[Day-01-Lecture-04]

관련 논문

> **권기성, 최운호, 김동건 (2022), "문학 작품의 거리 측정을 활용한 야담의 이본 연구",「한국고전연구 57집」87~120쪽, 한국고전연구학회.**

Levenshtein Distance

- Minimum Edit Distance
- Dynamic Programming
    - 최적의 해를 찾기 위한 동적 계획법(dynamic programming)을 사용하려고 합니다.
    - 자연어처리(nlp)에서는 tabular parsing 또는 CYK algorithm이라고도 소개되어 있습니다.
    - 동적 프로그래밍은 하나의 큰 문제를 여러 개의 작은 문제로 나누어서 그 결과를 저장하여 다음 계산에 활용하는 방식으로 계산 부담을 줄여 줍니다.
    - Levenshtein Distance 또는 Minimum Edit Distance는 Source String을 Target String으로 변환할 때 Insertion/Deletion/Substitution 연산을 적용하여 가장 적은 비용이 드는 경로를 찾는 방법을 사용합니다.
- okssw_02 폴더의 파이썬 프로그래밍을 사용하여서 두 스트링의 거리를 LD (Levenshtein Distance)를 사용하여서 적용해 봅시다.
- 분절음(segmental sound), 문자(character) 단위의 비교가 아닌 feature를 사용한다든지 함으로써 Insertion/Deletion의 Level을 feature 단위로 다루려면 distance metric을 직접 적용한 자신만의 모듈을 만들어야 합니다.
- 우리 수업에서는 이렇게 자세히 들어가지 않으니 difflib를 사용하도록 하겠습니다.

```
$> pip install difflib
```

```
#! lv_test.py
#
#!-*-coding: utf-8-*-
#

from difflib import ndiff

def lv_align(str1, str2):
    result = ""
    pos, removed = 0, 0

    for x in ndiff(str1, str2):
        if pos < len(str1) and str1[pos] == x[2]:
            pos += 1
            result += x[2]
            if x[0] == "-":
```

```
                removed += 1
            continue
        else:
            if removed > 0:
                removed -= 1
            else:
                result += "-"
    return result

def lv_align_max(str1, str2):
    res01 = lv_align(str1, str2)
    res02 = lv_align(str2, str1)

    len01 = len(res01)
    len02 = len(res02)

    if (len01 > len02):
        return res01

    return res02


def iterative_levenshtein(s, t):

    rows = len(s) + 1
    cols = len(t) + 1
    dist = [ [0 for x in range(cols)] for x in range(rows)]

    # source prefixed can be transformed into empty strings
    # by deletions:

    for i in range(1, rows):
        dist[i][0] = i

    # target prefixes can be created from an empty source string
    # by inserting the characters

    for i in range(1, cols):
        dist[0][i] = i

    for col in range(1, cols):
        for row in range(1, rows):
            if s[row-1] == t[col-1]:
                cost = 0
            else:
                cost = 1
            dist[row][col] = min(dist[row-1][col] + 1,     # deletion
                    dist[row][col-1] + 1,          # insertion
                    dist[row-1][col-1] + cost)   # substitution

    return dist[row][col]


if __name__ == "__main__":

```

```
    str01 = "Korea"
    str02 = "Koreea"
    print("Distance: ", iterative_levenshtein(str01, str02), sep="\t")
    res01 = lv_align("Korea", "Koreea")
    res02 = lv_align("Koreea", "Korea")
    print(len(res01), res01)
    print(len(res02), res02)
    print("*" * 80)


    """
    str01 = "橘屬木從登聲"
    str02 = "橘屬从木登聲"
    print("Distance: ", iterative_levenshtein(str01, str02), sep="\t")
    res01 = lv_align(str01, str02)
    res02 = lv_align(str02, str01)
    print(len(res01), res01)
    print(len(res02), res02)
    print("*" * 80)
    """


    """
    str01 = "與之相狎恩情之篤如山如海"
    str02 = "與之相狎恩情如山女如海"
    print("Distance: ", iterative_levenshtein(str01, str02), sep="\t")
    res01 = lv_align(str01, str02)
    res02 = lv_align(str02, str01)
    print(len(res01), res01)
    print(len(res02), res02)
    """
```