparray[i+1][j+1][1]

            if gw_i1j1 == 'w':
                c_i1j1 = min(c_ij+1, c_ij1+1, c_i1j+1)
            else:
                c_i1j1 = min(c_ij, c_ij1+1, c_i1j+1)

            df.iat[i+1, j+1][0] = c_i1j1
```

```
            j += 1
        j=0
        i += 1

    return df


def get_length_of_shortest_path(levenshtein_matrix:pd.DataFrame) -> int:
    """_summary_
    return the length of the shortest path (must be initialized)
    Args:
        levenshtein_matrix (df): the levenshtein matrix

    Returns:
        int: the length of the shortest path
    """
    return levenshtein_matrix.iat[-1,-1][0]

df = create_levenshtein_matrix(word1='end_word', word2='begin_word')
length_of_shortest_path = get_length_of_shortest_path(df)

print(df)
print('The shortest path is:', length_of_shortest_path)
```

```
          Row Index       b       e       g       i       n       _       w  \
Col Index    [0, w]  [1, w]  [2, w]  [3, w]  [4, w]  [5, w]  [6, w]  [7, w]
e            [1, w]  [1, w]  [1, g]  [2, w]  [3, w]  [4, w]  [5, w]  [6, w]
n            [2, w]  [2, w]  [2, w]  [2, w]  [3, w]  [3, g]  [4, w]  [5, w]
d            [3, w]  [3, w]  [3, w]  [3, w]  [3, w]  [4, w]  [4, w]  [5, w]
_            [4, w]  [4, w]  [4, w]  [4, w]  [4, w]  [4, w]  [4, g]  [5, w]
w            [5, w]  [5, w]  [5, w]  [5, w]  [5, w]  [5, w]  [5, w]  [4, g]
o            [6, w]  [6, w]  [6, w]  [6, w]  [6, w]  [6, w]  [6, w]  [5, w]
r            [7, w]  [7, w]  [7, w]  [7, w]  [7, w]  [7, w]  [7, w]  [6, w]
d            [8, w]  [8, w]  [8, w]  [8, w]  [8, w]  [8, w]  [8, w]  [7, w]

                o       r        d
Col Index  [8, w]  [9, w]  [10, w]
e          [7, w]  [8, w]   [9, w]
n          [6, w]  [7, w]   [8, w]
d          [6, w]  [7, w]   [7, g]
_          [6, w]  [7, w]   [8, w]
w          [5, w]  [6, w]   [7, w]
o          [4, g]  [5, w]   [6, w]
r          [5, w]  [4, g]   [5, w]
d          [6, w]  [5, w]   [4, g]
The shortest path is: 4
```